The Programming Game
An Alternative to GP for Expression Search

DAASE/COW Open Workshop
Tuesday 23rd April 2013

David R White
SICSA Research Fellow · University of Glasgow
A Confession
DAASE and Genetic Programming

An Alternative Program Search Method

Two Experiments

Wrap-Up
DAASE and Genetic Programming

An Alternative Program Search Method

Two Experiments

Wrap-Up
“There have been exciting recent breakthroughs in the use of genetic programming to re-design aspects of systems to fix bugs, to migrate to new platforms and languages and to optimise non-functional properties.”

Harman et al.,
Dynamic Adaptive Search Based Software Engineering,
ESEM 2012.
Genetic Programming as a Hyper-Heuristic

School of Computer Science and Information Technology
University of Nottingham
Jubilee Campus
NOTTINGHAM NG8 1BB, UK

Exploring Hyper-heuristic Methodologies with Genetic Programming

Edmund Burke, Mathew R. Hyde, Graham Kendall, Gabriela Ochoa, Eider Ozcan, and John R. Woodward

Abstract Hyper-heuristics represent a novel search methodology that is motivated by the goal of automating the process of selecting or combining simpler heuristics in order to solve hard computational search problems. An extension of the original hyper-heuristic idea is to generate new heuristics which are not currently known. These approaches operate on a search space of heuristics rather than directly on a search space of solutions to the underlying problem which is the case with most meta-heuristics implementations. In the majority of hyper-heuristic studies so far, a framework is provided with a set of human determined, and with good measures of performance, heuristics from which the purpose of this chapter is to discuss the Genetic Programming is the most widely used. The framework is presented including the steps needed to automate case studies, a literature review of related issues. Our aim is to convey the exciting potential of automating the heuristic design process.

A Genetic Programming Hyper-Heuristic Approach for Evolving 2-D Strip Packing Heuristics

Edmund K. Burke, Member IEEE, Matthew Hyde, Member IEEE, Graham Kendall, Member IEEE, and John Woodward

Abstract—We present a genetic programming (GP) system to evolve heuristics for the 2D strip packing problem. The evolved heuristics are constructive, and decide both which piece to pack next and where to place that piece, given the current partial solution. This paper contributes to a growing research area that represents a paradigm shift in search methodologies. Instead of using evolutionary computation to search a space of solutions, we employ it to search a space of heuristics for the problem. A key motivation is to investigate methods to automate the heuristic design process. It has been stated in the literature that humans are very good at identifying good heuristics for solving problems. However, the task of intelligently searching through all of the potential combinations of these heuristics is better suited to a computer. With such tools at their disposal, heuristic designers are then free to focus more of their time to the creative process of determining good constraints, which are measured as the distance from the base of the height to the piece edge furthest from the base. This problem is known to be NP hard [2], and has many industrial applications as there are many situations where a series of rectangles of different sizes must be cut from a sheet of material (for example, glass or metal) while minimizing waste.

Indeed, many industrial problems are not limited to just rectangles (for example textiles, leather, etc.) and this presents another challenging problem [3]. These are many other types of cutting and packing problems in one, two and three dimensions. A typology of these problems is presented by Wachter et al. in [4]. As well as their dimensionality, the problems are
The Demands of DAASE

- Dynamic, online, run-time optimisation.
- Continuous adaptation.
Anytime Algorithms

Evolutionary Algorithms are often viewed as anytime algorithms:

![Graph showing Quality vs. Time for Algorithm 1 and Algorithm 2]
Anytime Algorithms

Evolutionary Algorithms are often viewed as anytime algorithms:

... but I would argue that they are somewhat imperfect anytime algorithms. Especially GP.
Why GP is not so Anytime

- Bloat
- Parameter Setting
- Difficulty of Allocating Computational Budget
- Notions of Progress and Coverage
Why GP is not so Anytime

- Bloat
- Parameter Setting
- Difficulty of Allocating Computational Budget
- Notions of Progress and Coverage

How well do we understand a GP search? How can we hope to control it? (“Insight”)

### Good Theft vs. Bad Theft

<table>
<thead>
<tr>
<th>Honor</th>
<th>Degrade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Study</td>
<td>Skim</td>
</tr>
<tr>
<td>Steal from Many</td>
<td>Steal from One</td>
</tr>
<tr>
<td>Credit</td>
<td>Plagiarize</td>
</tr>
<tr>
<td>Transform</td>
<td>Imitate</td>
</tr>
<tr>
<td>Remix</td>
<td>Rip Off</td>
</tr>
</tbody>
</table>

Steal from Artificial Intelligence Research
DAASE and Genetic Programming

An Alternative Program Search Method

Two Experiments

Wrap-Up
Monte Carlo Tree Search
29 possible moves for White here.
Programming is a One-Player Game
Tristan Cazenave’s Work

Nested Monte-Carlo Expression Discovery, Cazenave, ECAI 2010.

A Stack Machine

Stack using Reverse Polish notation.

Each atom added is a move through the game tree.
Building the Game Tree

1. Selection
2. Expansion
3. Sampling
4. Update
```python
def uct(max_evals, terms, nonterms, ucb_constant, max_nodes, scoref):
    root = TreeNode(None, terms, nonterms, None, ucb_constant, 1, 0, max_nodes)
    for i in xrange(max_evals):
        if root.explored:
            break
        stack = ExpressionStack(max_nodes)
        leaf = tree_policy(root, terms, nonterms, ucb_constant, stack, max_nodes)
        score = playout(stack, terms, nonterms, scoref)
        backup(leaf, score)
    return root

def tree_policy(node, terms, nonterms, ucb_constant, stack, max_nodes):
    while stack.leaves > 0:
        if not node.all_atoms_tried():
            new_child = expand(node, stack, terms, nonterms, ucb_constant, max_nodes)
            stack.push(new_child.node_atom)
            if stack.leaves == 0:
                new_child.explored = True
                new_child.possible_atoms = []
                return new_child
        else:
            node = best_child(node)
            stack.push(node.node_atom)
    return node
```

Python Implementation
def expand(node, stack, terms, nonterms, ucb_constant, expr_size, max_nodes):
    atom = node.next_atom()
    e_leaves = stack.leaves + atom.arity - 1
    e_size = len(stack.expression) + 1
    c = TreeNode(atom, terms, nonterms, node, ucb_constant, e_leaves, e_size, max_nodes)
    node.add_child(c)
    return c

def backup(node, score):
    while node is not None:
        node.visits = node.visits + 1
        node.sum_scores = node.sum_scores + score
        if node.all_atoms_tried():
            done = True
            for c in node.children:
                done = done and c.explored
            node.explored = done
        node = node.parent
A Simple Example

Symbolic regression with the language \{+,* ,a, b\}. 
Example Game Tree Construction

Step 1
[null]

Step 2
[null], 1, 0.1
[+, 1, 0.1
[null], 1, 0.1

Step 3
[null], 2, 0.4
[+, 1, 0.1
[*, 1, 0.3
[null], 3, 0.5

Step 4
[null], 3, 0.5
[+, 1, 0.1
[*, 1, 0.3
[null], 3, 0.5

Step 5
[null], 4, 0.5
[+, 1, 0.1
[*, 1, 0.3
[a], 1, 0.1

Step 6
[null], 5, 1.0
[+, 1, 0.1
[*, 2, 0.8
[a], 1, 0.1

Score = 0.1
Score = 0.3
Score = 0
Score = 0.1
Score = 0
Score = 0.5
Choose child with highest UCT score.

\[ \frac{S_c}{n_c} + K \sqrt{\frac{2 \ln n_c}{n_p}} \]

- \( S_c \): total score for playouts involving this node.
- \( n_c \): number of visits to this node.
- \( n_p \): number of visits to the parent of this node.
- \( K \): constant
DAASE and Genetic Programming

An Alternative Program Search Method

Two Experiments

Wrap-Up
The Target Problem

Find an equation using the numbers \{1 \ldots 10\} exactly once and the arithmetic operators \(+,-,\div,*\) so that the result is as close to 737 as possible.
Target Problem: Results

Comparing Median Best Fitness on the Target Problem

- GP
- Nested
- UCT

Evaluations (log scale)

Fitness Score

1e+01 1e+03 1e+05
Prime Generation

Find an equation that generates unique prime numbers when fed with the natural numbers as input.

The function set is +,-,*,/ and the terminal set is \{1\ldots10\} and all the prime numbers under 100.
Prime Problem: Results

Comparing Median Best Fitness on the Prime Problem

- GP
- Nested
- UCT

Evaluations (log scale)

Fitness Score

0 10 20 30 40 50
Advantages of MCTS

Concise Solutions.

Game Tree is Human-Readable.

Parallelisation.
Relevant Previous Work

Real-time Games

Scheduling Problems
(includes a comparison to EAs)

Feature Selection
Feature Selection as a One-Player Game, Gaudel and Sebag, ML 2010.
DAASE and Genetic Programming

An Alternative Program Search Method

Two Experiments

Wrap-Up
What next?

A better paper!

Further adapting MCTS for program search. e.g. use of grammars to introduce typing.

Application to *challenging* problems.
Acknowledgements

Juan E. Tapiador

Tristan Cazenave
Further Reading

Highly recommended:

A Survey of Monte Carlo Tree Search Methods, Browne et al., IEEE Trans. on Computational Intelligence and AI in Games, 2012.