Introduction to Evolutionary Computation
Evolutionary Computation

- Different lines of investigation in simulated evolution
  - GA (Genetic Algorithms)
  - ES (Evolution Strategies)
  - EP (Evolutionary Programming)

* Each abstracts evolution in a slightly different manner
Evolutionary Theory for Simulation

• Neo-darwinian paradigm:
  – Synthesis of classic darwinism
    selectionism of Weismann
    genetics of Mendel

• Statistical processes within population
  – Reproduction
  – Mutation
  – Competition
  – Selection
Duality of Living Organism

- Genotype: underlying genetic coding
- Phenotype: manner of response contained in physiology, morphology and behavior of organisms

Epigenesis:
\[ f_1 : I \times G \rightarrow P \]
Selection:
\[ f_2 : P \rightarrow P \]
Genotypic survival:
\[ f_3 : P \rightarrow G \]
Mutation:
\[ f_4 : G \rightarrow G \]
Overview of EC

• Similarity
  – Population-based search
  – Impose random variation
  – Apply selection operation

• Difference
  – Typical representation
  – Emphasis on type of variation imposed
  – Promoting vs. culling view of selection

• Three main lines of investigation
  – GA, ES, EP
    \[ \text{EA} \]
    \[ \text{EC} \]
A Simple Evolutionary Algorithm

1. Generate the initial population $P(0)$ at random, and set $i \leftarrow 0$;

2. REPEAT
   
   (a) Evaluate the fitness of each individual in $P(i)$;
   
   (b) Select parents from $P(i)$ based on their fitness in $P(i)$;
   
   (c) Generate offspring from the parents using crossover and mutation to form $P(i + 1)$;
   
   (d) $i \leftarrow i + 1$;

3. UNTIL halting criteria are satisfied
# Variants of EC

<table>
<thead>
<tr>
<th></th>
<th>Genetic Algorithm</th>
<th>Evolutionary Strategy</th>
<th>Evolutionary Programming</th>
<th>Genetic Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representation</td>
<td>Binary</td>
<td>Real</td>
<td>Real</td>
<td>Tree</td>
</tr>
<tr>
<td>Mutation</td>
<td>Minor</td>
<td>Main</td>
<td>Main</td>
<td>Minor</td>
</tr>
<tr>
<td>Recombination</td>
<td>Main</td>
<td>Minor</td>
<td>None</td>
<td>Main</td>
</tr>
<tr>
<td>Initial Application</td>
<td>Adaptive System</td>
<td>Parameter Optimization</td>
<td>Artificial Intelligence</td>
<td>Program Evolution</td>
</tr>
<tr>
<td>Pioneer</td>
<td>John Holland</td>
<td>Rechenberg</td>
<td>Fogel</td>
<td>Koza</td>
</tr>
</tbody>
</table>
Representation

- Binary strings
- Real-valued vectors
- Permutations
- Finite-state representations
- Parse trees
Selection

- Proportional selection
- Tournament selection
- Rank-based selection
- Boltzmann selection
Advanced Techniques

- Population structures
  - Niching, speciation, island models
- Population sizing
- Mutation parameters
- Recombination parameters
- Parameter control
- Self-adaptation
- Meta-evolutionary approaches
- Coevolutionary algorithms
Journals and Conferences

• Journals
  – Evolutionary Computation (MIT Press)

• Conferences
  – GECCO (Genetic and Evolutionary Computation Conference)
    GP+ICGA
  – PPSN
  – CEC (Congress on Evolutionary Computation) EP+ICEC
Sources


Overview

• Invented by John Holland (early 1970s)

• Techniques for optimization and machine learning inspired by features of theory of biological evolution

• Solution encoded chromosomes

• Search proceeds through maintenance of a population of solutions

• Reproduction biased in favor of better individuals

• New encodings created by mutation and recombination
Five Components

- Encoding technique ("chromosome structure")
- Evaluation function ("environment")
- Initialization procedure ("creation")
- Genetic operators (mutation, recombination, etc)
- Parameter settings (practice and art)
Procedure

reproduction → children
parents
population
evaluation
modified children
modified children
deleted members
discard
Population of Chromosomes

- Bit string (0 1 1 0 … 1 0 1 1)
- Real number lists (23.6 -2.45 … 3.1 0.0)
- Permutation of elements (E26 E13 E18 … E4 E9)
- etc
Reproduction

- Parents are selected at random, with selection chances biased in relation to chromosome evaluations.
  
ex) parent selection (expected children = average / evaluation)
  
average = 10

<table>
<thead>
<tr>
<th>chromosome</th>
<th>evaluation</th>
<th>expected children</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>1.5</td>
</tr>
</tbody>
</table>

- cf) Roulette Wheel Parent Selection
  
  Given parent 1 with evaluation 10
  
  2         5
  
  3         15

  Chances of being chosen are proportional to evaluation
Chromosome Modification

- Triggered stochastically
- Operators: mutation / crossover (recombination)
  1. Mutation causes a local modification
     
     \[
     (0 1 1 0 1 1 0) \rightarrow (2.67, -49.3, 128.68, 33.1) \\
     (0 1 0 0 1 1 0) \rightarrow (2.67, -47.6, 128.69, 33.1)
     \]
  2. Crossover causes recombination
     
     \[
     (0 1 1 0 1 1 0) \times (0 1 0 1 1 0 1) \rightarrow (1 1 0 1 1 0 1) \\
     (1 1 0 1 1 0 1) \rightarrow (1 1 1 0 1 1 0)
     \]

- Important feature of GA
- Accelerate search early in evolution of a population
- Lead to effective combination of schemata (subsolutions)
Evaluation and Deletion

• Evaluation
  – evaluator decodes a chromosome and assigns it a number measuring its success
  – the only link between classical GA and problem

• Deletion
  – generational approach: entire population can be replaced at a time
  – steady-state approach: a few members can be replaced at a time
An Example (1)

\[ f(x) = x^2 \quad \text{where} \quad 0 \leq x \leq 31 \]

- decoding
  - 5-bit unsigned binary integer
  - ex) 13 = 01101

- initial population (at random)
  - population size = 4
  - tossing a fair coin 20 times
## An Example (2)

<table>
<thead>
<tr>
<th>Population</th>
<th>x value</th>
<th>f(x) value</th>
<th>$\chi^2$</th>
<th>$f_i / \sum f$</th>
<th>$f_i / f$ expected count</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01101</td>
<td>13</td>
<td>169</td>
<td>0.14</td>
<td>0.58</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11000</td>
<td>24</td>
<td>576</td>
<td>0.49</td>
<td>1.97</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>01000</td>
<td>8</td>
<td>64</td>
<td>0.06</td>
<td>0.22</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>10011</td>
<td>19</td>
<td>361</td>
<td>0.31</td>
<td>1.23</td>
<td>1</td>
</tr>
</tbody>
</table>

| sum        |         | 1170       | 1.00    | 4.00           | 4                         |       |
| average    |         | 293        | 0.25    | 1.00           | 1                         |       |
| max        |         | 576        | 0.49    | 1.97           | 2                         |       |

<table>
<thead>
<tr>
<th>mate</th>
<th>crossover</th>
<th></th>
<th></th>
<th></th>
<th>$\chi$</th>
<th>f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101</td>
<td>11000</td>
<td>2</td>
<td>4</td>
<td>01101</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>01000</td>
<td>11000</td>
<td>1</td>
<td>4</td>
<td>11000</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>10011</td>
<td>01000</td>
<td>4</td>
<td>2</td>
<td>01000</td>
<td>27</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>10011</td>
<td>4</td>
<td>2</td>
<td>10011</td>
<td>16</td>
<td>4</td>
</tr>
</tbody>
</table>

1754
439
729
Schema

- Combination of chromosome features
  ex) $(0 \; 1 \; 1 \; 0 \; 1)$ contains schemata
    $(# \; 1 \; 1 \; # \; #), \; (# \; 1 \; # \; 0 \; #), \; (0 \; 1 \; 1 \; # \; 1), \; (0 \; 1 \; 1 \; 0 \; 1), \; (# \; # \; # \; # \; #)$

- Holland’s schema theorem
  - $N_t$: no. of occurrences $S$ at $t$
  - $E_t$: avg. evaluation of $S$ at $t$
  - $A_t$: avg. evaluation of entire population at $t$

  better than avg. : proliferate \[ \frac{N_{t+1}}{N_t} \approx \frac{E_t}{A_t} \]
  bad ones : die out
Hybrid GA

- Combination of GA and one or more other optimization algorithms
- If done correctly, it can’t do worse
- Combining search capability of GA with domain-based expertise
Creating a Hybrid GA

decoder
representation
operators
population
initialization
domain knowledge
domain algorithm
Applications

- Control: gas pipeline, pole balancing, missile pursuit
- Scheduling: production / facility
- Game playing: poker, prisoner’s dilemma
- Design: semiconductor layout, aircraft design, communication network design, keyboard configuration
- Combinatorial optimization: set covering, TSP, routing, graph coloring / partitioning
- Machine learning: designing NN, improving classification algorithms, classifier systems
- Signal processing: filter design
- Robotics: trajectory planning
Some Current Efforts

- Analyzing mathematical properties
- Artificial life simulations
- Practical real world applications
- Parallel distributed processing
Homework #3

• GA를 사용하여 실제 문제를 해결하고자 할 때, 다음의 parameter들을 결정하는 기준에 대해서 조사하시오.
  – Population 수
  – Mutation rate
  – Crossover rate
  – Selection 방법
LCS 구조

- Rule Discovery Subsystem (GA)
- Rule and Message Subsystem
- Apportionment of Credit Subsystem

Messages: 1010

Feedback:

- Add/Delete Classifiers
- Matching Classifiers

Classifier list:

- ###0:01
- #1##:10
- ... #010:11
- ...

Winning Action:

- Winner
Classifier

- Classifier list
  - 여러 개의 규칙들, 즉 classifier들로 구성
- Classifiers
  - 일반적으로 랜덤하게 초기화

```plaintext
0  #  #  1  #  0  #  1  0  0  0  1
```

- 조건부
  `{0, 1, #}
  #:don’t care`
- 실행부
  `{0, 1}`
학습 방법

• 규칙 학습 :
  – Credit assignment algorithm
    – Bucket brigade algorithm
    – 일종의 강화 학습 방법
  – 유전자 알고리즘(GA)
    – 규칙 발견(Rule Discovery)
Credit Assignment Algorithm

- “Strength”
  - 각 classifier마다 가지고 있는 값
  - Classifier의 가치를 평가하는 기준
- 입력 메시지와 일치하는 classifier들의 경쟁 ➔ 승자(Winner): 최종 출력
- 외부 환경으로부터의 피드백(Feedback)
  - 최종 출력이 외부로 보내지면, 피드백을 받음
  - 승자 classifier에 Credit이 주어짐

\[ B(C, t) = bR(C)s(C, t) \]
유전자 알고리즘 (GA)

- 규칙 발견 알고리즘
- 한 세대가 끝난 후, 유전 연산자(교차, 돌연변이)를 이용하여 다음 세대의 규칙들을 생성
- CS에 적용
  - 외부 환경에 적합한 classifier들을 찾기 위함
  - 외부 환경과 효율적인 상호작용 능력을 가져야 함
  - 환경에 가장 적합한 classifier들만을 선택
  - 새로운 classifier들을 생성
Why Niching?

- In evolutionary computation, niching refers to the formation of groups of individuals in a population.
- Individuals within a group are similar to each other.
- Individuals from different groups are dissimilar to each other.
  - Niching helps to explore and maintain diversity.
  - Niching helps in machine learning, e.g., classification.
  - Niching helps multi-objective function optimization.
  - Niching helps simulation of complex and adaptive systems.
  - Niching helps cooperative co-evolution.
  - Niching is fun!
Different Niching Techniques

- Can be divided roughly into two major categories
  - **Sharing**, also known as fitness sharing
  - **Crowding**
- Other niching methods include sequential niching and parallel hillclimbing
Fitness Sharing: Introduction

- Fitness sharing transforms the raw fitness of an individual into shared fitness.
- It assumes that there is only limited and fixed “resource” available at each niche. Individuals in a niche must share them.
- Sharing is best explained from a multimodal function optimization perspective.

How can we locate multiple peaks in one evolutionary process?
Fitness Sharing: Implementation

- Define **a sharing radius** $\sigma_{share}$: Anything within this radius will be regarded to be similar to the individual and thus needs to share fitness

  ![Diagram showing a sharing radius](image)

  The individual in the center needs to share fitness with all others in the circle.

- Define **a similarity measure**, i.e., distance: The shorter the distance between two individuals, the more similar they are

- Define a sharing function

  \[
  sh(d) = \begin{cases} 
  1 - (d / \sigma_{share})^\alpha & \text{if } d \leq \sigma_{share} \\
  0 & \text{otherwise}
  \end{cases}
  \]

- Define shared fitness

  \[
  f_{share}(i) = \frac{f_{raw}(i)}{\sum_{j=1}^{\mu} sh(d_{ij})}
  \]

  where $\mu$ is the population size.
Fitness Sharing: Extensions

- Sharing can be done at genotypic or phenotypic level
  - Genotypic: Hamming distance
  - Phenotypic: Euclidean distance (Overlap in test case covering in classification) → The key issue is how to define the “distance” measure
- Sharing radius $s_{\text{share}}$ can be difficult to set, the same, fixed: It should be sufficiently small in order to discriminate between two neighboring peaks
- Population size should be sufficiently large to locate all peaks
- Population may not be able to converge to exact optima
- Population may not be stable, i.e., may lose peaks located
- Calculate shared fitness needs time
- Fitness sharing often needs raw fitness scaling.

$$f_{\text{share}}(i) = \frac{f_{\text{raw}}(i)}{\sum_{j=1}^{\mu} s_{\text{h}}(d_{ij})}$$

Why?

scaling factor
Why Fitness Scaling

- Let $f'_i = f_{\text{share}}(i)$, $f_i = f_{\text{raw}}(i)$, $m_i = \sum_{j=1}^{\text{sh}}(d_{ij})$
- Then fitness sharing is $f'_i = f_i / m_i$
  - Fitness sharing with scaling is $f'_i = f_i^\beta / m_i$, $\beta \geq 1$

![top-down view of a 2-d space](image)

![side-view of the 2-d space](image)

raw fitness

shared fitness

without scaling

with scaling using sufficiently large $\beta$
A Dilemma

- With low scaling factor: individuals won’t go to the real optimum because it’s not attractive
- With high scaling factor: We may not be able to find all peaks, because a high scaling factor creates “super individuals”, even a very soft selection scheme won’t help

→ A possible solution: Anneal the factor $\beta$

1. Start the evolution with a small $\beta$, e.g., $\beta = 1$, in order to explore and locate the peak regions
2. Then increase $\beta$ gradually to attract individuals to the optima
Implicit Fitness Sharing (1)

• The idea comes from an **immune system**: antibodies which best match an invading antigen receive the payoff for that antigen

• Similar situation occurs in **games**: a strategy receives payoff when it achieves the best score against a test case

• Implicit fitness sharing is most often used in learning. While **(explicit) fitness sharing** is done through individuals, **implicit fitness sharing** is test data based!

• The algorithm for calculating fitness:
  • For each data point \( i \) to be matched, do the following \( C \) times
  • 1. **Select** a sample of \( s \) individuals from the population
  • 2. **Find** the individual in the sample that achieves the highest score against the data point \( i \)
  • 3. This best individual receives the payoff. In the case of a tie, payoff is shared equally
Implicit Fitness Sharing (2)

• It has been shown that implicit and explicit fitness sharing have the same theoretical basis. Here plays the role of $\sigma_{\text{share}}$ in (explicit) fitness sharing.
• Larger C $\rightarrow$ better result but more time-consuming.
• Comparison between implicit and explicit sharing: they are better under different circumstances.
  – Implicit fitness sharing covers optima more comprehensively, even when those optima have small basin of attraction, when the population is large enough for a species to form at each optimum.
  – (Explicit) fitness sharing can find the optima with larger basins of attraction and ignore the peaks with narrow bases, when the population is not large enough to cover all optima.
Niching vs. Speciation

- Although some people distinguish between the two, we will treat them as **the same thing**
- If there is any difference:
  - niching is concerned more with locating peaks (basins of attraction), while speciation is more focused on actually converging to optima
Crowding

- Crowding techniques insert new elements into the population by replacing similar elements. It’s about *replacement strategies*.
- Crowding techniques strive to maintain the preexisting diversity of a population.
- Crowding methods do not modify fitness.
Crowding: Early Days

- **Cavicchio’s preselection scheme:**
  - Purpose: prevent premature convergence, preserve diversity
  - What it is: an offspring (usually the fitter one) replaces the worse parent
  - How well it works for niching: poorly, unable to form niches

- **De Jong’s Crowding** (crowding factor model):
  - Purpose: preserve population diversity
  - What it is: restricted replacement, offspring replaces the most similar parent in a same
  - How well it works for niching: poorly, only finding two peaks

- Common feature: **both involve modification of replacement strategies**. A restriction is often imposed
Deterministic Crowding: Algorithm (1)

Problems with De Jong’s crowding: high replacement error (i.e., replacing an individual of one class by another one from a different class)

Mahfoud’s deterministic crowding algorithm:
Input: $g$ – number of generations to run the algorithm
$\mu$ – population size
Output: $P(g)$ – final population
Algorithm:

1. $P(c) \leftarrow$ initialize ($\mu$)
2. FOR t $\leftarrow$ 1 to g DO
   1. $P(t) \leftarrow$ shuffle ($P(t-1)$)
   2. FOR i $\leftarrow$ 0 TO $\mu/2$ – 1 DO
      1. DCA ($\mu$)
      2. OD
   3. OD
Deterministic Crowding: Algorithm (2)

DCA ():
\[ P_1 \leftarrow a_{2i+1}(t) \]
\[ P_2 \leftarrow a_{2i+2}(t) \]
\{C_1, C_2\} \leftarrow \text{recombine } (P_1, P_2)
\[ C_1' \leftarrow \text{mutate } (C_1) \]
\[ C_2' \leftarrow \text{mutate } (C_2) \]
IF \[d(P_1, C_1') + d(P_2, C_2')\] <= \[d(P_1, C_2') + d(P_2, C_1')\] THEN
   IF \( f(C_1') > f(P_1) \) THEN \( a_{2i+1}(t) \leftarrow C_1' \) FI
   IF \( f(C_2') > f(P_2) \) THEN \( a_{2i+2}(t) \leftarrow C_2' \) FI

ELSE
   IF \( f(C_2') > f(P_1) \) THEN \( a_{2i+1}(t) \leftarrow C_2' \) FI
   IF \( f(C_1') > f(P_2) \) THEN \( a_{2i+2}(t) \leftarrow C_1' \) FI

FI
Deterministic Crowding: Discussion

- Capable of niching, i.e., locating and maintaining multiple peaks
- Minimal replacement error
- Few parameters to tune
- Fast, no distance calculations needed
- **Population size is important. Should be large if unsure**
- **Should use full crossover, i.e. crossover rate = 1.0**
- The following appears to all crowding methods:
  - Unlike sharing methods, crowding methods do not allocate individuals proportional to peak fitness. Instead, the number of individuals congregating about a peak is largely determined by the size of that peak’s basin of attraction under crossover
  - Similarity can be measured at either genotypic or phenotypic level
Speciation (in a narrow sense)

- In a broad sense, niching and speciation can be regarded as the same thing.
- In a narrow sense, the main emphasis of niching is on distributing individuals among different peaks, not finding exact optima efficiently. **Speciation** focuses on search within each peak.
- A speciation method **restricts mating** to similar individuals and discourages mating of individuals from different peaks (species).
- In order to apply speciation techniques properly, individuals representing each peak must be found first. Speciation techniques cannot be used independently.
- Niching and speciation are complementary.
- Similarity can be measured at either genotypic or phenotypic level.
Mating Restriction

- Use tags: each individual consists of a tag and a functional string

<table>
<thead>
<tr>
<th>template</th>
<th>tag</th>
<th>functional string</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1#0</td>
<td>1010</td>
<td>1010 …… 101</td>
</tr>
</tbody>
</table>

- Tag participate in (crossover and) mutation, but not fitness evaluation
- Templates can also be used
- Has been shown to be effective for multimodal function optimization
- **Only individuals with the same tag are allowed to mate**

- Use distance: either in the genotypic or phenotypic space

Define a threshold parameter $\sigma_{\text{mate}}$

- **Two individuals are allowed to mate only when their distance is smaller than $\sigma_{\text{mate}}$**

- GAs with niching and mating restriction were found to better distribute the population across the peaks than GAs with sharing alone

Mating restriction always applies during recombination