An Introduction to Nature-inspired Computation

Kenneth De Jong
Computer Science Department
George Mason University
Introduction

• What is nature-inspired computation?

  – Two uses of the term:
    • Computational models of natural systems
      – Weather models, cosmos models, …
    • Problem-solving methods based on abstractions of natural processes
      – Simulated annealing, artificial neural networks, evolutionary computation, …
Introduction

• Distinction at goal level:
  – Understand natural systems via computational models
  vs
  – Solving difficult computational problems in a nature-inspired way
    (the focus of this lecture)
Introduction

• Interesting tension:
  – Increasing model fidelity generally decreases problem-solving power
  – Increasing problem-solving power generally decreases model fidelity
Introduction

• Standard CS:
  – Design algorithms for specific problems

• Nature-inspired CS:
  – Construct computational abstractions
  – Figure out what problems they’re good for
Introduction

• Example: Simulated Annealing

  – Natural process:
    • Cooling from liquid to solid
    • End state depends on:
      – Initial temperature, rate of cooling, structure of the material, ..
    • E.g., different crystal structure end states
      – View end states as energy minima
      – Goal: anneal so as to achieve a desired energy minimum
Introduction

• Example: Simulated Annealing

  – Abstraction:
    • Stochastic local search
    • Accept better solution with a probability inversely proportional to “temperature”.
    • High temps: lots of exploration
    • Low temps: lots of exploitation
    • Annealing: dial down the temperature so as to end up at the global minimum (optimum) with high probability.
Introduction

• Example: Simulated Annealing
  – Useful for “black box” function optimization:
    • Relatively little know/assumed about the function.
    • Gain info by sampling (requesting values of f(x)).
    • Exploit accumulating information.
  – Exploration/exploitation tradeoffs
  – Local vs. global optima
Introduction

• Heuristics and meta-heuristics:
  – Heuristic: two uses of the term
    • Efficiency via problem-specific knowledge
      – E.g., greedy algorithm leads to optimum
    • Efficient approximations for hard problems
      – E.g., greedy algorithm produces acceptable solutions
Introduction

• Heuristics and meta-heuristics:
  
  – **Meta-heuristics**: confusing term
    
    • In general, meta-X means a higher-level discussion/analysis/study of X:
      
      – Meta-physics, meta-mathematics, ..
    
    • So, meta-heuristics would be the discussion/analysis/study of heuristics “from above”. 
Introduction

• Heuristics and meta-heuristics:
  – **Meta-heuristic:** several meanings
    • General strategy for constructing a heuristic
      – E.g., the general notion of a greedy algorithms
    • Heuristic template
      – E.g., simulated annealing
    • Robust optimization techniques
      – E.g., stochastic gradient descent
Introduction

• In general, nature-inspired computation is the study of nature-inspired meta-heuristics:
  – Interesting computational abstractions
  – Pseudo-code templates to be instantiated in problem-specific ways.
Introduction

• Examples of nature-inspired meta-heuristics:
  – Simulated annealing
  – Artificial neural networks
  – Evolutionary computation
  – Cellular automata
  – Lindenmeyer systems
  – Ant colony algorithms
  – Particle swarm algorithms
  – Immune system algorithms
  – …
Introduction

• No Free Lunch perspective:
  – All have strengths, weaknesses, biases

• Effective use:
  – Match problems with techniques
Introduction

• Problem classes:
  – Optimization:
    • Discrete
    • Continuous
    • Constrained
    • Multi-objective
Introduction

• Problem classes:
  – Search:
    • Different from optimization?
    • Structured spaces?
  – Machine Learning:
    • Different from optimization, search?
    • Executable objects?
An Introduction to Evolutionary Computation and its Applications

Kenneth De Jong

Computer Science Department
George Mason University
Evolutionary Computation: What is it?

- The use of evolutionary algorithms to solve difficult computational problems.
What is an Evolutionary Algorithm?

• An algorithm based on a Darwinian notion of an evolutionary system.

• Basic elements:
  – a population of “individuals”
  – a notion of “fitness”
  – a birth/death cycle biased by fitness
  – a notion of “inheritance”
A Simple Evolutionary Algorithm:

1. Randomly generate an initial population.

2. Do until some stopping criteria is met:
   
   Select individuals to be parents (biased by fitness).
   Produce offspring.
   Select individuals to die (biased by fitness).

   End Do.

3. Return a result.
Simple Example

- Individuals have 2 traits: \( <t_1, t_2> \)
- Each trait can vary from -25 to 20
- Unknown fitness surface defined over this trait space.
- Goal: find highly fit individuals quickly
Generation 0 (random)
Generation 5
Generation 10
Generation 20
Generation 30
Generation 0
Generation 5
Generation 10
Generation 20
Generation 30
Important Properties:

• Parallel, adaptive search

• Competition for resources:
  – survival (space)
  – reproduce (cpu)

• Stochastic

• Strong sense of fitness “optimization”
Key Issues:

• How to represent the space to be searched.

• How to generate “interesting” new samples from existing ones.

• How “greedy” should selection be.
Historical answers:

- **Evolution Strategies (ESs):**
  - developed by Rechenberg, Schwefel, etc. in 1960s.
  - focus: real-valued parameter optimization
  - individual: vector of real-valued parameters
  - reproduction: Gaussian “mutation” of parameters
  - $M$ parents, $N \gg M$ offspring
Historical answers:

• **Evolutionary Programming (EP):**
  – Developed by Fogel in 1960s
  – Goal: evolve intelligent behavior
  – Individuals: finite state machines
  – Offspring via mutation of FSMs
  – M parents, M offspring
Historical answers:

- **Genetic Algorithms (GAs):**
  - developed by Holland in 1960s
  - goal: robust, adaptive systems
  - used an internal “genetic” encoding of points
  - reproduction via mutation and recombination of the genetic code.
  - $M$ parents, $M$ offspring
Present Status:

• wide variety of evolutionary algorithms (EAs)

• wide variety of applications
  – optimization
  – search
  – learning, adaptation

• well-developed analysis
  – theoretical
  – experimental
Useful Optimization Properties:

- applicable to continuous, discrete, mixed optimization problems.
- no *a priori* assumptions about convexity, continuity, differentiability, etc.
- relatively insensitive to noise
- easy to parallelize
Problems other than function optimization?

- Heuristic search
- Machine learning
- Adaptation
- ...

How to apply EAs to problems?

• What elements are evolving?
  • Parameters
  • Structures
  • Code

• What are the mechanisms of change?
  • Mutation
  • Recombination
1. Tuning parameters:

- Identify key parameters of a solution

![Model parameters diagram]

**Individual 1**

\[ \begin{align*}
    a_1 & \quad a_2 & \quad a_3 & \quad a_4 & \quad a_5 & \quad a_6 & \quad a_7 \\
    a_{11} & \quad a_{12} & \quad a_{13} & \quad a_{14} & \quad a_{15} & \quad a_{16} & \quad a_{17}
\end{align*} \]

**Individual 2**

\[ \begin{align*}
    & a_{21} & \quad a_{22} & \quad a_{23} & \quad a_{24} & \quad a_{25} & \quad a_{26} & \quad a_{27}
\end{align*} \]

- \[ \cdots \]

**Individual n**

\[ \begin{align*}
    & a_{n1} & \quad a_{n2} & \quad a_{n3} & \quad a_{n4} & \quad a_{n5} & \quad a_{n6} & \quad a_{n7}
\end{align*} \]

**Genome:** parameter set

**Mutation:** alter individual parameters

**Recombination:** combine parameters from different parents
2. Evolving data structures:

- Modify internal structures that affect a solution

- Genome represents a graph, tree, ...
- Requires specialized recombination/mutation operators
3. Evolving executable code:

- Modify solution procedures

<table>
<thead>
<tr>
<th>Individual 1</th>
<th>IF a and b and c THEN A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IF d and e and c THEN B</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual 2</td>
<td>IF q and r THEN C</td>
</tr>
<tr>
<td></td>
<td>IF s and e and c THEN B</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual n</td>
<td>IF a and b and c THEN A</td>
</tr>
<tr>
<td></td>
<td>IF e and c THEN B</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Genome: entire program
- Recombination: combine rules from different parents
- Mutation: alter individual rules
Example: evolving behaviors for autonomous agents

• Evolutionary Robotics
  – High level behavioral rules evolved offline in simulation.
  – Evolved behaviors downloaded onto robots.
Collision Avoidance and Navigation

**Condition primitives (16):**
sonar[1]..sonar[6]; infrared[1]..infrared[6]
range/bearing to goal
current trans/steering rates

**Action primitives (2):**
translation rate (-4” to 12”/sec)
Steering rate (-30 to 30°/sec)

**Methodology:**
Each trial, robot placed at origin with random pose
Random placement of obstacles/goal, variable density
Noise (gaussian; false positives and negatives) added to sensors
Navigation and Collision Avoidance

Evolved Behaviors
Evolving Herding Behavior:

Shepherd must learn to herd sheep into pasture

**Sensors:**
- range/bearing to pasture
- range/bearing/heading of sheep

**Actions:**
- Trans rate (-4” to 10”/sec)
- Steering rate (-24° to 24°/sec)

**Methodology:**
“Sheep” programmed to move in random walk (small bias to go straight); but wants to avoid objects; will move in opposite direction. Error (gaussian) in range, bearing, and heading values. Assumed vision system would properly track sheep.
Herding
Evolved Behavior
Adapting to Partial System Failures

Monitor detects change in internal systems.

Robot performs simulation in virtual world to learn new behaviors.

Successful behaviors performed on-line.
Adapting to System Failures:
Multiple Interacting Agents

“Aerobics is down the hall - this is robotics.”
Micro Air Vehicles

- 6-12 inch wingspan, 50 - 100 grams gross mass
- Payloads (15 grams or less)
Multi-Agent Evolutionary Design

- Evolve decision rules for teams of cooperative agents

Task simulation
Multi-Agent Evolutionary Design

- RoboCup soccer:
  - www.cs.gmu.edu/~robotics
Alternative: Evolve Test Agents

• **How to validate complex systems?**
  – Prove theorems?
  – Hire test engineers?

• **Interesting alternative:**
  – Use EAs to search scenario spaces.
  – Scenario’s fitness related to the difficulties it creates for agents.
The Traditional Test Cycle

- **intelligent controller** under test
- **simulator**
- **fault scenario**
- Evaluate performance and create new fault scenario
Adaptive Testing

- Intelligent controller
- Fault scenario
- Evolutionary Algorithm
- Population of fault scenarios
- Simulator
- Under test

Current scenario

Fitness of scenario
Adaptive Testing of an Autonomous Underwater Vehicle

- Very high fidelity simulation of autonomous underwater vehicle.
- Modeled real vehicle.
Adaptive Testing of AUV:

- Controller failures found included:
  - Vehicle exceeds critical rate (e.g., maximum roll rate, oscillatory behavior)
  - Vehicle crashes into bottom
  - Vehicle unable to complete mission in time

- Some missions demonstrated failures that had not been observed before.

- Fault scenarios judged very interesting by controller and vehicle design teams.
Evaluating Network Security

- Assess ability of hackers to disrupt large networks

- Hacker’s goal: gain access to a large number of nodes on the network.
Evolutionary Testing

• Used Agent-based Network model

• Assessed hacker fitness via model simulation.

• Evolved hackers capable of exploiting security weaknesses.
Evolving Hacker Skills
New EC developments and directions:

- **Scaling up by exploiting parallelism:**
  - coarsely grained network models
    - Loosely coupled computer clusters (e.g., Beowulf)
    - EA: isolated islands with occasional migrations
  - finely grained diffusion models
    - Tightly coupled computer clusters (e.g., GPUs)
    - EA: continuous interaction in local neighborhoods
New EC developments and directions:

• Scaling up via co-evolutionary models:
  – competitive co-evolution (Rosin, ...)
    • improve performance via “arms race”
  – cooperative co-evolution (Potter, ...)
    • evolve subcomponents in parallel
New EC developments and directions:

- Scaling up by exploiting morphogenesis:
  - sophisticated genotype --> phenotype mappings
  - evolve plans rather than objects
  - applications to engineering design
    - E.g., Arciszewski, Gero, ...
New developments and directions:

• **Self-adaptive EAs:**
  – dynamically adapt to problem characteristics:
    • varying population size
    • varying selection pressure
    • varying representation
    • varying reproductive operators
  – goal: robust “black box” optimizer
New developments and directions:

- **Hybrid Systems:**
  - combine EAs with other techniques:
    - EAs and gradient methods
    - EAs and TABU search
    - EAs and ANNs
    - EAs and symbolic machine learning
New developments and directions:

- Time-varying environments:
  - fitness landscape changes during evolution
  - goal: adaptation, tracking
New developments and directions:

• Agent-oriented problems:
  – individuals more autonomous, active
  – fitness a function of other agents and environment-altering actions
  – E.g., the MASON toolkit
EA Generalizations:

• Meta-heuristics:
  – Heuristic for designing heuristics
    • E.g., hill climbing, greedy, …
  – Instantiate EA template in a problem-specific manner
EA Generalizations:

- **Nature-Inspired Computation:**
  - Early example: simulated annealing
  - Today: evolutionary algorithms
  - Others: particle swarm, ant colony, …
Conclusions:

• Powerful tool for your toolbox.

• Complements other techniques.

• Best viewed as a paradigm to be instantiated, guided by theory and practice.

• Success a function of particular instantiation.
More information:

• **Journals:**
  – Evolutionary Computation (MIT Press)
  – Trans. on Evolutionary Computation (IEEE)
  – Genetic Programming & Evolvable Hardware

• **Conferences:**
  – GECCO, CEC, PPSN, FOGA, …

• **Internet:**
  – [www.cs.gmu.edu/~eclab](http://www.cs.gmu.edu/~eclab)

• **My book:**
  – Evolutionary Computation: A Unified Approach
    • MIT Press, 2006