From Fitness Landscape Analysis to Designing Evolutionary Algorithms: The Case Study in Automatic Generation of Function Block Applications

Vladimir Mironovich
ITMO University
Saint-Petersburg, Russia
JetBrains Research
Saint-Petersburg, Russia
mironovich.vladimir@gmail.com

Maxim Buzdalov
ITMO University
Saint-Petersburg, Russia
mbuzdalov@gmail.com

Valeriy Vyatkin
ITMO University
Saint-Petersburg, Russia
Aalto University
Helsinki, Finland
Luleå University of Technology
Luleå, Sweden
vyatkin@ieee.org

ABSTRACT

Search-based software engineering, a discipline that often requires finding optimal solutions, can be a viable source for problems that bridge theory and practice of evolutionary computation. In this research we consider one such problem: generation of data connections in a distributed control application designed according to the IEC 61499 industry standard.

We perform the analysis of the fitness landscape of this problem and find why exactly the simplistic (1 + 1) evolutionary algorithm is slower than expected when finding an optimal solution to this problem. To counteract, we develop a population-based algorithm that explicitly maximises diversity among the individuals in the population. We show that this measure indeed helps to improve the running times.

CCS CONCEPTS

• **Software and its engineering** → **Search-based software engineering**; • **Theory of computation** → *Evolutionary algorithms*;

KEYWORDS

Evolutionary computation, population diversity, search-based software engineering, program synthesis

ACM Reference Format:
Vladimir Mironovich, Maxim Buzdalov, and Valeriy Vyatkin. 2018. From Fitness Landscape Analysis to Designing Evolutionary Algorithms: The Case Study in Automatic Generation of Function Block Applications. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3205651.3208230

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208230

1 INTRODUCTION

Evolutionary computation consists of theory and practice. Most recent advances in theory come from analysing relatively simple algorithms on artificial problems in terms of the number of fitness evaluations needed to reach the optimum in order to understand and improve the working principles of randomised search heuristics. Practice employs algorithms with rather complicated designs to solve problems that are often large and not understood well, and solving these problems to optimality is often an exception. Given these differences, bridging theory and practice is a difficult task, although successful cross-fertilisation always gives bright results.

Search-based software engineering [4] is one of the practical domains where evolutionary computation is applicable which, unlike many other domains, often requires solving problems to optimality. Program synthesis is a prominent example of such a problem: when the fitness of a program is based on its correctness any sub-optimal solution is not acceptable as a final result. Although there exist some benchmark problems of program synthesis that can be theoretically studied [7], no definite answers are known about synthesis of even middle-sized practically usable programs. The best forces are now concentrated on evolutionary software improvement [6]. Theoretical analysis is a rare thing even in the neighbouring discipline of automated software testing [5].

Our approach towards closing the gap between theory and practice, which we undertake in this paper, is to analyse small but realistic problems from the area of industrial control software synthesis, and to develop algorithms which overcome problems that are specific to this field. We chose a particular domain of distributed control software designed according to the IEC 61499 standard [1]. This standard defines software as a network of communicating *functional blocks*, which describe the state and the behaviour of a certain micro-controller in the distributed system. We further refine ourselves to the problem of synthesis of data connections for the given set of functional blocks in order to satisfy the conditions, expressed in terms of formal verification and model checking [2], that define how the entire system should behave.

Our previous research [8] considered evolutionary synthesis of such data connections by a simple (1 + 1) evolutionary algorithm (EA), which applies a mutation operator to the current solution at each iteration and the result replaces the current solution if it is at least as good as the current one. The fitness was defined as the

number of satisfied formulae. In this paper, we aim to improve the performance of this method by analysing the fitness landscape in more detail, which reveals the weak points of the simple algorithms in this particular case and suggests the algorithmic design changes towards better convergence to the global optimum.

2 PROBLEM DESCRIPTION

We use a simple closed-loop system from an automated pick-and-place manipulator as our model problem. It consists of a pneumatic cylinder with two proportional valves, position sensor and the joystick. This system follows an Internet-of-things architecture, thus each valve and the sensor has its own micro-controller. The system must reach the point shown by the joystick. The reader is kindly asked to refer to [8] for more details regarding this system.

In order to compute the fitness function we model each function block in the UPPAAL model checker [3]. UPPAAL uses a subset of computation tree logic to specify the properties of the system. Using the UPPAAL query language allows to describe the expected behaviour of the model using path and state formulae.

Compared to the previous paper [8], which had six formulae to check, we introduce two additional formulae – a copy of an original formula with different variables, which checks similar behaviour characteristic of a different block, and a new formula which checks a different behaviour characteristic which was not considered in the original paper. We now have eight formulae to check, so the fitness function, which is the number of satisfied formulae, takes values from 0 to 8.

Each individual is a model with $n = 6$ decision variables, each representing an input of a function block, which can take values from 0 to 6, each representing a signal output. In the original paper eight inputs were present, but for this paper we removed two of them, as they were used only for the visualization in the original application, and do not affect the functionality of the system.

The relatively small search space size consisting of 117 649 individuals made it possible to precompute fitness values for each possible individual, which makes the fitness landscape analysis much easier. We give the number of individuals for each fitness value below:

- 0: 99 (0.0841%);
- 1: 2016 (1.7136%);
- 2: 15956 (13.5624%);
- 3: 20958 (17.8140%);
- 4: 30148 (25.6254%);
- 5: 36164 (30.7389%);
- 6: 12257 (10.4183%);
- 7: 47 (0.0399%);
- 8: 4 (0.0034%).

Unfortunately, the search space size is quite regular, and still too large to enable exact computation of expected running times of variations of the (1+1) EA provided that neutral mutations are accepted. Thus our methods of analysis will still be either experimental (with the use of precomputed fitness values) or coarse-grained (such as computing the average probability of getting from fitness x to fitness y).

3 LANDSCAPE ANALYSIS

Initially we evaluated the performance of the (1 + 1) EA with the modified fitness function. We consider two mutation operators:

- Uniform mutation (uf): replaces each gene of an individual with a random different value with probability of $1/n$. We additionally guarantee that at least one gene is changed.
- Single gene mutation (sng): replaces one random gene with a new value different from it.

For each mutation operator 200 runs were conducted with a budget of 10^5 fitness evaluations. The median number of fitness evaluations until the optimum is found was 2783 in the case of uniform mutation and 3193.5 for single gene mutation. The corresponding interquartile ranges were 3556.5 and 5445.25. What is more, single gene mutation failed to obtain the optimum within the budget in 14 out of 200 runs, while uniform mutation found the optimum in all cases.

The transition matrices in Table 1, computed by aggregating data for all individuals and their fitness values, show the average probability to move from an individual with fitness f_1 to an individual with fitness $f_2 > f_1$. From this we can see that, on average, it is very easy to leave the fitness values of $[0; 5]$, as the probability of leaving these fitness values is at least $6.28 \cdot 10^{-2}$, which maps to approximately 16 iterations. However, leaving the fitness of 6 is much more difficult, and requires at least about 10^3 iterations on average. Transition from the fitness of 7 to the optimum is several times faster.

The fact that the actual running times are much higher than the sum of the average transition times shows that there is a significant number of individuals from which it is much more difficult to leave towards an individual of higher fitness.

Our investigations of the individuals with the fitness values of 6 and 7 show that the majority of individuals with fitness function of 6 are far from the optimum, as presented in Table 2. We consider it to be one of the possible reasons for the low performance of the (1 + 1) EA using the single gene mutation operator. As single gene mutation always produces a new individual with Hamming distance of 1 from the parent, it seems plausible that the single gene mutation cannot escape the plateau of individuals with fitness of 6 and gets stuck. Contrast to that, the average distance from an individual with the fitness of 7 to the optimum is much smaller.

4 INTRODUCING DIVERSITY

In order to check if the considered idea that the performance of the (1 + 1) EA is hindered by the fitness plateau distant from the optimal solutions, we propose a population based algorithm with enforced population diversity shown in Algorithm 1.

The initial population consists of N randomly generated individuals. At each iteration, N child individuals are produced from each parent via the chosen mutation operator. All $(N \times N) + N$ individuals are sorted by their fitness and for the next iteration N best and most diverse individuals are chosen. The algorithm stops when an individual with the maximum fitness value of 8 is found.

In order to determine the order of the best possible effect from introducing diversity, we try to maximise diversity by an exact deterministic algorithm rather than a heuristic. In our algorithm, the population diversity is enforced via the Diversity(P, k) function. It chooses k individuals from the set of individuals P such that the sum of Hamming distances between them is maximum. For the set of individuals P we compute the matrix M of their Hamming

Table 1: Transition matrices: upper triangle for uniform mutation, lower triangle for single gene mutation.

uf	0	1	2	3	4	5	6	7	8	
0	–	$8.41 \cdot 10^{-2}$	$1.87 \cdot 10^{-1}$	$1.07 \cdot 10^{-1}$	$9.12 \cdot 10^{-2}$	$1.08 \cdot 10^{-1}$	$1.10 \cdot 10^{-2}$	$4.16 \cdot 10^{-6}$	$7.25 \cdot 10^{-8}$	8
1	$5.91 \cdot 10^{-3}$	–	$2.17 \cdot 10^{-1}$	$1.25 \cdot 10^{-1}$	$9.75 \cdot 10^{-2}$	$9.07 \cdot 10^{-2}$	$1.59 \cdot 10^{-2}$	$5.92 \cdot 10^{-5}$	$8.50 \cdot 10^{-7}$	7
2	$6.57 \cdot 10^{-5}$	$1.02 \cdot 10^{-3}$	–	$1.40 \cdot 10^{-1}$	$1.17 \cdot 10^{-1}$	$9.57 \cdot 10^{-2}$	$2.98 \cdot 10^{-2}$	$1.49 \cdot 10^{-4}$	$1.73 \cdot 10^{-5}$	6
3	$1.77 \cdot 10^{-5}$	$3.83 \cdot 10^{-4}$	$8.37 \cdot 10^{-2}$	–	$1.61 \cdot 10^{-1}$	$1.05 \cdot 10^{-1}$	$2.44 \cdot 10^{-2}$	$1.78 \cdot 10^{-4}$	$3.88 \cdot 10^{-5}$	5
4	$1.75 \cdot 10^{-5}$	$2.43 \cdot 10^{-4}$	$4.45 \cdot 10^{-2}$	$2.17 \cdot 10^{-1}$	–	$1.61 \cdot 10^{-1}$	$3.76 \cdot 10^{-2}$	$2.28 \cdot 10^{-4}$	$1.66 \cdot 10^{-5}$	4
5	$6.49 \cdot 10^{-5}$	$1.96 \cdot 10^{-4}$	$2.38 \cdot 10^{-2}$	$1.24 \cdot 10^{-1}$	$2.28 \cdot 10^{-1}$	–	$6.28 \cdot 10^{-2}$	$2.47 \cdot 10^{-4}$	$1.18 \cdot 10^{-5}$	3
6	$1.04 \cdot 10^{-5}$	$1.34 \cdot 10^{-4}$	$3.32 \cdot 10^{-2}$	$1.08 \cdot 10^{-1}$	$1.47 \cdot 10^{-1}$	$2.03 \cdot 10^{-1}$	–	$5.06 \cdot 10^{-4}$	$3.42 \cdot 10^{-5}$	2
7	0.00	$6.89 \cdot 10^{-5}$	$1.03 \cdot 10^{-2}$	$9.60 \cdot 10^{-2}$	$1.12 \cdot 10^{-1}$	$1.70 \cdot 10^{-1}$	$3.65 \cdot 10^{-1}$	–	$2.82 \cdot 10^{-3}$	1
8	0.00	0.00	$8.42 \cdot 10^{-4}$	$1.23 \cdot 10^{-1}$	$9.54 \cdot 10^{-2}$	$1.40 \cdot 10^{-1}$	$3.09 \cdot 10^{-1}$	$1.60 \cdot 10^{-1}$	–	0
	8	7	6	5	4	3	2	1	0	sng

Table 2: Distribution of individuals with $f(x)$ close to optimum by Hamming distance

Hamming distance	1	2	3	4	5	6
$f(x) = 7$	6	17	8	16	0	0
$f(x) = 6$	9	56	195	715	2754	8528

Algorithm 1 Population based algorithm with diversity

```
Population P ← N randomly initialized individuals
while optimal solution has not been found do
    Initialize C ← ∅
    for p ∈ P do
        for i ∈ [1; N] do
            C ← C ∪ Mutate(p)
        end for
    end for
    A ← P ∪ C
    Best ← {x ∈ A | ∀x_i ∈ A : f(x) ≥ f(x_i)}
    if |Best| ≥ N then
        P ← Diversity(Best, N)
    else
        P ← Best ∪ Diversity(A \ Best, N − |Best|)
    end if
end while
```

distances between each other: $\{m_{i,j} \in M \mid i,j \in [1,|P|] : m_{i,j} = \textsc{HammingDistance}(P_i, P_j)\}$. In order to find k most diverse individuals we have to find $k \times k$ submatrix of M, such that the sum of its elements is the maximum possible.

The number of possible sub-matrices is equal to $\binom{|P|}{k}$, thus it may be unfeasible to solve this problem for large N. To overcome this problem, we evaluate only a small set of randomly chosen combinations, determined by the parameter L of the algorithm.

Initial experiments with single gene mutation and small N showed that the algorithm tends to get stuck with the same set of parents: new children are as good as parents but closer to each other in terms of the Hamming distance. In order to ensure progress in the optimization process we implement additional age correction in the $\textsc{Diversity}(P, k)$ function: for each individual we count the number of times it was used in the parent populations a, and reduce its Hamming distance to other individuals by a factor of $a + 1$.

5 EXPERIMENTS AND RESULTS

For each combination of mutation operator, population size ($N \in [2, 18]$) and diversity enforcement (no population diversity enforced, population diversity enforced and diversity enforced with age correction) 200 runs of the proposed algorithm were made. The number of evaluated combinations in Diversity function was limited to $L = 20\,000$.

The resulting median number of fitness evaluations required to reach optimum is presented on Figure 1 for the population algorithm with uniform mutation and on Figure 2 for the algorithm with single gene mutation. Each plot presents baseline results for (1 + 1) EA using corresponding mutation operator, as well as the results for three versions of the population algorithm: algorithm without enforcing population diversity (Pop), algorithm with diversity, but without age correction (Div) and algorithm with diversity and age correction (DivAge). Closer comparison of the (1 + 1) EA with uniform and single gene mutations as well as the corresponding population based algorithm with diversity and age correction is shown on Figure 3.

It can be seen that with sufficient population size ($N > 5$) both Div and DivAge algorithms outperform their (1+1) EA counterparts. Age correction significantly improves performance of the algorithms with small population sizes, but makes almost no difference with larger values of N. The Div algorithm with small population size and single gene mutation operator fails to reach optimum within the budget of 100 000 computations, but DivAge version's performance is almost on par with the (1 + 1) EA sng algorithm.

The differences in performance between the population-based algorithm and its (1 + 1) EA counterpart were checked for statistical significance using the Wilcoxon rank-sum test with Holm correction implemented in R programming language. The p-values given in Table 3 suggest that for each version of the population algorithm there are population sizes which render it statistically better than the (1 + 1) EA at the confidence level of 0.05. The medians observed for these sizes are typically less than those of the (1 + 1) EA by roughly 1/3.

6 CONCLUSION

We have empirically analysed the fitness landscape of a practical problem of automatic generation of data connections in IEC 61499 function block applications for industrial control systems. As this problem features fitness plateaus at large genotypical distances to

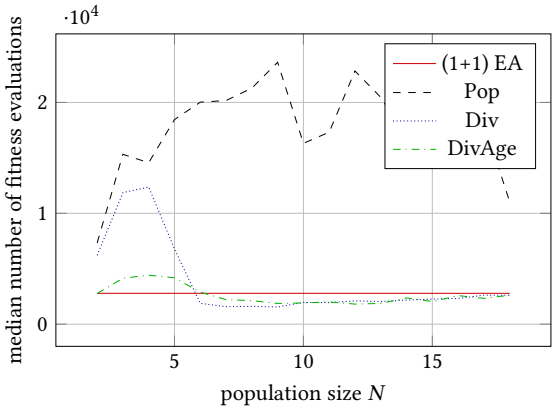

Figure 1: Results with the uniform mutation operator.

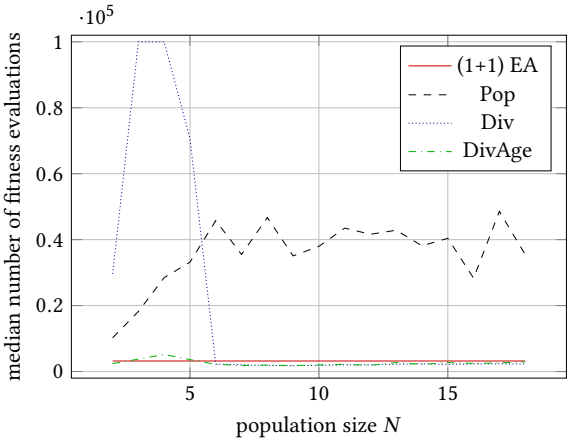

Figure 2: Results with the single gene mutation operator.

the optimum, hill climbers such as the (1 + 1) EA with various mutation operators experience problems in reaching the optimum. We introduced a population-based algorithm with a diversity preservation technique based on explicit diversity maximization, which reduced the observed running times by approximately a third for optimal choices of the population size.

In industrial applications, fitness evaluation can be expensive. Derandomized and non-heuristic mechanisms for improving various algorithmic properties are not only affordable in these conditions, but can also be desirable, since in the master-slave fitness evaluation framework the master node would otherwise stay idle most of time. In a sense, there is always a trade-off between the fitness cost and the cost of evolutionary operations. We showed that even solving an NP-hard problem to improve population diversity can enhance the overall performance.

Finally, we observe that our population-based algorithm seems to have an optimal range for the population size. Making this parameter self-adjusted can be another good showcase for introducing a theoretical investigation into a practice-oriented algorithm.

This work was financially supported by the Government of Russian Federation (Grant 08-08).

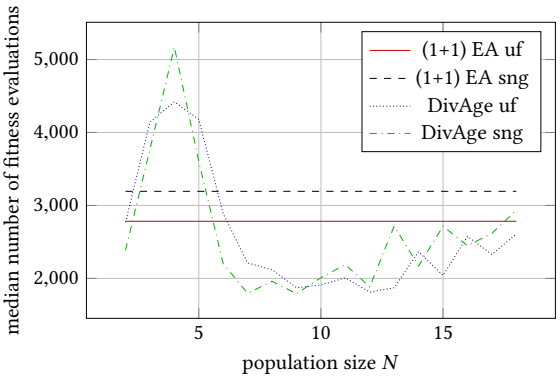

Figure 3: Closer comparison of best algorithms.

Table 3: The p-values for algorithms compared to the corresponding (1 + 1) EA. The cells with values $p < 0.05$ are highlighted in gray.

N	Div uf	DivAge uf	Div sng	DivAge sng
2	1.00	1.00	1.00	$5.47 \cdot 10^{-1}$
3	1.00	1.00	1.00	1.00
4	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00
6	1.00	1.00	1.00	$2.19 \cdot 10^{-1}$
7	$3.39 \cdot 10^{-4}$	1.00	1.00	$3.11 \cdot 10^{-6}$
8	$6.23 \cdot 10^{-4}$	$2.18 \cdot 10^{-1}$	$1.03 \cdot 10^{-1}$	$4.21 \cdot 10^{-4}$
9	$1.46 \cdot 10^{-4}$	$1.35 \cdot 10^{-3}$	$3.82 \cdot 10^{-4}$	$2.09 \cdot 10^{-8}$
10	$3.15 \cdot 10^{-2}$	$1.54 \cdot 10^{-2}$	$5.16 \cdot 10^{-4}$	$2.31 \cdot 10^{-6}$
11	$1.92 \cdot 10^{-1}$	$2.34 \cdot 10^{-2}$	$1.81 \cdot 10^{-3}$	$6.20 \cdot 10^{-4}$
12	$1.79 \cdot 10^{-1}$	$1.86 \cdot 10^{-4}$	$2.01 \cdot 10^{-3}$	$4.76 \cdot 10^{-7}$
13	$4.50 \cdot 10^{-2}$	$1.66 \cdot 10^{-2}$	$1.23 \cdot 10^{-2}$	$1.63 \cdot 10^{-1}$
14	$3.73 \cdot 10^{-1}$	1.00	$5.12 \cdot 10^{-2}$	$2.14 \cdot 10^{-2}$
15	1.00	1.00	$9.34 \cdot 10^{-3}$	$5.48 \cdot 10^{-1}$
16	1.00	1.00	$1.83 \cdot 10^{-1}$	$5.45 \cdot 10^{-2}$
17	1.00	1.00	$1.01 \cdot 10^{-1}$	$2.56 \cdot 10^{-1}$
18	1.00	1.00	$3.17 \cdot 10^{-1}$	1.00

REFERENCES

[1] 2012. *International Standard IEC 61499-1: Function Blocks – Part 1: Architecture, 2nd ed.* International Electrotechnical Commission, Geneva.

[2] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. 2008. *Principles of model checking.* MIT press.

[3] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Petterson, Wang Yi, and M. Hendriks. 2006. UPPAAL 4.0. In *Third International Conference on the Quantitative Evaluation of Systems - (QEST'06).* 125–126. https://doi.org/10.1109/QEST.2006.59

[4] Mark Harman, S. Afsin Mansouri, and Yuanyuan Zhang. 2009. *Search Based Software Engineering: A Comprehensive Analysis and Review of Trends, Technologies and Applications.* Technical Report TR-09-03. Department of Computer Science, King's College London.

[5] Joseph Kempka, Phil McMinn, and Dirk Sudholt. 2013. A theoretical runtime and empirical analysis of different alternating variable searches for search-based testing. In *Proceedings of Genetic and Evolutionary Computation Conference.* 1445–1452.

[6] William B. Langdon and Gabriela Ochoa. 2016. Genetic improvement: A key challenge for evolutionary computation. In *Proceedings of Congress on Evolutionary Computation.* https://doi.org/10.1109/CEC.2016.7744177

[7] William B. Langdon, Nadarajen Veerapen, and Gabriela Ochoa. 2017. Visualising the Search Landscape of the Triangle Program. In *European Conference on Genetic Programming.* Number 10196 in Lecture Notes in Computer Science. 96–113.

[8] Vladimir Mironovich, Maxim Buzdalov, and Valeriy Vyatkin. 2017. Automatic generation of function block applications using evolutionary algorithms: Initial explorations. In *Industrial Informatics (INDIN), 2017 IEEE 15th International Conference on.* IEEE, 700–705.