Evolutionary Algorithms/Computation
An introduction to evolutionary algorithms EA’s
-- Donald H. Cooley

Stochastic Processes
• A process involving some degree of randomness, regardless of the characterization of that randomness, is a stochastic process.
  – A random number generator is an example of a stochastic process
    • Whether the distribution of those random numbers is uniform, normal, etc. makes no difference as to the characterization of the process as stochastic.

Stochastic Process
• If a process is stochastic
  – Successive runs may not give the same result(s)
  – Cannot duplicate a run, unless duplicate the process of random number generation
  – Any random number generator used, must be truly characterized as random
  – Never report the results of a single run. Always report results as an average of several runs

AI & Evolutionary Computation
• Is there a single definition for AI? (Artificial Intelligence)
  – No
  – All of the fields of AI represent attempts to simulate human intelligence
• Evolutionary Computation
  – Machine learning optimization and classification paradigms based on evolutionary mechanisms

AI & Evolutionary Computation
• EC is about self-organization
  – Simple processes that lead to complex results, e.g. a genetic algorithm (GA) is simple
  – The whole is > the sum of its parts
  – “There is no conservation of simplicity” Wolfram

AI & Evolutionary Computation
• EC field consists of
  – Genetic Algorithms
  – Genetic Programming
  – Evolutionary Strategies (small perturbations of “best” solution to search)
  – Particle Swarm Optimization
Applications

- Generally (~always) EA’s are applied as optimization techniques
- In an optimization problem, we are looking for the solution that gives the “best” result
  - Solution is often a set of parameter values which characterize a function
  - Best requires some method to measure relative quality of solutions

Optimization

- Optimization can be thought of as occurring in one of three (search) spaces concurrently
  - Parameter space
    - Space of legal parameter values
  - Function space
    - Space of values that function(s) evaluate to for the given set of parameters
  - Fitness space
    - Space of values of fitness or objective function

Optimization

- Optimization can be thought of as searching the space of legal parameter values for that set which gives the best fitness value for the associated function
- Generally, the fitness surface is multimodal
  - Global Optimum
  - Local Optimum

Optimization

- There are two general categories of techniques used to solve optimization problems
  - Deterministic techniques
  - Stochastic techniques

Deterministic Techniques

- Deterministic techniques are called such because for the same initial conditions, e.g. initial values of parameters, they always give the same result

Deterministic Techniques

- Gradient-based
  - Find derivative or pseudo derivative of fitness function, error, etc. at current solution
  - Update (increment/decrement) parameter(s) according to that derivative
    - For example, go in direction of minus of derivative to decrease (minimize) some value, where minimum is goal
    - When derivative = 0 we have reached our “goal”
Deterministic Techniques

- Gradient based
  - Result is dependent on start condition
  - May not be stable if update value is too large
  - Usually terminates when fitness gradient is 0
  - Any local optimum can give a gradient of 0, and hence these techniques can get “stuck” in local minima/maxima

Hill Climbing (Deterministic)

- Assume we have a black box & want to optimize:

```
INPUT          OUTPUT
```

Hill Climbing – (Deterministic)

- Multi-modal surfaces (more that one maximum, but only one global maximum)
  - use iterated hill climbing
  - start at a random point and climb until can’t climb any higher
  - try another point and repeat
  - works best if the maximum is a large maximum
  - use largest of maximums found

Deterministic Techniques

- Heuristic
  - Use some “algorithm” to guide search
  - Example – Traveling salesperson problem: minimize total distance in a circuit
    - From the current city, as the next city choose the closest one not already visited.
    - Assumes all cities are connected to all other cities. If not must add a backup capability

Deterministic Techniques

- Heuristic
  - Like gradient, very dependent on initial condition, like which city we start with in circuit
  - Nothing guaranteed about the optimality of the solution, i.e. may or not even be a local optimum
Stochastic Techniques

- Stochastic techniques imply there is an element of randomness in the solution or search.
  - For example, in the traveling salesperson problem, let’s say the next city is chosen with some probability based on the distances to adjacent cities not yet visited.
    - e.g. Use roulette wheel selection based on the inverse of the distance

- Because of the randomness of stochastic processes, solutions are not necessarily repeatable
- There is more than one type of stochastic technique – we will consider
  - Simulated annealing
  - Evolutionary

Simulated Annealing

Simulated Annealing

- Does not use gradient information to guide its search
- Is thus applicable to a wider range of problems, i.e. those for which the gradient is expensive to compute or cannot be computed.
• Let’s say we want to **minimize** \( y(x) \).

1. **initialize** \( x \) (random)
2. compute \( x' = x + \Delta x \) where \( \Delta x \) is a small uniform random perturbation
3. compute \( \Delta y = y(x') - y(x) \)
4. compute \( P = \exp(-\Delta y / T) \) \( T \) is the temperature
5. if \( \Delta y < 0, x = x' \) else \( x = x' \) with probability \( P \)
6. repeat steps 2 through 5 until equilibrium, i.e. number of accepted transitions becomes small
7. update \( T \) according to \( T(t) = \frac{T_0}{1 + \ln t} \) stop when

\[ T = 0 \] \( (t = \text{iteration number}) \)

---

**Simulated Annealing**

• On the plus side, it can be shown (statistically) that by using simulated annealing properly, one is assured to reach the global optimum.
• On the minus side, simulated annealing is very time intensive, i.e. slow.

---

How are EA’s different from other methods of optimization?

• EA’s work with a coding of the parameter set not the parameters themselves
• Because they don’t need a priori knowledge (e.g. the gradient), EA’s tend to be more **robust** than other techniques

---

How are EA’s different from traditional methods of optimization?

• EA’s search from a population of points, not a single point
• EA’s use fitness not derivatives or other such measures. Hence GA’s are called “blind” or “weak methods”

---

What are EA Applications?

• There is no real limitation to the application of EA’s.
• Simply stated, EA’s are most useful in cases where an optimization is being performed.
• Nearly every problem can be stated as an optimization problem.
Applications

- Most EA applications are nonlinear and the fitness function may not be everywhere differentiable
- Most often the parameters to be optimized are not independent, i.e. there are interactions so $K>0$
- Often the parameters have constraints, hence some solutions (sets of parameter values) may be illegal

Categorizing EA’s

- While it is not universal, we will categorize EA’s on the basis of two criteria or goals
  - Find optimal set of parameter values – Genetic Algorithm (GA)
    - Single objective
    - Multiple objective
  - Find algorithm for optimal solution – Genetic programming (GP)
    - Single objective
    - Multiple objective

Applications

- In what follows, we will consider four problems
  - Function Optimization
  - 8 Queens problem
  - The traveling salesperson problem
  - The Mouse in a maze problem

Function Optimization

- For some function $F(X,Y,Z)$, find the set of parameters (values for $X$, $Y$, and $Z$, that maximizes (or minimizes) $F$

8 Queens

- On an 8x8 chess board, find locations such that 8 queens can be placed on the board so that none is threatening any other

Traveling Salesperson

- Visit a set of cities once and only once for minimum cost
Mouse in a Maze

• From a given start location, give that set of directional steps which will get the mouse to the goal (the cheese)
  – Steps are 1 step N, S, E, or W

Encoding

• Most difficult, and important part of any GA
• Encode so that illegal solutions are not possible
• Encode to simplify the “evolutionary” processes, e.g. reduce the size of the search space
• Most GA’s use a binary encoding of a solution, but other schemes are possible

GA Fitness

• At the core of any optimization approach is the function that measures the quality of a solution or optimization.
• Called:
  – Objective function
  – Fitness function
  – Error function
  – measure
  – payoff function
  – etc.

GA Example

The Objective Function \( O() \)

• Let’s say that the problem is to find a solution to the function:
  \[ f(x) = x^2 + 2x + 1 \]
• We know the “solution” to this function is \( x = -1 \).
• How “good” is a solution like \( x = 3 \)?
• To answer that question we need an objective function \( O() \).

• In this case, we know that a solution to \( f(x) \) is a value for \( x \) for which the function \( f(x) = 0 \). Hence, a possible objective function can simply be \( O(x) = f(x) \), where we want to minimize \( f(x) \).
• \( O(x=3) = 16 \)
• The most important feature of an objective function is that it be able to measure the relative quality of two solutions, i.e. tell us which is better and by “how much”

Elements of a Genetic Algorithm

• Every genetic algorithm consists of:
  – encoding scheme
  – initial population (size and scheme to generate)
  – new generation scheme
  • mate selection scheme (reproduction)
  • mating crossover scheme
  • mutation scheme (probability of mutation)
  • replacement scheme
  – stopping criteria
  – objective or fitness function
Problem Dependent Components

• What elements or components of a GA are problem dependent?
  – Always
    • Encoding scheme
    • Fitness function
  – Sometimes
    • Everything else

Elements of a Simple Genetic Algorithm

Encoding Scheme

• The simplest and most common scheme is to encode the solution as a binary vector - called a chromosome or genotype.
• Often, along with the encoding scheme is a decoding scheme, i.e. convert the chromosome into a value(s) to be used by the objective function.

Encoding Scheme for Examples

• Function Optimization
• 8 Queens
• Traveling Salesperson
• Mouse in a maze

Elements of a Simple Genetic Algorithm

Initial population (size and scheme to generate)

• Population size = number of chromosomes
• In simple terms, the size of a population is generally chosen proportional to the size of the search space, or the number of possible solutions.
  – As the number of possible solutions increases, the population size may need to increase.

Elements of a Simple Genetic Algorithm

Encoding Scheme

• While the chromosome is a string of bits, sets or groupings of the bits represent values for various parameters (genes)
• As an example, say you want to find the minimum of some function \( f(x,y) \).
  – We could use a 20-bit chromosome with the first 10 bits for \( x \) and the other 10 for \( y \)
  – Or a 30-bit chromosome with the first 25 for \( x \) and the last 5 for \( y \) … etc.
Elements of a Simple Genetic Algorithm
Initial population (size and scheme to generate)

- The initial population is generally generated randomly
  - e.g. if pop. size = 100, and the length of a chromosome is 32 bits, *randomly* generate 100 32-bit values
  - Sometimes the population is “seeded” with a few “good” solutions

The Search Space

- If the parameters interact in a known fashion, say linear, then you will need fewer parameters and thus fewer bits.
  - e.g. three parameters (x,y,z) for which there are 1024 possible values each, and we don’t know how they interact: we need a 30-bit chromosome
  - e.g. three parameters (x,y,z) for which there are 1024 possible values each, and we know that y = x^2: we need only 20 bits

The Search Space

- Sometimes the interactions are more subtle
- Consider the problem of placing 8 queens on a chess board.
  - Must position each queen so that it does not threaten any other queen
  - A chess board is 8 rows X 8 columns
  - For the position of each queen we could use 6 bits, i.e. 3 bits for the row and 3 bits for the column. Total bits = 8*6 = 48 => 2^48 ~ 256 x 10^12

The Search Space

- For the preceding example, is the given encoding scheme the best?
- What makes one encoding scheme better than another?
  - Are illegal solutions possible?
  - Is the search space as small as possible?

The Search Space

- The goal of an encoding scheme is to
  - Eliminate, if possible, any illegal solutions – why?
  - Make the search space size (in most cases this correlates to the length of the chromosome, i.e. shorter is better) as small as possible
- We ask the question again for the 8 Queens problem: “Is the encoding scheme given the best?”

The Search Space

- If number of good or optimal solutions is *small* in a *large* search space, random or enumerative searches are a poor choice.
  - Random search: continually guess at a solution until get a “good” one
  - Enumerative (exhaustive) search: try all possible solutions
Elements of a Simple Genetic Algorithm
Pseudo-Code

step 1: generate the initial population
step 2: for each chromosome in the population
        \( C_i \) compute its fitness \( O(C_i) \)
step 3: while the stopping condition is not met
        repeat steps 4 through 5
step 4: reproduce and generate a new population
step 5: compute \( O(C_i) \) for all new chromosomes

Example - TSP

- Consider the traveling salesperson problem TSP (illegal routes included)
- Search == find a solution that’s \textit{acceptable}
- Optimize == find the \textit{best} solution

Notice that something new has been added, i.e., we have a case where a guess may not be an \textit{acceptable} solution.

Example – TSP – illegal solutions

- For thirty (30) cities, we know the solution is a transit with 30 elements. However, if there is a repeat of any of the 30 cities, we know it is an illegal solution.

Initial Population – 32 cities

- Let’s say we have decided on a population of 50 chromosomes (transits), each 160 bits in length (why?)
  - For the initial population
    - 50 times: generate a random string of 1’s & 0’s of length 160
  - What’s the problem??

Mate Selection - Reproduction

- Given a population of \( M \) chromosomes, the generation scheme stochastically selects a pair of chromosomes to use to generate an offspring
- The selection process is called \textit{mate selection}.
- A common mate selection scheme is roulette wheel selection.
  - The probability of selection of a chromosome for mating is a function of its fitness \( O() \). The higher the fitness, the higher the probability of selection.

Roulette Wheel Mate Selection Example

- Assume there are 4 chromosomes in a population \( C_1, .., C_4 \), and \( O(Ci) \) is as follows (where larger \( O \) is better):
  \( O(C1) = 10, O(C2) = 20, O(C3) = 5, O(C4) = 30 \)
- To perform roulette wheel selection we can envision dividing a roulette wheel into 4 (population size = 4) sections. Each section will be of an area equal to its associated chromosome’s relative fitness.
Roulette Wheel Mate Selection Example

For C1: relative fitness = 10/65 = 0.154
where 10 = fitness of C1
65 = \sum_{i=1}^{\text{population size}} \text{fitness of } C_i
For C2: relative fitness = 20/65 = 0.308
For C3: relative fitness = 5/65 = 0.077
For C4: relative fitness = 30/65 = 0.461

Elements of a Simple Genetic Algorithm

Roulette Wheel Mate Selection Example

- Continuing with our population of 4 chromosomes, we want to generate 2 pairs of chromosomes to mate, i.e. the mating will produce 4 new chromosomes (2 offspring for each mating) which will replace the current population.
- Sometimes, each mating produces only one offspring and thus there needs to be 4 matings.

According to the fitness proportions just computed, divide the roulette wheel as follows:

- 0 < X <= 0.154 for C1
- 0.154 < X <= 0.462 for C2 (0.154 + 0.308)
- 0.462 < X <= 0.539 for C3 (0.462 + 0.077)
- 0.539 < X <= 1.0 for C4 (0.539 + 0.461)

The roulette wheel reproduction scheme is then as follows:

for I = 1 to population size/2
  for mate 1
    generate a random number (0 < X < 1)
    if 0<=X<=0.154, then C1 is the chosen chromosome, etc.
  for mate 2
    generate a random number (0 < X < 1)
    if 0<=X<=0.154, then C1 is the chosen chromosome, etc.
    if mate 1 = mate 2 chose another mate 2, i.e. repeat this step
  perform reproduction with mate 1 and mate 2
end for
replace original population with new population

Stochastic Error

- When we generate something randomly, there is always (and it usually happens) a chance for stochastic errors.
- As an example, consider tossing a “perfect” coin 100 times. According to the probabilities, we should have 50 heads and 50 tails. A stochastic error has occurred if we have say 49 heads and 51 tails.
  - Would it be possible to get 99 tails and 1 head?
Other Mating Schemes

• Most of the other mate selection schemes try to
  – Reduce stochastic errors, or
  – Retain best solution(s) to date (elitist), or
  – Maintain a diverse population of solutions, or
  – Speed the mate selection process

Elitist Schemes

• Take the best of population to date and copy it into the next generation
  – Means that over time the population size will grow
• Only replace a parent with a child if the child is more fit
  – Hill climbing because average fitness of population always increases or stays the same

Elitist Schemes

• Using all of the parents and all of the children
  – Gives a population of size 2N
  – Sort in descending order of fitness
  – Keep only the top N chromosomes from this population for the next generation
• This schemes does more than just keep the best of population to date

Hill Climbing Techniques

• The preceding are examples of hill climbing techniques
• Have some similarities with a gradient search technique in that are always (sort of) climbing higher up the fitness plane for every solution
• It’s a “sort of” climb because a child that replaces a parent may be on an entirely different “hill”

Scaling Problem

• Some reproduction methods have a scaling problem
  – If one chromosome in population is much better than others, it will “overpower” all of the others in generating the pool of parents, i.e. it will always be a parent.

Recombinative Hill Climbing (RHC)

• A hill climbing technique in that population average never decreases (usually increases)
• With RHC cannot lose best to date in family.
• Recombinative
  – Mate selection is purely random, i.e. not based on fitness
• Hill climbing
  – Offspring only replaces a parent if its fitness is better
RHC Algorithm

1. Given a population of N chromosomes
2. generate a random number R, where 1 \leq R \leq N-1.
3. Using R as the offset, mate chromosome i with chromosome (i+R) \text{ MOD N}.
4. Generates one offspring rather than two
5. If fitness of offspring > fitness of chromosome i (i.e. it’s first parent), replace chromosome i with offspring.
6. Repeat steps 3 through 5 until all N chromosomes have been mated.

• Note from the preceding that with RHC there is no fitness criteria for mate selection.
• This is a technique that helps to maintain diversity in a population

Tournament Selection

• A popular reproduction scheme is called tournament selection – It works as follows
• Randomly select two solutions and best goes into pool.
• Selection is systematic so each solution is selected exactly twice
  – Best solution in population will win twice, etc.
  – Any solution will have 0, 1, or two copies in the pool

• Once the pool is created
  – Randomly select a pair to mate and continue until all pairs have been selected
• Deb and Goldberg claim this technique is at least as good as any other method in terms of computational complexity and speed of convergence
• Also note that this method has no scaling problem

Crossover

• Reproduction doesn’t create any new solutions
• Reproduction only creates duplicates
• Creation of new solutions (search of solution space) is done by crossover and mutation

• Single Point Crossover (SPC) with equal length chromosomes

For a binary chromosome of length n bits, generate a random number x such that
\[ 1 < x = n \]
For SPC, swap bits x through n of the two chromosomes (bits are numbered 1 through n)
Elements of a Simple Genetic Algorithm

Single Point Crossover Example

- Consider two binary chromosomes:
  - A = 101101100, B = 01001000010
  - Thus, n = 11
- Assume that the generated crossover value (X from the previous slide) is 6. Then the new chromosomes, after crossover are:
  - A' = 10110100010
  - B' = 01001001100

Uniform Crossover

- Does not have linkage between bits.
- Works as follows:
  - At each bit position in a child X, randomly pick a bit from parent A or B. The bit not chosen then goes into the other offspring, i.e. child Y

2 or n-point crossover

- Rather than a single point for crossover, select two points in the chromosome, and swap that section.
- Where n is odd may assume that location L (last of chromosome) is included.
- Ex- 15 bit chromosome, with 3 point crossover, and crossover points are 2, 7,10 (= 2,7,10,15), i.e. bits 2-7 and 10-15 are swapped.

Transposition

- The bases for most searching in GA’s are crossover and mutation.
- In nature, there are other mechanisms for genetic recombination
- Transposition in nature was discovered in 1950’s
- Involves inter or intra chromosome movement of genes (groups of bits).

Transposition

- In 1950’s Barbara McClintock first discovered transposition
  - Found that in corn, certain genetic elements move to produce new colors that couldn’t be result of mutation or crossover
  - First viewed such movement as an abnormality – have since discovered several
Transposition

• McClintock received Nobel Prize in 1983 for her work
• Have since found that transposons are responsible for growth of cancers and resistance to antibiotics by bacteria

• Transposition
  – Process in which genes or genetic units move from one location in a chromosome to another
  – Movement can be in same or different chromosome
  – Mobile units are called transposons or jumping genes
  – Process can involve movement or duplication
  – Transposon is identified by what is termed a flanking sequence

• In nature, transposons can consist of one or several genes, or even a part of a gene
• Can move in several ways – the “how” in nature is not fully understood
  – (Inter) [not] Leave a copy and duplicate n times at other locations in self[other chromosome]

• Insertion point for a transposon can be
  – Chosen at random
  – Show a regional preference
  – At a corresponding flanking sequence in the chromosome

• A transposon is identified by a flanking sequence
  – Flanking sequence may be two duplicate sequences or a sequence and its reverse.
  – When a transposon moves to another zone of a chromosome, one of the flanking sequences goes with it.

• Many ways to implement transposition. The following is an example of such a method.
  – Chromosomes are of fixed length
  – FSL = flanking sequence length
  – CL = chromosome length
  – T = a gene in the chromosome
Transposition

– Randomly choose a gene (a bit T) between 0 and CL
– FSL genes immediately before T are the first flanking sequence
– Second flanking sequence can be same or inverse
– From bit T, look for a possible second flanking sequence

Transposition

– Transposon is formed as all bits from bit T to the last bit of the second flanking sequence
– Second flanking sequence always moves with the transposon

Transposition

• In the second parent, look for a sequence of bits equal or inverse to the flanking sequence (if not found don’t swap)
  – Insertion point is first bit after that sequence
• After finding insertion point, same number of bits, equal to the transposon length will be exchanged between parents

Parent Selection

• Can use any means to select two parents
• In tournament selection, only the fittest of the two parents transfers genetic material, i.e. to the other.
  – In this way, the good parent seeds the bad parent

Transposition - Example

• In this example,
  – X1 = 18 bits
  – X2 = 15 bits
  – Used roulette wheel with elitism (elite size = 20%)
  – Mutation rate = 0.01
  – Crossover/transposition rate = 0.7

Transposition - Example

– Used 1, 2, and uniform crossover
– Transposition used flanking sequences of 1 to 10 bits
– Populations sizes of 50, 100, and 200 were tested 10 times each
– 1000 generations
– Results are the average of the best individual fitness obtained in 10 trials with each experiment
Transposition - Example

• As flanking sequence length increases, the transposon length will get bigger
• In many (most) cases transposition won’t work because second flanking sequence won’t be found.

Transposition vs “Regular” Crossover

• Generally, transposition works better

Mutation

• Crossover does most of the searching, even though mutation does some
• Mutation’s main “job” is to maintain or introduce diversity

Mutation

• Mutation is generally performed after crossover, bit-by-bit, on each of the new chromosomes.

Given:
chromosome length = n and probability of mutation of $p_m$

For $I = 1$ to $n$
    generate a random number $X$
    if $X < p_m$, complement bit $I$ of the chromosome
end for

Mutation

• A problem with the bit-by-bit mutation method is that it requires the generation of a random number for every bit of every chromosome
• Goldberg(1989) suggested a mutation clock operator to reduce the complexity of the mutation operation
• After the first bit is mutated, the location of the next bit to mutate is created as a random function, i.e. skip some number of bits

Mutation

• A question that sometimes arises is what happens to mutation if rather than a binary chromosome we have say a ternary chromosome?
  – Each chromosome element is a 0, 1, or 2
  – We can’t simply complement
  – Can we mutate say a 2 to a 2 now?
  – What’s the difference between randomly choosing any one of the three possible values versus only one of the two different values?
Next Generation
• Once a complete generation has been created the old population must in some way be merged with the new.
• To replace all of old with new is called **generational replacement**
• Remember, this replacement is not done until a new population is created. Don’t replace old chromosomes as new ones are generated. The other replacement schemes utilize some form of elitism as we have already discussed

Putting it Together
• Selection/Reproduction
  – Selects good strings
• Crossover/Transposition
  – Combines good substrings to (hopefully) create better strings
• Mutation
  – Alters a string locally to (hopefully) create a better string, and/or introduce/maintain diversity

Elements of a Simple Genetic Algorithm
Stopping Condition(s) or Strategies
• There are several different stopping conditions. The stopping condition used depends on the application:
  – Continue until at least one chromosome represents an optimal solution. Say at least one chromosome \( X \) such that \( F(x) = \text{optimum} \)
  – Means must know optimum

Elements of a Simple Genetic Algorithm
Stopping Condition(s) or Strategies
– Continue until at least one chromosome represents an acceptable solution.
– This could simply be \( |\text{optimum} - f(x)| < \) 

Elements of a Simple Genetic Algorithm
Stopping Condition(s) or Strategies
– Continue until the average fitness of population is some value
– Continue until all fitness values are equal
  • Homogeneous population
– Continue for \( K \) generations
  • Note: in this case, and possibly others, and because of the replacement technique used for each generation, one normally saves the best chromosome to date throughout the process.

EA’s Today
• Almost any problem that can be viewed as an optimization problem can be “solved” with a GA.
• Not all problems should be solved with a GA.
• GA’s are still as much an art form as a science.
  – There is no algorithm for determining any of the parts of a GA given a particular problem.
  – Considerable experimenting, tweaking, art work, etc. must be performed on a GA for a particular problem.
No Free Lunch Theorem

- Wolpert & Macready (1997) proposed the No Free Lunch Theorem.
  - Any two algorithms for problem solution, e.g. evolutionary and gradient descent, when averaged for “goodness” over all possible problems (that’s quite a lot of problems) are exactly the same.

No Free Lunch Theorem

- The NFL theorem means there is no “silver bullet” algorithm for all problems.
- It is still the case the GA’s are appropriate for a large class of problems. The challenge is to find them.

OK – What Does it mean, i.e. What’s best?

- A GA consists of
  - Encoding
  - Initial population: generation technique & size
  - Mate selection
  - Crossover
  - Mutation
  - Replacement
  - Fitness function
  - Stopping criteria

What’s best? Encoding

- Generally -
  - Binary
  - A scheme that does not allow for illegal solutions
  - A scheme that minimizes the search space size

What’s best? Initial Population – Generation Technique & Size

- Generation technique
  - As random as possible
  - Generates only legal solutions
- Size
  - No rule, only rules of thumb
    - Search space size
    - Time to evaluate fitness

What’s best? Mate Selection

- Tournament Selection or
- Stochastic expected value
  - removes errors by assuring best in population are used to mate
  - Once mating population is generated, select mates randomly
- Or – completely random as in RHC
What’s best? Crossover

• Tough one
  – Single point should be first choice
  – If convergence is a problem, use uniform

What’s best? Mutation

• No “best” method, depends on problem
• Goals
  – Mutate only to acceptable/legal solutions
  – Generate diversity

What’s best? Replacement

• Elitist (keep better ones) forms are best
  – Replace only if better
  – Sort both children and parents and keep only best half
  – Replace most similar parent/chromosome
• If possible, replacement scheme should maintain diversity in population
  – Discard duplicate chromosomes

What’s best? Fitness Function

• Should accurately measure fitness
  – May have to normalize as range changes from generation to generation
  – Biggest problem is giving “reasonable” fitness to illegal solutions
  – Calculation needs to be fast, e.g. a control problem with many states/cases to consider

What’s best? Stopping Criteria

• Depends on problem
  – Searching for best/optimum value
  – Searching for acceptable solution
  – Run only for certain time/# of generations

Real Parameter GA’s

• For binary GA’s string length must be chosen a priori to achieve certain precision
  – The greater the precision, the greater the string length
  – Longer string length requires (generally) larger population
Real Parameter GA’s

• Real variables allow for arbitrary precision
• Remember that for a real, the precision is not uniform
  – How close can we come to 0.0 with a floating point number?
  – How close can we come to 999,999,999,999 with a floating point number?

Challenges

– How to combine (crossover) two chromosomes – can’t simply split a real value
  • Why?
– How to perturb a variable (mutation) – can’t simply complement a bit
  • Why?

Will discuss several schemes for crossover and mutation.
• Although all are different, with proper settings of parameters, most become the same or very similar

Linear Crossover

• Creates three solutions from two parents and two best solutions are chosen as the children

\[ x_{i}^{1,t} \Rightarrow \text{gene } i \text{ of parent chromosome 1 at generation } t \]

Naïve Crossover

• Crossover sites only chosen at variable boundaries, hence no floating point gene values are ever split.
• Has poor search capability
• Relies heavily on mutation to augment search process
Blend Crossover- BLX-β

• For two parent solutions, assuming \( x_i^{(1, t)} < x_i^{(2, t)} \) randomly pick a solution in the range

\[
\left[ x_i^{(1, t)} - \alpha (x_i^{(2, t)} - x_i^{(1, t)}), x_i^{(2, t)} + \alpha (x_i^{(2, t)} - x_i^{(1, t)}) \right]
\]

Blend Crossover- BLX-β

Let \( I \) be a random number in the range 0 to 1, then the following is an offspring

\[
x_i^{(1, t+1)} = (1 - \gamma_i) x_i^{(1, t)} + \gamma_i x_i^{(2, t)}
\]

where \( \gamma_i = (1 + 2\alpha) \mu - \alpha \)

if \( \alpha = 0 \), creates random solution in \( x_i^{(1, t)}, x_i^{(2, t)} \)

Tests indicate \( \alpha = 0.5 \) works best

Blend Crossover- BLX-β

• If the difference between parents is small, the difference between offspring and parents will be small
• Thus for large diversity populations, search is broad, for low diversity offspring, search is narrow

Real Parameter GA’s

• There are many other schemes
  – Simulated Binary (Deb & Agrawal)
  – Fuzzy Recombination (Voigt)
  – Etc.
• Most operators use some sort of prob. Distribution function as the basis for changing a gene’s value for crossover.

Other Techniques

• There are many other GA/EA techniques in use. Most are experimental, and few have shown real promise.
• We will consider one as an example – it has shown considerable promise
  – Particle Swarm Optimization

Particle Swarm Optimization (PSO)

• PSO was proposed by Kennedy & Russell (1995)
• Strictly speaking is not really a GA, because does not have things like crossover
• Does use a population and fitness, but update is not the same
Particle Swarm Optimization (PSO)
• We’ve all seen how a flock of birds flies around.
• Agents/solutions/chromosomes are particles in a flock of such particles
• Assume that we are trying to solve an optimization problem of 2 variables (all of this can very easily be expanded to any number of variables)

• The problem with PSO is the number of parameters involved. The basic idea is simple
• At each iteration, each chromosome (bird in the flock) has its genes updated so that it moves closer to its best solution to date and the best solution of the whole flock to date.
• The rate of this update is its velocity.

Particle Swarm Optimization (PSO)
• Weighting factors determine the influence of its best vs the best of the flock in the update, and the speed/velocity with which the chromosome(s) move
• Currently, PSO’s work only for real values functions (you’ll see from the update function)

• We begin by generating a population of N chromosomes, with each chromosome having D genes.
• Calculate the fitness of each chromosome and determine the fitness of the best of the population
• At each iteration (generation) update each chromosome as follows:

\[
\begin{align*}
    x_{i,k}^{t+1} &= x_{i,k}^t + v_{i,k}^{t+1} \\
    v_{i}^{t+1} &= \omega v_{i}^t + c_1 r_1 (p_i - x_i^t) + c_2 r_2 (p_g - x_i^t)
\end{align*}
\]

• In the preceding
  – \( p_i \) is the value of gene i that gave the best fitness to date for chromosome k
  – \( p_g \) is the value of gene i in some chromosome that gave the best fitness to date. i.e. best of population
  – \( \omega \) is an inertia weight
  – \( c_1 \) and \( c_2 \) scale the effect of the best value of the chromosome vs the best of population
  – \( r_1 \) and \( r_2 \) are random numbers in the range 0 to 1, they also adjust the update speed
Particle Swarm Optimization (PSO)

- Each particle moves toward both “best’s” with some speed.
- Whichever of the c’s is largest determines whether the tendency is to move toward the best of the population (uniformity) or toward their own best (diversity)

V is not really a velocity, it’s really an update amount
- Note that as particles get closer to the global optimum, the differences become 0, thus the whole flock will stop moving.
- Some authors do not use two c’s and two r’s, they only use one of each.

If \( c_1 > c_2 \), the emphasis is on diversity
- \( \beta \) controls impact of previous history.
  - Large value facilitates more global search
  - Small value facilitates local search
  - Start \( \beta \) large and then over time reduce it for a more local search

For problems with many global minima swarm can have problems
- Solution
  - Break population into subgroups and swarm on each independently.

Constraint Handling in GA’s

- Most methods can be categorized into
  - Preserve feasibility of solutions
  - Penalty functions
  - Bias feasible over infeasible solutions
  - Based on decoders

Preserving Feasibility

- Where constraints are based on equality, parameters can be eliminated
- e.g.
  - If the constraint is \( 2x - y^2z = 0 \)
    - Using \( x = 0.5 y^2z \) eliminates the need for \( x \), and now only have two constraints.
    - Two methods developed are Lamarckian’s and Baldwin’s
Penalty Functions

• User assigns a penalty value as a function of a constraint that is violated
  – Penalties must be normalized
  – Difficult to set penalty value
  – How should worst feasible solution compare to an infeasible solution
    • Among two infeasible solutions, the one having the smaller violation value is preferred

Feasible Over Infeasible Solutions

• Run GA initially to generate feasible solution on first constraint. Once enough have been found, use these to generate feasible solutions on the second constraint, holding the first constraint fixed, etc.

Decoder/Encoder

• Simply encode the constraint variable so that it cannot violate its constraint
• This is the simplest and most common strategy

Genetic Programming

History/Goal

• Invented by Cramer 1985
• Expanded on by Koza 1992
• What we have done up to this point is to find the parameters of a function that optimize that function
• GP tries to find the function

History/Goal

• Want to get a computer to solve a problem
  – This means a program
  – How can this be done without explicitly programming it?
Attributes of a GP System (Goals)

• Starts with what needs to be done – starts from a high level description of the problem
• Produces result in the form of a program-sequence of steps that can be run on a computer – hence – real code
• Automatically can determine the size of the program, i.e. number of steps
• Problem independent
• Produces results competitive with humans

What Language

• One of the challenges in developing a GP environment is the issue of encoding, i.e. in what form (language) should we encode.
  – Remember, it is from this encoding that we want to be able to do things like crossover and mutate.
  – As always, we don’t want illegal solutions

What Language

• One of the challenges of GP is that we now have a new realm of illegal solutions.
  – Previously, an illegal solution was one in which the changes to the parameters gave them values that were in some way illegal, e.g. X>100 or maybe we change X and Y so that their sum exceeds some maximum even though neither X or Y is larger than their respective maximums.

What Language

• If we make changes to a program, we have a whole new realm of possible illegal solutions.
  – Change a variable in a statement to one that is not declared
  – Create an illegal statement because of type, number of parameters, etc.
• The resolution of this issue requires a language that embodies a structure that can be manipulated, and very “loose” rules on parameter typing and declaration.
  • LISP does this.

What Language

• LISP – List Processing, embodies these characteristics
  – An expression in LISP can be represented as a linear (1D) string, which actually represents a 2D expression.
  – The expression 2*3+4 is represented in LISP as (+ (* 2 3) 4) (called an S-expression)
LISP

- Consider the expression
- \((+ 1 2 (IF (> \text{TIME} 10) 3 4))\)

Crossover

- While crossover actually occurs on a 1D string, it can most easily be understood in terms of the 2D tree representation. Consider the following two chromosomes
- \(0.2Z + X - 0.5\)
- \(ZY(Y+0.3Z)\)
- The crossover operation is performed at a node, i.e. swap the remainder of the trees

Crossover

- Let’s say that we perform the crossover at the two (randomly selected) nodes labeled with `+` for each of the two expressions
- In the next slide they are labeled in red
Crossover => Mutation

- Even in LISP there are some problems with crossover, but the preceding at least gives the idea of its implementation in a GP environment using LIST
- The next question is how do we do mutation?

Mutation

- Mutation can be most easily understood if we first address the question of how do we generate an initial program.
  - We begin by deciding on a set of functions (operators for now), and a set of terminals, i.e. variables and constants.
  - We then randomly call from this set to generate our program

Program Generation

- As an example, let’s say we have the following set:
  - +, -, /, *, A,B,random_constant
- What are the lengths of the expressions that can be generated?
- How many different programs can be generated from this set?

Mutation

- Now that we have a way to randomly generate a program(s), what happens on mutation?
  - Select a node at which to perform a mutation
  - From that node down, remove all code
  - Randomly generate another program segment from that node down

GEP

Genetic Expression Programming
GEP

• One of the problems with the aforementioned GP technique is that the manipulation of the chromosome, while fairly easy to visualize, requires more coding because of the linear LISP string involved. Also, the fact that it is LISP means that it will generally run rather slowly.
  – There are other problems that we may discuss later

GEP

• Gene Expression Programming is a relatively new technique developed by Candida Ferreira - ~2000
• In this case, the chromosomes are normally of fixed length
• Like GP, each chromosome represents a program (for now it will just be an expression)

GEP Material

• In addition to these notes, there are several papers on GEP on the WEB. I have placed two such papers (.pdf) on my web site
• There are also located at:

GEP Length

• The length of the head is set by the user. H
• The length of the tail is determined from the length of the head – \( H \)
• The length of the tail is a function of the length of the head H, and the max arity of any of the operators \( n \)

GEP Length

• Arity of an operator is the number of operands
  – Arity of square root is 1
  – Arity of * is 2
  – Arity – is 1 (unary minus) or 2
• The total length of a GEP chromosome is
  \[ T = \text{chromosome length} = H(n-1)+1 \]
GEP Length

• Thus for a head of length 10, and unary and binary operators, the tail would be of length
  \[-T=10*(2-1)+1 = 11\]

Consider the following expression

• Consider the following expression
  \[\frac{a\cdot b}{c}\cdot\sqrt{d-e}\]
  • The tree for this expression would be:

```
g +
c |
 a  b
```

Question

• What tree would the following K-expression (that’s what Ferreira calls them) code for?

```
01234567890123456789012345678901234567890
+/-Q*c-abde
```

• The integers above the string are simply to index the elements

More GEP’s

• Consider a gene for which the set of functions \(F = \{Q, *, /, -, +\}\) and the set of terminals \(T = \{a, b\}\).
  • In this case, \(n = 2\); and if we chose an \(h = 15\), then \(t = 16\).
  • The length of the gene \(g\) is \(15+16=31\).
  • One such gene is
    \[0123456789012345678901234567890\]
    \[/aQ/b*ab/Qa*b*-ababaababbabbbbba\]
GEP’s

• Now draw the tree for this expression

What happens if?

• What happens if at position 2, the Q (square root) is changed to a “+”? 

0123456789012345678901234567890 
/af/b*ab/Qa*b*-ababaababbabbbba

What is the new tree?

GEP Implementation

• In implementing GEP, one must decide on: (for now, we will later expand this)
  – The operands to be used
  – The operators to be used
  – The maximum length of an expression

Multigenic K-expressions

• Later, we will talk about K-expressions that represent multiple K-expressions
  – Author calls these multi-genic, i.e. they contain multiple genes or K-expressions
• For now
  – We will only consider single gene K-expressions
  – Every K-expression will evaluate to a single value
  • Thus, if there are operands (variables) in a K-expression, they must be assigned values prior to evaluating the K-expression
GEP Algorithm Steps

• After deciding on GE length, operands, and operators
  – Generate a GE from the list of operands and operators. (I am using GE – genetic expression, and K-expression synonymously)
  • Be sure that the tail of the GE has only operands
  – Now everything is like a regular GA

GEP as a GA – Given a population of GE’s

• Calculate fitness of every GE
• Select mates according to some criteria
• Perform crossover on pairs
• Perform mutation on pairs
• Form new generation
• Repeat as needed until stopping condition met

GE vs Binary Chromosome

• It would be problematic to implement a K-expression as a binary string with the same generality for that string as we have used
  – Because there are restrictions on the individual element values, e.g. how would you distinguish between an operand, say an integer, and an operator
  – To do this you’d need another string with each element to define its type as operand or operator

GE Elements and Structure

• This assumes that each of the strings are of the same length.
  – An interesting problem would be to consider GE’s of different lengths
• You must maintain some means to distinguish between the head and the tail of a GE
  – Because mutation in the head is different from mutation in the tail, i.e. in the tail you only mutate operands, in the head it is operands and operators

GE Elements and Structure

• Since the GE’s are of fixed length, in our code we could simply have a global structure variable containing the:
  – Total GE length
  – Length of the GE head
• Crossover is straightforward.
  – Generate a crossover point and cross the two selected GE’s

GE Elements and Structure

• For mutation, we must be more careful
  – If the mutation point is in the head, then can mutate from anything to anything
  – It mutation point is in tail, can only mutate from operand to operand, which can be a variable or possibly a constant.

  • Note: Because of the way GE’s are evaluated, a mutation may have no effect – why?
Viability

• When generating a population of GE’s which represent expressions, it is possible to generate one or more that are not viable, i.e. cannot be evaluated
  – For example, what’s wrong with the following?
    \[ +a/-aa+*/a-aaaaaaaaaaaa \]

Viability

• While the likelihood is slight, it is possible that in generating an initial population, all GE’s could be non-viable.
• The author suggests that it is sometimes prudent to be sure to generate at least one solution among the initial population that is viable
  – The claim is that even with such a “dummy” expression, a correct result can be evolved
• One could also have the rule that in generating the initial population, when a GE is generated, check it for viability, and if it isn’t viable, discard it and generate another

Elitism

• A population can be divided into 3 groups
  – Those with the highest fitness in the population
    – there will be one or more of these
  – Those that are viable but have a fitness less than the best
  – Those that are not viable

Fitness

• The fitness function used in EC is extremely important.
• Needed attributes
  – Values with low fitness may be eliminated if values with very high fitness exist – homogeneity too great
  – Values with high fitness may overwhelm all others

\[
f_i = \sum_{j=1}^{C_i} (M - |C_{i,j} - T_j|)\]

\[f_i = \text{fitness of chromosome i} \]
\[M = \text{range value.} \]
\[M \text{ can vary by iteration} \]
\[C_{i,j} = \text{value returned by chromosome i} \]
\[T_j = \text{target value} \]
\[C_i = \text{number of fitness cases} \]

For a perfect fit, \(C_{i,j} = T_j\) and thus \(f_i = f_{\text{max}} = C_i M\)

\[
f_i = \sum_{j=1}^{C_i} (M - \frac{C_{i,j} - T_j}{T_j} * 100)\]

\[f_i = \text{fitness of chromosome i} \]
\[M = \text{range value.} \]
\[M \text{ can vary by iteration} \]
\[C_{i,j} = \text{value returned by chromosome i} \]
\[T_j = \text{target value} \]
\[C_i = \text{number of fitness cases} \]

For a perfect fit, \(C_{i,j} = T_j\) and thus \(f_i = f_{\text{max}} = C_i M\)
Fitness

- For the two preceding fitness functions, the first is used when minimum absolute error is the goal. The second is chosen when minimum relative error is the goal.
- Also, if one sets the error at a value say $< 0.03$, then if a value $|C_{ij} - T_j| < 0.03$, then set $|C_{ij} - T_j|$ to 0

Mate Selection

- The author of GEP always uses roulette wheel selection – along with elitism in that the best of each generation is copied unchanged into the next generation.
- Even though roulette wheel has stochastic errors, she claims that the diversity maintained is a greater benefit than the errors are a detriment

Constants in a GE

- There are three approaches
  - Approach 1:
    - In generating a GE, let the set of generated elements be operators, operands (variables), and random constants
    - When performing mutation, if the element to mutate is a random constant, just randomly generate it again

- Approach 2:
  - The inventor of GEP uses a much more elaborate scheme for generating random constants. She adds a third segment to a GE. We will call that third segment the constant-pointers segment.
  - Constant-pointers
    - A string of integer pointers to randomly generated constants – Thus there will also be an array of constants that can be mutated

Constants in a GE

- Approach 2:
  - In the following slides, the “?” represent one of the random constants.
  - They are replaced left-to-right top-to-bottom from the constant pointers list.
  - If there are more ?’s than constant pointers, then it is done mod length of constant pointers, i.e. the list begins to repeat from the beginning after it is exhausted

```
01234567890123456789012345678901234567890123456789012345678901234
*??/*a-*/a??a??a??a281983874486
```
### Constants

- The number represented in the constants pointer list is actually an index into a list of constants.
- Let’s say that list consisted of the following:
  \[ A = \{-2.829, 2.55, 2.399, 2.979, 2.442, 0.662, 1.797, -1.272, 2.826, 1.618\}, \]
- We would then have the following GEP

\[
\begin{align*}
\{\text{a} \} \times \{\text{2.399} \} + \{\text{2.826} \} / \{\text{1.818}\} \times \{\text{2.979}\} \times \{\text{a}\} \times \{\text{2.56}\} \\
\{\text{2.399}\} + \{\text{2.826}\} / \{\text{2.56}\} \times \{\text{a}\} \\
\{\text{a}\} \times \{\text{2.399}\} + \{\text{2.826}\} / \{\text{1.818}\} \times \{\text{2.979}\} \times \{\text{a}\} \times \{\text{2.56}\}
\end{align*}
\]

- Note that the addition of the constant pointers (CP) element in a GE requires the following changes
  - When designing the chromosome, now have 3 parts, head, tail, and CP
    - Like the head and tail lengths that must be set, the CP length must be set, and the number of elements in the array of random values
  - When generating a chromosome, the head and tail portions have one more element that can be randomly inserted, i.e. the “?”

- In generating the CP segment of a chromosome, for each position in the CP, randomly generate an integer that is the pointer into the list of random constants
  - Must also randomly generate the list of random constants
  - When applying mutation to the CP, you should generate a random constant in the range of the length of the list of random numbers
  - When applying mutation, you should also apply mutation to the list of random constants

- The third approach that the author actually prefers is to allow GEP to generate the constants.
  - This generally means that rather than a single gene chromosome, one must have a multi-gene chromosome
  - With a multi-gene chromosome, one must have a method of combining the genes. A simple approach is to use addition.
Generating the next generation, assuming that the fitness of each chromosome has been determined, and the population size is $N$.

- Place best of population in next generation
- Select $N$ chromosomes for next generation
- Mutate
- Transposition
- Mutate

Mutation

- We have already discussed the how of mutation, the only issue is the frequency
  - The author of GEP suggests an average mutation rate equal to 2 mutations per chromosome
    - If chromosome length = 10 then want $P_m = 0.2$
    - Note that mutation may
      - Have no effect
      - Make a chromosome non-viable, i.e. set its fitness to 0

Transposition

- There are three types of transposition
  - IS (insertion sequence) transposition
  - RIS (root insertion sequence) transposition
  - Gene transposition
    - This case only arises if the chromosome consists of a group of genes
- For each type of transposition, the system randomly chooses chromosomes to perform transposition on

IS Transposition

- Randomly choose a duplication site in the chromosome, and a length ($n$)
- Randomly choose an insertion site in the chromosome ($k$) – anywhere in the head+tail
  - The insertion site cannot be the root (first) element of the chromosome.
  - Replace the $n$ elements starting at location $k$
    - Note that this has now duplicated the sequence into the chromosome

IS Transposition

- Example: Population size = 10, $P_{IS} = 0.1$, $L1=3$, $L2=4$, $L3=6$, chromosome length = 100
  - This means we would randomly select 1 ($0.1*10 = 1$) chromosome to perform IS transposition on
    - Randomly generate IS start location (0-99) assuming indices are 0-99, let’s say 46
    - Randomly generate (1 to 3) the desired transposon length, let’s say we generate a 2, so the length will be 4.
    - Randomly generate insert location (1-99) let’s say 27
**IS Transposition**

- Now insert the 4 elements starting at location 46 into the location starting at 27.
  - The inserted sequence does not replace elements, it increases the length of the chromosome so that elements are lost off the end.
  - The insertion can occur in the tail only if there are only tail elements in the IS, in which case, elements drop off the end of the tail.
- **Note** if the length of the IS transposon causes the insertion sequence to go past the tail, don’t perform the IS transposition, i.e. in this case 46+3 > length of the chromosome.

**RIS Transposition**

- From anywhere in the head, choose a site k.
- Scan starting at k until an operator is found.
- For a random number n of elements, create a duplicate sequence.
- Starting at the head of the chromosome, replace the n elements.
- **Note:** if, starting at k an operator is not found, go to the next chromosome.

**RIS Transposition**

- The author suggests a $P_{IS} = 0.1$, and 3 different transposon lengths.

**Gene Transposition (GT)**

- In GT a chromosome is randomly chosen.
- An entire gene in that chromosome is randomly chosen (any except the first gene).
- That gene replaces the first gene and all genes move down in the sequence.
Recombination

- Recombination is similar to the crossover that we have seen
- The author uses three types
  - 1-point
  - 2-point
  - Gene recombination

1-point Recombination
- Randomly chose a crossover point (the same) for the two chromosomes and swap material
- The author uses a crossover rate of 0.3 to 0.7
- As a rule of thumb, she suggests that the sum of the three crossover rates be 0.7

2-point Recombination
- 2 points for crossover are chosen.
- The material between these two points is then exchanged

Suppose gene 3 was chosen to undergo gene transposition. Then the following chromosome is obtained:

012345678901234567890123456789012
+/Qa*bbaaabaa*a*/Qbhbhbhbhb/Q-aabbaabb

Recombination is similar to the crossover that we have seen.

The author uses three types
- 1-point
- 2-point
- Gene recombination

0123456789012345601234567890123456
+*-b-Qa*aabbbbaaa-Q-//b/*aabbbabba
++//b/-bbbbbhbhb-**-ab/b+bbbabbaaa

Suppose bond 6 in gene 1 (between positions 5 and 6) was randomly chosen as the crossover point. Then, the paired chromosomes are cut at this bond, and exchange between them the material downstream the crossover point, forming the offspring below:

0123456789012345601234567890123456
+*-b-Q/-bbbbbhbhb-**-ab/b+bbbabbaaa
++//b/a*aabbbbaaa-Q-//b/*aabbbabba

0123456789012345601234567890123456
+*+/Q/QaaabbbbabQQab*++-aababaabb
Q/-b-++/abaabbaab/*-aqaa*babbabbbab

Suppose bond 5 in gene 1 (between positions 4 and 5) and bond 7 in gene 2 (between positions 6 and 7) were chosen as the crossover points. Then, the following chromosomes are created:

0123456789012345601234567890123456
+*-+/abaabbaab/*-aqaa*-aabbabaab
Q/-b-Q/QaaabbbbabQQab*++babababb
Gene Recombination

- Genes in each of two randomly chosen chromosomes are randomly chosen, and the genetic material of each entire gene is swapped.

Extensions to GEP

- What we have discussed to date are chromosomes in which there is a single gene, i.e. a single head+single tail, and possibly an extension for constants.
- The most important extension to the basic GE format is what the author terms **multigenic chromosomes**
- Multigenic chromosome
  - A chromosome in which there is more than one gene

Multigenic Chromosome

- Multiple genes in a GE facilitates the exploration of the search space
- Multigenic chromosome
  - Each chromosome in the population has the same number of genes
  - Number of genes and head length and thus the gene length is fixed at time of definition
  - Each gene forms a sub-expression
  - Final result is formed by some forming or combining of the sub-expressions

Linking Sub-ET’s

- The author calls these individual genes – sub-ET, or sub-expression trees.
- There are several approaches to linking the genes
  - The requirement is that the form used fit the number and type of the genes.
Linking Sub-ET’s

• For example, let’s say we want to link the genes with an if.
  – Would need 3 genes
  – First gene evaluates as a Boolean
  – Next two genes (and only two) evaluate as expressions
  – This means there would be a total of three genes in the chromosome

• Note that when you have genes of different types, the operator and operand types may be different for the different genes
  – This will especially impact the mutation operation
• The most common linking functions are
  – If, And, Or, addition, multiplication

Linking Sub-ET’s

• It is also possible to let the interaction of the sub-ET’s evolve
  – In such a case, say for a 3-gene chromosome, a fourth gene would be added
    • The author calls this gene a homeotic gene
    • This gene codes for the interaction of the other genes.
    • It can have a different or the same set of operators
    • Its operand set consists of integers which point to the three genes, and possibly the other set of operands, etc.

Multi-Geneic GEP

• There are basically three approaches in GEP
  – The single gene approach
  – Use a multi-gene approach in which the genes are combined based upon some predetermined function, e.g. “+”
  – Use a multi-gene approach as follows:

• A multi-gene chromosome consists of N genes. The first N-1 genes are all of the same length and use the same operand and operator set. Each of these genes specifies the code for what could be termed a function. Gene N is a homeotic gene which differs from the other genes in that it:
  – May have a different length.
  – May or may not use all of the operators as used in the other genes
  – May or may not use all of the operands as used in the other genes
  – Will use as potential operands the values of the other genes
• When evaluating a multi-gene chromosome, first evaluate the N-1 genes, then use the values returned by these evaluations in the evaluation of the homeotic gene. The value returned by the homeotic gene is the value returned by the chromosome.