Stochastic Search: Part 2

Genetic Algorithms

Vincent A. Cicirello

Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

cicirello@ri.cmu.edu

The Genetic Algorithm (GA)

The genetic algorithm is based on the mechanics of natural genetics and natural selection.

- Population based
- Survival of the fittest
- Variation
- Mutations

The Simple GA: Representation

You begin with a population of random bit strings. Each bit string encodes some problem configuration.

For example, you can encode SAT by representing each boolean variable by a position in the bit string:

A	B	C	D	E	F	G	H
1	0	0	0	1	0	0	1
0	1	1	1	0	0	0	1
1	0	0	1	0	1	1	1
0	1	1	0	0	1	0	1

$$
\begin{array}{c}
A \vee \neg B \vee C \\
\wedge \\
\neg A \vee C \vee D \\
\wedge \\
B \vee D \vee \neg E \\
\wedge \\
\neg C \vee \neg D \vee \neg E \\
\wedge \\
\neg A \vee \neg C \vee E
\end{array}
$$

And you begin with a population of n randomly generated individuals.

The Simple GA: Selection

Selection determines which from among your initial population reproduce.

As in Nature, the fittest survive.

We need a measure of fitness....

For example, for SAT we can use the number of satisfied clauses.

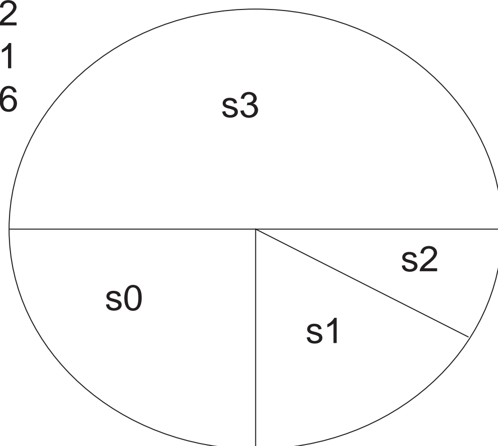
The Simple GA: Selection

Once we have a fitness function, how do we choose which individuals survive?

Fitness proportionate (Weighted roulette wheel)

1. Each individual gets a chunk of a roulette wheel proportional to its fitness relative to the rest of the population.
2. Spin the roulette wheel n times where n is the size of the population.
3. Repeat selection is allowed.

f0 = 3
f1 = 2
f2 = 1
f3 = 6



Stochastic Universal Sampling (SUS)

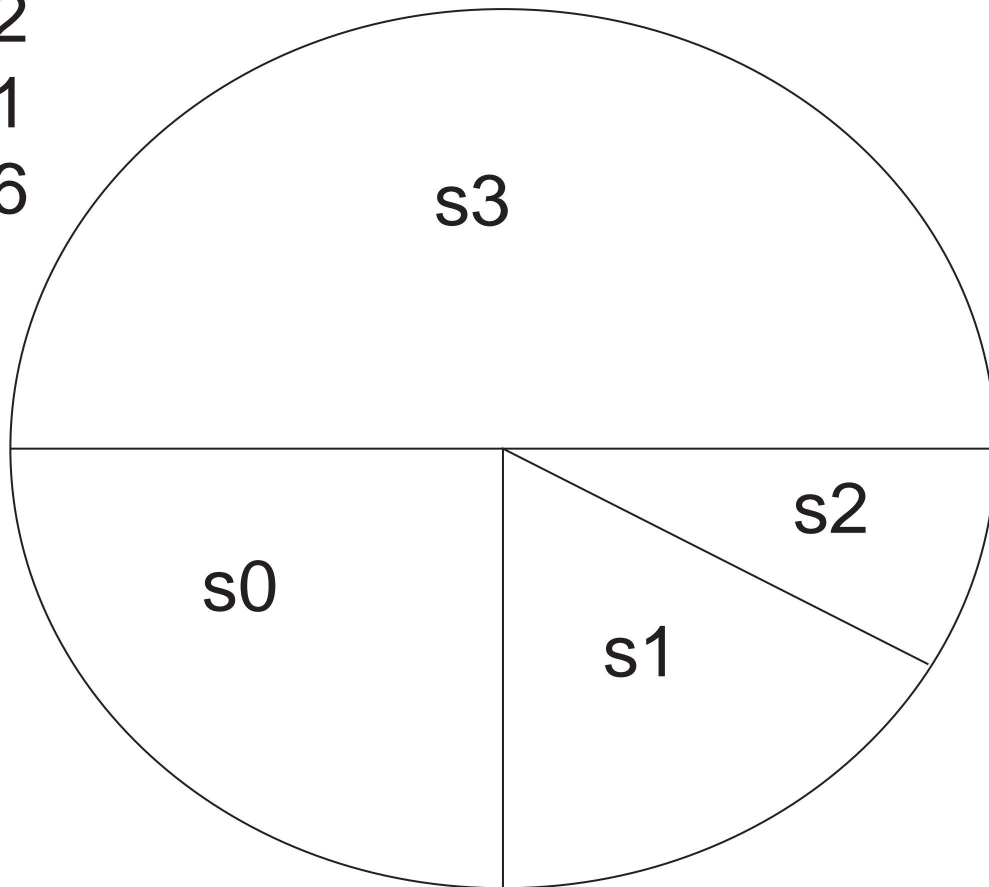
Spin n-pointer roulette wheel once

f0 = 3

f1 = 2

f2 = 1

f3 = 6



The Simple GA: Selection

Other selection methods: ranking, expected number, tournaments, truncation

What happens if we repeat this process iteratively?

- With high probability we get population full of best individual in the initial population.
- i.e., a noisy expensive way of choosing the best from a set (not very useful by itself)

The Simple GA: Recombination

Single point crossover

- Two individuals are mated at random
- Cross site chosen at random.
- Cut and splice pieces of parents to form two new individuals:

1	0	0	0	1	0	0	1

0	1	1	1	0	0	0	1

1	0	0	0	0	0	0	1

0	1	1	1	1	0	0	1

The Simple GA: Recombination

k-point crossover

- Instead of above, k cross sites are chosen at random
- More disruptive.

Uniform crossover

- Don't choose any cross sites.
- Line up parents
- Iterate through strings and with probability p_c swap bit values between strings.

The Simple GA: Mutation

With some small probability, randomly alter a bit position (i.e., flip a 0 to a 1 or vice versa).

1. Iterate through the bit string.
2. And for each bit, with probability p_m flip the bit.

0	1	1	**1**	1	0	0	1

0	1	1	**0**	1	0	0	1

What do we get if we repeat mutation iteratively on an initial population?

- Get a random population (i.e., perform a random walk).
- Not very useful by itself.

What do we have so far?

So far we have seen three operators that separately are rather useless.

- Selection (by itself): inefficient way of choosing best from an unordered set.

- Crossover (by itself): a random shuffle

- Mutation (by itself): a random walk

Can we make something useful out of a combination of these?

Putting it all together

Selection + Crossover = Innovation

- Selection gives us a population of the strongest individuals.
- Crossover attempts to combine parts of good individuals to make even better new ones.

Selection + Mutation =
Stochastic Hill Climbing

- Selection gives us a population of the strongest individuals.
- Mutation makes slight alterations to these.
- Repeating this process will weed out the bad mutations and keep the good.
- We essentially have the equivalent of stochastic hill climbing.

Putting it all together

Selection + Crossover + Mutation = The Power of the GA

- So "selection + mutation = stochastic hill climbing"

- Add crossover to that, and we have stochastic hill climbing with a means of jumping to potentially "interesting" parts of the search space.

- Mutation is also often seen as an insurance policy against the irreversible loss of key bits.

The Simple GA

1. Let P := a random population of n bitstrings
2. Until "Convergence" or "bored" do
3. Let f_i = Fitness(P_i) for $i = 1 \ldots n$
4. Let P' = SelectNewPopulation(P, f)
5. Pair up the individuals in P' at random and for each pair with probability C perform crossover replacing the parents with the children otherwise keep the parents unaltered.
6. For each P'_i in P' perform mutation i.e. iterate through bitstring flipping each bit with probability M
7. Let $P = P'$

So how long do we iterate this?

When to Stop

Some possibilities:

- If we've found an individual with the maximum value of the fitness function; or

- After iterating entire process some predefined maximum number of "generations"; or

- When population converges upon a single individual (i.e., they are all the same).

- When $f_i = f_j$ for all P_i, P_j in P.

- After some number of generations without improving upon the best so far.

- ...

Usually you use two or three stopping criteria.

Example: SAT

$$A \vee \neg B \vee C \ \wedge$$
$$\neg A \vee C \vee D \ \wedge$$
$$B \vee D \vee \neg E \ \wedge$$
$$\neg C \vee \neg D \vee \neg E \ \wedge$$
$$\neg A \vee \neg C \vee E \ \wedge$$
$$E \vee F \vee \neg G \ \wedge$$
$$B \vee C \vee D$$

1 1 1 1 1 1 1 ($f_0 = 6$)
1 0 0 0 1 0 1 ($f_1 = 4$)
0 1 1 1 0 0 1 ($f_2 = 6$)
0 0 0 0 0 0 0 ($f_3 = 6$)
1 0 0 1 0 1 1 ($f_4 = 6$)
0 1 1 0 0 1 1 ($f_5 = 6$)

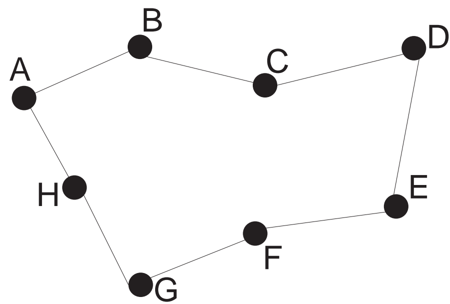
Let's simulate one iteration of the GA given this initial population.

What about the TSP?

So, how should we encode the TSP? Cities on a stack some number of bits to choose which by position.

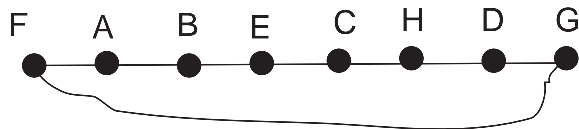
City stack

A
B
C
D
E
F
G
H



encoding

101..111..110..111..000..010..110..001

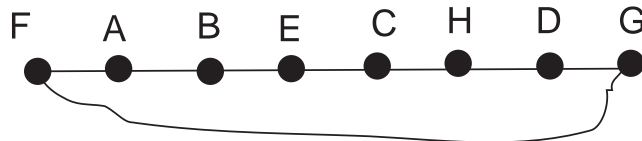


Why you might not want to do this

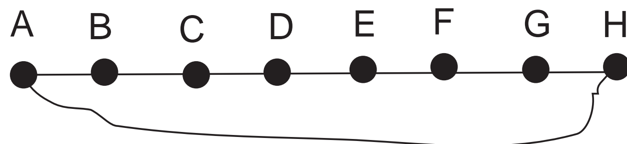
Crossover

parents

101..111..110..111..000..010..110..001

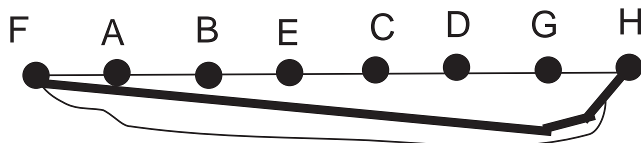


000..000..000..000..000..000..000..000

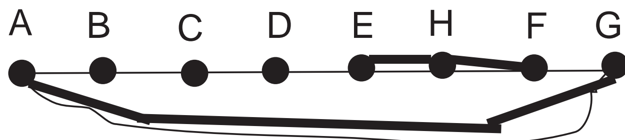


children

101..111..110..111..000..000..000..000



000..000..000..000..000..010..110..001



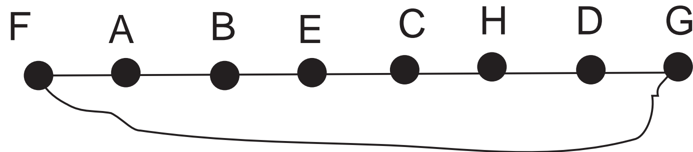
--children tend to differ from parents
--most noticeable in 2nd child above
--can be much worse!!

Why you might not want to do this

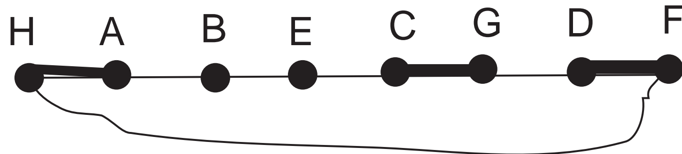
Mutation

encoding

101..111..110..111..000..010..110..001



111..111..110..111..000..010..110..001



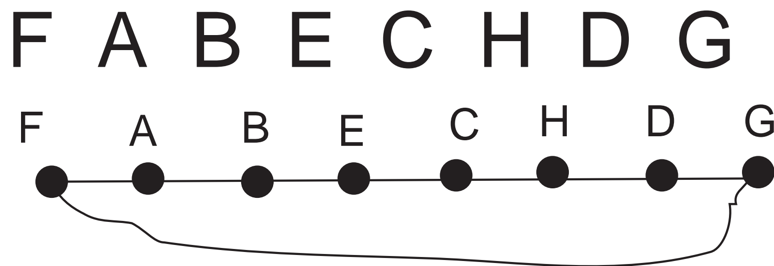
A single bit mutation has replaced 4 edges from parent tour!!!

Can we do any better? Any suggestions?

Other Representations: Permutations

no bitstrings

F A B E C H D G



but what happens with xover?

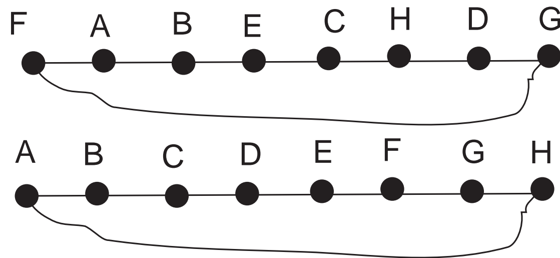
F A B E C H D G

A B C D E F G H

F A B **E** **E** **F** G H

A B **C** **D** **C** H **D** G

Partially Matched Crossover (PMX)

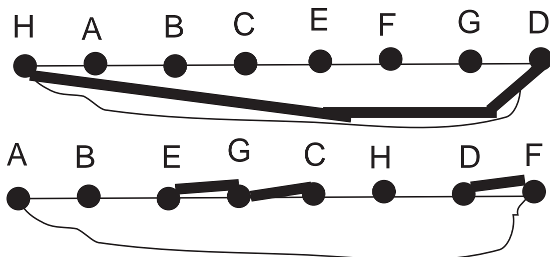


choose 2 cross sites

F A B E | C H D | G
A B C D | E F G | H

--work left-to-right in cross region

H A B C | E F G | D
A B E G | C H D | F



hmmm...not any better?

Respects absolute position

Order Crossover (OX)

choose 2 cross sites

F A B E | C H D | G
A B C D | E F G | H

↓

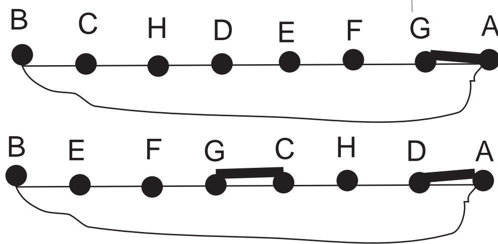
-- A B -- | C H D | --
A B -- -- | E F G | --

↓

B C H D | -- -- -- | A
B E F G | -- -- -- | A

↓

B C H D | E F G | A
B E F G | C H D | A



Respects relative position

Control Parameters

So far, we've defined many parameters:

- Crossover rate
- Mutation rate
- Max number of generations
- Population size

And there can be many others depending on choice of operators (for example, uniform crossover has an additional parameter).

Even what crossover operator we use can be seen as a parameter.

Some of the possible stopping criteria have their own parameters.

How do we set these? Any ideas?

Parameter Tuning

GA parameter tuning is a wide open research area. There is no general set of rules or guidelines to follow in parameter tuning.

Unfortunately, the most often used method is that of hand-tuning, or trial-and-error (i.e., make up a set, try it and see what happens, repeat).

This doesn't sound pleasant. Can we do any better?

Parameter Tuning

- De Jong (1975) systematically studied the effects of the control parameters on a class of GAs.
- These parameters are often blindly used.

- Population Size
- Typically people set this to between 50 and 100 for the simple GA.
- Some feel that the appropriate population size is related to the encoding length in some way.

- Mutation Rate
- Usually set to some "low" value
- Some design it in such a way to lead to an expected number of mutated bits per individual (if length is 100 and 1 bit mutation desired then $p_M = 0.01$).
- Why not decay mutation rate? high rate early for exploration and decay rate as we begin converging toward a solution.

Parameter Tuning

- Metalevel Optimization
- Employing a second GA to optimize the parameters.
- Fitness evaluation expensive! (have to execute the primary GA some number of times! Argh!)
- Also, what determines fitness?
  Quality of result?
  Convergence time?
  Combination of both and how?

- Adapting Control Parameters Over Time
- Adding parameters to encoding and evolving with solution to problem.
- Using problem related feedback in some way and adapting parameters based on current population in some way.

A Representation: Vector of Reals

Are bit strings and permutations the only thing we can evolve with GAs? No.

Let's take Metalevel Optimization as an example. How would we encode the control parameters of the primary GA? One possibility: bit strings.

Another possibility: a vector of real valued parameters

$(R_1, R_2, \ldots, R_n)$

Crossover: same as with bit strings only with reals rather than bits.

Mutation:
Add g to some real parameter where g is drawn from a Gaussian distribution with $(\mu = 0, \sigma)$.
Decay σ over time.

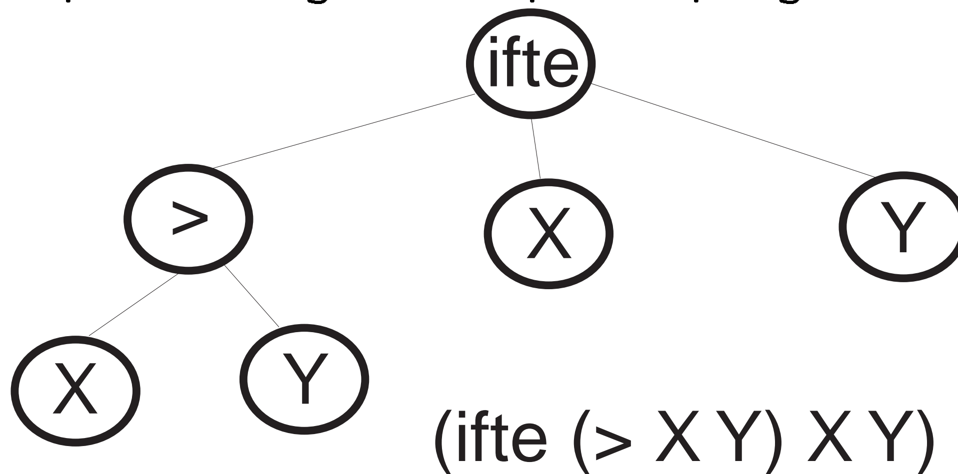
Genetic Programming (GP)

Formulated by Koza (1990?) as a means of automatic programming.

Terminals: X, Y, 1, 2, zero-arg-functions, etc.

Functions: $+$, $-$, $*$, $/$, $>$, if, ifte, etc.

Programs are encoded by parse trees typically representing a computer program in Lisp.



GP: Random Creation

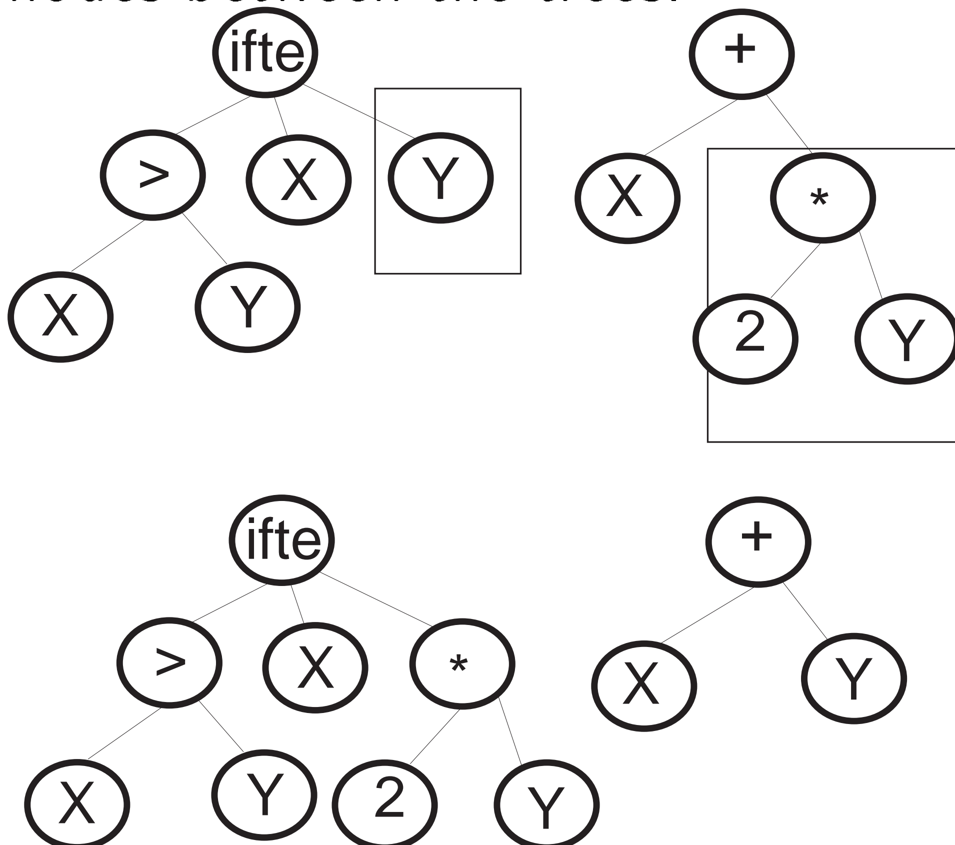
After defining your terminal set and function set you randomly generate an initial population.

Each random parse tree is generated as follows:

1. Let T equal an empty parse tree.
2. Let C = a random function or terminal.
3. Add C to T.
4. If C at predefined maximum initial depth then choose random terminals for each of the children of C and add these to T
5. Otherwise recurse on this procedure for each of the children of C.

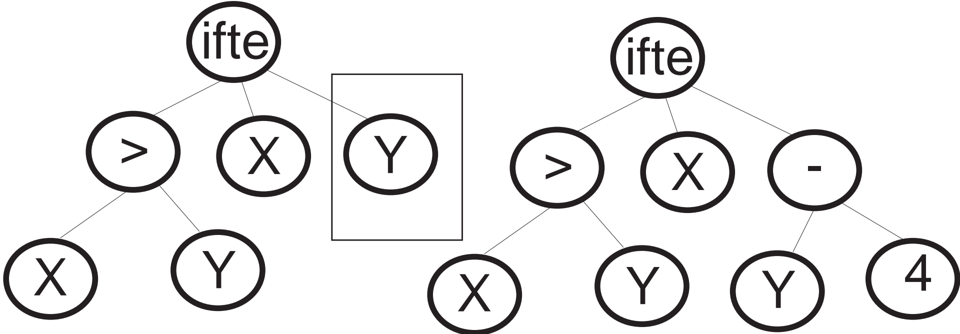
GP: Crossover

- Pick two parents at random based on fitness.
- Independently, pick a random node in each of the parse trees.
- Swap the two subtrees identified by these nodes between the trees.



GP: Mutation

- Pick one parent at random based on fitness.
- Pick a random node in this tree.
- Remove the subtree rooted here.
- Grow a new subtree in the same manner as the initial population was grown.



GP

1. Let P = random initial population.
2. Until Stopping Criterion
3. Choose $R\%$ of next population P' from P unaltered.
4. Choose $M\%$ of next population P' from P and perform Mutation.
5. Choose $C\%$ of next population P' from P, pair up at random and perform crossover.
6. Note: $(R + M + C = 100\%)$
   Repeat selection allowed.
   All selections based on fitness.

Population sizes tend to be quite large for GP (i.e., 500-1000 individuals would not be unreasonable).

GA Issues

- Choice of representation is critical (bit strings not always the best choice).
- Choice of genetic operators often critical.
- Design of fitness function is often critical.
- A "bad" choice of encoding / fitness function combo may result in poor performance.
- Control parameter tuning is critical and no good guidelines for doing so.

GA Discussion

- Often the "second best way" to solve a problem.

- But relatively easy to implement.

- The simple GA is blind and doesn't care about problem specifics other than the fitness function.

- Can sometimes improve performance with problem specific heuristic operators.