ECAI 2010
H. Coelho et al. (Eds.)
IOS Press, 2010

doi:10.3233/978-1-60750-606-5-1057

Nested Monte-Carlo Expression Discovery

Tristan Cazenave[1]

Abstract. Nested Monte-Carlo search is a general algorithm that gives good results in single player games. Genetic Programming evaluates and combines trees to discover expressions that maximize a given evaluation function. In this paper Nested Monte-Carlo Search is used to generate expressions that are evaluated in the same way as in Genetic Programming. Single player Nested Monte-Carlo Search is transformed in order to search expression trees rather than lists of moves. The resulting program achieves state of the art results on multiple benchmark problems. The proposed approach is simple to program, does not suffer from expression growth, has a natural restart strategy to avoid local optima and is extremely easy to parallelize.

1 Introduction

Recently Monte-Carlo Tree Search (MCTS) has been very successful in games such as Go [2, 4], General Game Playing [3] and puzzles [1]. We propose to adapt the method to discover expressions.

Expression discovery is usually addressed with Genetic Programming [5]. In Genetic Programming a set of random expressions is built, then the set of expressions is evolved for multiple generations. At each generation the expressions are evaluated and sorted according to their evaluation (i.e. their fitness or their score). The highest rated expressions (i.e. individuals) of a generation are then breeded to create the next generation. Breeding two expressions consists in exchanging two subtrees of the two expressions represented as trees. The principle underlying Genetic Programming is that the subtrees of the expressions are building blocks that can be used to build new expressions. Subtrees that are useful in one expression are often useful in other expressions and these successful subtrees guide the search toward the good expressions. In contrast, Nested Monte-Carlo Search favors expressions that have the same successful atoms at the start of the stack representing the expression.

Genetic Programming can suffer from bloat (i.e. uncontrolled growth of the expressions with increasing numbers of generations) and from over-specialization which leads to a lack of diversity in the population and to suboptimal expressions. Restart strategies are usually used to overcome this undesired behavior.

When using MCTS to generate expressions, the size of the generated expressions as well as the restarts of the algorithm are naturally handled. In our experiments the expression generated with MCTS are usually more simple and provide scores at least equivalent to the scores of the expressions generated with Genetic Programming for the same problems. Moreover the algorithm is more simple and requires less tuning than Genetic Programming.

The second section explains Nested Monte-Carlo Search, the third section details its application to expression discovery, the fourth section gives experimental results for different problems.

[1] LAMSADE, Université Paris-Dauphine, Paris, France, email: cazenave@lamsade.dauphine.fr

2 Nested Monte-Carlo Search

The basic idea of Nested Monte-Carlo Search is to perform a principal playout with a bias on the selection of each move based on the results of a Monte-Carlo tree search [1].

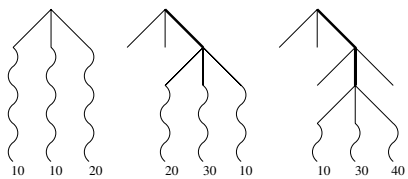

Figure 1. At each step of the principal playout shown here with a bold line, an NMC of level n performs a NMC of level $n-1$ (shown with wavy lines) for each available move and selects the best one. At level 0, a simple pseudo-random playout is used.

The base level of the search plays random games (i.e. playouts), random moves are played until the end of the game at this level. When the game is over the score of the position that has been reached is sent back.

At each move of a playout of level 1 it chooses the move that gives the best score when followed by a random playout. Similarly for a playout of level n it chooses the move that gives the best score when followed by a playout of level $n-1$.

When a search at the highest level is finished and there is time left, another search is performed at the highest level, and so on until the thinking time is elapsed.

Nested Monte-Carlo search has been successful in establishing world records in single player games such as Morpion Solitaire or SameGame. It provides a good balance between exploration and exploitation and it automatically adapts its search behavior to the problem at hand without parameters tuning.

3 Nested Monte-Carlo Expression Discovery

Expressions are generated with sampling at level zero and with nested searches at higher levels. The first subsection explains the sampling algorithm. The second subsection explains the nested search algorithm.

3.1 Random sampling

Expressions can be seen as trees. An atom is a node of a tree. A terminal atom is a node that has no children, usually terminal atoms are the variables and the constants of a problem. For example +, -, *, / are non terminal atoms since they have two children, whereas 1, 2, 3, x are terminal atoms.

In random sampling the tree is represented as a stack. Sampling consists in randomly filling the stack, stopping when the expression is complete (i.e. there are no more leaves to complete in the corresponding tree). A maximum number of nodes is used to prevent too large expressions. When the minimal number of potential nodes in the final tree is greater than this threshold, only terminal atoms are added to the tree in order to have less than the maximum number of nodes.

3.2 Nested search

The nested search maintains a stack for each level of search. At a given level; and at each step, all the possible atoms are tried and followed by a lower level search. The atom that results in the best expression is then chosen and the playout continues until the tree is completed (i.e. there are no more leaves to develop in the tree).

Algorithm 1 Nested search

nested ($index$, $numberLeaves$, $maximumNodes$, $level$)
best expression $\leftarrow \{\}$
while $numberLeaves > 0$ **do**
  for a in all possible atoms **do**
    if $index$ + $numberLeaves < maximumNodes$ or nbChildren (a) = 0 **then**
      $stack[level][index] \leftarrow a$
      $index \leftarrow index + 1$
      $numberLeaves \leftarrow numberLeaves$ + nbChildren(a) - 1
      for $i \leftarrow 0$ to index **do**
        $stack[level-1][i] \leftarrow stack[level][i]$
      end for
      if $level = 1$ **then**
        $score$ = sample ($index$, $numberLeaves$, $maximumNodes$)
      else
        $score$ = nested ($index$, $numberLeaves$, $maximumNodes$, $level$ - 1)
      end if
      if $score >$ score of best expression **then**
        best expression $\leftarrow stack[level-1]$
      end if
      $index \leftarrow index - 1$
      $numberLeaves \leftarrow numberLeaves$ - nbChildren(a) + 1
    end if
  end for
  $stack[level][index] \leftarrow$ best expression [index]
  $numberLeaves \leftarrow numberLeaves$ + nbChildren($stack[level][index]$) - 1
  $index \leftarrow index + 1$
end while
return score ($stack[level]$)

4 Experimental Results

The algorithm was successfully applied to the Prime generating polynomials problem, and to the Finite algebra problem.

The goal of the Prime generating polynomials problem [7] is to find a polynomial that generates as many different primes in a row as possible.

A level two nested search found + (* (- (+ (4 , 13) , * (1 , x)) , - (4 , x)) , 83) which generates 51 primes in a row and 40 different primes. It is better than the polynomial found in [7] ($x^2 - 3x + 43$) that generates 43 primes in a row and 40 different primes.

The Finite Algebra problems consist in finding terms that satisfy some equalities [6]. We did some tests on the A_2 primal algebra from [6].

For algebra A_2, a nested level three search quickly found (after 8,704,083 evaluations) a discriminator term containing 31 operations (in [6] an optimized GP found a 51 operations term after 238,000,000 evaluations). The 31 operations discriminator term is:

* (* (* (x , x) , * (x , * (* (* (* (* (* (y , * (x , x)) , x) , x) , z) , * (* (z , x) , x)) , y))) , * (* (* (z , * (* (z , * (y , * (* (* (* (z , y) , x) , z) , x))) , z)) , * (x , * (z , * (y , x)))) , * (* (x , y) , * (* (* (z , y) , z) , x))))

5 Conclusion

Nested Monte-Carlo search improves much on random search for difficult expression discovery problems. It is competitive with state of the art Genetic Programming since it produces shorter expressions that have equal or better scores with less evaluations for some problems such as the finite algebra problem or the prime generating polynomial problem. Moreover it is a simple and easy to program algorithm that keeps a good balance between exploration of new expressions and exploitation of already found ones. It has a natural restart strategy that ensures diversification and it avoids bloat. It is also very easy to parallelize it massively, simply running the same algorithm in parallel on many computers with a different random seed.

Concerning future works, it would certainly improve nested searches to memorize the results of previous attempts in order to direct its search. For example using a tree of previously evaluated expressions would at least make the search faster because it would not evaluate again a previously encountered expression, moreover knowing the results of previous attempts contained in the tree could help the algorithm direct its search. It would also be interesting to understand the properties of a problem that makes nested Monte-Carlo Search relevant for it.

Acknowledgments

This work has been supported by French National Research Agency (ANR) through COSINUS program (project EXPLO-RA ANR-08-COSI-004)

REFERENCES

[1] Tristan Cazenave, 'Nested Monte-Carlo search', in *IJCAI*, pp. 456–461, (2009).
[2] R. Coulom, 'Efficient selectivity and back-up operators in monte-carlo tree search', in *Computers and Games 2006*, Volume 4630 of LNCS, pp. 72–83, Torino, Italy, (2006). Springer.
[3] Hilmar Finnsson and Yngvi Björnsson, 'Simulation-based approach to general game playing', in *AAAI*, pp. 259–264, (2008).
[4] Sylvain Gelly and David Silver, 'Achieving master level play in 9 x 9 computer go', in *AAAI*, pp. 1537–1540, (2008).
[5] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection.*, MIT Press, Cambridge, MA, 1992.
[6] Lee Spector, David M. Clark, Ian Lindsay, Bradford Barr, and Jon Klein, 'Genetic programming for finite algebras', in *Genetic And Evolutionary Computation Conference*, pp. 1291–1298. ACM New York, NY, USA, (2008).
[7] James Alfred Walker and Julian Francis Miller, 'Predicting prime numbers using cartesian genetic programming', in *Genetic Programming*, Volume 4445 of LNCS, pp. 205–216. Springer, (2007).