Evolutionary Computation

What is evolutionary computation?

- Any of a number of techniques/approaches based on or inspired by natural evolution, e.g.
  - Genetic algorithms
  - Genetic programming

Natural evolution

Adaptation based on a combination of *competition* (e.g. for food), *selection* (i.e. surviving long enough to be able to reproduce in a given enviroment), *mutation* (i.e. spontaneous change in the gene itself), and *reproduction* (i.e. producing a new individual with the same or similar genetic makeup).

Evolutionary fitness

The ability of an individual (or a population) to survive and reproduce in a given environment. Darwinian evolution optimizes evolutionary fitness.

As environmental conditions change so does the *adaptive topology* – the multidimensional fitness function. In other words, the goalposts are always moving.

Genetic algorithms

What are GAs?

- A class of stochastic (i.e. probabilistic) search algorithms based on biological evolution.
- Rely on well-defined termination criteria and fitness functions.
- Nodes in the search space (i.e. potential solutions) are represented as *chromosomes* (typically a binary series). Each element in a chromosome is called a *gene*.

A GA run

1. Initialization. Represent the problem domain as a chromosome of length L. Choose population size (N), crossover probability (p_c) and mutation probability (p_m).
2. Define a *fitness function*, which quantitatively measures the success of an individual chromosome.
3. Randomly generate an initial population of size N.
4. Calculate the fitness of each chromosome. If the termination condition is satisfied, stop. Otherwise, continue.

A GA run (2)

5. Select parent chromosomes probabilistically, based on their fitness.
6. Apply the genetic operators (crossover and mutation)
7. Place the generated offspring in the new population. Repeat from 5 until population size is N.
8. Repeat from 4, with the new population.

Representing candidate solutions as chromosomes

- A chromosome is typically a binary string (although other representations exist).
- If potential solutions are numbers, or series of numbers, then encoding in this representation is straightforward.
- If potential solutions are more complex entities, **encoding can be the hardest part of the problem**!

The fitness function

- Should be a relatively simple (i.e. quick to calculate) measure of an individual’s fitness (i.e. how close it is to a solution).
- A domain-specific, heuristic measure
- In a GA, N individuals are evaluated in each generation, and there are typically many (hundreds or thousands) of generations – so a complex fitness function can slow things down a lot.
- Coming up with a good fitness function can also be very difficult! Try a weighted sum of desirable features…

The crossover operator

Corresponds to bisexual reproduction in natural selection (typically, p_c=.7). If it fires:

1. A point in the length of the parent chromosomes is randomly selected.
2. The two offspring are:
    1. The first part of Parent A plus the second part of Parent B, and
    2. The first part of Parent B plus the second part of Parent A.

If it doesn’t fire, the offspring are (typically) simply reproductions of the parents.

Other kinds of crossover

- Multi-point crossover: As before, but with several points selected.
- Uniform crossover: Genes are randomly selected from each parent.
- Arithmetic crossover: The offspring's genes are some function of parents', e.g. AND.

Roulette Wheel Selection: most common method for selecting ‘breeders’

1. Calculate the fitness for each member of the population N.
2. Calculate the fitness ratio for each member:
3. Give each member a portion of the ‘roulette wheel’ (i.e. range from 1 to 100) corresponding to its fitness ratio.
4. Choose a random number between 1 and 100, and select the member corresponding to that number.
5. Repeat N times – this is your breeding population.

The mutation operator

Represents random mutation in nature. Typically very low probability (e.g. p_m=.001). If it fires:

1. A gene in the chromosome is randomly chosen.
2. It is flipped to the opposite value (or, if the chromosome is not binary, some other change function is applied).

If this operator doesn't fire, the chromosome is unaffected.

Why have mutation?

- Helps knock the population out of local maxima.
- Introduces diversity, just in case the best solution can’t be reached from the initial population.

Elitism

If the GA is set to use *elitism*, the most fit members of the current population are simply copied over to the new population, before the other operators are applied.

Examples

- http://rednuht.org/genetic_cars_2/
- http://rednuht.org/genetic_walkers/

Exercise: Play around with the parameters on these. What works best?

Genetic programming

What is genetic programming?

- Like GAs, GPs use evolutionary techniques to solve problems.
- Unlike GAs, the solutions are *programs.*
- The goal is to produce programs which can solve problems, without explicitly programming them.
- Chromosomes are both *data* (i.e. can be manipulated) and *programs* (i.e. can be run to determine fitness).

The GP process

1. *Determine the set of terminals* (i.e. inputs to the program)
2. *Select the set of primitive functions* (e.g. the set of mathematical operators, or perhaps more sophisticated functions)
3. *Define the fitness function*. Typically, the sum of the absolute errors over a number of fitness cases (sample inputs).
4. *Determine key parameters* (e.g. population size, max number of generations, as with GAs).
5. Run as with GAs.

Encoding programs as chromosomes

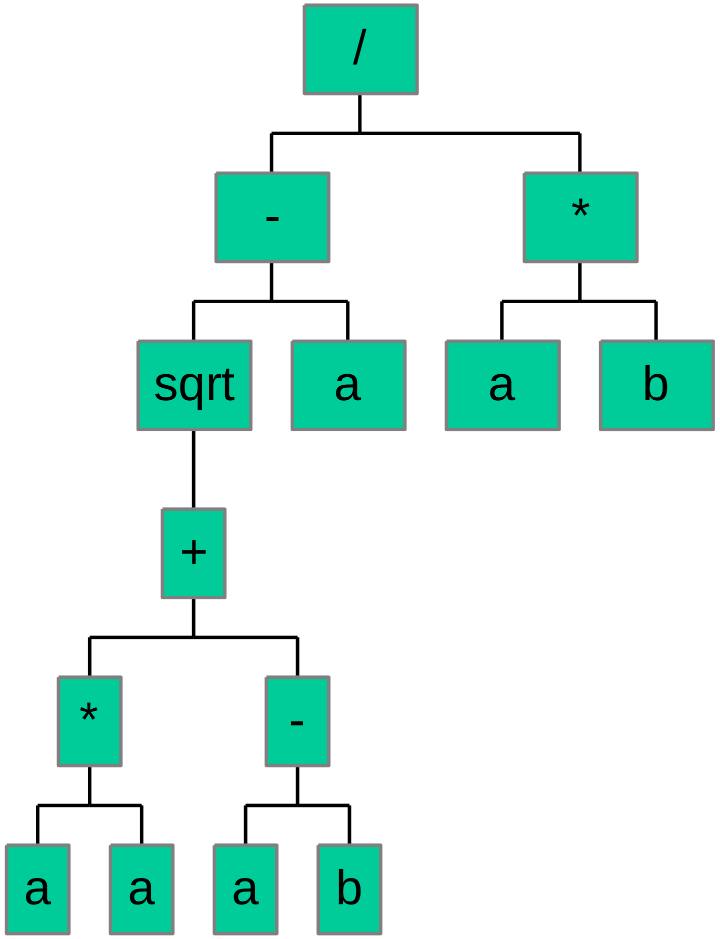
Consider the LISP 'program':

(/ (- (sqrt (+ (* a a)(- a b))) a)(* a b))

Which is equivalent to:

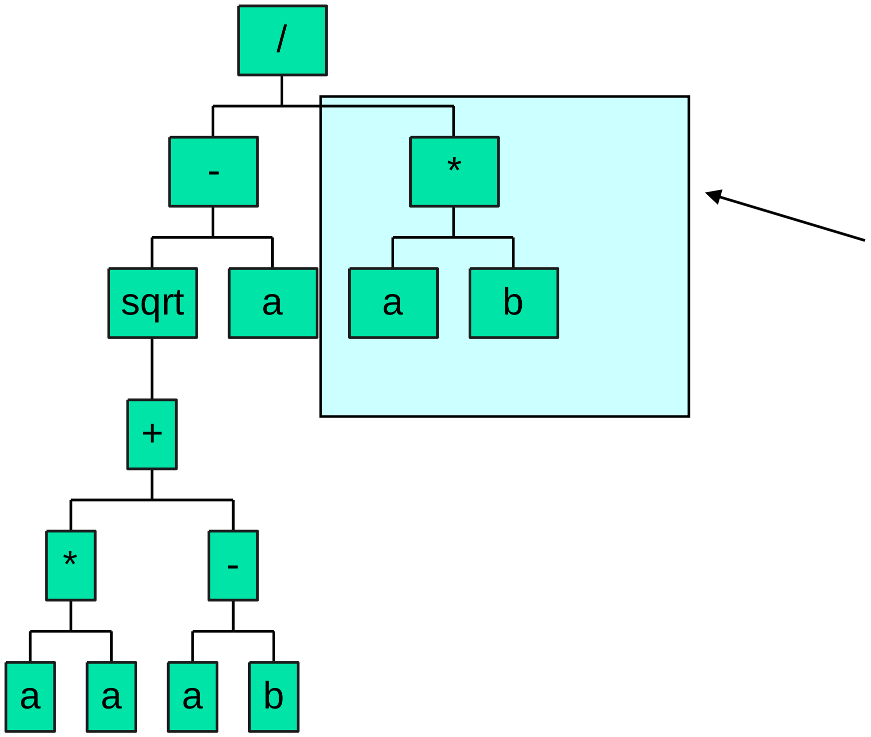
$$\frac{\sqrt{a^2 + (a-b)} - a}{ab}$$

Which can be represented as the tree:



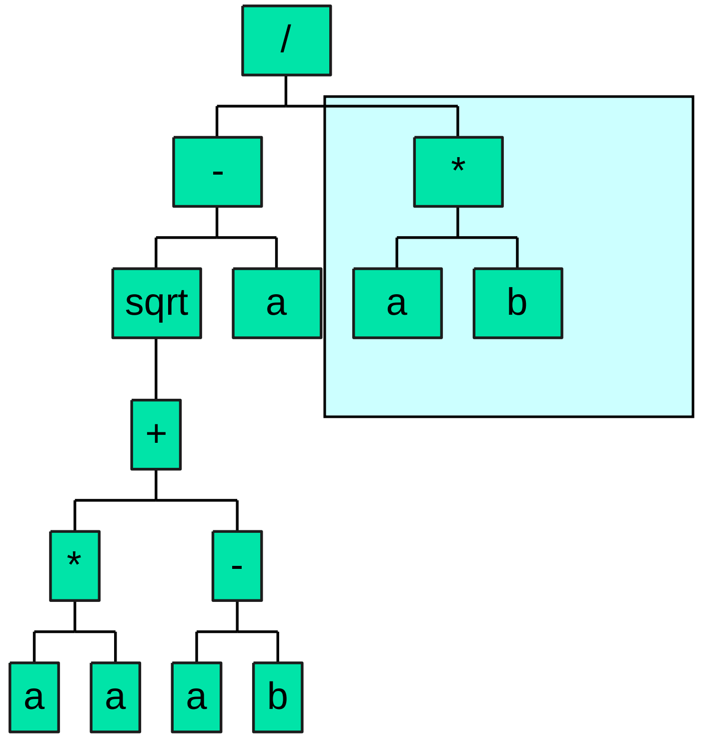
Crossover

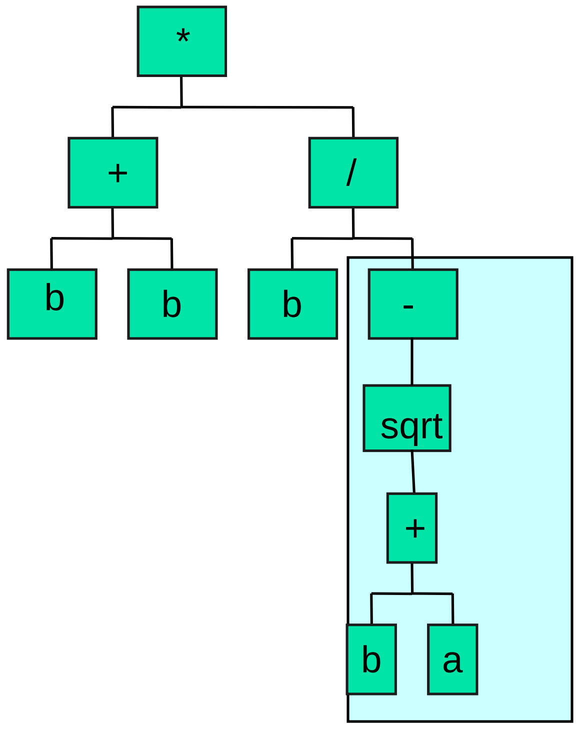
A branch is randomly chosen as the crossover point, e.g.:



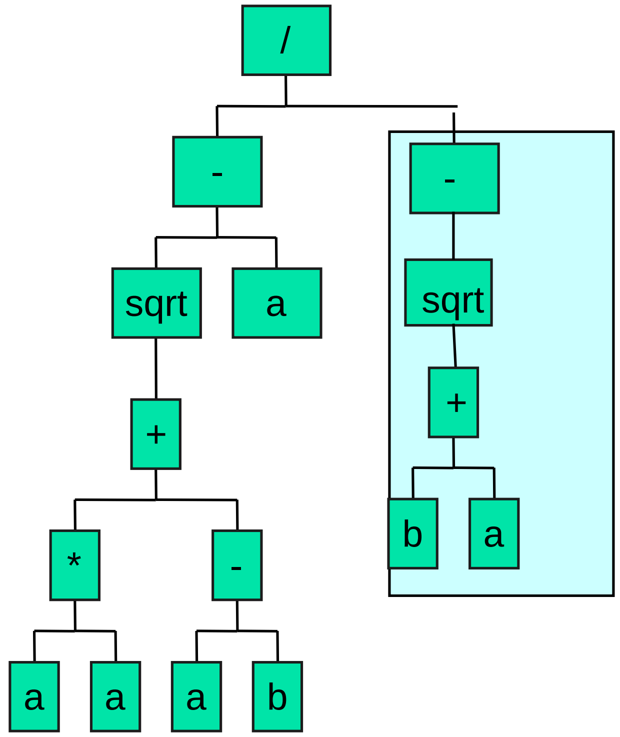
Fragment to be exchanged with fragment from other parent

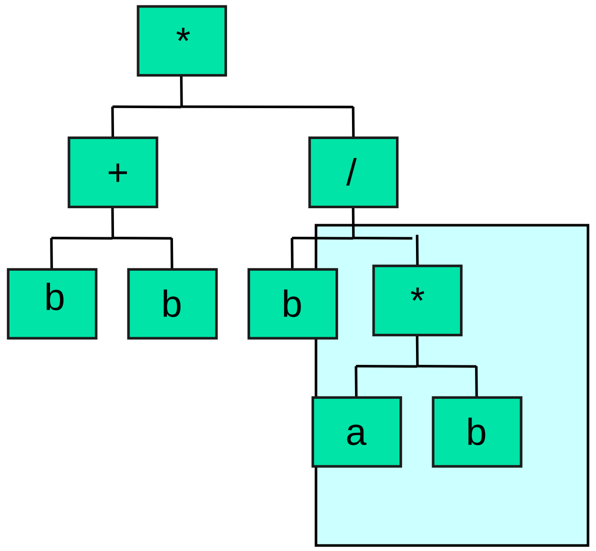
Crossover example (parents)





Crossover example (children)





Mutation

There are two types of mutation:

- An operator node can be randomly changed to another operator
- A terminal node can be randomly changed to another terminal.

Fitness-based selection methods:

- Probabilistically based on fitness
- *Tournament*: Potential mates for a given individual are compared, and the fittest is chosen
- *Ranking*: Individuals are ranked based on fitness, and top individuals are chosen.

GAs: A case study
Maintenance Scheduling

Scheduling

Scheduling problems tend to be:

- NP complete
- Unmanageable for uninformed search
- Difficult to specify heuristics for
- Highly constrained, with many different kinds of constraints
- Tolerant to non-optimal solutions, as long as they are *near* optimum.

All of these features make them appropriate problems for GAs.

Step 1: Define the problem

- Seven power units, with differing unit capacity (e.g. 20MW) and maintenance requirements (e.g. how long it takes to do maintenance).
- Four maintenance intervals per year.
- When being maintained, a power unit must be shut down.
- The power reserve (total capacity of system – power loss due to maintenance – maximum load forecast during maintenance period) must never drop below zero.

Step 2: Represent possible solutions as chromosomes

- Here, we can divide the chromosome into seven parts – one for each unit.
- We can encode one constraint (i.e. how long it takes to do maintenance) into the chromosome itself, by making the gene a 4-bit string, where 1 indicates that the unit will be maintained in that interval (e.g. 0100).
- So, a chromosome is 7 4-bit genes, one for each unit.
- There are other ways to represent solns as chromosomes – any better ideas?

Step 3: Define the fitness function

- The main fitness constraint is that no interval have less than zero net reserve. How would you define a function to evaluate the fitness of a given schedule (i.e. chromosome)?

Step 4: Construct genetic operators

Make each 4 bit gene indivisible (why?).

- Crossover: randomly select a point between genes, and exchange halves of parents. $p_c=0.7$
- Mutation: randomly replace a gene with another from the pool for that unit (so that only legal schedules are constructed). $p_m=0.001$

Step 5: Run the GA

1. Choose population size N and stopping condition (max number of generations and/or fitness threshold).
2. Generate initial population.
3. Stopping condition met? If so, stop.
4. Generate new population, and go to 3.

Another exercise:

Design a GA that can generate an observation schedule (1 night) for an automatic telescope. A schedule is an ordered list of observations selected from a set of observation requests. Each observation has a different viewing window (e.g. Mars can be seen between 1am and 3am), location in the sky, probability of success, and priority. The schedule should (in order of importance): minimize the amount of time during the night that the telescope is idle, maximize the priority of the scheduled observations, maximize the probability of success, and minimize the distance moved between observations.