# GA

**Bernoulli's Principle of Insufficient Reason** if we don't know anything about our problem space, all events have equal probability

**Phenotypic Traits** the physical or behavioral features observed by the environment

**Genotypic Traits** the features observed by the algorithm

**Termination Criteria**

- number of generations
- minimum threashold reached
- no improvement
- memory/time constraints

**Crossover** is explorative

**Mutation** is exploitative

**Population Models** Generational model (GGA) is each individual survives for only one generation, and steady-state model (SSGA) is that only part of the population is replaced

## Survivor Selection

Which of a parents and offspring go into the next generation (what part of current population carries over)

**Age-Based Selection** Delete a random element or use FIFO

**Fitness-Based Selection** delete or replace based on fitness

**Elitism** Always keep the best individuals

**Genitor** Always kill the worst individuals

## Parent Selection

Which parents will get to produce offspring

**Fitness Proportionate** A parent is selected using probability based on its fitness over the total fitness of all parents

**Roulette Wheel Algorithm** Spin a one armed roulette wheel n times to chose n parents, a different version of FPS

**Baker's Stochastic Universal Sampling** Spin a roulette wheel with n evenly spaced arms, different version of FPS

**Rank Based Selection** Sort parents by fitness then make probability based on rank, deals with the fact that a very fit parent can take over in FPS

**Tournament Selection** pick k parents at random and select the best of these, deals with the overhead of needing global knowledge of other parents

## Where GAs are Useful

- hightly multimodal functions
- discrete or discontinuous functions
- high-dimensionality functions
- non-linear dependencies between parameters

## Difficulties with GA

- premature convergence
- unable to overcome deception
- need more evaluations than time permits
- bad match of representation making operators destructive
- diased or incomplete representation
- problem too hard/easy

## Adaptation

**Parameter Tuning** finding suitable parameters before the algorithm is run

**Deterministic Parameter Control** replace any parameter p with a function p(t) where t is the generation number, this does not take into account the progress of the search. For example changing $\sigma$ to $\sigma(t) = 1 - \frac{0.9t}{T}$ where t is the generation number and T is the maximum generation number.

**Adaptive Parameter Control** take in feedback from the current state and use some heuristic to control the parameters. For example, Rechenberg's success rule. If we have too many successful children multiply $\sigma$ by c and if we have too few divide $\sigma$ by c, where c is some constant. Moves that are not successful are moving too much and should get closed to what was a good state. Moves that are very successful should try larger steps to increase efficiency of search.

**Rechenberg's 1/5 Success Rule** We want to have roughly 1/5 of our mutations result in successful offspring (produces an individual better than its parent).

**Self Adaptive Parameter Control** incorporate the parameter values into the chromosomes so that their control with be driven through GA, see ES for more examples

**Adaptive Crossover** Change the operation being used for crossover, or change the probability of crossover, or change the parameter used by the crossover function (for instance, the point of swapping)

# ACO

**Stigmergy** indirect agent interaction and modification of environment, use environmental modification as external memory, work continued by any individual

**Binary Bridge Experiment** Two routes to a food source, ants converged to using only one bridge (converged on shortest if lengths were unequal or random one if equal)

**Ant Density Model** Ants apply constant Q to edges

**Ant Quantity Model** Ants apply Q/d to edge where d is lenght of edge

## Ant System

ACO, but add online delayed update where ants update edges with Q/L where L is the total length of their path once they have finished the path

## Max-Min Ant System

AS, but only the very best ant gets to update pheromones (either in the current iteration or the best solution overall). Values of pheromone are restricted between a max and min to allow high exploration at the beginning, near max, and high exploitation later, near min. The max and min are chosen experimentally or they can be calculated if the optimal solution is already known

## Ant Colony System

Based on AS, but uses elitist strategy. Adds constant q0 to give us a probability of just selecting the best possible edge, else we just chose a random edge with fitness in mind. Ants use only best solution when updating and incorporate an offline update for each ants last traversed edge.

## Where ACO is Useful

- combinatorial problems
- not solvable with classical optimization methods
- discrete optimization problems
- some variants of ACO can handle continuous

## Advantages

- stocahstic, population based (like GA)
- retains memory of entire colony instead of just previous generation
- less affected by poor initial soluitons due to combinations of random path selection
- proved successful in many applications
- can handle dynamic environments

## Disadvantages

- mostly experimental, proofs are hard
- lots of parameters
- make take a while to converge

## Adaptation

**ACSGA** Have GA running on top of ACS to optimize its parameter values. Encode $q_0, \alpha, \beta$ in chromosome and run GA.

**Near Parameter Free** Each ant can select the suitable values for its parameters $q_0, \alpha, \beta, \rho$. Keep a separate pheromone matrix for learning parameters. We have to discretize parameter values to store them in matrix (break up parameter range into chunks and chose one). Also have to keep upper and lower bounds in mind.

## Cooperation

**Heterogeneous** Ants in colonies have different behaviors (optimizing different criteria)

**Homogeneous** Ants in colonies have the same behavior

## PSO

### Properties

- Flexibility - system performance is adaptive with respect to internal or external changes
- robustness - system always performs even if some individuals fail
- decentralization - control is distributed among individuals
- self-organization - global behaviors appear out of local individual interactions

### Flocking

- homogenity - all the entities are similar, the flock moves without a leader
- locality - any entitiy communicates with nearby entities only
- velocity matching - attempt to match velocity of nearby entities
- collisoin avoidance - avoid colliding with nearby entities
- flock centering - attempt to stay close to nearby entities

### Principles

- proximity - the swarm should be able to carry out simple space and time computations
- quality - the swarm should be able to respond to quality factors in the environment
- diverse response = the swarm should not commit its activities along excessively narrow channels
- stability - the swarm should not change its mode of behavior every time the environment changes
- adaptability - the swarm must be able to change behavior mode when it is worth the computational price

### Behaviors

- separation - each agent tries to move away from its nearby mates if they are too close (collision avoidance)
- alignment - each agent steers towards the average heading of its nearby mates (velocity matching)
- cohesion - each agent tries to go towards the average position of its nearby mates (centering)

**Termination Criteria**

- max number of iterations
- max number of function evaluations
- acceptable solution has been found
- no improvement over some iterations

**Neighborhood topologies**

- Star - use global best, fast propagation of data, gets stuck in local optima
- Ring - use local best, slow propagation, more exploration

**Neighbor Selection** Can calculate distance to all neighbors and take the closest (**Physical Neighbors**). This is expensive to calculate. Can instead chose neighbors based on data structure you are storing your particles in (**Social Neighbors**). A common form of this is to store particles in grid and just grab the four in cardinal directions, called **square topology**.

**Parameters** Most of the time $c_1 = c_2$. Small values will result in smooth curves and large values will result in more acceleration. W is used to balance exploration (large vales) and exploitation (small values).

**Convergence** Study replaced random numbers with expected value of 1/2. To get convergence you need:

$$w < 1$$
$$2 > c_1 + c_2 > 0$$

Usually use w = 0.8, cs = 1.5

**Adaptation** Create tribes of particles with something in common, good tribes (as defined by the ratio of good particles to total particles) will delete their worst particle and bad tribes will spawn a new random particle, these new particles come together to form a new tribe connected to their parents.

## Cooperation

**Concurrent PSO** Multiple swarms are updated in parallel using different algorithms, these swap their gbest values every few iterations

**Cooperative PSO** Multiple swarms are optimizing different values of the solution so fitness of a particle is dependent on the gbest of other swarms. Combine these into a context vector

**Hybrid PSO** One swarm doing regular PSO and one swarm doing cooperative PSO they swap gbest and context vector, CPSO uses gbest to update random particles of its swarms, PSO uses context vector to replace a randomly chosen particle