10. Genetic Algorithms

- General-purpose "black-box" optimisation method proposed by J. Holland (1975) and K. DeJong (1975).
- Method has attracted lots of interest, but theory is still incomplete and the empirical results inconclusive.
- Advantages: general-purpose, parallelisable, adapts incrementally to changing cost functions ("on-line optimisation").
- Disadvantages: typically very slow – should be used with moderation for simple serial optimisation of a stable, easily evaluated cost function.
- Some claim that GA's typically require fewer function evaluations to reach comparable results as e.g. simulated annealing. Thus the method may be good when function evaluations are expensive (e.g. require some acutal physical measurement).

10.1 The Basic Algorithm

- We consider the so called "simple genetic algorithm"; also many other variations exist.
- Assume we wish to maximise a cost function c defined on n-bit binary strings:

$$c : \{0,1\}^n \to \mathbb{R}.$$

  Other types of domains must be encoded into binary strings, which is a nontrivial problem. (Examples later.)
- View each of the candidate solutions $s \in \{0,1\}^n$ as an *individual* or *chromosome*.
- At each stage (*generation*) t the algorithm maintains a *population* of individuals $p_t = (s_1, \ldots, s_m)$.

Three operations defined on populations:

- *selection* $\sigma(p)$ ("survival of the fittest")
- *recombination* $\rho(p)$ ("mating", "crossover")
- *mutation* $\mu(p)$

The *Simple Genetic Algorithm*:

function SGA(σ, ρ, μ):
  $p \leftarrow$ random initial population;
  while p "not converged" **do**
    $p' \leftarrow \sigma(p)$;
    $p'' \leftarrow \rho(p')$;
    $p \leftarrow \mu(p'')$
  end while;
  return p (or "fittest individual" in p).
end.

Selection (1/3)

Denote $\Omega = \{0,1\}^n$. The selection operator $\sigma : \Omega^m \to \Omega^m$ maps populations probabilistically: given an individual $s \in p$, the expected number of copies of s in $\sigma(p)$ is proportional to the *fitness* of s *in* p. This is a function of the cost of s compared to the costs of other $s' \in p$.

Selection (2/3)

Some possible fitness functions:

- Relative *cost* ($\Rightarrow$ "canonical GA"):

$$f(s) = \frac{c(s)}{\frac{1}{m}\sum_{s' \in p} c(s')} \triangleq \frac{c(s)}{\bar{c}}.$$

- Relative *rank*:

$$f(s) = \frac{r(s)}{\frac{1}{m}\sum_{s' \in p} r(s')} = \frac{2}{m+1} \cdot r(s),$$

  where $r(s)$ is the rank of individual s in a worst-to-best ordering of all $s' \in p$.

Selection (3/3)

Once the fitness of individuals has been evaluated, selection can be performed in different ways:

- *Roulette-wheel selection* ("stochastic sampling with replacement"):
  - Assign to each individual $s \in p$ a probability to be selected in proportion to its fitness value $f(s)$. Select m individuals according to this distribution.
  - Pictorially: Divide a roulette wheel into m sectors of width proportional to $f(s_1), \ldots, f(s_m)$. Spin the wheel m times.
- *Remainder stochastic sampling*:
  - For each $s \in p$, select deterministically as many copies of s as indicated by the integer part of $f(s)$. After this, perform stochastic sampling on the fractional parts of the $f(s)$.
  - Pictorially: Divide a fixed disk into m sectors of width proportional to $f(s_1), \ldots, f(s_m)$. Place an outer wheel around the disk, with m equally-spaced pointers. Spin the outer wheel once.

Recombination (1/2)

- Given a population p, choose two random individuals $s, s' \in p$. With probability p_ρ, apply a *crossover operator* $\rho(s, s')$ to produce two new offspring individuals t, t' that replace s, s' in the population.
- Repeat the operation $m/2$ times, so that on average each individual participates once. Denote the total effect on the population as $p' = \rho(p)$.
- Practical implementation: choose $\frac{p_\rho}{2} \cdot m$ random pairs from p and apply crossover deterministically.
- Typically $p_\rho \approx 0.7 \ldots 0.9$.

Recombination (2/2)

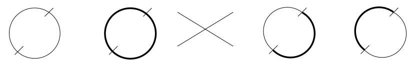
Possible crossover operators:

- *1-point crossover:*

  1 1 0 1 | 0 0 1 1 0 0 1 → 0 1 1 0 | 0 0 1 1 0 0 1
  0 1 1 0 | 1 0 1 1 0 1 1 → 1 1 0 1 | 1 0 1 1 0 1 1

- *2-point crossover:*

  1 1 | 0 1 0 0 1 | 1 0 0 1 → 1 1 | 1 0 1 0 1 | 1 0 0 1
  0 1 | 1 0 1 0 1 | 1 0 1 1 → 0 1 | 0 1 0 0 1 | 1 0 1 1



- *uniform crossover:*

  1 1 0 1 0 0 1 1 0 0 1 → 0 1 0 1 1 0 1 1 0 0 1
  0 1 1 0 1 0 1 1 0 1 1 → 1 1 0 1 1 0 1 1 0 0 1

Mutation

- Given population p, consider each bit of each individual and flip it with some small probability p_μ. Denote the total effect on the population as $p' = \mu(p)$.
- Typically, $p_\mu \approx 0.001 \ldots 0.01$. Apparently good choice: $p_\mu = 1/n$ for n-bit strings.
- Theoretically mutation is disruptive. Recombination and selection should take care of optimisation; mutation is needed only to (re)introduce "lost alleles", alternative values for bits that have the bits that have the same value in all current individuals.
- In practice mutation + selection = local search. Mutation, even with quite high values of p_μ, can be efficient and is often more important than recombination.

10.2 Analysis of GA's

Hyperplane sampling (1/4)

- A heuristic view of how a genetic algorithm works.
- A *hyperplane* (actually subcube) is a subset of $\Omega = \{0,1\}^n$, where the values of some bits are fixed and other are free to vary. A hyperplane may be represented by a *schema* $H \in \{0,1,*\}^n$.
- E.g. schema '$0*1**$' represents the 3-dimensional hyperplane (subcube) of $\{0,1\}^5$ where bit 1 is fixed to 0, bit 3 is fixed to 1, and bits 2, 4, and 5 vary.
- Individual $s \in \{0,1\}^n$ *samples* hyperplane H, or *matches* the corresponding schema if the fixed bits of H match the corresponding bits in s. (Denoted $s \in H$.)
- *Note*: given individual generally samples many hyperplanes simultaneously, e.g. individual '101' samples '$10*$', '$1*1$', etc.

Hyperplane sampling (2/4)

- *order* of hyperplane H:

$$\begin{aligned} o(H) &= \text{number of fixed bits in } H \\ &= n - \dim H \end{aligned}$$

- *average cost* of hyperplane H:

$$c(H) = \frac{1}{2^{n-o(H)}} \sum_{s \in H} c(s)$$

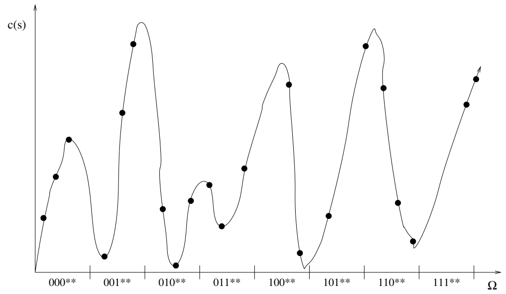
- $m(H,p) =$ number of individuals in population p that sample hyperplane H.
- *average fitness* of hyperplane H in population p:

$$f(H,p) = \frac{1}{m(H,p)} \sum_{s \in H \cap p} f(s,p)$$

Heuristic claim: selection drives the search towards hyperplanes of higher average cost (quality).

Hyperplane sampling (3/4)

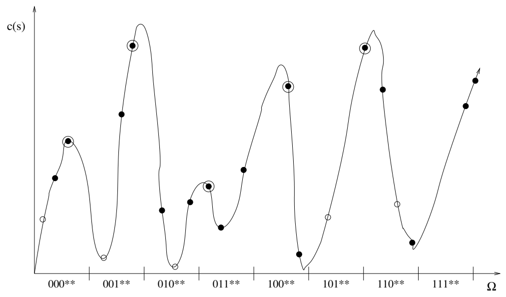
Consider e.g. the following cost function and partition of Ω into hyperplanes (in this case, intervals) of order 3:



Here the current population of 21 individuals samples the hyperplanes so that e.g. '$000**$' and '$010**$' are sampled by three individuals each, and '$100**$' and '$101**$' by two individuals each. Hyperplane '$010**$' has a rather low average fitness in this population, whereas '$111**$' has a rather high average fitness.

Hyperplane sampling (4/4)

Then the result of e.g. roulette wheel selection on this population might lead to elimination of some individuals and duplication of others:



Then, in terms of expected values, one can show that

$$E[m(H,\sigma(p))] = m(H,p) \cdot f(H,p).$$

The effect of crossover on schemata (1/2)

- Consider a schema such as

$$H = **\underbrace{11**01*1}_{\Delta(H)=7}**$$

  and assume that it is represented in the current population by some $s \in H$.
- If s participates in a crossover operation and the crossover point is located between bit positions 3 and 10, then with large probability the offspring are no longer in H (H is *disrupted*).
- On the other hand, if the crossover point is elsewhere, then one of the offspring stays in H (H is *retained*).

The effect of crossover on schemata (2/2)

- Generally, the probability that in 1-point crossover a schema $H = \{0,1,*\}^n$ is retained, is (ignoring the possibility of “lucky combinations”)

$$\Pr(\text{retain } H) \approx 1 - \frac{\Delta(H)}{n-1},$$

  where $\Delta(H)$ is the *defining length* of H, i.e. the distance between the first and last fixed bit in H.
- More precisely, if H has $m(H,p)$ representatives in population p of total size m:

$$\Pr(\text{retain } H) \geq 1 - \frac{\Delta(H)}{n-1}\left(1 - \frac{m(H,p)}{m}\right).$$

The Schema “Theorem” (1/2)

Heuristic estimate of the changes in representation of a given schema H from one generation to the next. Proposed by J. Holland (1975).

Denote:

$m(H,t) =$ number of individuals in population at generation t that sample H.

Then:

(i) Effect of selection:

$$m(H,t') \approx m(H,t) \cdot f(H)$$

(ii) Effect of recombination:

$$m(H,t'') \approx (1-p_\rho)m(H,t') + p_\rho \left(m(H,t')\,\mathrm{Pr}(\text{retain } H) + \underbrace{m \cdot \mathrm{Pr}(\text{luck})}_{\geq 0} \right)$$
$$\geq (1-p_\rho)m(H,t') + p_\rho m(H,t') \left(1 - \frac{\Delta(H)}{n-1}\left(1 - \frac{m(H,t')}{m}\right)\right)$$
$$= m(H,t')\left(1 - p_\rho \frac{\Delta(H)}{n-1}\left(1 - \frac{m(H,t')}{m}\right)\right)$$

(iii) Effect of mutation:

$$m(H,t+1) \approx m(H,t'') \cdot (1-p_\mu)^{o(H)}$$

The Schema “Theorem” (2/2)

In summary, then:

$$m(H,t+1) \gtrsim m(H,t) \cdot f(H) \cdot \left(1 - p_\rho \frac{\Delta(H)}{n-1}\left(1 - \frac{m(H,t')}{m}\right)\right) \cdot (1-p_\mu)^{o(H)}$$

The formula leads to so called *“Building Block Hypothesis”*:

- In a genetic search, short, above-average fitness schemata of low order (“building blocks”) receive an exponentially increasing representation in the population.

Criticisms

- Many of the approximations used in deriving the “Schema Theorem” implicitly assume that the population is very large. In particular, it is assumed that all the relevant schemata are well sampled. This is clearly not possible in practice, because there are 3^n poss. schemata of length n.
- The result cannot be used to predict the development of the population for much more than one generation:
  1. the long-term development depends on the coevolution of the schemata, and the “theorem” considers only one schema in isolation;
  2. an “exponential growth” cannot continue for long in a finite population.
- Proper treatment: analyse the genetic search as a stochastic process (Markov chain). This is unfortunately very difficult.

10.3 Data Representations

General comments on coding:

- If the function to be optimised is not naturally defined on binary strings, then the domain must be *coded*. This is a nontrivial task for GA's, because the representation influences the computation.
- Real numbers can be block-coded into sequences of integers.
- For integers, the *Gray code* should be considered as an alternative to the standard binary representation. In the Gray code the binary representation of integer $k+1$ differs from that of integer k in only one bit. Thus, mutating a Gray coded integer by one bit can only change its value by ± 1.

Gray code conversion

integer (k)	*standard* ($a_1 a_2 a_3$)	*Gray* ($b_1 b_2 b_3$)
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

- standard $\rightarrow$ Gray conversion: $b_i = \begin{cases} a_i, & i = 1, \\ a_{i-1} \oplus a_i, & i > 1 \end{cases}$
- Gray $\rightarrow$ standard conversion: $a_i = \bigoplus_{j=1}^{i} b_j$

Other coding issues

- Cycles/permutations
- Trees
- Graphs
- ...

10.4 Evolutionary Strategies

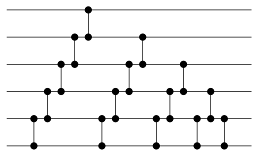
- Evolutionary methods for continuous optimisation (Bienert, Rechenberg, Schwefel et al. 1960's onwards). Unlike GA's, some serious convergence theory exists.
- Goal: maximise objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Use population consisting of individual points in $\mathbb{R}^n$.
- Genetic operations:
  - Mutation: Gaussian perturbation of point.
  - Recombination: Weighted interpolation of parent points.
  - Selection: Fitness computation based on f. Selection either completely deterministic or probabilistic as in GA's.
- Typology of deterministic selection ES's (Schwefel):
  - Population size μ. λ offspring candidates generated by recombinations of μ parents.
  - $(\mu+\lambda)$-selection: best μ individuals from μ parents and λ offspring candidates together are selected.
  - (μ,λ)-selection: best μ individuals from λ offspring candidates alone are selected; all parents are discarded.

10.5 Coevolutionary Genetic Algorithms (CGA)

- Hillis (1990), Paredis et al. (from mid-1990's)
- Idea: coevolution of interacting populations of solutions and tests/constraints as "hosts and parasites" or "prey and predator".
- Goals:
  1. Evolving solutions to satisfy a large & possibly implicit set of constraints.
  2. Helping solutions escape from local minima by adapting the "fitness landscape".

Coevolution of sorting networks (1/4)

- Sorting networks: explicit designs for sorting a fixed number n of elements.
- E.g. sorting network representing "bubble sort" of $n = 6$ elements:



- Interpretation: elements flow from left to right along lines; each connection ("gate") indicates comparison of corresponding elements, so that smaller element continues along upper line and bigger element along lower line.
- Quality measures: size = number of gates (comparisons), depth ("parallel time").

Coevolution of sorting networks (2/4)

- Quite a bit of work in the 1960's (cf. Knuth Vol. 3); size-optimal networks known for $n \leq 8$; for $n > 8$ the optimal design problem gets difficult.
- "Classical" challenge: $n = 16$. A general construction of Batcher & Knuth (1964) yields 63 gates; this was unexpectedly beaten by Shapiro (1969) with 62 gates, and later by Green (1969) with 60 gates. (Best known network.)
- Hillis (1990): Genetic and coevolutionary genetic algorithms for the $n = 16$ sorting network design problem:
  - Each individual represents a network with between 60 and 120 gates.
  - Genetic operations defined appropriately.
  - Individuals not guaranteed to represent proper sorting networks; behaviour tested on a population of test cases.
  - Population sizes up to 65536 individuals, runs 5000 generations.

Coevolution of sorting networks (3/4)

- Result when population of test cases not evolved: 65-gate sorting network.
- Coevolution:
  - Fitness of networks = % of test cases sorted correctly.
  - Fitness of test cases = % of networks fooled.
  - Also population of test cases evolves using appropriate genetic operations.

Coevolution of sorting networks (4/4)

Result of coevolution: a novel sorting network with 61 gates: