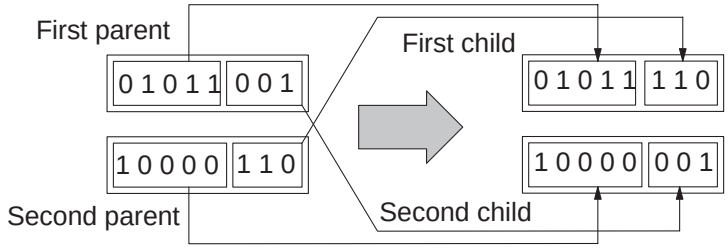
GENETIC ALGORITHMS (GAs)

Principles:

* Based on analogy with evolution process in nature.
* Works with a *population* of feasible solutions, instead of a single feasible solution.
* Each feasible solution is encoded in a linear data structure, usually a bit string, called a *chromosome.*
* Two *parent* chromosomes are combined by *crossover* to form one/two *child* chromosomes.
* Optimization based on ‘‘survival of the fittest’’: prefer parents with better costs for mating.
* *Derivative-free optimization:* it can be used both for continuous and discrete optimization. Example application: use GAs to find the weights in a neural network.

GENETIC ALGORITHMS: ILLUSTRATION



GENETIC ALGORITHMS: CODE

```
genetic()
{
   pop ← ∅;
   for (i ← 1; i ≤ pop_size; i ← i + 1)
      pop ← pop ∪ {“chromosome of random feasible solution”};
   do {
      newpop ← ∅;
      for (i ← 1; i ≤ pop_size; i ← i + 1) {
         parent1 ← select(pop);
         parent2 ← select(pop);
         child ← crossover(parent1, parent2);
         newpop ← newpop ∪ {child};
      }
      pop ← newpop;
   } while (!stop());
   “report best solution”;
}
```


SITUATION IN NATURE

* All genetic information is encoded in chromosomes. The total genetic information is called the *genotype.*
* After a complex chemical process, the genetic information comes into expression in a living being called the *phenotype.*
* A chromosome consists of two long strings of DNA.
* Each string in a chromosome consists of gene sequences.
* Only one of the pairs of genes comes into expression in the phenotype.
* Sexual reproduction involves separation of the DNA string pairs.
* Effects like crossover, inversion and mutation contribute to the creation of new genetic material.

Conclusion: GAs do not closely follow the natural process.

DARWINIAN AND LAMARCKIAN EVOLUTION

* A principle of *Darwinian* is that the genotype controls the phenotype but that the reverse is never true.
* Evolution in which the phenotype affects the genotype is called *Lamarckian.*
* In the context of GAs, Lamarckian evolution means that individuals are e.g. “repaired” before becoming part of the new population. This can be done with the goal of:
  + putting solutions whose encodings do not represent a feasible solution back into the set of feasible solutions;
  + applying a local optimization on an individual.

IMPORTANT ISSUES

* *Fitness calculation:* different ways to calculate the fitness may affect the performance of the algorithm.
* *Chromosome representation:* binary strings, numbers, or other application-dependent data structures.
* *Selection mechanism:* parents for the new generation can be selected in different ways.
* *Child creation:*
  + by means of various *crossover operators;*
  + by means of *mutation*, a random modification of part of a chromosome.
  + by means of “copying without modification”.
* *Replacement scheme:* the way a population maintains a constant size.

FITNESS

* The goal of optimization may be either *maximization* or *minimization* of a cost function.
* However, GA always maximizes *fitness*, which should be a *positive* value due to the way that fitness is used during selection.
* Minimization problems can, therefore, not simply be transformed to maximization problems by multiplication by -1.
* So, it is necessary to assign a suitable fitness value to each feasible solution such that a higher fitness value indicates a higher desirability of the associated solution.

FITNESS FUNCTIONS (1)

Recall optimization functions. An instance $I = (F, c)$, where:

* F is the *set of feasible solutions,* and
* c is a *cost function,* assigning a cost value to each feasible solution; $c : F \rightarrow R$

Consider a minimization problem:

* A possible choice for the fitness function $h(i)$ of some feasible solution s_i encoded by chromosome i ($1 \leq i \leq N_p$, N_p being the population size) is:

$$h(i) = \begin{cases} C_{\max} - c(s_i), & \text{if } c(s_i) < C_{\max} \\ 0, & \text{otherwise} \end{cases}$$

* The constant $C_{\max}$ can be fixed, dependent on the problem instance or dependent on the actual costs in the current generation.

FITNESS FUNCTIONS (2)

* If the goal is maximizing a cost function, but the cost may become negative, a possible choice for the fitness function h is:

$$h(i) = \begin{cases} c(s_i) + C_{\min}, \text{ if } c(s_i) + C_{\min} > 0 \\ 0, \text{ otherwise} \end{cases}$$

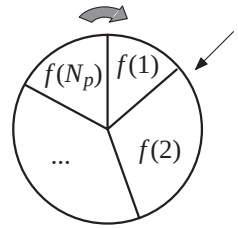
* *Fitness scaling* is applied to adjust the range of fitness values during the search in order to avoid that individuals with extreme low or high fitness values have too much influence on the search. Examples are:
  + *Linear scaling:* $h'(i) = ah(i) + b$.
  + *Ranking:* sort the individuals in the population according to their fitness values and assign new fitness values e.g. starting with 1 for the worst, 2 for the next, etc.

SELECTION MECHANISMS (1)

* Consider the *normalized fitness* $f(i)$ derived from the fitness $h(i)$:

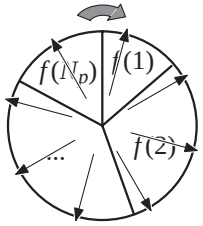
$$f(i) = \frac{h(i)}{\sum_{k=1}^{N_p} h(k)}$$

* The normalized fitness can be used as a probability for the selection of an individual for participation in the next generation. This is called *roulette-wheel* selection. It is one of the most popular selection mechanisms.
* Roulette-wheel selection cannot guarantee a proportional representation in the next generation.



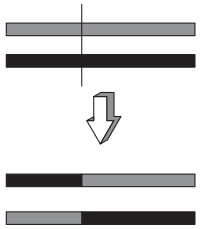
SELECTION MECHANISMS (2)

* In *tournament* selection, a subset of m individuals is uniformly drawn from the population and the one with the best fitness from this subset is selected. This mechanism will lead to a more than proportional representation of the individuals with better fitness.
* In *stochastic universal sampling* all individuals that need to be selected are selected at the same time by sampling the roulette wheel with equidistant pointers. This will lead to a representation of individuals proportional to their fitness. Example with 8 pointers:

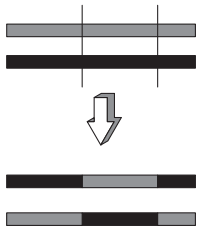


CROSSOVER OPERATORS (1)

* *One-point* crossover: cut the chromosomes at one position and combine. A disadvantage is that patterns at the left and rightmost ends of a chromosome are never kept together.

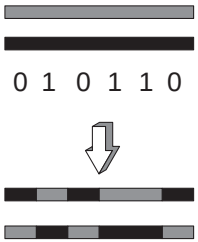


* *Two-point* crossover: cut the chromosomes at two positions and exchange.



CROSSOVER OPERATORS (2)

* *Uniform* crossover: generate a random bit string and copy from one parent or the other depending on bit value.



* Problem-specific crossover operators may also be very useful. An example for TSP follows later.

REPLACEMENT SCHEME

* One can either construct an entire new generation of the population in each iteration or have a *steady-state* population to and from which individuals are continually added and removed.
* In the latter case, one needs to have a selection mechanism for discarding individuals, e.g. based on $1 -$ *'normalized fitness'*. One can also consider the union of the old population and the set of new individuals and use fitness-based selection to obtain a new population of size N_p.
* In both cases, one can opt for an "elitist strategy", in which special measures are taken to make sure that the best solution is never discarded.

APPLICATION TO TSP (1)

* Given is a set of cities $\{c_1, \ldots, c_N\}$ and the pairwise distances between the cities. Find the shortest tour that visits all cities exactly once.
* *Encoding:* as opposed to many GA algorithms where the chromosome contains slots for subsequent parameter values that can be chosen independently, it is convenient here just to encode the sequence of cities in a tour. Example for $N = 8$:

$$c_1 \mid c_4 \mid c_5 \mid c_2 \mid c_3 \mid c_7 \mid c_8 \mid c_6$$

* Note that the crossover operators discussed earlier would not work on this encoding: they would produce illegal solutions.

APPLICATION TO TSP (2)

* A suitable crossover operator for the encoding chosen is *order crossover* (it is useful in many other combinatorial optimization problems that look for an optimal ordering). It uses a single cut point, copies the code from the first parent until the cut point into the child and then adds the missing cities using the order in which they appear in the second parent. Example:

First parent: $c_1 \mid c_4 \mid c_5 \mid c_2 \mid c_3 \mid c_7 \mid c_8 \mid c_6$

Second parent: $c_5 \mid c_8 \mid c_1 \mid c_2 \mid c_6 \mid c_7 \mid c_4 \mid c_3$

Child using cut point after second city: $c_1 \mid c_4 \mid c_5 \mid c_8 \mid c_2 \mid c_6 \mid c_7 \mid c_3$

* Many other crossover operators can be thought of that preserve correctness in chromosomes that encode permutations.

APPLICATION TO TSP (3)

* The approach just described is a non-Lamarckian GA for TSP.
* In practice, the most successful GAs for TSP are Lamarckian: solutions are optimized according to some locally optimal criterion before being added to the population.
* For a fixed computation time, special-purpose heuristics give the best solutions. If one is prepared to invest more time, better solutions are found by combinations of GAs (or simulated annealing) and special-purpose heuristics [3].
* Algorithms that combine general-purpose and special-purpose techniques are called *hybrid* algorithms.

SCHEMATA

* Consider an application with bit-string encodings.
* A pattern in a bit string is called a *schema* (plural: *schemata*). One can also say that schemata specify subspaces of the Boolean space containing all bit strings of a given length.
* Example bit string:

  1 0 1 1 0 1 0 0.
* Schemata in this string are (a * indicates a don't care):

  * 0 1 * * * * *

  1 * * 1 0 * 0 *
* One can assume that schemata have a fitness themselves: the subspaces they indicate may or may not be interesting for the search.

SCHEMA THEOREM (1)

* The normalized fitness of a schema S is given by $f(S)$: it is the average fitness of all individuals carrying schema S.
* Let $m(S,t)$ denote the number of individuals carrying schema S in a population present at time t.
* Then, assuming a selection mechanism proportional to fitness (where f_{ave} is the average fitness in the population):

$$m(S,t+1) = m(S,t)N_p \frac{f(S)}{\sum_{i=1}^{N_p} f(i)} = m(S,t)\frac{f(S)}{f_{ave}}$$

* Suppose that the fitness of schema S is slightly better (or worse) than average:

$$f(S) = (1+c)f_{ave}$$

SCHEMA THEOREM (2)

* Then:

$$m(S,t+1) = m(S,t)(1+c) = m(S,0)(1+c)^{t+1}.$$

* This means that the presence of a schema with a fitness above (below) average, will grow (decrease) exponentially in the population.
* A similar result holds if the effects of crossover and mutation is taken into effect. The conclusion is then that the non-don't-care bits of schemata should be closely grouped to have a higher chance to survive.

DISCUSSION

* GAs are a powerful optimization technique. However, one needs extensive experimentation before finding the right “settings of the buttons”: Which chromosome encoding should be used? Which crossover operator? Which replacement scheme? How large should the population size be chosen? etc.
* Many more variations exist, for example:
  + Parallel implementations that evolve independently but in which individuals from time to time migrate;
  + Implementations for which the fitness computation is expensive due to many test cases that have to be evaluated; a solution is co-evolve *parasites*, sets of difficult test cases (see book).

FURTHER READING

* A “classic” textbook on GAs is [1].
* A GA book that discusses many applications in electrical engineering is [2].
* The state-of-the-art for TSP solution techniques including genetic algorithms, neural-net approaches, etc. can be found in [3].

[1] Goldberg, D.E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison–Wesley, Reading, Massachusetts, (1989).

[2] Man, K.F., K.S. Tang and S. Kwong, Genetic Algorithms, Springer, London, (1999).

[3] Johnson, D.S. and L.A. McGeoch, ”The Traveling Salesman Problem: A Case Study”, In: E. Aarts and J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, John Wiley and Sons, Chichester, pp. 215–310, (1997).