Multitask Evolution with Cartesian Genetic Programming

Extended Abstract

Eric O. Scott
George Mason University
4400 University Drive
Fairfax, Virginia 22030
escott8@gmu.edu

Kenneth A. De Jong
George Mason University
4400 University Drive
Fairfax, Virginia 22030
kdejong@gmu.edu

ABSTRACT

We introduce a genetic programming method for solving multiple Boolean circuit synthesis tasks simultaneously. This allows us to solve a set of elementary logic functions twice as easily as with a direct, single-task approach.

CCS CONCEPTS

•Computing methodologies → Search methodologies;

KEYWORDS

Cartesian Genetic Programming, Multitask Learning

Figure 1: A hypothetical logic circuit that solves three distinct Boolean functions over two inputs. The tree that solves task C reuses subtrees of the tree that solves task B.

1 INTRODUCTION

Most evolutionary algorithms and metaheuristics built to date have been used as *single-task* problem solvers. With a few exceptions in areas like case-based reasoning [2] and robot shaping [3], traditionally there is no attempt to store solutions, subcomponents, or other evolved information from one task and to reuse it on other tasks. Instead, these search algorithms rely on direct guidance from an objective function to approach a solution via variation and selection.

Very recently, a disjoint handful of researchers have begun to design new kinds of evolutionary systems that depart significantly from this single-task, adaptationist paradigm. Preliminary efforts with *multitask* evolutionary systems have begun to reveal cases where transferring and re-using information across superficially dissimilar tasks can serve to help a problem-solving agent perform more efficiently, or to provide a means of solving problems that would otherwise be deceptive or intractable. A number of different mechanisms for effectively evolving solutions to multiple tasks simultaneously
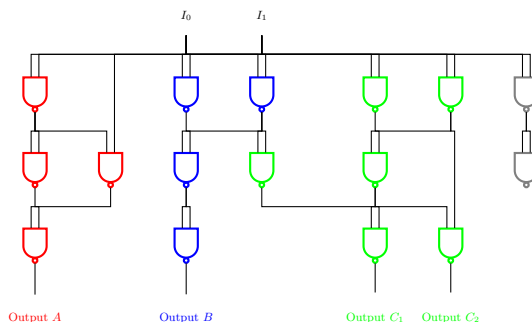
or in sequence have recently been proposed by authors working in different domains. New structured-population models such as MAP-elites, for instance, have been developed for robotics and design applications [8], 'multifactorial' methods have been introduced for combinatorial optimization [5], and agent-based 'path-reuse' mechanisms have been proposed to allow deep neural networks that have been evolved for different pattern recognition tasks to reuse subcomponents that were trained for previous tasks [4].

In this work, we are applying Cartesian genetic programming (CGP, [7]) in a multitask fashion to evolve logic circuits that solve elementary Boolean functions. Taking inspiration from the canonical literature on multitask learning in neural networks [1], we follow what we call a 'multi-behavior' approach to multitask evolution: each individual genome in the evolutionary algorithm encodes a solution to several tasks at once (Figure 1). In the resulting program tree, the solutions to each task all share the same inputs, but have their own designated outputs. This allows subtrees that are useful in the solution to one task to be reused as partial solutions to other tasks.

CGP has often been applied in much this way to synthesize solutions to single tasks with more than one output. Our focus here is somewhat different: we are interested in finding ways of sharing and transferring information across multiple *distinct tasks*. The question here is whether we can use multitask CGP to more efficiently solve a set of tasks that would traditionally be treated separately.

2 METHODOLOGY

The set of tasks used in this study is a simple suite $\mathcal{T}$ of 9 elementary logic function synthesis tasks that were originally used by Lenski et al. to demonstrate that complex tasks are sometimes easier to solve in conjunction with other tasks than they are to solve directly [6]:

$$\mathcal{T} = \{\texttt{AND}, \texttt{AND\_N}, \texttt{EQU}, \texttt{NAND}, \texttt{NOR}, \texttt{NOT}, \texttt{OR}, \texttt{OR\_N}, \texttt{XOR}\}.$$

Our primitive set consists only of $\{\texttt{NAND}\}$.

For a preliminary proof of concept we wish only to demonstrate the following claim:

Hypothesis: A multi-behavior approach will be able to solve all $|\mathcal{T}| = 9$ tasks using less computational effort than a traditional single-task approach.

To test this, we apply a standard $(1+4)$-style implementation of Cartesian genetic programming to a fitness function that evaluates each circuit on all 9 tasks simultaneously. We treat the average fitness across all 9 tasks as a scalar fitness value.

Now, in multitask algorithm applications it sometimes happens that progress is impeded by *task interference*: progress on one task may overwrite the solution to previously solved tasks, preventing the algorithm from encoding solutions to many tasks at once. In light of this, we further hypothesize that multi-behavior CGP will perform better if we avoid mutating genes that belong to a successful solution to a task. We effect this by tracking which outputs each circuit element contributes information to. We then configure the probability that each element's genes are mutated to be a decreasing function of the mean fitness of all the tasks it belongs to. We test this weighted mutation scheme with both linearly- and exponentially-decreasing weighting functions. All together, we implemented one single-task CGP algorithm and three multi-behavior algorithms (a constant-mutation case, the linear case, and the exponential case).

In order to achieve a fair comparison, we performed an extensive sweep over CGP's free parameters and our weighted mutation scheme parameters to select configurations of each algorithm that solve the suite of tasks in the least computational effort on average. As a result of this parameter-tuning process, some algorithms are configured to use different circuit sizes than others. To compare results with different circuit sizes, our measure of computational effort is *node-evaluations*—that is, the number of individuals that have been evaluated, multiplied by their size. To solve all 9 tasks with each algorithm, we run each multi-behavior algorithm once on the task suite, and then we run the single-task algorithm 9 times in sequence, once for each of the tasks, measuring the total number of node-evaluations expended along the way before a solution is found.

3 PRELIMINARY RESULTS

Results on the 9-logic suite suggest that the multi-behavior CGP algorithm is more efficient than a single-task approach when a constant or exponentially-weighted mutation rate is used. Figure 2 shows the distribution of effort each algorithm required to successfully find solutions to all 9 tasks. On
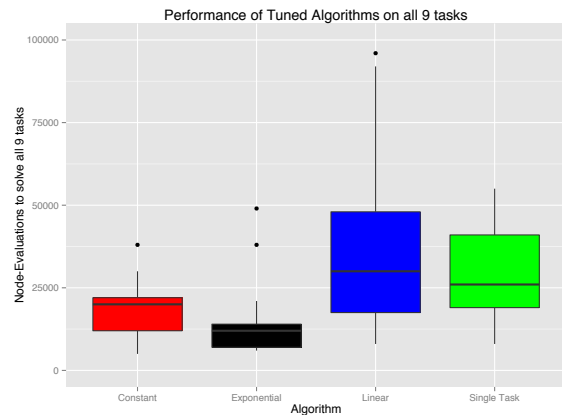

Figure 2: Preliminary results indicate that a multi-behavior approach is more efficient at solving all 9 tasks than a single-task approach.

average, we find that the exponentially-weighted algorithm solves the tasks twice as easily as the single-task algorithm does.

One of the potential advantages of this kind of multitask evolution is that no prior knowledge is needed about how information might be reused across different tasks: evolution dynamically discovers subproblems that are redundant across tasks. Much more work is needed, however, to understand how sharing information across tasks works in different domains, and what conditions or structural similarities among tasks are necessary to see performance improvements from multitask evolution.

As this research continues, we intend to analyze the circuits generated by the multi-behavior approach for patterns of reuse, and to extend this methodology to more complex task suites and primitive sets.

REFERENCES

[1] Rich Caruana. 1998. Multitask learning. In *Learning to learn*. Springer, 95–133.
[2] Pádraig Cunningham and Barry Smyth. 1997. Case-based reasoning in scheduling: reusing solution components. *International Journal of Production Research* 35, 11 (1997), 2947–2962.
[3] Marco Dorigo and Marco Colombetti. 1998. *Robot Shaping: An Experiment in Behavior Engineering*. MIT Press, Cambridge, MA, USA.
[4] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. 2017. PathNet: Evolution Channels Gradient Descent in Super Neural Networks. *arXiv preprint arXiv:1701.08734* (2017).
[5] Abhishek Gupta, Yew-Soon Ong, and Liang Feng. 2016. Multifactorial Evolution. *IEEE Transactions on Evolutionary Computation* 20, 3 (2016), 343–357.
[6] Richard E. Lenski, Charles Ofria, Robert T. Pennock, and Christoph Adami. 2003. The evolutionary origin of complex features. *Nature* 423 (May 2003), 139–144.
[7] Julian F Miller. 2011. Cartesian genetic programming. In *Cartesian Genetic Programming*. Springer, 17–34.
[8] A Nguyen, J Yosinski, and J Clune. 2016. Understanding Innovation Engines: Automated Creativity and Improved Stochastic Optimization via Deep Learning. *Evolutionary Computation* (2016).