Genetic Algorithms

- Overview of GAs
- Architectural/Implementation issues
- Formal basis of GAs
- Misc: alternative algorithms, applications

Genetic Algorithms As Function Optimizers

- Genetic algorithms search a population representing different sample points in the search space.
- Each sample point is represented as a string which can be recombined with other strings to generate new sample points in the space.
- Algorithms are based on biological analogies with "population genetics" and "simulated evolution".

Genetic Algorithms: Machine Learning or Search?

- "a variant of stochastic beam search in which successor states are generated by combining two parent states…" [Russell and Norvig]
- Performance is often compared to that of search algorithms (e.g., local search).
- What do they learn?

GA Applications

- function optimization (maximize/minimize some function):
  - best control parameter settings
- combinatorial optimization (order a set of objects to achieve some objective):
  - manufacturing scheduling
- machine learning:
  - interpreting layers in geophysical data
Why Genetic Algorithms?

- **No Gradient Information Is Needed.** These algorithms do not search along the contours of the function, but rather by hyperplane sampling in Hamming space.
- **The Resulting Search is Global.** Since they do not hill-climb, they avoid local optima and so can be applied to multimodal functions.
- **Potential for Massive Parallelism.** Can effectively exploit thousands of processors in parallel.
- **They Can Be Hybridized** with conventional optimization methods.

Basic Algorithm

```
function Genetic-Algorithm(population, Fitness-Fn)
    returns an individual

inputs: population, a set of individuals
        Fitness-Fn, measures the fitness of an individual

repeat
    parents — Selection(population, Fitness-Fn)
    population — Reproduction(parents)
until some individual is fit enough
return the best individuals in population, according to Fitness-Fn
```

Issues: Encoding

- **Binary:** parameter values are encoded as binary numbers
- **Nonbinary:** larger alphabets, real-valued encodings, permutations

Arguments against:

- there will be fewer explicit hyperplane partitions
- the alphabetic characters will not be as well represented in a finite population.

Issues: Fitness Function

- Domain specific to goals of problem
- Single value output: multi-objective must be combined into single function
- Fast! May need to be executed hundreds of thousands of times
- Sometimes approximate to achieve speed
Genetic Algorithm Process

Selection (Duplication)
- String1
- String2
- String3
- String4
- ...

Recombination (Crossover)
- OffspringA
- OffspringB
- OffspringC
- OffspringD
- ...

Current Generation t

Intermediate Generation t

Next Generation t+1

Algorithm (R&N, modified)

```
Genetic-Algorithm(pop, fitness-fn)
repeat
    new-pop = {}
    loop for j from 1 to |pop| do
        x = Random-selection(pop, fitness-fn)
        y = Random-selection(pop, fitness-fn)
        child = Reproduce(x, y)
        if (random(1.0) < thresh) then child = Mutate(child)
        add child to new-pop
    pop = new-pop
until some # of iterations
return best from pop
```

Issues: Initializing Population

Random: generate n strings randomly, within encoding requirements.

Domain specific: use heuristic method to generate “ok” solutions that can be refined.

Selection in the Canonical Genetic Algorithm

- Population is evaluated according to a fitness function.
- Parents are selected for reproduction by ranking according to their relative fitness $f_i$
- Text method: probability of selection is proportional to fitness.
Issues: Selection

- **Fitness Selection:**
  - Stochastic sampling with replacement
    - Map individuals to space on a roulette wheel, more fit individuals are allocated proportionally more space.
    - Spin wheel repeatedly until desired population size is achieved

Population Example, **Stochastic Sampling w/Replacement**

<table>
<thead>
<tr>
<th>String</th>
<th>Fit</th>
<th>Space</th>
<th>copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>001000000</td>
<td>2.0</td>
<td>.995</td>
<td></td>
</tr>
<tr>
<td>101010101</td>
<td>1.9</td>
<td>.186</td>
<td></td>
</tr>
<tr>
<td>111100011</td>
<td>1.8</td>
<td>.371</td>
<td></td>
</tr>
<tr>
<td>010001100</td>
<td>1.7</td>
<td>.352</td>
<td></td>
</tr>
<tr>
<td>111000000</td>
<td>1.6</td>
<td>.429</td>
<td></td>
</tr>
<tr>
<td>101000110</td>
<td>1.5</td>
<td>.5</td>
<td></td>
</tr>
<tr>
<td>011001110</td>
<td>1.4</td>
<td>.567</td>
<td></td>
</tr>
<tr>
<td>001110000</td>
<td>1.3</td>
<td>.629</td>
<td></td>
</tr>
<tr>
<td>001010100</td>
<td>1.2</td>
<td>.886</td>
<td></td>
</tr>
<tr>
<td>100100011</td>
<td>1.1</td>
<td>.738</td>
<td></td>
</tr>
<tr>
<td>010000111</td>
<td>1.0</td>
<td>.786</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>Fit</th>
<th>Space</th>
<th>copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>011001111</td>
<td>0.9</td>
<td>.829</td>
<td></td>
</tr>
<tr>
<td>000100110</td>
<td>0.8</td>
<td>.387</td>
<td></td>
</tr>
<tr>
<td>110001100</td>
<td>0.7</td>
<td>.9</td>
<td></td>
</tr>
<tr>
<td>110000111</td>
<td>0.6</td>
<td>.929</td>
<td></td>
</tr>
<tr>
<td>100100100</td>
<td>0.5</td>
<td>.952</td>
<td></td>
</tr>
<tr>
<td>011010011</td>
<td>0.4</td>
<td>.971</td>
<td></td>
</tr>
<tr>
<td>000110000</td>
<td>0.3</td>
<td>.986</td>
<td></td>
</tr>
<tr>
<td>001001000</td>
<td>0.2</td>
<td>.995</td>
<td></td>
</tr>
<tr>
<td>100110110</td>
<td>0.1</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>010010011</td>
<td>0.0</td>
<td>--</td>
<td></td>
</tr>
</tbody>
</table>

Random #s: .93, .65, .02, .51, .20, .93, .20, .37, .79, .28, .13, .70, .80, .51, .76, .45, .61, .07, .76, .86, .29

Other Selection Methods

- **Tournament Selection:** randomly select two strings, place the best into the new population, repeat until intermediate population is full
- **Ranking:** order individuals by rank rather than fitness value

Another Fitness Selection Method

- Remainder stochastic sampling
  - Again map to roulette wheel, but this time add outer wheel with N evenly spaced pointers.
  - Spin once to determine all population members
Population Example, 
Remainder Stochastic Sampling

<table>
<thead>
<tr>
<th>String</th>
<th>Fitness</th>
<th>Random</th>
<th>copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010000</td>
<td>2.0</td>
<td>--</td>
<td>2</td>
</tr>
<tr>
<td>10101010</td>
<td>1.9</td>
<td>0.93</td>
<td>2</td>
</tr>
<tr>
<td>11110011</td>
<td>1.8</td>
<td>0.65</td>
<td>2</td>
</tr>
<tr>
<td>01000110</td>
<td>1.7</td>
<td>0.02</td>
<td>1</td>
</tr>
<tr>
<td>11100000</td>
<td>1.6</td>
<td>0.51</td>
<td>2</td>
</tr>
<tr>
<td>10100010</td>
<td>1.5</td>
<td>0.20</td>
<td>1</td>
</tr>
<tr>
<td>01100110</td>
<td>1.4</td>
<td>0.93</td>
<td>2</td>
</tr>
<tr>
<td>00111000</td>
<td>1.3</td>
<td>0.20</td>
<td>1</td>
</tr>
<tr>
<td>00011010</td>
<td>1.2</td>
<td>0.37</td>
<td>1</td>
</tr>
<tr>
<td>10000011</td>
<td>1.1</td>
<td>0.79</td>
<td>1</td>
</tr>
<tr>
<td>00000111</td>
<td>1.0</td>
<td>--</td>
<td>1</td>
</tr>
</tbody>
</table>

Random #s: .93, .85, .02, .51, .20, .93, .20, .37, .79, .28, .13, .70, .80, .51, .76, .45, .61, .07, .76, .86, .29

Reproduction: Recombination/Crossover

Two parents: binary strings representing an encoding of 5 parameters that are used in some optimization problems.

Recombination occurs as follows:

Producing the following offspring:

Issues: Combination Operators

1 point crossover: pick single crossover point, split strings at this point, recombine

2 point crossover: pick two crossover points, split strings at these points, recombine (think of ring for string)

Uniform crossover: randomly pick each element from one of the two parents
Other Combination Operators

**HUX**: exactly half of the differing bits are swapped

Given parents:
- 10011010111100010110
- 1011001110001100001

A new individual is:
- 1001001111100011010

More Combination Operators

**Reduced Surrogate**: Crossover points chosen within differing bits only

- 10011010111100010110
- 1011001110001100001

May become:
- 100110111100001100001

**Domain specific**: operations designed to match demands of the domain (e.g., reorder portions for scheduling application)

Crossover and Hypercube Paths

Why Might Reduced Surrogate Be Important

- Closely related to HUX
- Key idea: look only at portions of strings that differ

- 000111110110100111
- 00010011010010010

- How does probability of new string change with reduced surrogate versus 1-point crossover?
Mutation
- For each bit in population, mutate with probability $p_m$
- $p_m$ should be low, typically < .01
- can mean randomly select a value for the bit or flip the bit

Issues: Which Strings in New Generation
- Replace with offspring
  - Assumption of canonical GA
  - Best of offspring and parents
    - Alternative view which guarantees always keep best and outs intensive pressure on population for improvement

Issues: Termination Criteria
- **Quality of solution**: best individual passes some pre-set threshold
- **Time**: certain number of generations have been created and tested
- **Diminishing returns**: improvement over each generation does not exceed some threshold

Alternatives to the Canonical GA Model
- Genitor
- CHC
- Parallel Algorithms
### Genitor (Whitley et al.)

Differences with canonical genetic algorithm:
- Reproduction proceeds one individual at a time.
- Worst individual in population is replaced by new offspring.
- Fitness is assigned by rank.

---

### Genitor Algorithm (Whitley)

- **Selection**
- **Crossover & Recombination**
- **Insert In population**

### CHC (Eshelman)

- **Crossover** using generation elitist selection
- **Heterogeneous** recombination by incest prevention
  - Origin of HUX terminology
- **Cataclysmic** mutation, when population starts to converge

### Parallel Genetic Algorithms

- **Simple parallel genetic algorithms**
  - Use tournament selection,
  - Implement on N/2 processors where each has two strings, each processor holds two independent tournaments, both processors get the winners
- **Parallel island models:**
  - Each processor has subpopulations
  - Run GA on each processor
  - Every 5 generations or so, allow subpopulations to swap some strings (*migration*)
Parallel GAs (cont.)

- Fine grain parallelism
  - One string per processor, strings seek best “mates” from adjacent processors
  - Produce one offspring which is kept in both processors

Underlying Theory: Hyperplane Sampling

Another View of Hyperplane Sampling

Population Example for Hyperplane Sampling
### Some Schemata and Fitness Values

<table>
<thead>
<tr>
<th>Schema</th>
<th>Mean (μ)</th>
<th>Count</th>
<th>Expect (E)</th>
<th>Observe (O)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10<em>1</em></td>
<td>3.70</td>
<td>2</td>
<td>3.4</td>
<td>3</td>
</tr>
<tr>
<td>11<em>1</em></td>
<td>3.70</td>
<td>2</td>
<td>3.4</td>
<td>4</td>
</tr>
<tr>
<td>*<em>01</em></td>
<td>3.38</td>
<td>5</td>
<td>6.9</td>
<td>6</td>
</tr>
<tr>
<td>**<em>1</em></td>
<td>3.30</td>
<td>10</td>
<td>13.0</td>
<td>14</td>
</tr>
<tr>
<td>*<em>11</em></td>
<td>3.22</td>
<td>5</td>
<td>6.1</td>
<td>8</td>
</tr>
<tr>
<td>11**</td>
<td>3.175</td>
<td>4</td>
<td>4.7</td>
<td>6</td>
</tr>
<tr>
<td>001**</td>
<td>3.166</td>
<td>3</td>
<td>3.5</td>
<td>3</td>
</tr>
<tr>
<td><em>1</em>*</td>
<td>3.089</td>
<td>9</td>
<td>9.8</td>
<td>11</td>
</tr>
<tr>
<td>01**</td>
<td>3.033</td>
<td>6</td>
<td>6.2</td>
<td>7</td>
</tr>
<tr>
<td>10**</td>
<td>3.020</td>
<td>5</td>
<td>5.1</td>
<td>5</td>
</tr>
<tr>
<td>*<strong>0</strong></td>
<td>3.010</td>
<td>10</td>
<td>10.1</td>
<td>12</td>
</tr>
<tr>
<td>*****</td>
<td>3.000</td>
<td>21</td>
<td>21.0</td>
<td>21</td>
</tr>
</tbody>
</table>

### Hyperplane Deception

Since genetic algorithms are driven by hyperplane sampling a misleading problem can be constructed as follows.

- \( f(0^{**}) > f(1^{**}) \)
- \( f(*0*) > f(*1*) \)
- \( f(**0) > f(**1) \)
- \( f(00*) > f(01*), f(10*), f(11*) \)
- \( f(0*0) > f(0*1), f(1*0), f(1*1) \)
- \( f(*00) > f(*01), f(*10), f(*11) \)
- \( f(111) > f(000) \)

where \( f(x) \) gives the average fitness of all strings in the hyperplane slice represented by \( x \).

### The Schema Theorem

**Selection Only**

\[
M(H, t + \text{ intermediate}) = M(H, t) \frac{f(H)}{f}
\]

**An Exact Calculation**

\[
M(H, t + 1) = \left( 1 - p_c \right) M(H, t) \frac{f(H)}{f} + p_c \left[ M(H, t) \frac{f(H)}{f} \left( 1 - \text{ losses} \right) + \text{ gains} \right]
\]

Strings that do not undergo crossover
Crossover causes losses and gains.

### Schema Theorem – Lower Bound

**A Common Version of the Schema Theorem**

\[
P(H, t + 1) \geq P(H, t) \left( 1 - p_c \right) \frac{f(H)}{f} \left[ 1 - L^{-1} \left( 1 - P(H, t) \frac{f(H)}{f} \right) \right]^{1 - p_m}
\]

**Disruptions** depend on the vulnerable proportion of the string times its inverse proportion in the population.

**Mutations** indicate the proportion left unaffected by flipping each vulnerable bit according to mutation probability.
Criticisms of the Schema Theorem

- Inequality, thus loss of information.
- Average fitness is only relevant for the first generation or two.

*However…provides a lower bound for first few generations.*

A Generator For String Losses

Given strings:
- 00000000000
- 0001000100

In what cases will we get losses during crossover?

For strings $B$ and $B'$, the middle $\Delta+1$ bits have pattern $b##...#b$

$\overline{b}##...#\overline{b}$

The b's are sentry bits and define the probability of disruption.

Visualizing the Generators

A Generator For String Gains

<table>
<thead>
<tr>
<th>Region -&gt;</th>
<th>Beginning</th>
<th>Middle</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length -&gt;</td>
<td>A</td>
<td>R</td>
<td>W</td>
</tr>
<tr>
<td>Q characteristics</td>
<td>#...#R</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>R characteristics</td>
<td>=</td>
<td>=</td>
<td>$\overline{R}$...#</td>
</tr>
</tbody>
</table>
Visualizing String Gains

Total pairs: \[ \sum_{j=1}^{\frac{n}{2}} j \times 2^j \]

10000
00001

#1000
00001

10000
0001#

##100
0001#

#100
001##

##100
001##

###10
001##

####10
0001#

####10
0001#

10000
00001

10000
0001#

10000
001##

10000
01###


spring 2005 ©cs640- Howe