UFME7K-15-M Lecture 10 11/04/2018

Evolutionary Computing

Dr J. Charles Sullivan

What is Evolutionary Computing?

- the collective name for a range of problem-solving techniques based on principles of biological evolution, such as *natural selection* and *genetic inheritance* (Eiben & Smith)
- The algorithms which implement specific instances of these techniques are known as Evolutionary Algorithms (EA)
- There are many varieties of EA
  - GA Genetic Algorithms (the most well-known and widely used)
  - ES Evolution Strategies (more specialised for numerical optimisation)
  - GP Genetic Programming
  - EP Evolutionary Programming
  - And many more ...

Elements of an EA

- A population of candidate solutions or hypotheses
- Encoding
  - Binary string
  - Real number string
  - Trees or other structures
- Evaluation
  - Assign “fitness” to individuals in a population
  - Multiple objectives
- Genetic operators
  - Selection
  - Recombination
  - mutation

Important characteristics of EAs

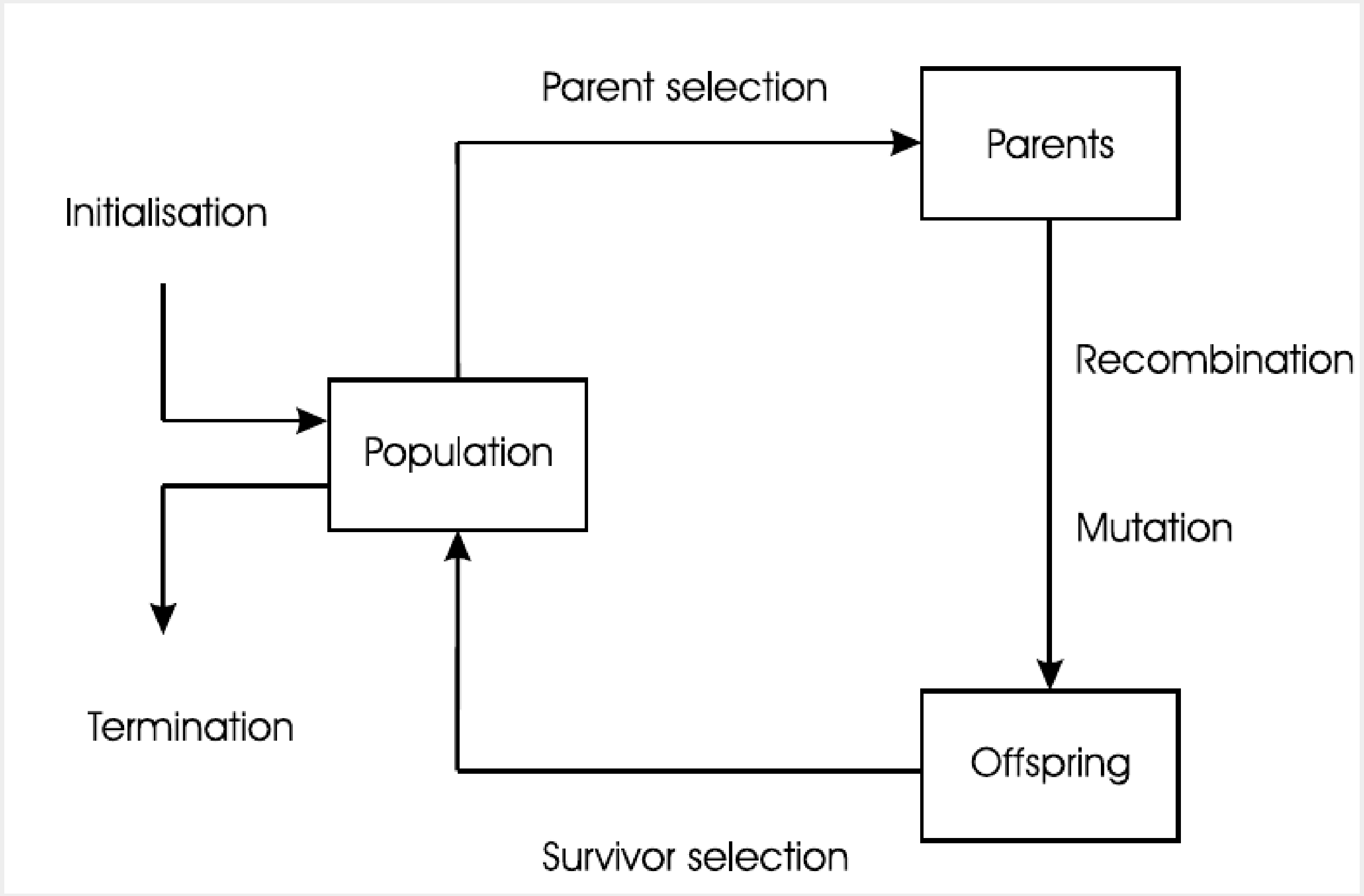
- They belong to the category of *generate-and-test algorithms*.
  - The evaluation (fitness) function represents a heuristic estimation of solution quality and the search process is driven by the variation and the selection operators

- EAs are population based, i.e., they process a whole collection of candidate solutions simultaneously

- EAs mostly use recombination to mix information of more candidate solutions into a new one

- EAs are stochastic.

EA “Flow chart”



Basic Evolutionary Algorithm Pseudocode

```
BEGIN
    INITIALISE population with random candidate solutions
    EVALUATE each population member
    REPEAT UNTIL ( TERMINATION CONDITION is satisfied )
            1 SELECT parents
            2 RECOMBINE pairs of parents
            3 MUTATE the resulting offspring
            4 EVALUATE new offspring
            5 SELECT individuals for the next generation
END
```

Evolutionary Operators

- Executing recombination and mutation leads to a set of new candidates (the offspring)
  - that compete – based on their fitness (and possibly age)– with the old ones for a place in the next generation

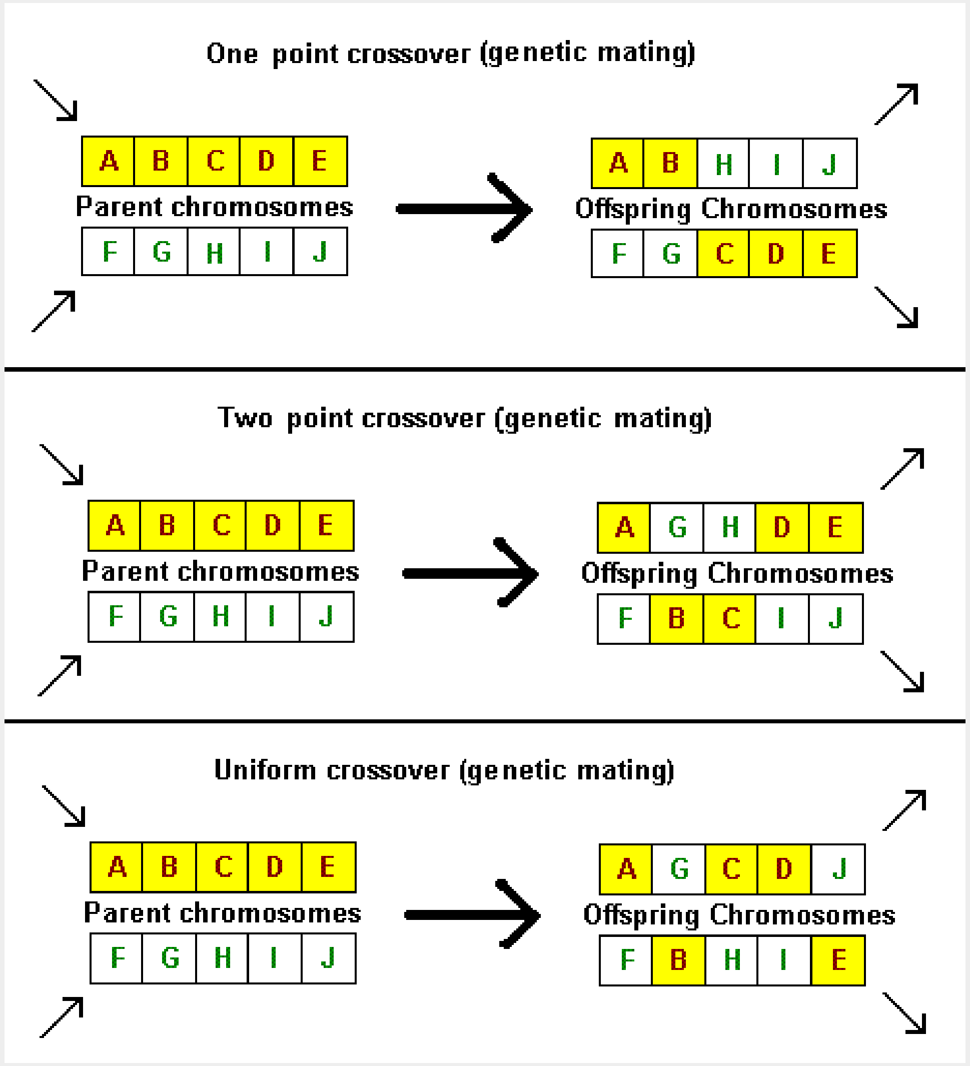
- This process can be iterated until a candidate with sufficient quality (a solution) is found or a previously set computational limit is reached

- In this process there are two fundamental forces that form the basis of evolutionary systems.
  - ***Variation*** operators (recombination and mutation) create the necessary diversity and thereby facilitate **novelty**
  - ***selection*** acts as a force pushing **quality**

Recombination

- recombination (or crossover) is usually a binary operator
- merges information from two parent genotypes into one or two offspring genotypes.
- recombination is a stochastic operator:
  - the choice of what parts of each parent are combined, and the way these parts are combined, depend on random choices
- recombination operators with a higher arity (using more than two parents) are mathematically possible and easy to implement, but have no biological equivalent
  - Perhaps this is why they are not commonly used, although several studies indicate that they can have positive effects

Recombination (Crossover) Operators in Binary Encoded GA



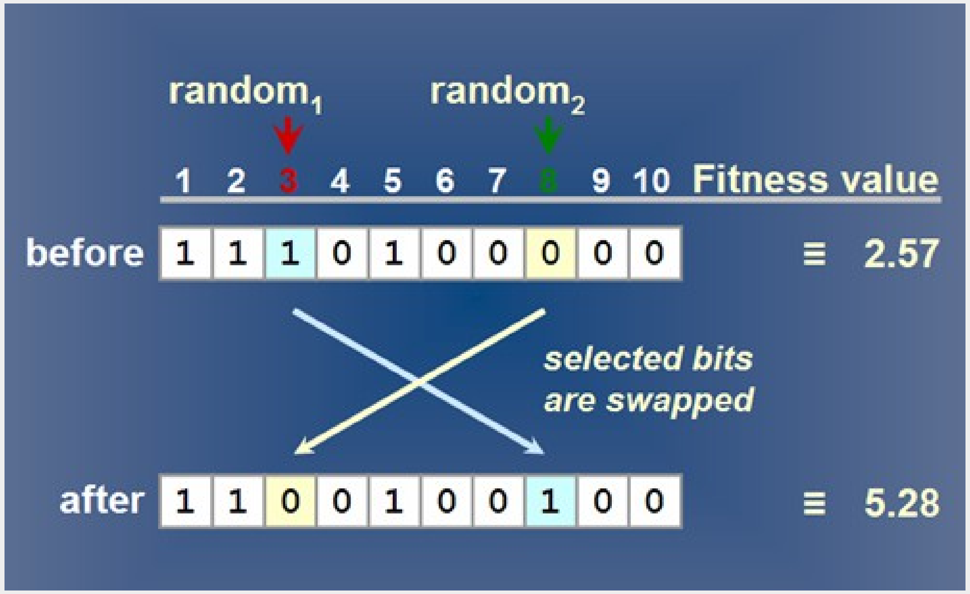
The principal behind Recombination

- The principal behind recombination is simple –
  - that by mating two individuals with different but desirable features, we can produce an offspring which combines both of those features
- has a strong supporting case –
  - successfully applied for millennia by breeders of plants and livestock, to produce species which give higher yields or have other desirable features
- Evolutionary Algorithms create a number of offspring by random recombination
  - some will have undesirable combinations of traits
  - most may be no better or worse than their parents
  - hope that some have improved characteristics.

Mutation

- unary variation operator is commonly called mutation
- applied to one genotype and delivers a (slightly) modified mutant, the child or offspring of it
- mutation operator is always stochastic:
  - its output – the child – depends on the outcomes of a series of random choices
- an arbitrary unary operator is not necessarily seen as mutation
  - in general, mutation is supposed to cause a random, unbiased change

Example Mutation Operator in Binary Coded GA

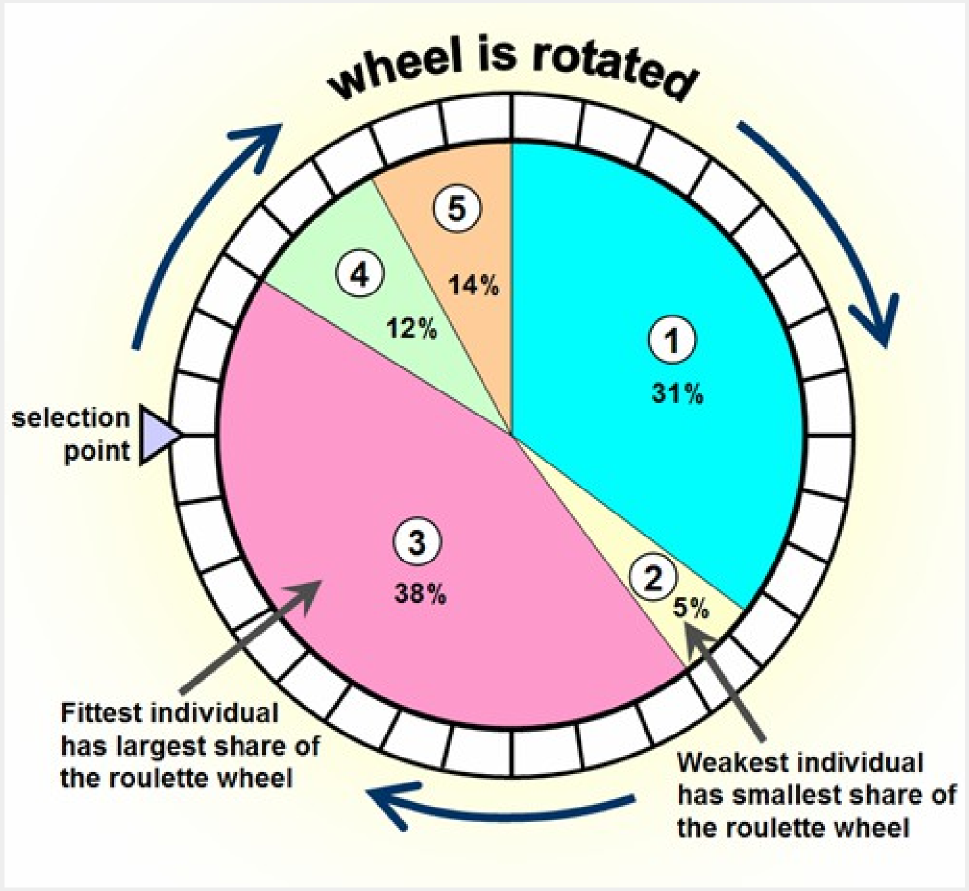


random$_1$
random$_2$
1 2 3 4 5 6 7 8 9 10 Fitness value
before 1 1 1 0 1 0 0 0 0 0 ≡ 2.57
selected bits are swapped
after 1 1 0 0 1 0 0 1 0 0 ≡ 5.28

Parent Selection

- The role of parent selection or mating selection is to distinguish among individuals based on their quality
- in particular, to allow the better individuals to become parents of the next generation
- an individual is a parent if it has been selected to undergo variation in order to create offspring
- together with the survivor selection mechanism, parent selection is responsible for pushing quality improvements
- In EC, parent selection is typically probabilistic
- Examples:
  - Roulette Wheel Selection
  - Tournament Selection

Roulette Wheel Selection


wheel is rotated
5
14%
4
12%
1
31%
selection point
3
38%
2
5%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

Survivor Selection

- role is to distinguish among individuals based on their quality
  - similar to parent selection, but it is used in a different stage of the evolutionary cycle
  - called after having created the offspring of the selected parents

- population size is (usually) constant, thus a choice has to be made on which individuals will be allowed in the next generation
  - this decision is usually based on their fitness values
  - favouring those with higher quality
  - the concept of age is also frequently used

- as opposed to parent selection which is typically stochastic, survivor selection is often deterministic,
  - for instance ranking the unified multiset of parents and offspring and selecting the top segment (fitness biased)
  - or selecting only from the offspring (age-biased)

- also often called **replacement** or replacement strategy

Initialisation

- Initialisation is kept simple in most EA applications
- the first population is seeded by randomly generated individuals
- in principle, problem specific heuristics can be used in this step aiming at an initial population with higher fitness.

Exploration vs. exploitation

.

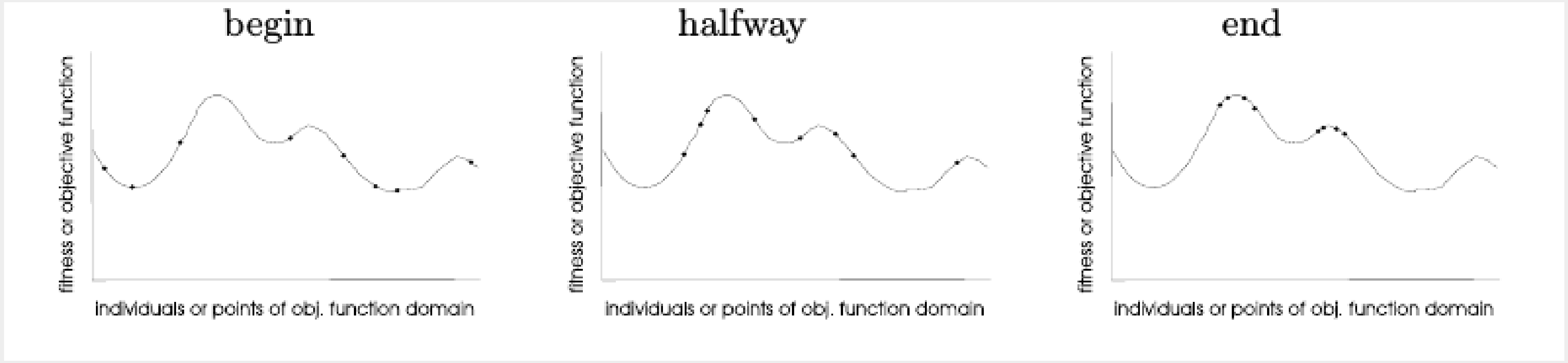
- Evolutionary search processes are often referred to in terms of a trade-off between **exploration** and **exploitation**

- too much of the former leading to inefficient search, and too much of the latter leading to a propensity to focus the search too quickly

- premature convergence is the well-known effect of losing population diversity too quickly and getting trapped in a **local optimum**

- this danger is generally present in Evolutionary Algorithms, and is perhaps their greatest weakness in solving real-world problems which are usually multi-modal

EA population convergence
(from Eiben & Smith,2003)



Genetic Algorithms (GA)

- Genetic algorithms became popular through the work of John Holland in the early 1970s, and particularly his book “Adaptation in Natural and Artificial Systems” (1975)

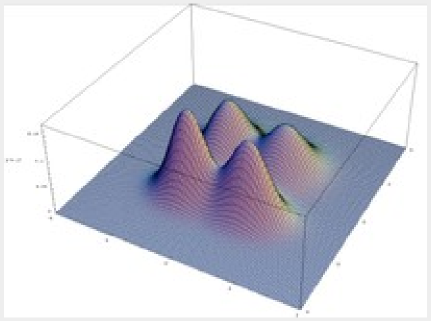
- His work originated with studies of cellular automata, conducted by Holland and his students at the University of Michigan

- Holland introduced a formalized framework for predicting the quality of the next generation, known as Holland's Schema Theorem

- The Schema Theorem and most of the implementations of the GA used bit-string representations.
  - These are attractive from the point of Biological plausibility and certain Computer Science applications
  - less well suited to Engineering Problems which often involve real-valued function optimisation. For example designing a car body with minimum aerodynamic drag or a bridge with maximum strength to weight ratio.

Schema Theorem

- says that short, low-order schemata with above-average fitness ***increase exponentially in successive generations***
- proposed by John Holland in the 1970s.
- a schema is a template that identifies a subset of strings with similarities at certain string positions
- e.g. schema 1*10*1 describes the set of all strings of length 6 with 1's at positions 1, 3 and 6 and a 0 at position 4. The * is a wildcard symbol
- the fitness of a schema is the ***average fitness*** of all strings matching it
- holds under the assumption of a genetic algorithm with an ***infinitely large population,*** but does not always carry over to (finite) practice: due to sampling error in the initial population
- genetic algorithms may converge on schemata that have no selective advantage
  - in particular in multimodal optimization, where a function has multiple peaks: the population may drift to prefer one of the peaks, ignoring the others



Evolution Strategies (ES)

- Originally devised in the 1960s by Schwefel and Rechenberg in Berlin, using a population size of only 2, known as a [1+1]-ES
- Rechenberg (1971) presented an approximate analysis of the behaviour of the [1+1]-ES in optimisation of **real-valued functions**
  - Also at this time, step size adaptation was intoduced using simple heuristic rules
- Schwefel (1975) introduced ES algorithms with larger population sizes, the (μ,λ)-ES has separate parent (μ) and offspring (λ) populations
  - It was shown that in these more elaborate ES implentations the parameters for the operators, such as mutation step size could be incorporated in the genotype and evolve to optimise the rate of evolution as well as the solution

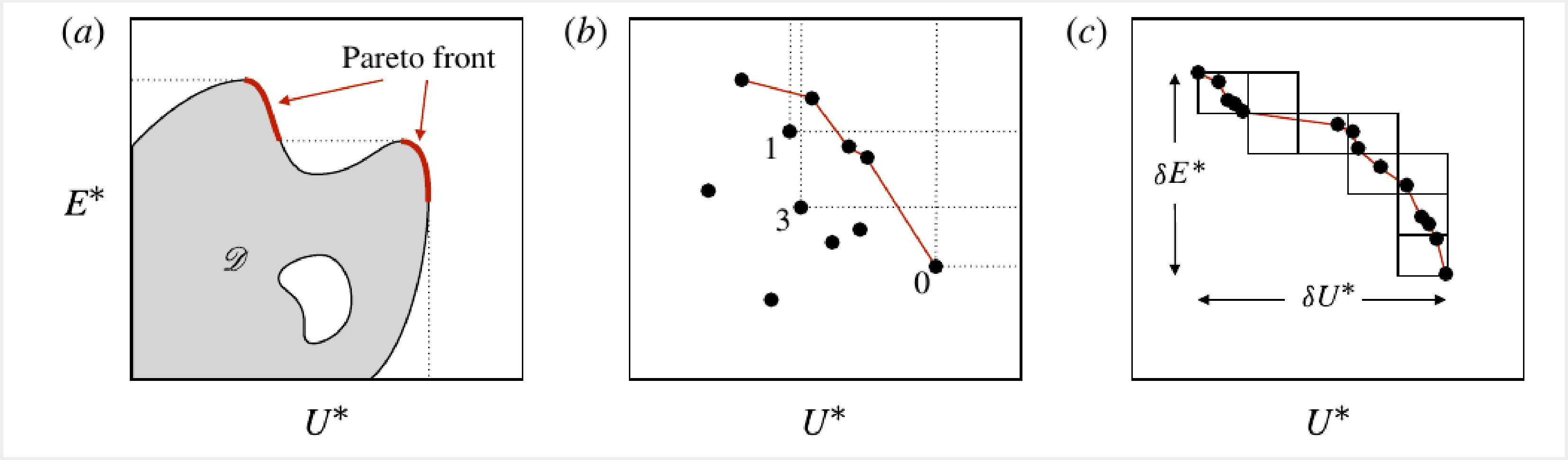
Evolution Strategy Characteristics

- Real number vector genotypes
  - highly suited to function optimisation problems in Engineering contrasting with the Genetic Algorithm (GA) which more often uses a bit-string representation
- Mutation important as variation operator (with adaptive step size)
- Parameters for operators incorporated in genotype and optimised in parallel with solution candidates to accelerate progress
- Often effective even with small population sizes
- Some of these features have subsequently been incorporated into Genetic Algorithms (GA) as well

Multi-objective Optimization

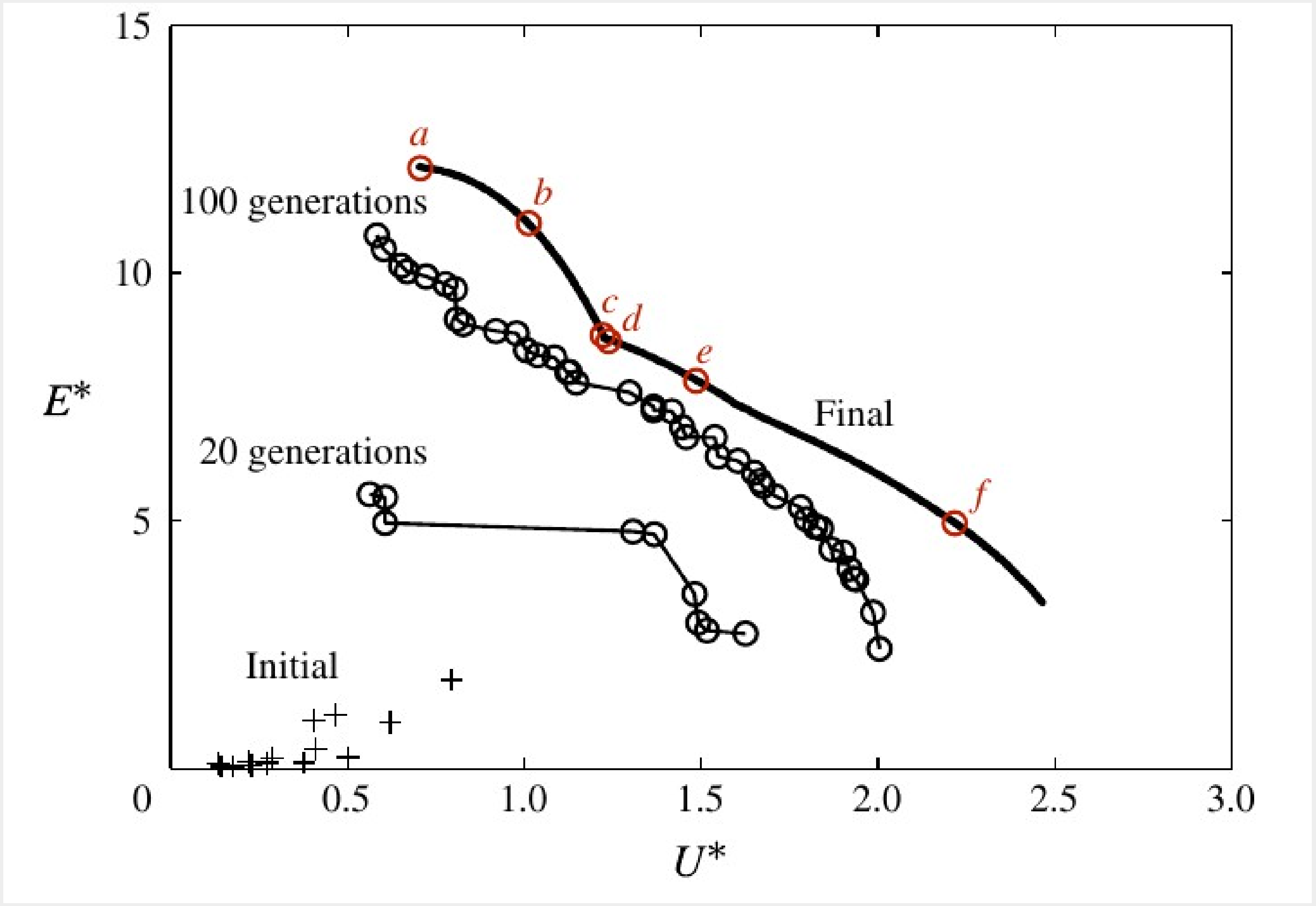
- For a nontrivial multi-objective optimization problem, no single solution exists that simultaneously optimizes each objective
- In that case, the objective functions are said to be conflicting, and there exists a (possibly infinite) number of ***Pareto optimal*** solutions.
- A solution is called ***nondominated***, Pareto optimal, Pareto efficient or noninferior, if none of the objective functions can be improved in value without degrading some of the other objective values
- Without additional subjective preference information, all Pareto optimal solutions are considered equally good

Multi-objective Optimization
An Example



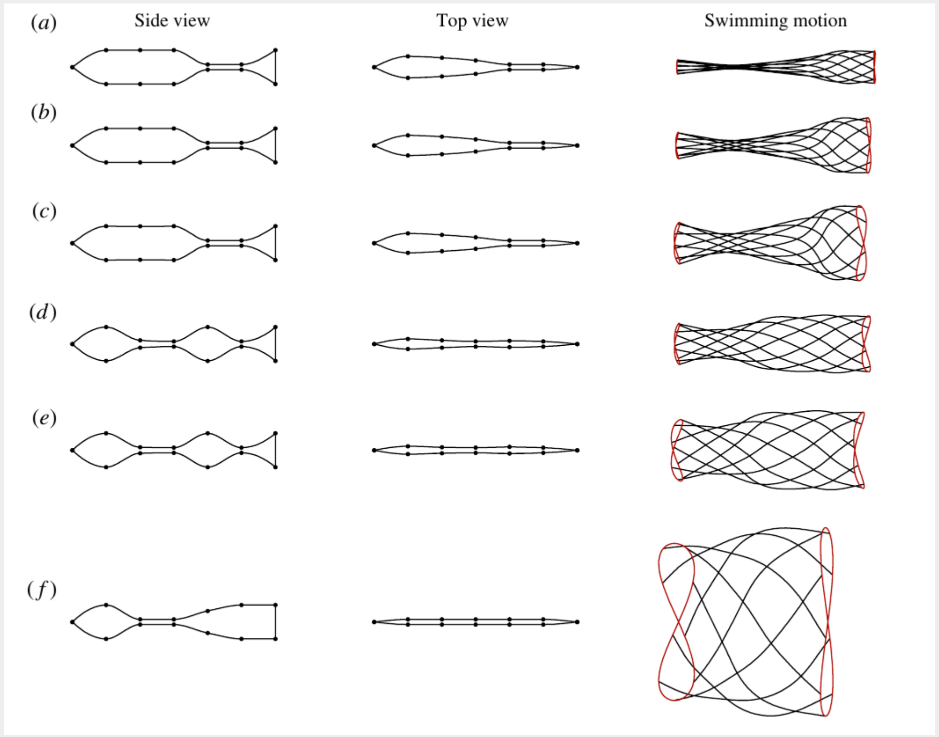
Christophe Eloy, *On the best design for undulatory swimming, J. Fluid Mech. (2013), vol. 717, pp. 48–89. doi:10.1017/jfm.2012.561*

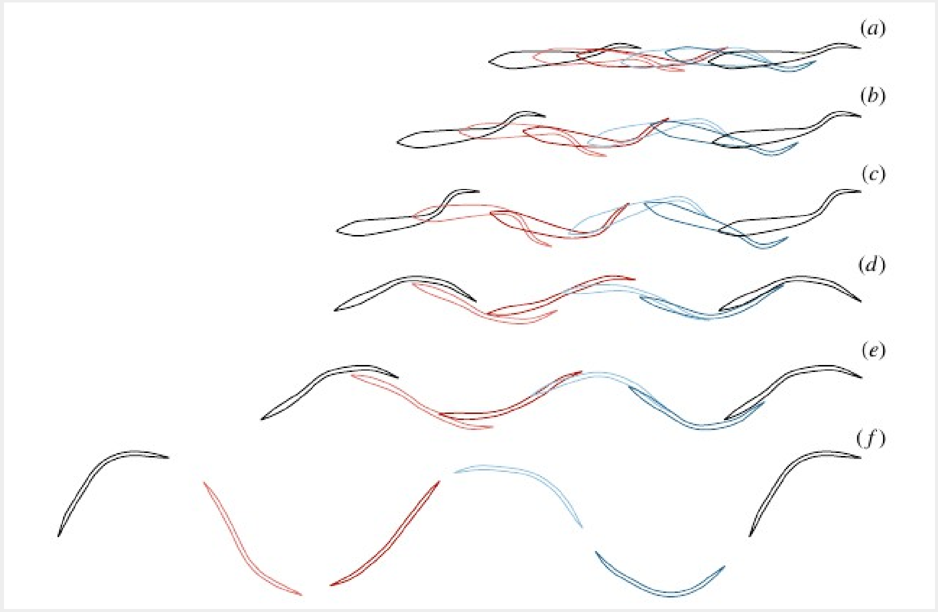
Multi-objective Optimization Results



Multi-objective Optimization

Shapes and Kinematics of Best Swimmers





Darwinian Evolution

- The notion of the “survival of the fittest” was introduced by Darwin in Origin of the Species(1859), to explain how selection acts as a force for increasing **quality**

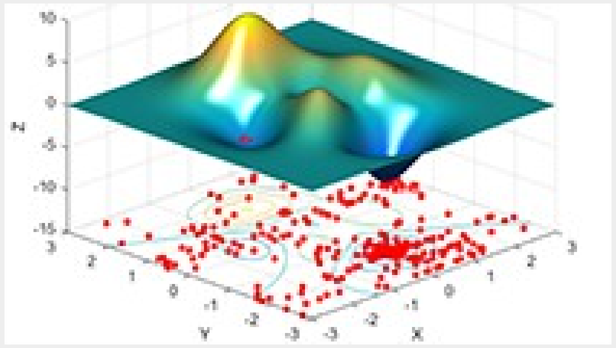
- The idea is central in the writings of Richard Dawkins
  - in particular the **Blind Watchmaker** (1986) which emphasises that the evolutionary process is driven forward by a completely random search “strategy” in combination with the natural pressures on survival

GA in MATLAB

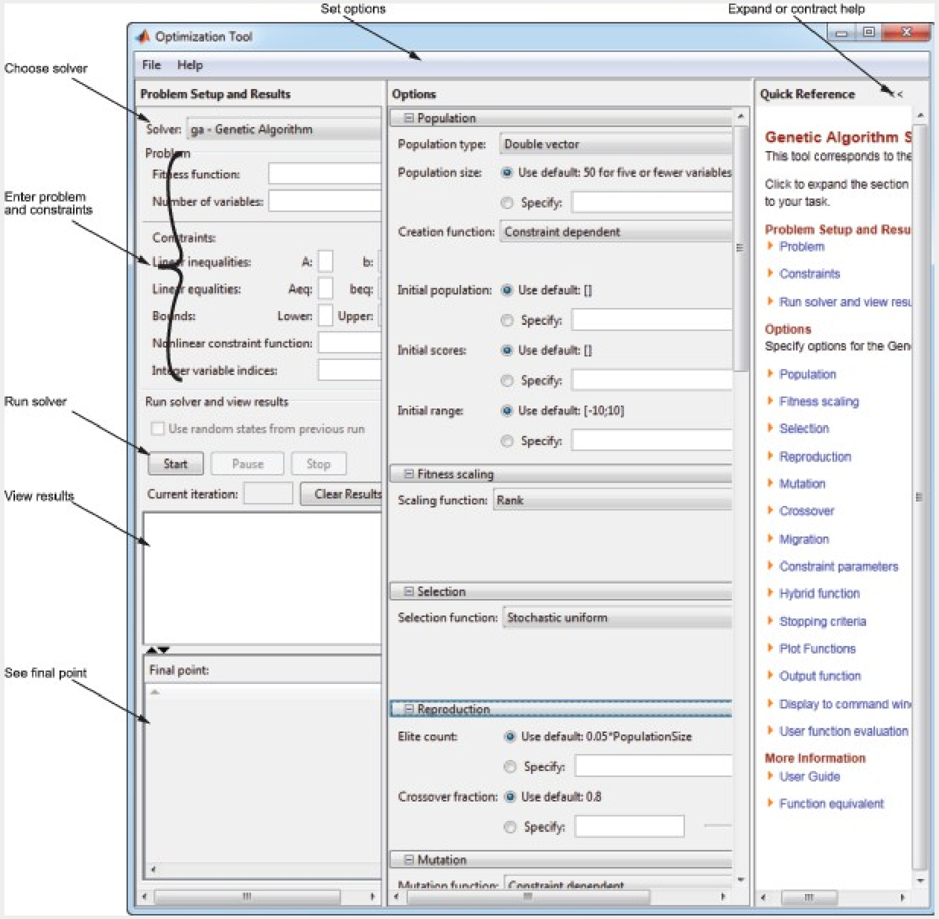
Calling the Function ga at the Command Line

To use the genetic algorithm at the command line, call the genetic algorithm function ga with the syntax

```
[x fval] = ga(@fitnessfun, nvars, options)
where
    @fitnessfun is a handle to the fitness function.
    nvars is the number of independent variables for the
            fitness function.
    options is a structure containing options for the genetic
        algorithm.
        .
The results are given by
    x — Point at which the final value is attained
    fval — Final value of the fitness function
```



MATLAB optimtool



Solver: ga - Genetic Algorithm

Problem
Fitness function:
Number of variables:

Constraints:
Linear inequalities: A: b:
Linear equalities: Aeq: beq:
Bounds: Lower: Upper:
Nonlinear constraint function:
Integer variable indices:

Run solver and view results
Use random states from previous run
Current iteration:

Final point:

Population
Population type: Double vector
Population size: Use default: 50 for five or fewer variables
Specify:
Creation function: Constraint dependent
Initial population: Use default: []
Specify:
Initial scores: Use default: []
Specify:
Initial range: Use default: [-10;10]
Specify:

Fitness scaling
Scaling function: Rank

Selection
Selection function: Stochastic uniform

Reproduction
Elite count: Use default: 0.05*PopulationSize
Specify:
Crossover fraction: Use default: 0.8
Specify:

Mutation
Mutation function: Constraint dependent

Genetic Algorithm S
This tool corresponds to the
Click to expand the section to your task.
Problem Setup and Resu
Problem
Constraints
Run solver and view resu
Options
Specify options for the Gen
Population
Fitness scaling
Selection
Reproduction
Mutation
Crossover
Migration
Constraint parameters
Hybrid function
Stopping criteria
Plot Functions
Output function
Display to command win
User function evaluation
More Information
User Guide
Function equivalent

- Global Optimization Toolbox provides methods that search for global solutions to problems that contain multiple maxima or minima.
- It includes global search, multistart, pattern search, genetic algorithm, and simulated annealing solvers.
- You can use these solvers to solve optimization problems where the objective or constraint function is continuous, discontinuous, stochastic, does not possess derivatives, or includes simulations or black-box functions with undefined values for some parameter settings

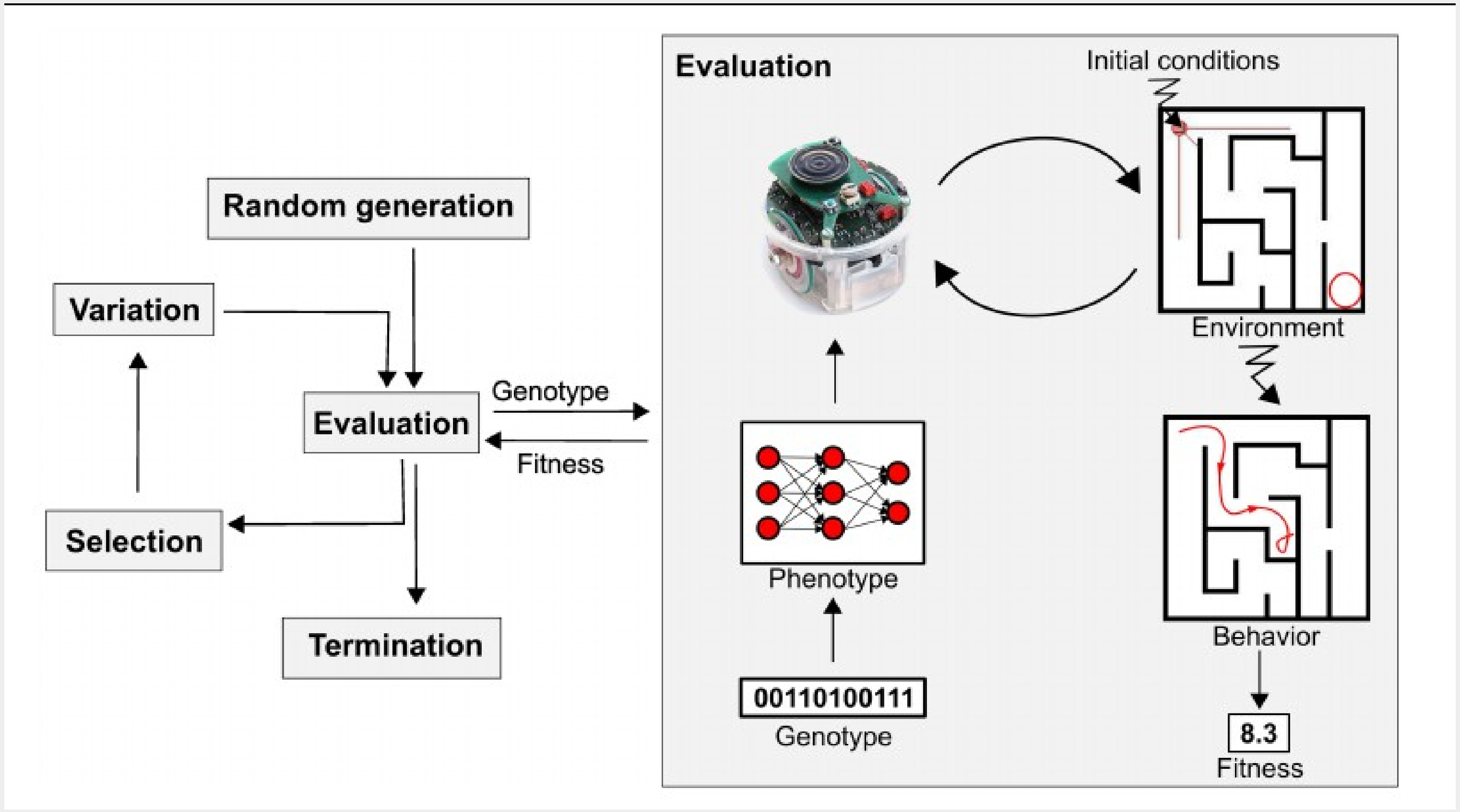
Robotics Applications

- Can be applied to almost any optimisation problem
  - In other words almost any Engineering problem
- Many examples in robotics literature
  - Trajectory planning
  - Inverse kinematics and dynamics
  - Walking gait optimisation for humanoid robots
  - Structural optimisation
  - Drive train optimisation
  - Implementations often including Neural Network
    - e.g. optimisation of weights and biases of an MLP

Evolutionary Robotics

http://www.evolutionaryrobotics.org/

- Evolutionary robotics uses population-based artificial evolution to evolve autonomous robot controllers (i.e. robot brains) and sometimes robot morphologies (i.e. robot bodies)
- Generally, the robots are evolved to perform tasks requiring some level of intelligence, for example moving around in an environment without running into things



Further Reading (recommended for revision)

- A.E. Eiben and J.E. Smith, *Introduction to Evolutionary Computing*, Springer 2003

  http://www.cs.vu.nl/~gusz/ecbook/ecbook.html

- Dawkins, Richard [1986]. *The Blind Watchmaker*. New York: W. W. Norton & Company, Inc. ISBN 0-393-31570-3.

  Dawkins, in contrasting the differences between human design and its potential for planning with the workings of natural selection, dubbed evolutionary processes as analogous to a blind watchmaker. This book is an interesting and thought-provoking book about evolution which provides a good introduction to the ideas which motivate Evolutionary Computing

- J. Holland, *Adaptation in Natural and Artificial Systems*, The MIT Press; Reprint edition 1992 (originally published in 1975).