Recent Developments in Autoconstructive Evolution

Lee Spector
School of Cognitive Science
Hampshire College
893 West Street
Amherst, Massachusetts 01002
lspector@hampshire.edu

Eva Moscovici
College of Information and Computer Sciences
University of Massachusetts Amherst
140 Governors Drive
Amherst, MA 01003
evamoshkovich92@gmail.com

ABSTRACT

This is an extended abstract for an invited keynote presentation at the 7th Workshop on Evolutionary Computation for the Automated Design of Algorithms (ECADA). We first outline the motivation, primary mechanisms, and prior results of the evolutionary computation technique called "autoconstructive evolution." We then briefly describe a collection of recent enhancements to the technique, along with a few preliminary results of ongoing experimental work.

CCS CONCEPTS

•**Computing methodologies** → **Genetic programming;** *Artificial life;* •**Software and its engineering** → **Genetic programming;**

KEYWORDS

genetic programming, autoconstructive evolution, software synthesis

ACM Reference format:
Lee Spector and Eva Moscovici. 2017. Recent Developments in Autoconstructive Evolution. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 3 pages.
DOI: http://dx.doi.org/10.1145/3067695.3082058

1 AUTOCONSTRUCTIVE EVOLUTION

Autoconstructive evolution is an evolutionary computation technique that has been a topic of intermittent research and development since the year 2000 [1, 2, 11, 12, 16, 17].

Autoconstructive evolution systems can be considered to be instances of genetic programming systems, insofar as they are employed to solve computational problems for which the solutions are executable computer programs, and insofar as they find solutions by processing populations of executable computer programs that undergo cycles of variation and selection [9].

In contrast to other genetic programming systems, however, in an autoconstructive evolution system the evolving programs are responsible not only for solving a specified computational problem, but also for producing their own children in the evolutionary process. This allows the methods of reproduction and variation themselves to vary and to evolve. The way that it does so is roughly analogous to the ways in which reproductive methods (considered broadly to include everything from karyotype to mate selection behaviors) evolve in biological systems; the reproductive methods are embodied in the organisms themselves, and may evolve along with other features of the organisms through variation and selection.

One motivation for research on autoconstructive evolution is the speculation that it may have the potential to solve problems beyond the reach of ordinary genetic programming systems, because evolved methods of reproduction and variation may be more effective than those designed by hand. However, the task faced by an autoconstructive evolution system is clearly more difficult than that faced by an ordinary genetic programming system, because it must "invent" and refine its reproductive mechanisms while also solving its target computational problem.

Prior research on autoconstructive evolution has explored a variety of options for the aspects of reproduction and variation that are under the control of the evolving programs, the ways in which child-production is integrated into the genetic programming generational loop, and the representations and instruction sets available to programs for the construction of children. Some of this prior work was summarized in our paper for the 2016 Workshop on Evolutionary Computation for the Automated Design of Algorithms, which also introduced the AutoDoG system on which our current research is based [16].

2 AUTODOG

The AutoDoG system (named for "Autoconstructive Diversification of Genomes") is implemented on the foundation of the PushGP genetic programming system, which has been used for a variety of purposes and projects unrelated to autoconstructive evolution (for example, [3, 4, 6, 7, 15]).

AutoDoG is simply PushGP, which implements a standard generational genetic programming loop, run in a configuration that uses only the single `autoconstruction` genetic operator in place of the collection of hand-written mutation and recombination operators that are normally employed. The `autoconstruction` genetic operator takes two parent genomes as inputs and produces a child genome by *executing* the program encoded by one of the parent genomes, with both parent genomes available as data from which the child genome can be constructed. Details are available in [16], but the following are a few of the other key features of AutoDoG relative to previous autoconstructive evolution systems:

GECCO '17 Companion, Berlin, Germany

978-1-4503-4939-0/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3067695.3082058

- The use of a diversity-maintaining parent selection algorithm (lexicase selection [7, 13]), which may help to prevent the collapse of population diversity as methods for reproduction and variation evolve.
- The representation of both parents and children using linear (Plush) genomes for structured (Push) programs [5], which permits the use of relatively simple primitive instructions for generating and varying children.
- Explicit diversification constraints that prevent the survival of children that do not produce diverse children themselves, and are therefore unlikely to contribute to the evolution of either problem-solving or reproductive functionality.

The 2016 workshop paper demonstrated that AutoDoG could use autoconstruction to solve a non-trivial problem, the Replace Space with Newline problem from our software synthesis benchmark suite [6]. It also presented evidence that the reproductive methods used by individuals in AutoDoG populations evolve in significant ways as runs proceed.

3 RECENT DEVELOPMENTS

Since the 2016 workshop, AutoDoG has been enhanced in several ways, including:

- Several new diversification tests have been developed. Previously, a child was permitted to survive if and only if two of its own children (generated only for the purpose of this test and then discarded) differed from their parent by different, non-zero amounts. Among the new tests that have been developed are tests that require differences more specifically in program sizes and instruction sets, and tests that require variation over more than one generation of descendants.
- The domain-specific language for genome manipulation, which programs use to generate and vary the genomes of their children, has been enriched. Among the new elements are high-level instructions that implement components of the hand-written genetic operators that are used in non-autoconstructive runs of PushGP, such as uniform mutation and alternation [14]. Evolving programs may use these genetic operator components, with evolved parameters, in evolved combinations, and in conjunction with other genome-manipulation instructions, to construct their children.
- A mechanism has been added for the application of "entropy," in the form of random gene deletion, to the results of autoconstructive production of children. Because we want variation methods to be encoded in individuals, so that they may evolve, we do not implement exogenous mutations upon which the evolutionary process may come to rely. Exogenous *deletions*, however, are permissible, because they cannot introduce new genetic material. Initial experiments suggest that the presence of entropy favors individuals that repair or at least add material to their children, presumably because these behaviors offset the random deletions caused by entropy. This may facilitate the emergence of meaningful variation.
- New mechanisms have been added to favor individuals with relatively *more* diversifying methods for reproduction and variation. AutoDoG's diversification constraints are binary: children that meet them are allowed to survive, while those that do not are not. The new mechanisms provide more discrimination among those that do survive, using stochastic screening procedures that are applied to the population before each parent selection event. With one method, for example, a degree of parental similarity is randomly chosen for each parent selection event, and individuals that are more similar to their parents than the chosen degree are excluded from consideration. Another method screens on the basis not of the similarity of parent and child, but on the similarity of the parent's and child's reproductive behaviors. Yet another mechanism screens on the basis of genealogical "age," using a method that we call "AMPS" for "Age-Mediated Parent Selection." AMPS builds on the ideas of ALPS [8] and Age-Pareto Optimization [10], providing a mechanism that can be added with little effort to standard (parent-selection-based) genetic programming systems. When used in conjunction with autoconstructive evolution, AMPS can provide selection in favor of more-diversifying reproductive methods, by incorporating child/parent differences into the function that is used to compute the ages of children from those of their parents.

4 PRELIMINARY RESULTS

Experiments with the methods described above, alone and in various combinations, are ongoing. As of this writing, we do not have sufficient data to draw firm conclusions about the efficacy of any of these techniques, or about the long-term prospects for autoconstructive evolution using the ideas outlined here. However, we do have preliminary results that exhibit some points of interest.

For example, we have found that autoconstructive evolution can sometimes achieve high success rates even within the computational limits that we normally use for non-autoconstructive genetic programming, expressed in terms of population sizes and generation limits. This was a surprise. We have been motivated by the prospect that autoconstructive evolution could solve problems not solvable by non-autoconstructive genetic programming, but we expected it to require significantly greater resources to solve problems that could also be solved non-autoconstructively, since it must invent and refine its own methods for reproduction and variation even as it works to solve the target problem.

Prior work accorded with this expectation. We had previously documented non-autoconstructive genetic programming solving the Replace Space with Newlines problem in roughly 50% of runs, with a population size of 1000 and within 300 generations [6]. In [16] we reported that AutoDoG had also solved this problem, but that it had done so only in 5–10% of our runs, even though we allowed runs to continue for more generations.

Using some of the new developments described above, however, we are now able to find solutions to the Replace Space with Newlines problem using autoconstructive evolution at least as frequently as we previously could with non-autoconstructive evolution, within

the same resource constraints. For example, 75% of a recent suite of 20 AutoDoG runs with population size 1000 succeeded within 300 generations.

We have also recently conducted 20 runs of AutoDoG on the Mirror Image software synthesis problem [6], and achieved 100% success, with 16 runs (80%) finding solutions in fewer than 300 generations. The highest success rate within 300 generations for non-autoconstructive evolution on this problem documented in [6] was 78%.

The numbers of runs described here are too low to claim with any certainty that autoconstructive evolution performs better than non-autoconstructive genetic programming, on problems that both can solve, using the same computational resources. But these results do suggest that the performance of autoconstructive evolution is at least sometimes in the same ballpark as that of non-autoconstructive genetic programming, which is notable because of the extra work that an autoconstructive evolution system must do to bootstrap the evolutionary process.

Recent experiments have also provided what may be the first evidence that autoconstructive evolution can indeed extend the reach of genetic programming to solve problems that it could not otherwise solve. No previous run of non-autoconstructive genetic programming, or of any other program synthesis method of which we are aware, has solved the String Differences problem described in [6]. However, we have recently found a (single) solution to this problem with AutoDoG, using some of the enhancements described above.

These results are preliminary and largely anecdotal, and we caution the reader against concluding anything definitive about the power or potential of autoconstructive evolution on their basis alone. They do, however, serve to illustrate the kinds experiments that we are conducting, and they provide a preview of the more complete results that we plan to present at the 7th Workshop on Evolutionary Computation for the Automated Design of Algorithms.

ACKNOWLEDGMENTS

We thank the members of the Hampshire College Computational Intelligence Lab for helpful discussions, J. Erikson for systems support, and Hampshire College for support for the Hampshire College Institute for Computational Intelligence. This material is based upon work supported by the National Science Foundation under Grants No. 1617087, 1129139 and 1331283. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Kyle Harrington, Emma Tosch, Lee Spector, and Jordan Pollack. 2011. Compositional Autoconstructive Dynamics. In *Unifying Themes in Complex Systems Volume VIII: Proceedings of the Eighth International Conference on Complex Systems (New England Complex Systems Institute Series on Complexity)*. NECSI Knowledge Press, 856–870.

[2] Kyle I. Harrington, Lee Spector, Jordan B. Pollack, and Una-May O'Reilly. 2012. Autoconstructive evolution for structural problems. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion (GECCO Companion '12)*. ACM, 75–82. DOI: http://dx.doi.org/10.1145/2330784.2330797

[3] Thomas Helmuth. 2015. *General Program Synthesis from Examples Using Genetic Programming with Parent Selection Based on Random Lexicographic Orderings of Test Cases*. Ph.D. Dissertation. College of Information and Computer Sciences, University of Massachusetts Amherst, USA. https://web.cs.umass.edu/publication/details.php?id=2398

[4] Thomas Helmuth, Nicholas Freitag McPhee, and Lee Spector. 2016. The Impact of Hyperselection on Lexicase Selection. In *GECCO '16: Proceedings of the 2016 Annual Conference on Genetic and Evolutionary Computation*. ACM, Denver, USA, 717–724. DOI: http://dx.doi.org/doi:10.1145/2908812.2908851 Nominated for best paper.

[5] Thomas Helmuth, Nicholas Freitag McPhee, and Lee Spector. 2016. Plush: Linear Genomes for Structured Push Programs. In *Genetic Programming Theory and Practice XIV*. Springer.

[6] Thomas Helmuth and Lee Spector. 2015. General program synthesis benchmark suite. In *GECCO '15: Proceedings of the 2015 Conference on Genetic and Evolutionary Computation* (July, 2015).

[7] Thomas Helmuth, Lee Spector, and James Matheson. 2015. Solving Uncompromising Problems With Lexicase Selection. *Evolutionary Computation, IEEE Transactions on* 19, 5 (Oct 2015), 630–643. DOI: http://dx.doi.org/10.1109/TEVC.2014.2362729

[8] Gregory S. Hornby. 2006. ALPS: the age-layered population structure for reducing the problem of premature convergence. In *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, Vol. 1. ACM Press, Seattle, Washington, USA, 815–822. DOI: http://dx.doi.org/doi:10.1145/1143997.1144142

[9] John R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[10] Michael Schmidt and Hod Lipson. 2010. Age-Fitness Pareto Optimization. In *Genetic Programming Theory and Practice VIII*, Rick Riolo, Trent McConaghy, and Ekaterina Vladislavleva (Eds.). Genetic and Evolutionary Computation, Vol. 8. Springer, Ann Arbor, USA, Chapter 8, 129–146. http://www.springer.com/computer/ai/book/978-1-4419-7746-5

[11] Lee Spector. 2001. Autoconstructive Evolution: Push, PushGP, and Pushpop. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. Morgan Kaufmann, 137–146. http://hampshire.edu/lspector/pubs/ace.pdf

[12] Lee Spector. 2011. Towards Practical Autoconstructive Evolution: Self-Evolution of Problem-Solving Genetic Programming Systems. In *Genetic Programming Theory and Practice VIII*, Rick Riolo, Trent McConaghy, and Ekaterina Vladislavleva (Eds.). Genetic and Evolutionary Computation, Vol. 8. Springer New York, 17–33. DOI: http://dx.doi.org/10.1007/978-1-4419-7747-2_2

[13] Lee Spector. 2012. Assessment of Problem Modality by Differential Performance of Lexicase Selection in Genetic Programming: A Preliminary Report. In *1st workshop on Understanding Problems (GECCO-UP)*. ACM, 401–408. DOI: http://dx.doi.org/doi:10.1145/2330784.2330846

[14] Lee Spector and Thomas Helmuth. 2013. Uniform Linear Transformation with Repair and Alternation in Genetic Programming. In *Genetic Programming Theory and Practice XI*, Rick Riolo, Jason H. Moore, and Mark Kotanchek (Eds.). Springer, Chapter 8, 137–153. DOI: http://dx.doi.org/doi:10.1007/978-1-4939-0375-7_8

[15] Lee Spector, Jon Klein, and Maarten Keijzer. 2005. The Push3 execution stack and the evolution of control. In *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, Vol. 2. ACM Press, Washington DC, USA, 1689–1696. http://www.cs.bham.ac.uk/~wbl/biblio/gecco2005/docs/p1689.pdf

[16] Lee Spector, Nicholas Freitag McPhee, Thomas Helmuth, Maggie M. Casale, and Julian Oks. 2016. Evolution Evolves with Autoconstruction. In *GECCO '16 Companion: Proceedings of the Companion Publication of the 2016 Annual Conference on Genetic and Evolutionary Computation*. ACM, Denver, Colorado, USA, 1349–1356. DOI: http://dx.doi.org/doi:10.1145/2908961.2931727

[17] Lee Spector and Alan Robinson. 2002. Genetic Programming and Autoconstructive Evolution with the Push Programming Language. *Genetic Programming and Evolvable Machines* 3, 1 (March 2002), 7–40. DOI: http://dx.doi.org/doi:10.1023/A:1014538503543