Lecture 15 of 42

Genetic and Evolutionary Computation 1 of 3:
The Simple Genetic Algorithm

Friday, 16 February 2007

William H. Hsu
Department of Computing and Information Sciences, KSU
http://www.kddresearch.org
http://www.cis.ksu.edu/~bhsu

Readings:
Sections 9.1-9.4, Mitchell
Chapter 1, Sections 6.1-6.5, Goldberg
Section 3.3.4, Shavlik and Dietterich (Booker, Goldberg, Holland)

Lecture Outline

• Readings
  – Sections 9.1-9.4, Mitchell
  – Suggested: Chapter 1, Sections 6.1-6.5, Goldberg
• Evolutionary Computation
  – Biological motivation: process of natural selection
  – Framework for search, optimization, and learning
• Prototypical (Simple) Genetic Algorithm
  – Components: selection, crossover, mutation
  – Representing hypotheses as individuals in GAs
• An Example: GA-Based Inductive Learning (GABIL)
• GA Building Blocks (aka Schemas)
• Taking Stock (Course Review): Where We Are, Where We’re Going
**Simple Genetic Algorithm (SGA)**

- Algorithm *Simple-Genetic-Algorithm (Fitness, Fitness-Threshold, p, r, m)*
  
  // p: population size; r: replacement rate (aka generation gap width), m: string size
  
  - P ← p random hypotheses // initialize population
  - FOR each h in P DO \( f[h] \leftarrow \text{Fitness}(h) \) // evaluate Fitness: hypothesis \( \rightarrow \) R
  - WHILE (Max(\( f \)) < Fitness-Threshold) DO
    - 1. Select: Probabilistically select (1 - r)/p members of P to add to \( P_{S} \)
    - 2. Crossover: Probabilistically select \((r \cdot p)/2\) pairs of hypotheses from P
      - FOR each pair \(<h_{1}, h_{2}>\) DO
        - \( PS \leftarrow \text{Crossover}(<h_{1}, h_{2}>) \) // \( PS[t+1] = P_{S}[t] + <\text{offspring}_{1}, \text{offspring}_{2}>\)
    - 3. Mutate: Invert a randomly selected bit in \( m \cdot p \) random members of \( PS \)
    - 4. Update: \( P \leftarrow P_{S} \)
    - 5. Evaluate: FOR each \( h \) in P DO \( f[h] \leftarrow \text{Fitness}(h) \)
  - RETURN the hypothesis \( h \) in P that has maximum fitness \( f[h] \)

**GA-Based Inductive Learning (GABIL)**

- GABIL System [Dejong *et al.*, 1993]
  
  - Given: concept learning problem and examples
  - Learn: disjunctive set of propositional rules
  - Goal: results competitive with those for current decision tree learning algorithms (e.g., C4.5)

- Fitness Function: \( \text{Fitness}(h) = (\text{Correct}(h))^{2} \)

- Representation
  
  - Rules: IF \( a_{1} = T \land a_{2} = F \) THEN \( c = T \); IF \( a_{2} = T \) THEN \( c = F \)
  - Bit string encoding: \( a_{1}[10] \cdot a_{2}[01] \cdot c[1] \cdot a_{1}[11] \cdot a_{2}[10] \cdot c[0] = 1001111100 \)

- Genetic Operators
  
  - Want variable-length rule sets
  - Want only well-formed bit string hypotheses
Crossover: Variable-Length Bit Strings

- **Basic Representation**
  - Start with
    
    \[
    \begin{array}{cccc}
    a_1 & a_2 & c & a_1 & a_2 & c \\
    h_1 & 1 & 0 & 0 & 1 & 1 \\
    h_2 & 0 & 1 & 1 & 1 & 0 & 0 \\
    \end{array}
    \]
  - Idea: allow crossover to produce variable-length offspring

- **Procedure**
  - 1. Choose crossover points for \( h_1 \), e.g., after bits 1, 8
  - 2. Now restrict crossover points in \( h_2 \) to those that produce bitstrings with well-defined semantics, e.g., \(<1, 3>, <1, 8>, <6, 8>\)

- **Example**
  - Suppose we choose \(<1, 3>\)
  - Result
    
    \[
    \begin{array}{cccccc}
    a_1 & a_2 & c & a_1 & a_2 & c \\
    h_3 & 1 & 1 & 0 & 1 & 1 \\
    h_4 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
    \end{array}
    \]

GABIL Extensions

- **New Genetic Operators**
  - Applied probabilistically
    - 1. **AddAlternative**: generalize constraint on \( a_i \) by changing a 0 to a 1
    - 2. **DropCondition**: generalize constraint on \( a_i \) by changing every 0 to a 1

- **New Field**
  - Add fields to bit string to decide whether to allow the above operators
    
    \[
    \begin{array}{ccccccc}
    a_1 & a_2 & c & a_1 & a_2 & c & AA & DC \\
    01 & 11 & 0 & 10 & 01 & 0 & 1 & 0 \\
    \end{array}
    \]
  - So now the learning strategy also evolves!
  - aka genetic wrapper
GABIL Results

• Classification Accuracy
  – Compared to symbolic rule/tree learning methods
    • C4.5 [Quinlan, 1993]
    • ID5R
    • AQ14 [Michalski, 1986]
  – Performance of GABIL comparable
    • Average performance on a set of 12 synthetic problems: 92.1% test accuracy
    • Symbolic learning methods ranged from 91.2% to 96.6%

• Effect of Generalization Operators
  – Result above is for GABIL without AA and DC
  – Average test set accuracy on 12 synthetic problems with AA and DC: 95.2%

Building Blocks
(Schemas)

• Problem
  – How to characterize evolution of population in GA?
  – Goal
    • Identify basic building block of GAs
    • Describe family of individuals

• Definition: Schema
  – String containing 0, 1, * (“don’t care”)
  – Typical schema: 10**0*
  – Instances of above schema: 101101, 100000, ...

• Solution Approach
  – Characterize population by number of instances representing each possible schema
  – $m(s, t)$ = number of instances of schema $s$ in population at time $t$
Source: Kansas State University

**Selection and Building Blocks**

- **Restricted Case: Selection Only**
  - \( \bar{f}(t) \) = average fitness of population at time \( t \)
  - \( m(s, t) \) = number of instances of schema \( s \) in population at time \( t \)
  - \( \bar{u}(s, t) \) = average fitness of instances of schema \( s \) at time \( t \)

- **Quantities of Interest**
  - Probability of selecting \( h \) in one selection step
    \[
    P(h) = \frac{f(h)}{\sum_{h_i} f(h_i)}
    \]
  - Probability of selecting an instance of \( s \) in one selection step
    \[
    P(h \in s) = \sum_{h \sim h_i} \frac{f(h)}{n \cdot f(t)} = \frac{\bar{u}(s, t)}{n \cdot f(t)} \cdot m(s, t)
    \]
  - Expected number of instances of \( s \) after \( n \) selections
    \[
    E[m(s, t + 1)] = \frac{\bar{u}(s, t)}{f(t)} \cdot m(s, t)
    \]

**Schema Theorem**

- **Theorem**
  \[
  E[m(s, t + 1)] \geq \bar{u}(s, t) \cdot m(s, t) \cdot \frac{1 - p_c}{1 - \left(1 - \frac{d}{l}ight)} \cdot \left(1 - p_m\right)^{\alpha(s)}
  \]

- \( m(s, t) \) = number of instances of schema \( s \) in population at time \( t \)
- \( f(t) \) = average fitness of population at time \( t \)
- \( \bar{u}(s, t) \) = average fitness of instances of schema \( s \) at time \( t \)
- \( p_c \) = probability of single point crossover operator
- \( p_m \) = probability of mutation operator
- \( l \) = length of individual bit strings
- \( \alpha(s) \) = number of defined (non "*"*) bits in \( s \)
- \( d(s) \) = distance between rightmost, leftmost defined bits in \( s \)

- **Intuitive Meaning**
  - “The expected number of instances of a schema in the population tends toward its relative fitness”
  - A fundamental theorem of GA analysis and design
Terminology

- **Evolutionary Computation (EC):** Models Based on Natural Selection
- **Genetic Algorithm (GA) Concepts**
  - **Individual:** single entity of model (corresponds to hypothesis)
  - **Population:** collection of entities in competition for survival
  - **Generation:** single application of selection and crossover operations
  - **Schema aka building block:** descriptor of GA population (e.g., 10**0**)
  - **Schema theorem:** representation of schema proportional to its relative fitness
- **Simple Genetic Algorithm (SGA) Steps**
  - **Selection**
    - Proportionate reproduction (aka roulette wheel): \( P(\text{individual}) \propto f(\text{individual}) \)
    - **Tournament:** let individuals compete in pairs or tuples; eliminate unfit ones
  - **Crossover**
    - Single-point: 11101001000 × 00001010101 → { 11101010101, 00001001000 }
    - Two-point: 11101001000 × 00001010101 → { 11001011000, 00101000101 }
    - Uniform: 11101001000 × 00001010101 → { 10001000100, 01101011001 }
  - **Mutation:** single-point (“bit flip”), multi-point
- **Schema Theorem:** Propagation of Building Blocks

Summary Points

- **Evolutionary Computation**
  - **Motivation:** process of natural selection
    - Limited population; individuals compete for membership
    - Method for parallelizing and stochastic search
  - **Framework for problem solving:** search, optimization, learning
- **Prototypical (Simple) Genetic Algorithm (GA)**
  - **Steps**
    - Selection: reproduce individuals probabilistically, in proportion to fitness
    - Crossover: generate new individuals probabilistically, from pairs of “parents”
    - Mutation: modify structure of individual randomly
  - **How to represent hypotheses as individuals in GAs**
- **An Example:** GA-Based Inductive Learning (GABIL)
  - **Schema Theorem:** Propagation of Building Blocks
- **Next Lecture:** Genetic Programming, The Movie