The Programming Game: Evaluating MCTS as an Alternative to GP for Symbolic Regression

David R. White
School of Computing Science
University of Glasgow, UK
david.r.white@glasgow.ac.uk

Shin Yoo
Dept. of Computer Science
University College London, UK
shin.yoo@ucl.ac.uk

Jeremy Singer
School of Computing Science
University of Glasgow, UK
jeremy.singer@glasgow.ac.uk

ABSTRACT

We develop previous work by Cazenave, applying Monte Carlo Tree Search (MCTS) to programming. We compare MCTS to Genetic Programming (GP) and find that MCTS is competitive with GP for standard benchmarks.

Categories and Subject Descriptors

SD I.2.8 [**Artificial Intelligence**]: Heuristic methods

Keywords

Genetic Programming, Monte Carlo Tree Search

1. INTRODUCTION

Monte Carlo Tree Search (MCTS) [2] is a family of heuristic search algorithms that explore game trees, most successfully in playing the game of Go [5]. MCTS relies on a combination of heuristics and stochastic playouts to assess the likely result of taking an action from the current state: beyond a certain distance of lookahead, the consequences of a given action are estimated by randomly playing the remaining moves and assessing the final state. Although originally designed to play games, Cazenave [4] demonstrates that MCTS can be used for symbolic regression. We extend Cazenave's approach and make comparisons with Genetic Programming (GP) on four standard benchmarks [7].

GP maintains a population, making it a global search algorithm unlike MCTS, and has been applied more widely than MCTS. However, MCTS offers some advantages: it is inherently resistant to bloat, has a sound mathematical underpinning, and the trade-off between exploration and exploitation can be explicitly controlled by a single parameter.

2. MCTS

We implement two MCTS algorithms: Upper Confidence Bound for Trees (UCT) [6] and Nested Search [3]. We apply them to incrementally build an expression. At each step, a function (terminal or nonterminal) is pushed onto a stack representation of the expression. Each node in the game tree branches according to the functions that may be used at that point in the abstract syntax tree.

UCT repeatedly descends the game tree from the root to a leaf node, selecting the child to follow (and hence the function to push onto the stack) that maximises the following:

$$\frac{S_c}{n_c} + K\sqrt{\frac{2\ ln\ n_c}{n_p}} \quad (1)$$

S_c is the sum of scores for the child, n_c and n_p the number of times the child and its parent have been visited, and K is the UCB constant that tunes the balance between exploration and exploitation. Once a leaf node is reached that has yet to be fully expanded, a new node is added to the search tree representing a function pushed onto the stack. A playout is then made from the new node, completing the program using stochastically chosen functions. The program is evaluated and its score used to update the search tree.

Nested Tree Search creates an empty stack to which functions are added based on a recursive search. After one recursive search for each possible function, the function resulting in the best score is pushed permanently onto the stack. The process is repeated until the stack expression is complete, using a playout below a given depth. Expression size is limited through the functions considered for insertion. In contrast, UCT incrementally builds the game tree.

3. EXPERIMENTS

We compare UCT, Nested Search and traditional GP. We provide our code online [8]. We use the version of ECJ from GPBenchmarks.org [7]. We instantiate Ephemeral Random Constants when adding them to a stack. We use four benchmarks from gpbenchmarks.org, given in Table 1.

Parameter Settings We limit fitness evaluations to a maximum of E, with an upper limit of 2^{20}. ECJ parameters ensure $p * g = E$, where p is population size, and g the number of generations. We measure performance using the test set fitness f of the best individual, normalised to $\frac{1}{1+f}$.

We set GP population size p to 1024, the probability of crossover to 0.9 and mutation to 0.1. Other settings are left at ECJ defaults. Generations are limited to E/p. For MCTS, the maximum expression length was 35 as per Cazenave's work. Nested "level", which determines the depth to which the expression space will be enumerated, was 4. For UCT the constant K, which determines the balance between exploration and exploitation to the popular value $1/\sqrt{2}$ [6].

GECCO '15 July 11-15, 2015, Madrid, Spain

ACM ISBN 978-1-4503-3488-4/15/07.

DOI: http://dx.doi.org/10.1145/2739482.2764655

Benchmark	Target Function	Function Set
Keijzer6	$\sum_{i=1}^{x} 1/i$	ADD, MULT, INV, NEG, SQRT, ERC
Nguyen7	$ln(x+1)+ln(x^2+1)$	ADD, MULT, DIV, SUB, SIN, COS, LOG, EXP
Pagie1	$\frac{1}{1+x^{-4}}+\frac{1}{1+y^{-4}}$	ADD, MULT, DIV, SUB, SIN, COS, LOG, EXP
Vladislavleva4	$\frac{10}{5+\sum_{i=1}^{5}(x_i-3)^2}$	ADD, MULT, DIV, SUB, SQUARE, ERC_A, ERC_B, ERC_C

Table 1: Benchmark symbolic regression problems

Benchmark	$p_{E>U}$	$p_{E>N}$	$p_{U<N}$
Keijzer6	**< 1e-5**	0.32	**< 1e-5**
Vladislavleva4	**< 1e-5**	**< 1e-5**	**< 1e-5**
Pagie1	**< 1e-5**	**< 1e-5**	**< 1e-5**
Nguyen7	**< 1e-5**	0.16	**< 1e-5**

Table 2: Wilcoxon rank-sum p-values. $p_{A>B}$ is the P-Value for the alternative hypothesis that Algorithm A outperforms B. E is ECJ, U is UCT, N is Nested.

4. RESULTS

Table 2 gives the results of Mann-Whitney-Wilcoxon significance tests for 30 runs of 2^{20} evaluations. UCT performs poorly on all benchmarks, possibly because it assumes normally distributed rewards in the game tree, which may not be the case for expression spaces. Nested Search performs as well as GP for two benchmarks, Keijzer and Nguyen, but it is outperformed by GP on the other two. See Figures 1 and 2 for representative examples. We conjecture GP's superior performance is due to our naive playout algorithm,; the final length of the expression is determined by the ratio of terminals to non-terminals in the function set.

The variance of GP results was greater than MCTS. For the Keijzer benchmark the median score even decreases at higher numbers of evaluations, an observation that can only be explained by stochastic variation. Given that Nested Search contains a large element of systematic recursive exploration as well as randomised playouts, it is likely that the difference in variance is due to its more systematic approach.

5. CONCLUSIONS

Our naive implementation of UCT performs poorly compared to GP and Nested Search, but Nested Search is competitive with GP on two problems. We recommend a further investigation of MCTS parameters, and alternative playout functions. Also, we search expressions in a depth-first fashion; breadth-first could be used, so that the functions highest in the implicit expression tree would be determined first. We conjecture that *programming is a one-player game*.

Thank you to: Juan E. Tapiador, Marc Schoenauer, Francis Maes, Sean Luke, and the GPBenchmarks Team.

6. REFERENCES

[1] B. Bouzy and B. Helmstetter. Developments On Monte Carlo Go. In *Advances in Computer Games 10*, 2003.

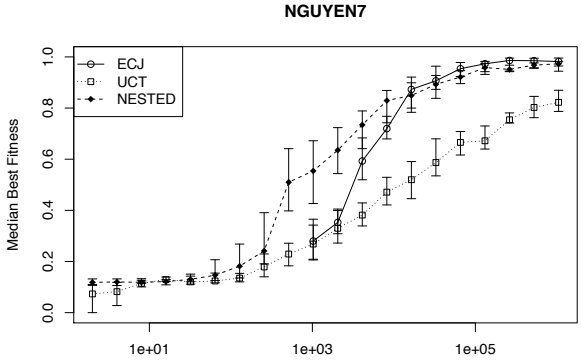

Figure 1: Results for the Nguyen7 Benchmark

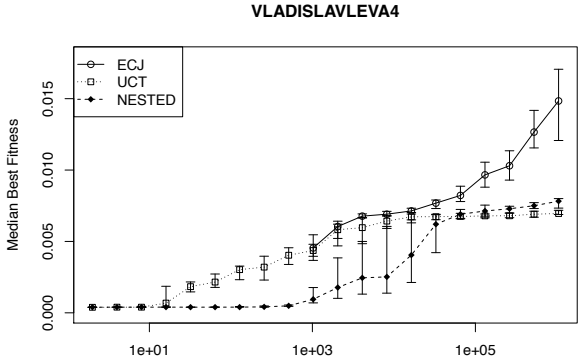

Figure 2: Results for the Vladislavleva4 Benchmark

[2] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A Survey of Monte Carlo Tree Search Methods. *Computational Intelligence and AI in Games, IEEE Trans.*, 4(1):1 –43, 2012.

[3] T. Cazenave. Nested Monte-Carlo Search. *IJCAI*, 9:456–461, 2009.

[4] T. Cazenave. Monte-Carlo Expression Discovery. *International Journal on Artificial Intelligence Tools*, 22(1), 2013.

[5] R. Coulom. The Monte-Carlo Revolution in Go. In *The Japanese-French Frontiers of Science Symposium (JFFoS 2008), Roscoff, France*, 2009.

[6] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo Planning. In *ECML-06*, 2006.

[7] J. McDermott, D. R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, and U.-M. O'Reilly. Genetic Programming needs better Benchmarks. In *GECCO 2012*, 2012.

[8] D. White, S. Yoo, and J. Singer. Online resources. http://www.dcs.gla.ac.uk/~whited/mcts, 2015.