Natural Computing

Lecture 3

Michael Herrmann
mherrman@inf.ed.ac.uk
phone: 0131 6 517177
Informatics Forum 1.42

27/09/2011

The Canonical Genetic Algorithm

The Canonical Genetic Algorithm: Conventions

1. Old population
2. Selection
3. Intermediate population
4. Recombination
5. Mutation
1. New population

one generation

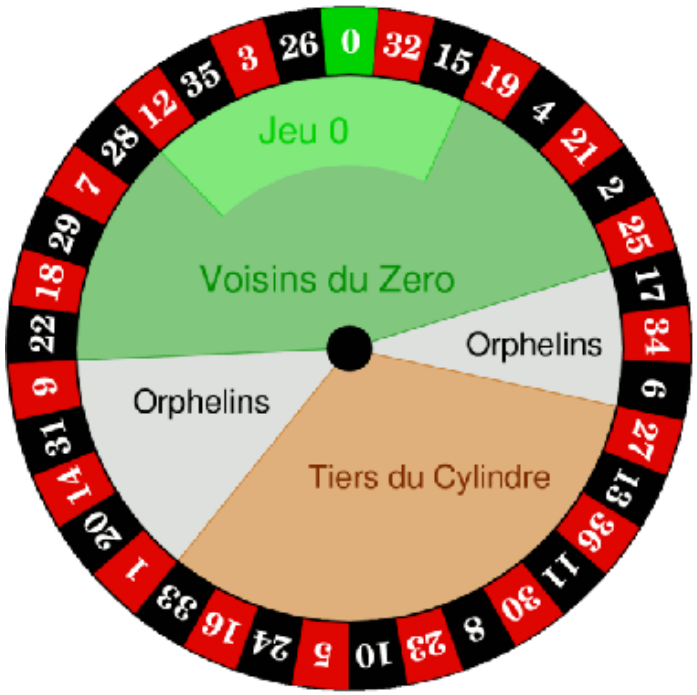
- A population is a (multi-) set of individuals
- An individual (genotype, chromosome) is a string $S \in \mathcal{A}^L$ ($\mathcal{A}$: alphabet, often: $\mathcal{A} = \{0, 1\}$)
- Fitness = objective function = evaluation function. Fitness values can be replaced by ranks (high to low)

Roulette Wheel Selection

I. Plain variant

Mean fitness $\bar{f} = \frac{1}{n}\sum_i f_i$ $\Longrightarrow$ Normalized fitness: $\frac{f_i}{\bar{f}}$ (from now on short: fitness)

- Each time the ball spins one individual is selected for the intermediate population
- Stochastic sampling with replacement
- Ratio of fitness to average fitness determines number of offspring, i.e. a new individual is a copy of an old individual (of fitness f_i) with probability $\frac{f_i}{n\bar{f}}$
- If $f_i = \bar{f}$ then the individual "survives" with probability $1 - \left(1 - \frac{1}{n}\right)^n$



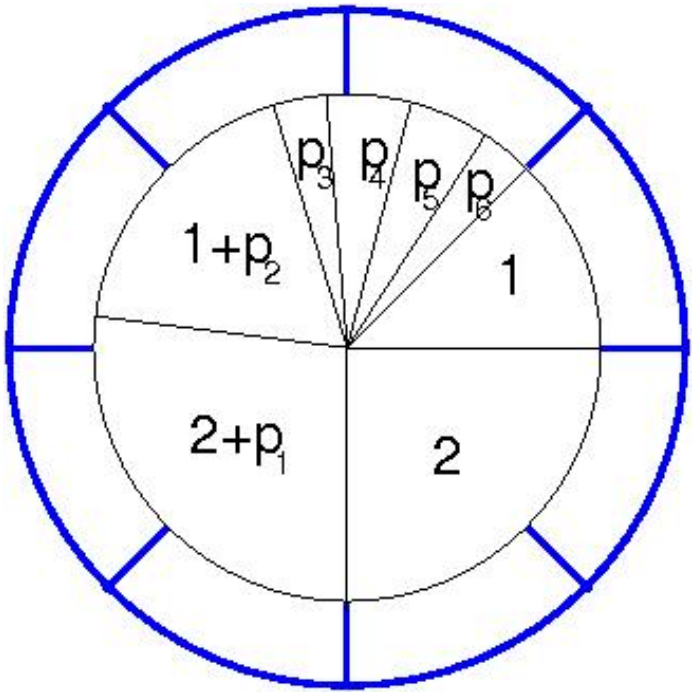
Sector (French) bets in roulette: Here, the size of the sector represents the relative fitness of an individual

Roulette Wheel Selection

II. A more practical variant

Mean fitness $\bar{f} = \frac{1}{n}\sum_i f_i$ $\Longrightarrow$ Normalized fitness: $\frac{f_i}{\bar{f}}$

- Remainder stochastic sampling
- Ratio of fitness to average fitness determines number of offspring
- If $f_i = \bar{f}$: the individual survives
- If $f_i < \bar{f}$: survives with prob. $\frac{f_i}{\bar{f}}$
- If $f_i > \bar{f}$: number of offspring $\text{int}\left(\frac{f_i}{\bar{f}}\right)$ and possibly one more with probability $\frac{f_i}{\bar{f}} - \text{int}\left(\frac{f_i}{\bar{f}}\right)$



Now: Only the outer wheel with equidistant pointers spins once and pointers in each sector are counted

Both variants are equivalent in the sense that they produce an unbiased sample of the fitness in the population, i.e. a new individual is a copy of an old individual (of fitness f_i) with probability $\frac{f_i}{n\bar{f}}$

From intermediate to new population

Preparation:

- Population was already shuffled by selection
  (but may contain multiple copies of the same string)
- Individuals are strings of equal length L
- Choose a probability p_c:

Crossover:

- Choose a pair of individuals
- With probability p_c:
  - choose a position from 1 to $L-1$
  - cut both individuals after this position
  - re-attach crossed: xyxxxyyy, abbabbab $\rightarrow$ xyxxbbab, abbaxyyy
- Move the obtained pair to the new population (even if not crossed over)
- Repeat for the remaining pairs (assert n even)

From intermediate to new population

Preparation:

- Crossover finished
- Individuals are strings of length L made from k different char's
- Choose a (small) probability p_m (possibly rank-dependent)

Mutation:

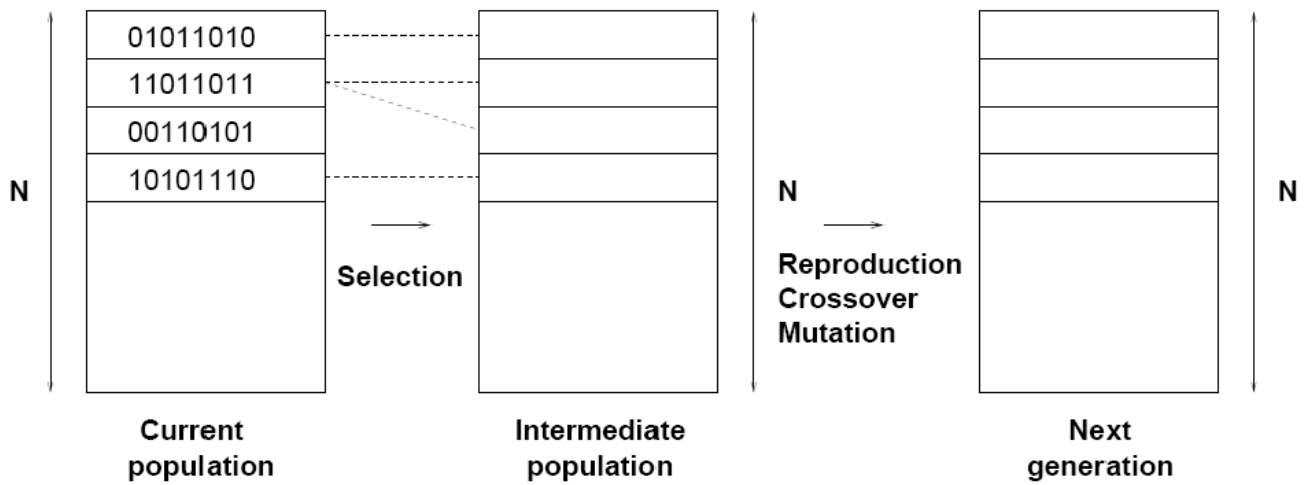
- For all individuals (from new population)
  - for each position from 1 to L
  - with probability p_m:
  - set the character (bit if binary) to a random value or change it [this gives $\frac{k}{k-1}$ (i.e. twice if binary) the effect! Canonical: binary, switch]
- The obtained mutants (possibly including some unmutated individuals) form the new population

The canonical GA in brief

Repeat

- Evaluate fitness
- Select intermediate population
- Do crossover or reproduction
- Do mutation

Until solutions are good enough



The canonical GA

- Evaluation function F (raw fitness) gives a score $F(i) = f_i$ to each individual solution $i \in \{1, \ldots, n\}$
- If $\bar{f}$ is the average evaluation over the whole population of n individuals then the **fitness** of i is $f_i/\bar{f}$
- Probability of **selection** of a solution with evaluation f_i is $f_i / \sum_i f_i$
- Select two parents at random from the intermediate population. Apply crossover with probability p_c, with probability $1 - p_c$ copy the parents unchanged into the next generation — reproduction.
  Crossover: from the 2 parents create 2 children using 1-point crossover. Select crossover point uniform-randomly
- **Mutation**: Take each bit in turn and flip it with probability $p_m (1 \to 0 \text{ or } 0 \to 1)$. $p_m < 0.01$ usually. Note that the probability p_m is applied differently from p_c
- This is one generation. Repeat for many generations until a **termination** criterion is met.

Termination of a GA

The generational process is repeated until a termination condition has been reached, e.g.

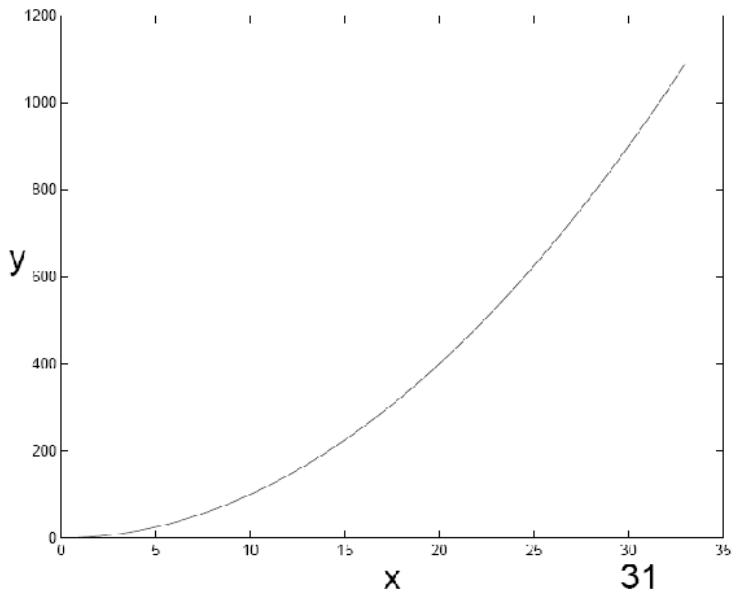
- A solution is found that has optimal fitness (or is sufficiently close to the optimum)
- Fitness indicates a sufficient improvement over alternative algorithms
- Fixed number of generations reached (only for safety!)
- Allocated budget (computation time/money) reached
- The diversity of the population has vanished (restart?)
- The fitness of the highest ranking solution is reaching or has reached a plateau such that successive iterations no longer produce better results (restart?)
- Combinations of the above

After Termination decide: Really finish or restart a variant of the GA on the same task

Simple Example: "All-Ones"

Maximise $f(x) = x^2$ for integer $x \in \{0, ..., 31\}$. (What is the answer?)

- Encoding:
  Write x in base 2 (here: more significant bits to the left).
- Initialisation:
  All strings $= (0, 0, 0, 0, 0)$
  (or random)
- Mutations:
  Generate 1s
- Cross-over:
  Combine 1s from different individuals
- Termination criterion



Simple Example: "All-Ones"

Represent x as 5 bits

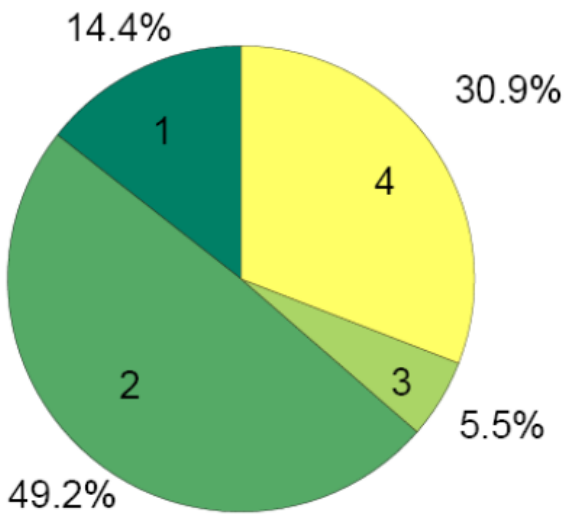
00000	0	0
00001	1	1
00010	2	4
⋮	⋮	⋮
11111	31	961

Here raw fitness values will be used instead of ranks. Does this make any difference?

Use a population size of 4 (far too small!)

i	genome	raw fitness	% of total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9

Roulette-wheel selection



Simple Example: "All-Ones"
Further Recombinations

So our new (intermediate) population might be: 1, 2, 2, 4

Create next generation:

Crossover: $p_c = 1.0 \quad \Rightarrow$ crossover is always applied here

Parents: 1 and 2

0110 1 $\Longrightarrow$ 0110 0

1100 0 1100 1

Parents: 2 and 4

11 000 $\Longrightarrow$ 11 011

10 011 10 000

Mutation:

$p_m = 0.001$, 20 bits $\Rightarrow$ no mutation here ($20 \times 0.001 = 0.02$)

(... wouldn't be a good idea if all individuals had a 0 in the same place!)

Simple Example: "All-Ones"

Results for the second generation:

i	genome	raw fitness	% of total
1	01100	144	8.2
2	11001	625	35.6
3	11011	729	41.6
4	10000	256	14.6

What is the average evaluation of this population?

How does it compare with the average evaluation of the previous generation?

Continue until no improvement in the best solution for k generations [or run for a fixed number of generations (?)]

Simple Example: "All-Ones"

So our new (intermediate) population might be: 1, 2, 2, 4

Create next generation:

Crossover: $p_c = 1.0 \quad \Rightarrow$ crossover is always applied here

Pair parents randomly, choose crossover points randomly

Parents: 1 and 2		2. Generation		3. Generation
0110 1	$\Longrightarrow$	01 100	$\Longrightarrow$	01 001
1100 0		11 001		11 100
Parents: 2 and 4		**2. Generation**		**3. Generation**
11 000	$\Longrightarrow$	11 011	$\Longrightarrow$	110 00
10 011		10 000		100 11

... assuming that by chance the same pairings have taken place.
Now there is a chance to find the optimal solution.

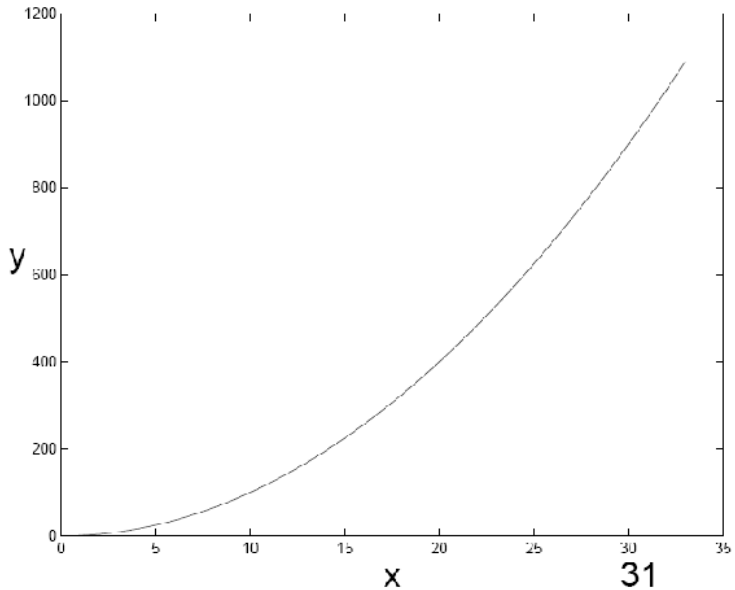
What chance? Choose (what is now) parents 2 and 4: probability 1/3; cut after third place probability 1/4, i.e. a chance of 1/12.

Simple Example: "All-Ones"
Conclusions

- Larger populations may improve exploration
- Large populations, may contain many identical individuals
  - redundant representation
  - if they are fit some of them will survive
- Choice of the representation is crucial: E.g. 10000 is better (higher fitness!) than 01111 although the latter is "closer" (in Hamming distance but not w.r.t. fitness!).
- Low evolutionary pressure can be helpful: The algorithm will typically find significant bits first, (e.g.) 11100 has much higher fitness than 00011, but together they could form the optimal solution if some of the latter individuals did survive
- Mutations are important!
- Termination is a non-trivial problem

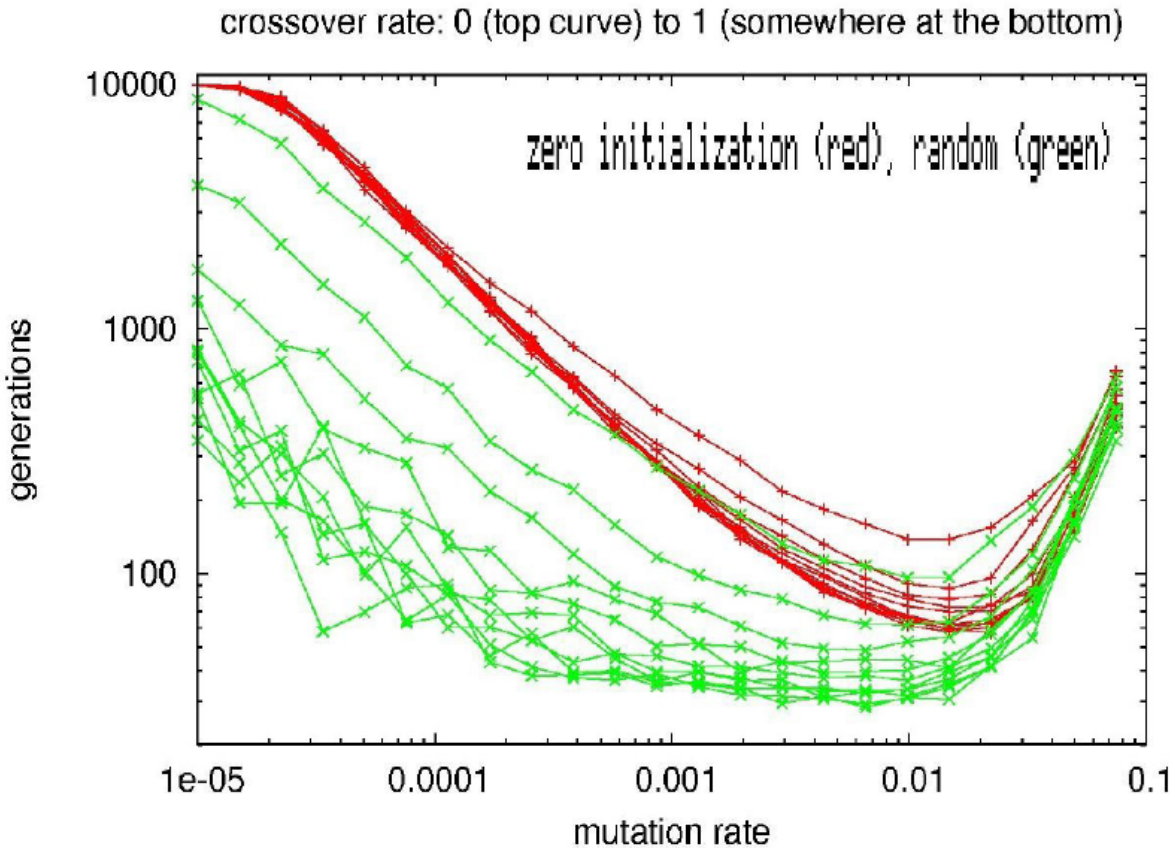
GA: How does it work?

- Climbing up the fitness curve (landscape)
- Promote fast climbers
- Putting together building blocks of good subsolutions



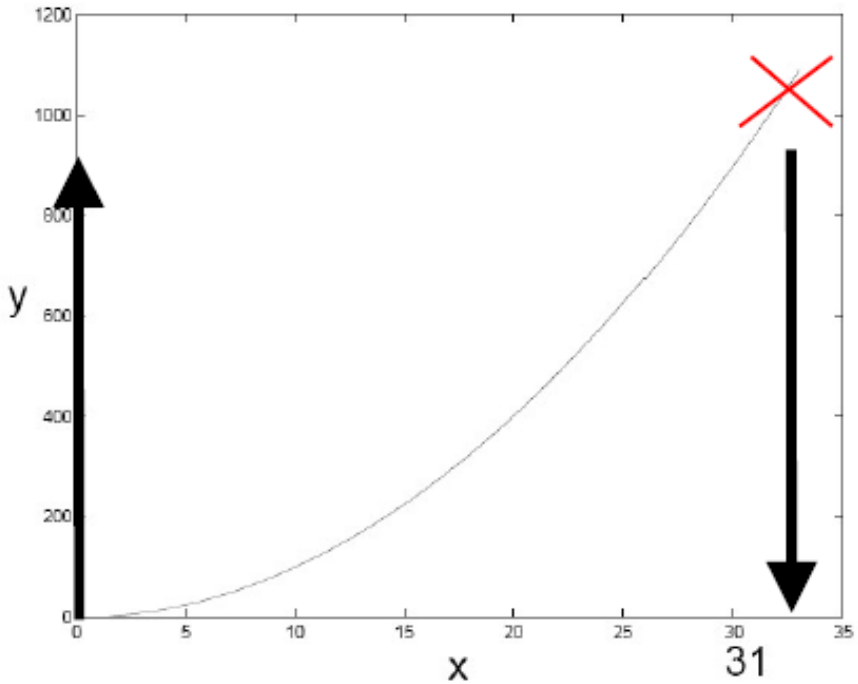
- What would happen if we choose a linear fitness function?
- What would happen if fitness just counts the bits?

Simple Example: "All-Ones" (bit-wise)



Number of generations required to discover the optimal solution
Strings of 20 characters $S_i \in 0, 1^{20}$, $n = 100$, $f(S) = \Sigma S_i$,
initialization: a) $S_i = 0$ and $S_i = 1$ with prob. $\frac{1}{2}$ each
or b) $S_i = (0, \ldots, 0)$

A “deceptive” fitness function



$$f(x) = \begin{cases} 961 & \text{for } x = 0 \\ x^2 & \text{for } 0 < x < 31 \\ 0 & \text{for } x = 31 \end{cases}$$

The "Philosophy" of GA

- Encoding: Create a space of solutions
- Fitness function: Discriminate good from bad solutions
- Initialization: Start with good candidate solutions
- Selection: Prefer better solutions to worse ones
- Recombination: Combine parental traits in a novel manner
- Mutation: Creating individual traits by random local search
- Termination: Comparing achieved and achievable fitness

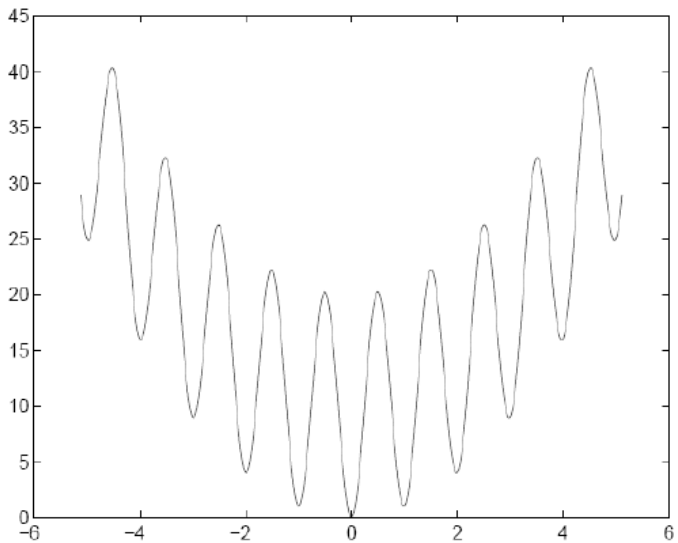
How do the simple mechanisms create something useful when combined?

- Selection + Mutation = Continual improvement
- Selection + Recombination = Innovation

Example: Function Optimization

Minimise Rastrigin's Function

$$f(x) = 10 + x^2 - 10\cos(2\pi x), \quad -5.12 \leq x \leq 5.12$$



Representation: binary strings

$x = x_{\text{min}} + (x_{\text{max}} - x_{\text{min}})\, b/(2^m - 1)$

So for 8-bit strings

$x = -5.12 + 10.24\ b/(2^8 - 1)$

E.g. if $b = 10011001$ (represents 153 in base 10)

$x = -5.12 + 10.24 \times 153/255 = 1.024$

Optimally, the algorithm finds (for this setting) $x = 0.0201$ with $f(x) = 0.0799$ rather than $x = 0$ and $f(x) = 0$. Why?

More on this example: Search for "Rastrigin" at www.mathworks.com, www.obitko.com/tutorials/genetic-algorithms/example-function-minimum.php

Outlook

- The schema theorem
  - What is a schema? A non-empty subset of a string (in other words: a string with some wildcards)
  - Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generation of a genetic algorithm.
- Variants of GA
- Performance analysis: More examples
- General formulation, theory, convergence etc.
- Other algorithms