Automating Algorithm Design through Genetic Programming Hyper Heuristics

Elsa M. Browning
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
brow3924@morris.umn.edu

ABSTRACT

actually write an abstract

Citing stuff to see what it looks like [4] and [3]

Keywords

Evolutionary Computation, Genetic Programming, Hyper Heuristics, Autoconstruction

1. INTRODUCTION

Mostly notes right now

Traditionally, genetic programming solutions to problems evolve, but almost everything else is specified by the system designer [5]. In autoconstruction, the variation methods are evolving as well.

In sections 2.1, 2.2, and 2.3 we describe necessary background for understanding the rest of the paper. Next, we describe the history of automating algorithm design (AAD) with a focus on hyper-heuristics (HH) optimization in section 2.4. Then, in section 3, we go over a few different genetic programming variants used in HH and the success of two in particular. Finally, we will go over current research being done with stack-based genetic programming (one of the GP variants) in section 4; the stack-based programming language called Push is used in this example along with a technique for AAD called autoconstruction.

this is long roadmap. Probably need to shorten

2. BACKGROUND

2.1 Evolutionary Computation

2.2 Genetic Programming

2.3 Hyper Heuristics

2.4 History of AAD

3. GENETIC PROGRAMMING VARIANTS

3.1 Tree-based Genetic Programming

3.2 Stack-based Genetic Programming

4. AUTOCONSTRUCTION

In genetic programming hyper heuristics (GPHH), the goal is to use genetic programming to evolve a program that will solve hard computational search problems. The individual programs serve as potential solutions. In GPHH, only the potential solutions are evolving – everything else is specified by the system designer.

not sure how to phrase previous sentence. Also not sure if I need to clarify what I mean by "potential solutions"

In autoconstruction, evolution is evolving as well. This happens by encoding the methods of variation into the programs that are evolving so that, as a program evolves, its variation methods also evolve (see section 4.1 for how this works).

Prior work on autoconstruction has explored a variety of system designs, but, until recently, none of them have been able to solve hard problems. A new system called Autoconstructive Diversification of Genomes (AutoDoG) has broken this trend by solving at least one hard problem: "Replace Space with Newline" [5]. And, in recent unpublished work, AutoDoG has actually solved a problem that has never been solved before [2].

Not sure how much of this is introduction stuff

4.1 Push and Plush

Push is a stack-based programming language with a separate stack for each data type. It was developed for program evolution and autoconstruction was one of the driving forces behind the original design [5]. Push programs are strings of instructions, constants, and parentheses with only one syntax requirement: the parentheses must be balanced [4].

I am not sure that I should mention prev sentence, because I may want to give examples if this is mentioned...

Instructions are executed by putting them on the `exec` stack. If the necessary arguments are not available for the instruction to be executed, the instruction is skipped. For example, assume all stacks are empty and assume a program says `(1 2 integer_add)`. This means push 1 and 2 onto the `integer` stack and push the `integer_add` onto the take the `exec` stack [1]. The `integer_add` instruction will try to execute: the first two integers (in this case 1 and 2) are popped off the top of the `integer` stack, added together,

UMM CSci Senior Seminar Conference, April 2017 Morris, MN.

and the result is pushed back on to the `integer` stack. If, instead, the program was `(1 integer_add)` then it would push 1 onto the `integer` stack, but skip the `integer_add` because there are not enough integers in the `integer` stack.

not sure I should include example inline like this, might want to separate and try to shorten somehow

- Talk about:
- Plush
- linear genomes
- how Plush/linear genomes are beneficial to autoconstruciton
- PushGP–a "reasonably standard generational genetic programming system"
- PUT PUSHGP IN WITH RESULTS STUFF... BRIEFLY MENTION WHAT IT IS (and that it's not autoconstruciton)

Turn list into paragraphs

4.2 AutoDoG

In this section, we briefly describe some of the key features of AutoDoG.

One thing that the designers of AutoDoG wanted to do was maintain diversity in parent selection. To do this, AutoDoG uses Lexicase Selection, described below:

In each parent selection event,

- start with pool of entire population
- filter pool based on how each program performs on individual fitness cases (performed in random order, and one at a time)

  HOW do I phrase prev bullet point?

- In each fitness case, only retain programs best on that case.

TO NIC: is above a correct summary of lexicase selection?

left off 10:30 in video, here (ish) in paper

Its name comes from the way it filters the population using a kind of "lexigraphic ordering" of cases [5].

> Lexicase selection performs significantly better than the standard approach and than implicit fitness sharing, allowing the solution of more problems, in fewer generations, on a benchmark suite of problems taken from introductory programming textbooks [5].

will probably paraphrase previous quote since it's so long

Most autoconstruction systems have some form of the"no cloning rule" which prevents an exact copy of a program from moving on to the next generation, and AutoDoG has a form of this as well [5]. AutoDoG broadens this to "require that offspring are only allowed to enter the population if they pass a more stringent diversification test" [5]. Test:

- make individual for next generation
- individual takes itself as mate and makes temp children.
- if temp children differ enough from individual and each other, individual enters next generation's population
- else, generate new random individual and repeat previous steps
- if random individual fails, generate individual with empty genome and add it to next generation
- discard any children after test

AutoDoG works similarly to PushGP, but is run with autoconstruction as the genetic operator rather than human designed mutation and crossover operators. Its main loop "iteratively tests the error of all individuals and then builds the next generation by selecting parents and passing them to genetic operators" [5].

Describe more/better, summarize long quotes, turn lists into paragraphs

4.3 Results

AutoDoG is unique among autoconstructive systems in that it can solve hard problems. However, as stated by Spector et al,

> We do not know which of these, or which combinations of these, may be responsible for the fact that AutoDoG appears to be capable of solving more difficult problems than previous autoconstructive evolution systems [5].

This is because it's hard to separate the pieces; there are a lot of intertwined elements of AutoDoG and Push. Separating them to find which elements contribute to the success without unraveling the entire system is difficult and has not yet been done.

TO NIC: Is this (above) more correct?

The "hard problem" we mentioned earlier is "Replace Space with Newline." This is a software synthesis problem which, when given a string, requires the solution to print the string replacing spaces with newlines. It also requires the return of the integer count of the non-whitespace characters. This is complex because it involves multiple data types and outputs, and requires conditionals and iteration or recursion. AutoDoG does not actually do better than standard PushGP on this problem.

> AutoDoG does not succeed as reliably on this problem as has PushGP in some other configurations, but it does solve the problem approximately 5-10 % of the time, producing general solutions [5]

"This is not surprising because AutoDoG has to learn how to evolve as well as learn how to solve the problem. That said, recent (unpublished) results look like there might be scenarios where autoconstruction may find solutions to problems we've never solved with regular PushGP, so we may soon have examples where autoconstruction is 'better' in some meaningful sense"

PARAPHRASE QUOTE FROM NIC

More info about new AutoDoG results

One interesting thing to note is how AutoDoG evolves. In standard PushGP, one can see a steady incline in differences between the genomes of parents and their children. Over time, change occurs at a fairly steady rate and is not very dramatic (from one generation to the next). In AutoDoG, there are huge differences between the genomes of some parents and their children. There was significant change for some individuals from one generation to the next and overall it was a bit erratic.

Clean up, more info about weird AutoDoG evolution. Maybe include DL graphs

5. CONCLUSIONS AND FUTURE WORK

6. ACKNOWLEDGMENTS

7. REFERENCES

[1] T. Friedrich and H. P. Institute, editors. *Expressive Genetic Programming: Concepts and Applications*. ACM New York, NY, USA copyright 2016.

[2] E. Moshkovich. private communication.

[3] G. L. Pappa, G. Ochoa, M. R. Hyde, A. A. Freitas, J. Woodward, and J. Swan. Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, 15(1):3–35, March 2014.

[4] L. Spector. Autoconstructive evolution: Push, pushgp, and pushpop. In L. Spector, E. Goodman, A. Wu, W. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, editors, *GECCO '01: Proceedings of the 2001 on Genetic and Evolutionary Computation Conference Companion*, pages 137–146. Morgan Kaufmann Publishers.

[5] L. Spector, N. F. McPhee, T. Helmuth, M. M. Casale, and J. Oks. Evolution evolves with autoconstruction. In T. Friedrich and H. P. Institute, editors, *GECCO '16: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 1349–1356. ACM, July 2016.

Add more of references to bib, update [2], cite Nic?