Evolutionary Algorithms

There are several types of machine learning based on evolutionary principles

- Genetic algorithms: general purpose, first developed by John Holland
- Evolutionary strategies: aim at optimization, emphasize parallelism and mutation
- Genetic programming: uses code pieces as building blocs to directly create programs

Evolutionary Algorithms

Why use evolutionary models for computational problems?

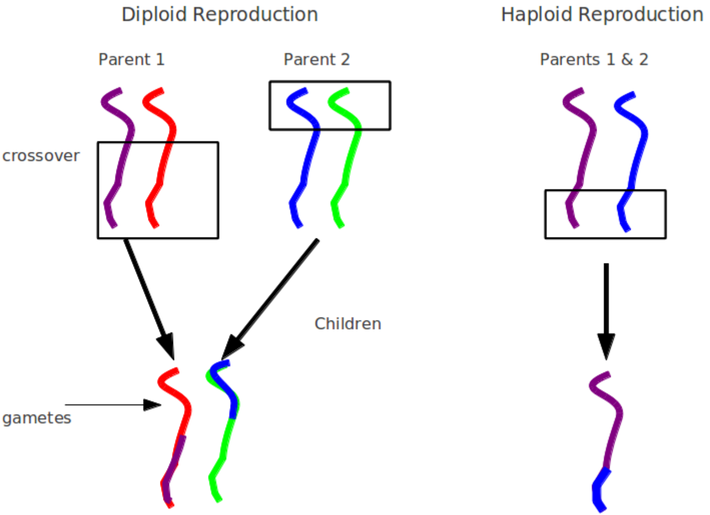
- You don't have a strong bias regarding possible solutions
- Parallel search may be necessary for large problems.
- Evolutionary history suggests genetic search can work.

Tenuous links to biological genetics

We are interested in inheritance, but not in cell replication.

- Much of the terminology is adopted from genetics, but is used very informally.
- All biological organisms consist of cells. Within cells are chromosomes, strings of deoxyribonucleic acid (DNA) that determine how the organism develops.
- Genes are subsequences within a chromosome. Genes encode particular traits. All of an organism's genes taken together make up its genome.
- Under development, the genome becomes a phenotype (genetic manifestation as an organism)
- Natural selection operates on the phenotype, but the genotype is the vehicle of inheritance

- Diploid organisms: chromosomes arranged in pairs. Animals that sexually reproduce usually have diploid chromosomes.
- Haploid organisms: single-stranded chromosomes (viruses).



Artificial Evolution

Artificial evolution is inspired by four elements of natural evolution:

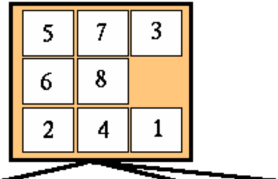
1. maintenance of a population
2. creation of diversity
3. a selection mechanism
4. a process of genetic inheritance

Evolution implements Search

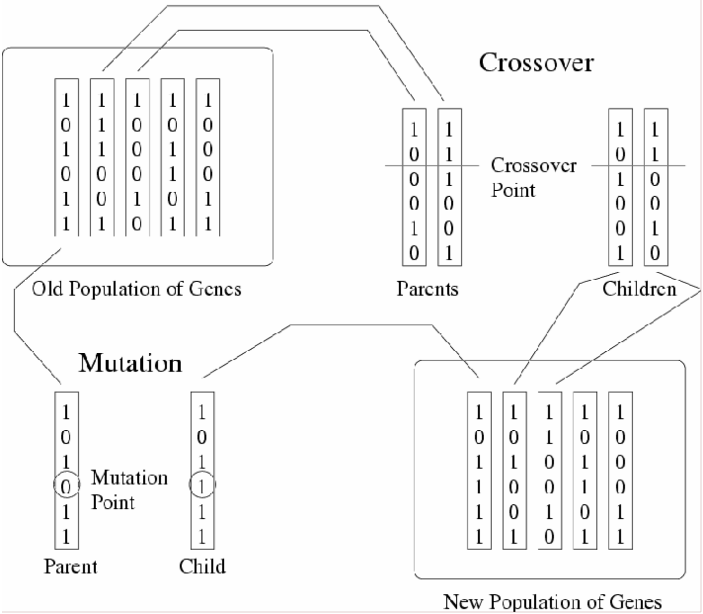
Search is a component of many problems that involve large quantities of information.

- Searching for stored data: (data retrieval) in presorted data.
- Searching for paths to goals. (DFS, BFS, A*). Search uses actions to move among a number of discrete states. States are linked by a successor function. This includes games (chess, backgammon) and even searching for causes in an expert system. Critically, path evaluation is possible.
- Search for any solution whatsoever. This is the general form of search implemented by GAs. It subsumes searching for paths to goals.

  Other general methods include hill-climbing and simulated annealing.



Genetic Algorithm



A Genetic Algorithm

GA(*Fitness*, *Fitness_threshold*, p, r, m)

- *Initialize:* $P \leftarrow p$ random hypotheses
- *Evaluate:* for each h in P, compute *Fitness*(h)
- While $[\max_h Fitness(h)] < Fitness\_threshold$
  1. *Select:* Probabilistically select $(1 - r)p$ members of P to add to P_S.

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

  2. *Crossover:* Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P. For each pair, $\langle h_1, h_2 \rangle$, produce two offspring by applying the Crossover operator. Add all offspring to P_s.
  3. *Mutate:* Invert a randomly selected bit in $m \cdot p$ random members of P_s
  4. *Update:* $P \leftarrow P_s$
  5. *Evaluate:* for each h in P, compute *Fitness*(h)

Return the hypothesis from P that has the highest fitness.

Selecting Most Fit Hypotheses

Fitness proportionate selection:

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

... can lead to *crowding*

Tournament selection:

- Pick h_1, h_2 at random with uniform prob.
- With probability p, select the more fit.

Rank selection:

- Sort all hypotheses by fitness
- Prob of selection is proportional to rank

Example: Knapsack Problem

The Knapsack Problem is NP-complete (basically, runs in exponential time in its inputs).
You want to pack your luggage for the holidays. You bought a big suitcase, but you can't take everything. If every items takes a certain amount of space, what is the greatest amount of space you can use up without going over?

- String Representation: Use 0/1 to represent presence of item.
- Fitness: Want to measure volume, but penalize over-volume configurations.
- Population: Just random binary strings.

Example: Knapsack Problem

The Knapsack Problem is NP-complete (basically, runs in exponential time in its inputs).
You want to pack your luggage for the holidays. You bought a big suitcase, but you can't take everything. If every items takes a certain amount of space, what is the greatest amount of space you can use up without going over?

- String Representation: Use 0/1 to represent presence of item.
- Fitness: Want to measure volume, but penalize over-volume configurations.
- Population: Just random binary strings.

Example: Knapsack Problem

The Knapsack Problem is NP-complete (basically, runs in exponential time in its inputs).
You want to pack your luggage for the holidays. You bought a big suitcase, but you can't take everything. If every items takes a certain amount of space, what is the greatest amount of space you can use up without going over?

- String Representation: Use 0/1 to represent presence of item.
- Fitness: Want to measure volume, but penalize over-volume configurations.
- Population: Just random binary strings.

Schemas

How to characterize evolution of population in GA?
Schema = string containing 0, 1, * (“don’t care”)

- Typical schema: 10**0*
- Instances of above schema: 101101, 100000, ...

Characterize population by number of instances representing each possible schema

- $m(s, t) =$ number of instances of schema s in pop at time t

Consider Just Selection

- $\bar{f}(t) =$ average fitness of pop. at time t
- $m(s,t) =$ instances of schema s in pop at time t
- $\hat{u}(s,t) =$ ave. fitness of instances of s at time t

Probability of selecting h in one selection step

$$\begin{aligned}\Pr(h) &= \frac{f(h)}{\sum_{i=1}^{n} f(h_i)} \\ &= \frac{f(h)}{n\bar{f}(t)}\end{aligned}$$

Probability of selecting an instance of s in one step

$$
\begin{aligned}
\Pr(h \in s) &= \sum_{h \in s \cap p_t} \frac{f(h)}{n\bar{f}(t)} \\
&= \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t)
\end{aligned}
$$

Expected number of instances of s after n selections

$$
E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)
$$

Schema Theorem

$$E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t) \left(1 - p_c \frac{d(s)}{l-1}\right) (1-p_m)^{o(s)}$$

$m(s,t) =$ instances of schema s in pop at time t
$\bar{f}(t) =$ average fitness of pop. at time t
$\hat{u}(s,t) =$ ave. fitness of instances of s at time t
$p_c =$ probability of single point crossover operator
$p_m =$ probability of mutation operator
$l =$ length of single bit strings
$o(s)$ number of defined (non "*") bits in s
$d(s) =$ distance between leftmost, rightmost defined bits in s