Outline

• **Variable Length Representation**, mostly used to solve modeling type of problems.
  – Parse Tree
  – Graph
  – Finite State Machine

• Unlike fixed-length representation, both contents and structure of a variable length structure are evolved.
Data Modeling

• In many scientific and engineering fields, one important task is to find significant and interesting patterns (model) from data that have been collected via experiment or computer simulation.
• The model can then be used to forecast the future from the same kind of data.

Data Modeling - Continue

• This modeling task can be formulated as a search problem.
  – search for a model explaining the data;
  – search for prediction rules;
• These models can be represented in many different data structures:
  – Parse tree; Graph; Finite state machines etc
Parse Tree

- Most of the initial GP work evolves parse tree models.
- A parse tree represents a LISP program (symbolic expression) that takes inputs and generates outputs.
- The internal nodes of a parse tree contain functions while the leaf nodes contain terminals.
- Both structure and contents of a parse tree evolve.

![Parse Tree Diagram]

Closure Requirement

- Each function in the function set $F$ should be able to take all values that it may encounter as input.
- $n$: number; $b$: boolean

![Closure Requirement Diagram]
Strongly Typed GP

- Aims: relax the closure requirement of the original GP and enrich the representation.[Montana, 1995]
- Method:
  - Specify the types of terminals and the types of function arguments and return values.
  - Type check all parse trees during parse tree creation and evolution using genetic operation.

Regression Analysis

- It models the relationship between a dependent variable Y and one or more independent variables X.
  - A regression model f relates Y to X: \( Y = f(X) \)
- To carry out regression analysis, the form of the function f must be specified. Sometimes the form of this function is based on knowledge about the relationship between Y and X that does not rely on the data. If no such knowledge is available, a convenient form for f is used.
Example

• Given some points in $\mathbb{R}^2$, $(x_1, y_1)$, … , $(x_n, y_n)$
• Find a function $f(x)$ s.t. $\forall i = 1, \ldots, n : f(x_i) = y_i$

Regression using GP

• Each parse tree represents a function $f$
• There is no need to specify the form of the regression model $f$.
• However, a modeler (you) has to provide:
  – a function set : e.g. $+, -, *, /, \sin, \exp, \sqrt{\,}$
  – a terminal set : e.g. $x_1$ … $x_n$, plus constants
• The function and terminal sets have to be sufficient (Koza, 1992) to express a model solution.
• Similar to all regression methods, the objective is minimizing sum squared errors (or similar measures):
\[
\sum_{i=1}^{n}(y_i - \hat{y}_i)^2
\]
Cart Centering Problem

- Objective: find a time-optimal bang-bang control strategy (push force in a positive or negative direction) to center a cart on a one-dimensional frictionless track.
- When the mass of the cart is 2.0 kilograms and the force is 1 newton, the known time-optimal strategy is:
  - push positive direction if \(-x > v^2 \text{sign}(v)\)
  - otherwise, push negative direction
  - \(x\) and \(v\) are position and velocity of the cart
  - \(\text{sign}(v)\) returns +1 if \(v\) is positive, -1 if \(v\) is negative

GP Modeling

- Variables:
  - independent variables: position (\(x\)) and velocity (\(v\)) of the cart.
  - dependent variable: +1 (push force to the right) or -1 (to the left)
- Function set: \{+, -, *, %, abs, gt\}
- Terminal set: \{x, v, rnd\}
- Search space: how many model candidates?
A GP evolved solution by Koza:

- It behaves exactly the same as
  \[-x > v^2 \text{sign}(v)\]
- But if you need to know the solution model in advance to design sufficient functions and terminals sets, why bother using GP to find the solution anyway?

Why Bother?

- All models (known or unknown) are approximation of the reality.
- In the parse tree representation (and many others), there is a many-to-one mapping between solution models and fitness.
- Evolutionary search exploits that property (neutrality) to optimize solution models.
- Evolved models may contain new insights that were previously unknown.
Why Bother? Seriously

- In real life, we are more likely to know something about the solution model but not the model; such knowledge can be integrated in designing the function and terminal sets.
- If there is already a known model, mostly likely it will used as a benchmark to evaluate against the quality of the new model.

Finite State Machines

- A FSM is specified by
  - Inputs $I$
  - States $S$
  - Outputs $O$
  - Transition function $\delta: S \times I \rightarrow S \times O$
  - Start state
  - End state

- A FSM can be used to
  - recognize a language
  - automatic controller
  - robot controller
Example I

• Task: evolve a FSM that predicts the next bit of an input streams of bit is 0 or 1.
• Inputs I: \{0,1\}
• Outputs O: \{0,1\}
• States: \{A, B\}
• Fixed-length

from “Evolutionary Computation for Modeling and Optimization”, Dan Ashlock

Example II

• A thermostat controller:
• Inputs I: \{hot, okay, cold\}
• Outputs O: \{air-conditioner, do-nothing, furnace\}
• States: \{ready, heating, cooling, just-heated, just-cooled\}
• Fixed length
FSM Controller

<table>
<thead>
<tr>
<th>Initial State: ready</th>
<th>make a transition</th>
<th>and respond with</th>
</tr>
</thead>
<tbody>
<tr>
<td>When current state</td>
<td>to state</td>
<td></td>
</tr>
<tr>
<td>and input are</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(hot, ready)</td>
<td>cooling</td>
<td>air-conditioner</td>
</tr>
<tr>
<td>(hot, heating)</td>
<td>just-heated</td>
<td>do-nothing</td>
</tr>
<tr>
<td>(hot, cooling)</td>
<td>cooling</td>
<td>air-conditioner</td>
</tr>
<tr>
<td>(hot, just-heated)</td>
<td>ready</td>
<td>do-nothing</td>
</tr>
<tr>
<td>(hot, just-cooled)</td>
<td>ready</td>
<td>do-nothing</td>
</tr>
<tr>
<td>(okay, ready)</td>
<td>ready</td>
<td>do-nothing</td>
</tr>
<tr>
<td>(okay, heating)</td>
<td>just-heated</td>
<td>do-nothing</td>
</tr>
<tr>
<td>(okay, cooling)</td>
<td>just-cooled</td>
<td>do-nothing</td>
</tr>
<tr>
<td>(okay, just-heated)</td>
<td>ready</td>
<td>do-nothing</td>
</tr>
<tr>
<td>(okay, just-cooled)</td>
<td>heating</td>
<td>furnace</td>
</tr>
<tr>
<td>(cold, ready)</td>
<td>heating</td>
<td>furnace</td>
</tr>
<tr>
<td>(cold, heating)</td>
<td>just-heated</td>
<td>do-nothing</td>
</tr>
<tr>
<td>(cold, cooling)</td>
<td>just-cooled</td>
<td>do-nothing</td>
</tr>
<tr>
<td>(cold, just-heated)</td>
<td>ready</td>
<td>do-nothing</td>
</tr>
<tr>
<td>(cold, just-cooled)</td>
<td>ready</td>
<td>do-nothing</td>
</tr>
</tbody>
</table>

from “Evolutionary Computation for Modeling and Optimization”, Dan Ashlock

Classroom Exercises I

• Design the function and terminal sets for GP to evolve a search algorithm.
  – Don’t forget about data type constraints
Classroom Exercise II

• Design the inputs I, states S, outputs O, start state and end state to evolve a FSM that recognizes the search algorithms generated by the GP in exercise I.
  – S-expressions use prefix notation, where the first element of every S-expression is an operator and all remaining elements are treated as data.
  – Examples: (* 4 (+ x 2))