Towards Competent Genetic Programming: What are the missing ingredients?

Moshe Looks, Washington University in St. Louis
Ben Goertzel, Novamente LLC
July 2006
Outline

- What is program learning?
- Why is program learning different?
- What new approach is being taken, and Why?
- What are some results & future directions?
Program Learning

- **Optimization**
  - Find a solution $s$ in $S$
  - Maximize/minimize $f(s)$
    - $f: S \rightarrow \mathbb{R}$

- **Program Learning**
  - Solutions encode executable *programs*
  - Execution maps programs to behaviors ($\text{exec}: P \rightarrow B$)
  - Find a program $p$ in $P$
  - Maximize/minimize $f(\text{exec}(p))$
    - $f: B \rightarrow \mathbb{R}$

- *For this to be useful, we need to make assumptions about exec*
Program Learning – Example

$n$-arry Boolean Formulae

- $\mathcal{P}$ is the set of Boolean AND/OR/NOT formulae over $n$ variables
- $\text{exec}$ maps formulae to length-$2^n$ bit-strings (truth tables)
- $f$ is Hamming distance to some prespecified truth-table
  - *It’s just a one-max!*

```
  or
   /\   /
  and  and
   |   |
  x y  not not
   |   |
  x y
```

$\text{exec} \begin{array}{c} \rightarrow \\ 1010 \end{array} \quad f \begin{array}{c} \rightarrow \\ |1010,1010| = 0 \end{array}$
Challenge & Opportunity

- Programs are hierarchical and open-ended
  - Smaller programs are easier to learn

- Execution is chaotic
  - small changes in the program may cause large changes in behavior
  - Even if \( f \) decomposable, \textit{exec} will not be!

- Many programs map to the same behavior

- Many different \( f \)s may have the same \textit{exec}
Challenge & Opportunity

- Hofstadter, 1985:

  1. Knob creation - discovering novel values to parameterize
  2. Knob twiddling - adjusting the values of existing parameters

Knob creation:
- For PMBs over fixed-length strings, a non-issue
- For PMBs over program-space, the central challenge!
Claim

- Compact problem decompositions rarely exist for:
  - generic representations:
    - E.g., trees
    - E.g., grammars
  - general expressions:
    - E.g., Boolean formulae
    - E.g., symbolic equations
    - E.g., finite automata
New Approach

- Assume that \texttt{exec} is known
  - Exploit known identities & transformations to simplify programs
  - Reduce the program search space

- Explicitly build a parameterized representation (good knobs)
  - Search a sub-space of general program-space
  - Chosen based on an existing population (or prior)
  - Provide a compact representation to the PBM, \textit{consisting of salient parameters to be tuned}
  - The representation may change as the population changes
Example – Boolean Formulae

- Heuristic reduction and normalization:
  - Flatten binary and/or to n-arry
  - Searching for contradictions / redundancies
    - Vertically (paths from the root to leaves)
    - Horizontally (across conjunctions and disjunctions)

- Incorporates “Elegant Normal Form”
  - Holman, 2001
Example – Boolean Formulae

\[ \text{and}(\text{and}(x_1 \; x_2) \; \text{or}(\text{and}(x_3 \; x_1) \; \text{and}(x_3 \; x_4))) \]
Example – Boolean Formulae

\[ \text{and(\text{and}(x_1, x_2) \text{ or } (\text{and}(x_3, x_1) \text{ and } (x_3, x_4)))} \]

- flatten:

\[ \text{and}(x_1, x_2 \text{ or } (\text{and}(x_3, x_1) \text{ and } (x_3, x_4))) \]
Example – Boolean Formulae

\[
\text{and(} \text{and(} x_1 \text{ } x_2 \text{)} \text{ } \text{or(} \text{and(} x_3 \text{ } x_1 \text{)} \text{ } \text{and(} x_3 \text{ } x_4 \text{)})\text{)}
\]
- flatten:
  \[
  \text{and(} x_1 \text{ } x_2 \text{ } \text{or(} \text{and(} x_3 \text{ } x_1 \text{)} \text{ } \text{and(} x_3 \text{ } x_4 \text{)})\text{)}
  \]
- search vertically
  \[
  \text{and(} x_1 \text{ } x_2 \text{ } \text{or(} x_3 \text{ } \text{and(} x_3 \text{ } x_4 \text{)})\text{)}
  \]
Example – Boolean Formulae

\[
\text{and} (\text{and}(x_1 \ x_2) \ \text{or} (\text{and}(x_3 \ x_1) \ \text{and}(x_3 \ x_4)))
\]

- flatten:
  \[
  \text{and}(x_1 \ x_2 \ \text{or}(\text{and}(x_3 \ x_1) \ \text{and}(x_3 \ x_4)))
  \]

- search vertically
  \[
  \text{and}(x_1 \ x_2 \ \text{or}(x_3 \ \text{and}(x_3 \ x_4)))
  \]

- search horizontally
  \[
  \text{and}(x_1 \ x_2 \ x_3)
  \]
Example – Boolean Formulae

- But does it work?
- How can we test it independently of learning?

Consider the effects on tree similarity:
- Syntactic similarity (based on dynamic programming tree-alignment)
- Semantic similarity (based on Hamming-distance)
- We can compute these pairwise and look for a ridge
Before Simplification + ENF
After Simplification + ENF
How can we use this?

- Embed reduction engine in GP?
  - Mixed results in the literature

- Hypothesis
  - Reduction reduces syntactic diversity
  - GP not designed to effectively search for linkages in a compact representation
  - cannot effectively exploit one, even if it exists
First Approach

- Align trees via pairwise dynamic programming, and model with BOA

- Too slow – each alignment is at least $O(n^2)$!

- Too much dissimilarity among trees
  - even after selection & normalization

- Even with a large population insufficient sample density
  - because of limited overlap
Second Approach

- Focus sampling on limited regions of the space

- Generate programs with *known* similarities, obviating the need for expensive alignment

- Incrementally shift the modeling space (and sample new points) as better solutions are found
Meta-Optimizing Semantic Evolutionary Search (MOSES)

1. Begin with an initially empty prototype tree
2. Create “knobs” representing meaningful variations on the prototype tree
3. Randomly sample a population of nearby trees by turning the knobs
4. Select promising trees from the population for modeling
5. Model the trees’ corresponding knob-settings (fixed-length vectors) using the hBOA (or some other PMB optimizer)
6. Sample the model to generate new knob-settings, apply the inverse transformation to convert them into trees, and integrate them into the population
7. Update the prototype tree(s) if necessary, and go to step 2 or 4.
Meta-Optimizing Semantic Evolutionary Search (MOSES)

- Advantages:
  - Similarity relations are known
    - Hamming distance on knob-space
  - Only need explore combinations of meaningful variations

- Disadvantages:
  - May be computationally expensive to manipulate sufficient knobs
  - Only searches a limited subspace at a time
  - Some details ad-hoc – more principled design & analysis needed
Results ~ Boolean Formulae

- AND/OR/NOT Formulae for n-Even-Parity and n-Odd-Parity are difficult to learn
  - Require $\Omega(n^2)$ nodes
    - (assuming XOR and $=$ are not available)
  - Size of any fixed-depth representation grows exponentially – no poly-sized CNF or DNF

*Extremely* common GP benchmark…
## Computational Effort

<table>
<thead>
<tr>
<th>Method</th>
<th>GP</th>
<th>EP</th>
<th>MOSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-Even-Parity</td>
<td>96,000</td>
<td>28,500</td>
<td>19,500</td>
</tr>
<tr>
<td>4-Even-Parity</td>
<td>384,000</td>
<td>181,500</td>
<td>53,000</td>
</tr>
<tr>
<td>5-Even-Parity</td>
<td>6,528,000</td>
<td>2,100,000</td>
<td>521,000</td>
</tr>
</tbody>
</table>
# Computational Effort

<table>
<thead>
<tr>
<th>Method</th>
<th>MOSES</th>
<th>MOSES without model-building</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-Even-Parity</td>
<td>19,500</td>
<td>16,284</td>
</tr>
<tr>
<td>4-Even-Parity</td>
<td>53,000</td>
<td>21,154</td>
</tr>
<tr>
<td>5-Even-Parity</td>
<td>521,000</td>
<td>1,200,810</td>
</tr>
</tbody>
</table>
Ongoing Work

- Make the system robust:
  - Population Sizing
  - Neutral Networks
  - Initial Sampling
Ongoing Work

- Population Size
  - A function of problem size
  - Problem size can change in program-learning
  - Can we adjust this dynamically?
Ongoing Work

- Neutral Networks (or Near-Neutral)

  - “Redundancy is necessarily neutral, but neutrality is not necessarily redundant”
    - Marc Toussaint, 2001

  - May need to reorganize structure of a problem before fitness can be improved

  - Should we prefer smaller representations? Or larger?
Ongoing Work

- Initial Sampling
  - Non-trivial for programs
  - How to generate initial / new search regions?
  - How to sample instances within a region?
Future Work

- Modularity
  - E.g., ADFs
  - Across problems?
  - An advantage of an EDA approach
Other Domains

- **Agent Control (e.g., Artificial Ant)**
  - “Indeed genetic programming and other search techniques do not perform enormously better than random search” – Landgon & Poli, “Why Ants are Hard”
  - AGI-SIM – agisim.sourceforge.net
  - Fetch, Tag, Block Stacking, …

- **Symbolic regression**
  - Normal form similar to Boolean case
  - Alternating levels of linear / nonlinear

- **List manipulation**

- **Open-ended computation**
  - Polynomial Time?
  - Primitive Recursive?
  - Turing-Complete?
Conclusions

- PMB optimization algorithms work well
  - When representation allows compact decomposition
  - Not generally the case for program-learning

- Applying PMB effectively to programs requires qualitatively new techniques - *the “meta”*
  - Exploit known semantics of program execution
  - Set up modeling *over* meaningful variations
  - Once in place, can build on existing methods