Using algorithm configuration tools to optimize genetic programming parameters: A case study

Nicholas Freitag McPhee
University of Minnesota, Morris
Morris, Minnesota, USA 56267
mcphee@morris.umn.edu

Thomas Helmuth
Washington and Lee University
Lexington, Virginia, USA 24450
helmutht@wlu.edu

Lee Spector
Hampshire College
Amherst, Massachusetts, USA 01002
lspector@hampshire.edu

ABSTRACT

We use Sequential Model-based Algorithm Configuration (SMAC) to optimize a group of parameters for PushGP, a stack-based genetic programming system, for several software synthesis problems. Applying SMAC to one particular problem leads to marked improvements in the success rate and the speed with which a solution was found for that problem. Applying these "tuned" parameters to four additional problems, however, only improved performance on one, and substantially reduced performance on another. This suggests that SMAC is "overfitting", tuning the parameters in ways that are highly problem specific, and raises doubts about the value of using these "tuned" parameters on previously unsolved problems. Efforts to use SMAC to optimize PushGP parameters on other problems have been less successful due to a combination of long PushGP run times and low success rates, which make it hard for SMAC to acquire enough information in a reasonable amount of time.

CCS CONCEPTS

•**Computing methodologies** → **Machine learning; Genetic programming;** *Search methodologies;*

KEYWORDS

parameter optimization, genetic programming, PushGP, SMAC, software synthesis

ACM Reference format:
Nicholas Freitag McPhee, Thomas Helmuth, and Lee Spector. 2017. Using algorithm configuration tools to optimize genetic programming parameters: A case study. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3067695.3076097

1 INTRODUCTION

There is a long history of optimizing parameters in evolutionary computation [7] including, for example, early work on optimizing mutation rates [2] and population sizes for genetic algorithms [1]. Recent developments in statistical modeling and machine learning have led to the design of powerful new techniques for parameter optimization. Sequential Model-based Algorithm Configuration (SMAC), for example, is a flexible tool for optimizing algorithm parameters; SMAC uses repeated runs of the target algorithm with different parameter values to estimate the relationship between parameters and performance [6]. In his GECCO 2016 keynote address [5], Holger Hoos argued that the parameter optimization field had reached a state where, instead of avoiding new parameters or making arbitrary choices for parameter values, researchers should expose as many parameters as possible, and then use tools like SMAC to optimize those (possibly large) sets of parameters.

Here we summarize a case study of applying SMAC to optimize a set of nine parameters for the Clojush[1] implementation of the PushGP system [8, 9] when applied to several software synthesis benchmark problems [4]. While SMAC found "tuned" parameters for one problem which substantially improved the success rate on that problem, those "tuned" parameters appear to be problem specific. Applying them led to no improvement on several other problems, and actively hurt the performance on another. This is a useful reminder that parameter optimization is, like all machine learning problems, subject to overfitting, and that just because a set of parameters works well on one problem, or even a set of problems, doesn't mean it will be a good choice for a new unsolved problem.

2 EXPERIMENTAL RESULTS

We initially used SMAC to optimize nine PushGP parameters [3] on the Replace Space With Newline software synthesis problem [4]. Both the default parameter settings used in previous work [4] and the parameter settings "discovered" by SMAC are listed in Table 1.

To ensure that changing the population size didn't affect the overall computational budget for the runs, the number of generations was calculated so the product of the population size and number of generations never exceeded the 300,000 individuals processed when using the default parameters (population size of 1,000 for 300 generations). The parameters *uniform mutation probability*, *uniform close mutation probability*, *alternation probability*, and *alternation followed by uniform mutation probability* need to add up to 1; we let SMAC explore any values in the range $[0, 1]$ and then normalized those four values so they summed to 1.

The final row of Table 1 shows that the SMAC parameters led to significantly higher success rates; 95% of the SMAC parameter runs find a solution, where only 54% of the runs with the the default setting find solutions. The runs using the SMAC parameter configuration also discovered solutions much earlier. By generation 100, for example, 87% of the runs had succeeded when using the SMAC parameters, where only 37% of the runs with the default settings had succeeded.

GECCO '17 Companion, Berlin, Germany
 978-1-4503-4939-0/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3067695.3076097

[1] https://github.com/lspector/Clojush

Table 1: Clojush parameters optimized by SMAC for the Replace Space With Newline problem, along with their ranges (as given to SMAC), their default values, and their "optimized" values as discovered by SMAC. Population size and alignment deviation are specified to have integer values. Population size is also specified to be on a log scale. The selection method is a categorical (non-numeric) variable. The default values are what were used in [4]. The final row shows the number of successes out of 110 independent trials for each collection of parameter settings.

Parameter	Range	Default	SMAC
Population size	$[1, 30K]$	1,000	150
Selection method	*tournament* or *lexicase*	*lexicase*	*lexicase*
Uniform mutation prob.	[0, 1]	0.2	0.36
Uniform close mut. prob.	[0, 1]	0.1	0.06
Alternation prob.	[0, 1]	0.2	0.03
(Alt. + Uni. mut.) prob.	[0, 1]	0.5	0.54
Alternation rate	[0, 1]	0.01	0.01
Alignment deviation	[0, 400]	10	121
Uniform mutation rate	[0, 1]	0.1	0.188
Successes (out of 110)		**59**	**104**

Table 2: Number of successful programs out of 100 runs with default parameters, and with the parameters that SMAC found when run on RSWN ; see Table 1.

Problem	Standard	SMAC
Replace Space with Newline	*54*	*91*
Double Letters	0	6
String Lengths Backwards	68	75
Syllables	22	17
X-Word Lines	17	3

Table 2 shows the success rates on 100 independent runs for each parameter set on four additional software synthesis benchmark problems [4]. Here we see that settings "tuned" by SMAC for Replace Space With Newline do *not* generalize well across the new problems. The SMAC settings are significantly (but not dramatically) better on Double Letters, and significantly worse on X-Word Lines, while the differences for the other two problems weren't statistically significant.[2]

3 CONCLUSIONS AND FUTURE WORK

SMAC was able to discover parameter settings that substantially improved performance on a specific problem (Replace Space With Newline) where we already had a reasonable success rate. Those new parameter settings did not generalize to additional problems, however, suggesting that SMAC, like most machine learning systems, is susceptible to "overfitting".

SMAC does support a training mode where it can explore parameters across a range of problems or problem instances, which might help address these "overfitting" issues and help SMAC find parameters that perform well more generally. Unfortunately, running SMAC across a broad collection of problems requires substantial computational effort. This problem is then made worse when several of the test problems have low success rates, as these runs often consume considerable computational effort before they fail, generating very little new information in the process. Limited exploratory work in this direction has been held up by SMAC's need for an adequate amount of information on successful runs. For these software synthesis problems, this can lead to hundreds of runs, each run lasting up to 24 hours.

Thus, while it is important to find ways to help SMAC (or similar tools) discover more generally applicable parameter settings, doing so on suites of problems that are computationally expensive to run and have low success rates is certainly problematic, and deserving of further attention.

[2]Using a pairwise χ^2 test of proportions with a significance level (before adjustment) of 0.05.

ACKNOWLEDGEMENTS

We are extremely grateful to participants on the SMAC forum for their patience in answering numerous questions as we learned our way around SMAC. Thanks also to Mitchell Finzel and Elsa Browning for their editing assistance.

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] Jarmo T Alander. 1992. On optimal population size of genetic algorithms. In *CompEuro'92.'Computer Systems and Software Engineering', Proceedings*. IEEE, 65–70.

[2] Thomas Back. 1993. Optimal mutation rates in genetic search. In *Proceedings of the fifth international conference on genetic algorithms*. Morgan Kaufmann, San Mateo, CA, 2–8.

[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. http://scholarworks.umass.edu/dissertations_2/465/

[4] 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).

[5] Holger H. Hoos. 2016. Taming the Complexity Monster or: How I Learned to Stop Worrying and Love Hard Problems. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16)*. ACM, New York, NY, USA, 3–4. DOI: http://dx.doi.org/10.1145/2908812.2908960

[6] F. Hutter, H.H. Hoos, and K. Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *LION-5 (LNCS)*. 507–523.

[7] FJ Lobo, Cláudio F Lima, and Zbigniew Michalewicz. 2007. *Parameter setting in evolutionary algorithms*. Vol. 54. Springer Science & Business Media.

[8] 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*. ACM Press, Washington DC, USA, 1689–1696. DOI: http://dx.doi.org/10.1145/1068009.1068292

[9] 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/10.1023/A:1014538503543