Automating Algorithm Design through Hyper Heuristic Genetic Programming

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

ABSTRACT

This paper provides a sample of a LaTeX document which conforms, somewhat loosely, to the formatting guidelines for the University of Minnesota, Morris, Computer Science Senior Seminar proceedings. It is based heavily on (and takes material directly from) a similar document illustrating the format of the ACM SIG Proceedings, which we have based our proceedings format on.

The original ACM document tried to include

> every imaginable sort of "bells and whistles", such as a subtitle, footnotes on title, subtitle and authors, as well as in the text, and every optional component (e.g. Acknowledgments, Additional Authors, Appendices), not to mention examples of equations, theorems, tables and figures.[2]

We've removed many of the more esoteric tricks here because either they'll never be used (e.g., multiple authors) or are used *very* rarely (e.g., appendices). Refer to the original ACM document for more of those fancy examples.

`Needs more work`

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

The *proceedings* are the records of a conference, and conference editors like ACM seek to give their conference by-products a uniform, high-quality appearance. We also would like our proceedings to look highly professional, so we're borrowing heavily from the ACM formatting guidelines. These include some rigid requirements for the format of the proceedings documents: there is a specified format (balanced double columns), a specified set of fonts (Arial or Helvetica and Times Roman) in certain specified sizes (for instance, 9 point for body copy), a specified live area (18×23.5 cm [7" × 9.25"]) centered on the page, specified size of margins (2.54cm [1"] top and bottom and 1.9cm [.75"] left and right; specified column width (8.45cm [3.33"]) and gutter size (.083cm [.33"]).

`I really like this section.`

The good news is, with only a handful of manual settings,[1] the LaTeX document class file handles all of this for you.

The remainder of this document is concerned with showing, in the context of an "actual" document, the LaTeX commands specifically available for denoting the structure of a proceedings paper, rather than with giving rigorous descriptions or explanations of such commands. Section **??** introduces the main examples of formatting, and Section **??** wraps things up.

1.1 My nifty subsection

I want to refer to Section 2.1 and Figure **??**. It would also be nice to cite [3] and [2] and [1].

Let's make an equation:

$$\text{area} = \pi r^2$$

I want to refer to Section **??** and Figure **??**. It would also be nice to cite [3]. We can also do inline equations: $s = \sum_{i=0}^{N} x_i$. I want to refer to Section **??** and Figure **??**. It would also be nice to cite [3].

$$s = \sum_{i=0}^{N} x_i$$

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

---

[1]One of these, the `\alignauthor` command, you have already used; another, `\balancecolumns` will be used in your very last run of LaTeX to ensure balanced column heights on the last page.

4. AUTOCONSTRUCTION

Traditionally, genetic programming *solutions* to problems evolve, but almost everything else is specified by the system designer [3]. In autoconstruction, the variation methods are evolving as well. Variation methods are things like mutation and crossover–things that cause *variation* in the population (see Section 2.1 and Section 2.2 for more details on how these work). Spector et al. generalizes how this is happening:

> In autoconstructive evolution the methods for variation are encoded in the same individuals that are being evolved as solutions to the target problem.

I say "variation" a million times... sorry guys

Here,"individuals" refers to programs. This means that variation methods are part of the actual program code so that the methods for evolution are evolving as well.

Not sure if I should describe quote here–I say this in different ways earlier in the section

Prior work on autoconstruction has explored a variety of system designs, but, until recently, none of them have been able to solve hard problems. This new system called Autoconstructive Diversification of Genomes (AutoDoG) has actually managed to solve a few hard problems [3]. And, in recent unpublished work, AutoDoG has actually solved a problem that has never been solved before. [1].

In Section 4.1, we will describe the format in which AutoDoG is expressed. In Section 4.2 we will describe how AutoDoG works. Finally, in Section 4.3 we will go over how well AutoDoG performs compared to other systems.

4.1 Push, Plush, and PushGP

Push is a programming language developed for program evolution [3]. It was designed to be a better language in which to express evolving programs. Autoconstruction was actually one of the driving forces behind the original design. Push can briefly be described as

> a stack based language with a separate stack for each data type [...] Programs are executed by putting them on `exec` stack, from which the interpreter continuously takes and processes items [3].

I will hopefully have described enough about stack based languages in Section 3.2 that I won't have to describe much more here

- Talk about: how Push works (maybe bullets for now)
- how Push = better for autoconstruction things
- Plush
- linear genomes
- how Plush/linear genomes are beneficial to autoconstruciton
- PushGP–a "reasonably standard generational genetic programming system"

Turn list into paragraphs

4.2 AutoDoG

AutoDoG is unique in that is can solve some hard problems. We are going to discuss some of its features, however

> 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 [3]

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

- Each parent selection event starts with pool of entire population
- Pool filtered based on performance of individual fitness cases, considered in random order one at a time.
- In each case, retains only individuals best on that case.

Its name comes from the way it filters the population using a kind of "lexigraphic ordering" of cases [3].

> 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 [3].

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 [3]. AutoDoG broadens this to "require that offspring are only allowed to enter the population if they pass a more stringent diversification test" [3]. 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" [3].

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

4.3 Results

The "hard problem" we discussed 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 [3]

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 [1].

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. So over time, change occurs at a fairly steady rate and is not very dramatic (from generation to generation). 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] Eva. private communication.
[2] 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.
[3] 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 [1], cite Nic?