Self-adaptive Crossover in Genetic Programming: The Case of the Tartarus Problem

Thomas D. Griffiths(✉) and Anikó Ekárt

Aston Lab for Intelligent Collectives Engineering (ALICE), Aston University, Aston Triangle, Birmingham B4 7ET, UK
{grifftd1,a.ekart}@aston.ac.uk

Abstract. The runtime performance of many evolutionary algorithms depends heavily on their parameter values, many of which are problem specific. Previous work has shown that the modification of parameter values at runtime can lead to significant improvements in performance. In this paper we discuss both the '*when*' and '*how*' aspects of implementing self-adaptation in a Genetic Programming system, focusing on the crossover operator. We perform experiments on Tartarus Problem instances and find that the runtime modification of crossover parameters at the individual level, rather than population level, generate solutions with superior performance, compared to traditional crossover methods.

Keywords: Self-adaption · Crossover · Tartarus problem

1 Introduction

In the field of Evolutionary Algorithms and specifically Genetic Programming, it is widely accepted that the on-the-fly modification and adaptation of parameters values at runtime can lead to improvements in performance [1]. This process of modifying parameter values can be conceptualised into two distinct processes, the first: *'when'* to modify and the second: *'how'* to modify.

A common approach for deciding '*when*' to trigger the parameter modifications, whether they be deterministic [2] or probabilistic [3], is decided by use of a pre-determined schedule or fixed time interval; we refer to these as *episodic modifications*. The primary benefit of episodic methods is that they allow for a regular and predictable sequence of parameter modifications to be performed over time without the need for any further interaction.

However, the rigid nature of this approach presents several drawbacks when utilised on dynamic or multi-dimensional optimisation problems, such as the Tartarus Problem (TP). An alternative to *episodic modification* is to create a mechanism which provides a continual opportunity to modify parameter values at any time; we refer to this as *continuous modification*. We therefore propose the introduction of a self-adaptive crossover bias method, allowing for the continual modification of individual crossover parameters at runtime.

A. Auger et al. (Eds.): PPSN 2018, LNCS 11101, pp. 236–246, 2018.
https://doi.org/10.1007/978-3-319-99253-2_19

The process of deciding *'how'* the parameter value is to be modified is often more complex, this can be divided into two smaller, sequential sub-tasks:

- Deciding the mechanism by which the parameter values are modified,
- Calculating the magnitude of the parameter value modifications.

This division between the mechanism and the magnitude allows for the methods by which the modifications are made and the impact of those modifications, to be tuned and controlled separately at runtime. There exist several different approaches to deciding *'how'* the parameter values should be modified that are utilised in Genetic Programming, these can be classified as either *deterministic*, *adaptive* or *self-adaptive*. An outline and comparative taxonomy of these approaches is presented in Sect. 2.

It is hypothesised that allowing the Genetic Programming system to trigger the parameter modifications '*as and when they are required*' will reduce the number of ineffective adaptations being executed, increasing efficiency and allowing for convergence to an optimal solution. The proposed self-adaptive crossover bias will create a more *continuous* parameter value modification process, which is more flexible compared to the rigid, traditional *episodic* approach, leading to an increase in solution performance.

In this paper we discuss the differences between adaptive and self-adaptive parameter modification and implement a self-adaptive crossover bias method in genetic programming. The paper is organised as follows: Sect. 2 discusses and defines the differences between adaptation and self-adaptation, presenting a taxonomy of the two parameter modification approaches. Section 3 describes the Tartarus Problem and the experimental setup. Section 4 presents the proposed self-adaptive crossover operator and compares the performance with that of individuals utilising a standard crossover operator. Finally, Sect. 5 addresses conclusions and future research aspirations on the topic.

2 Parameter Modification Approaches

The parameter modification approaches utilised in genetic programming can be generally classified into one of three categories [1], being *deterministic*, *adaptive* or *self-adaptive*[1] in nature. The characteristics, flexibility and complexity varies widely between the three categories of approach.

Deterministic Parameter Modification – The parameter value is modified on a *global level* according to a fixed, pre-determined rule. The modification receives no feedback from, and is not influenced by, the current status of the search [4,5].

Adaptive Parameter Modification – The parameter value is modified on a *global level* according to a mechanism, which receives input from, and is at least partly influenced by, the status of the search [6].

[1] The descriptive terms 'Adaptive' and 'Self-Adaptive' are used in the broad general context of Evolutionary Computation. These terms have distinct meanings in fields such as Artificial Life; based on strict Ecological and Psychological definitions.

Self-adaptive Parameter Modification – The parameter value is modified on an *individual level*, where the parameters are encoded into the genome of an individual in some form. The parameters undergo the same processes of mutation and recombination as the individuals themselves. The modification of these parameter values is coupled with the status of the search [7].

In Adaptive Parameter Modification (APM) the mechanism by which the parameter values are modified is defined in advance, leading to explicit exogenous parameter modification. The performance of APM is only as good as the information that it receives from the environment, care must be taken to ensure that the information received is applicable to the selected parameters.

Conversely, in the Self-Adaptive Parameter Modification (SAPM) the way in which the parameter values are modified is entirely implicit. In this approach the mutation and recombination processes of the evolutionary cycle itself are used and exploited. The parameter values are embedded in the representation [8], leading to an implicit endogenous parameter modification. The performance of SAPM is closely linked to the choice of evolutionary operators, therefore effective operator choice is essential. Table 1 outlines a taxonomy of approaches comparing the three methods of *deterministic*, *adaptive* and *self-adaptive* parameter modification.

Table 1. Taxonomy of parameter modification approaches. (× indicates a relationship.)

		Deterministic	APM	SAPM
Affected by	Explicitly-defined mechanisms	×	×	
	State of the search		×	×
	Operator selection			×
Modifies	Population level parameters	×	×	
	Individual level parameters			×

The taxonomy outlined in Table 1 allows for the comparison of the different parameter modification approaches to be made. Each approach is *affected by* a selection of factors, both internal and external, which influence the overall effectiveness and performance. The *self-adaptive* approach leads to *modifications* to be made at the *individual* level, in contrast the *adaptive* and *deterministic* approaches both lead to *modifications* to be made at the *global* level.

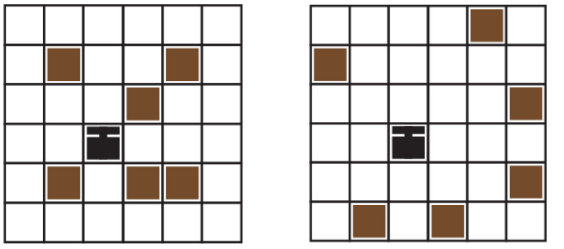
3 The Tartarus Problem

The Tartarus problem is a grid-based optimisation problem [9], which we introduced as a genetic programming benchmark [10]. The problem was chosen due to the fact that it satisfies many of the desirable benchmark characteristics outlined

by White et al. [11]. One of the most important characteristics of an effective benchmark problem is *tunable difficulty* [12], the ability to create several problem instances with a tunable and predictable level of difficulty.

A Tartarus instance comprises of an enclosed, non-toroidal $n \times n$ grid, a set number of movable blocks B and a controllable agent, as shown in Fig. 1(a). Unlike in other grid based problems, such as the Lawnmower problem [13], the agent is initially unaware of its location and orientation within the environment. The agent receives input from eight sensors, allowing it to detect both blocks and the environment boundary in the surrounding eight grid-squares. The goal is to locate and move the blocks to the environment boundary, as shown in Fig. 1(b). At the end of a run, the environment is analysed and the agent is awarded a score, the fitness score, based on its progress in achieving the goal. The agent is able to change its state by executing a finite number of actions m, chosen from the following three actions:

(1) *turn left*, **(2)** *turn right*, **(3)** *move forwards one square.*



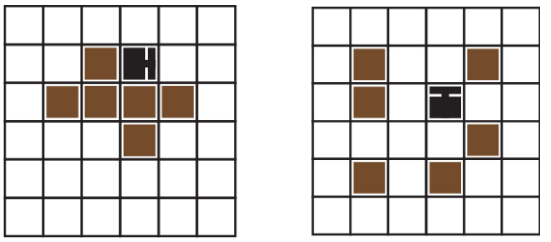
(a) Example Initial State (b) Example Final State

Fig. 1. Example states for the canonical 6×6 Tartarus instance.

3.1 Improved State Evaluation

We previously suggested that the original method of evaluating the state of Tartarus instances was insufficient to capture the progress of the agent [10]. The original method of state evaluation only rewarded individuals who had pushed blocks all the way to the edges of the grid. This binary success or fail approach works well for many benchmark problems where the absolute score achieved by a candidate solution is the only desired success measure. However, for GP, rewarding part-way solutions is essential during evolution, so that better solutions can evolve.

For example, the concentrated instance in Fig. 2(a) is very different from the dispersed instance in Fig. 2(b). However, under the original evaluation method [9] both of these states would have the same fitness score of zero. The blocks in the dispersed instance are visibly closer to the edge of the grid when compared to the blocks in the concentrated instance. Specifically, it would take a total of 32 movement actions to move the blocks to the edge of the grid in the concentrated instance (a), but only 27 actions to move the blocks in the dispersed instance (b).



(a) Concentrated Instance (b) Dispersed Instance

Fig. 2. Comparison of concentrated and dispersed instances

We proposed an improved method for evaluating the state of a Tartarus instance that utilises a more granular approach, rewarding blocks which have moved part-way as well as blocks which have been moved completely to the edge. This is done by calculating how close each block is to the edge of the environment, resulting in the following state evaluation *SE* [10]:

$$SE = 6 - \frac{12 \sum_{i=1}^{B} d_i}{\mathrm{B}(n-1)}, \tag{1}$$

where B is the total number of blocks, n is the size of the grid and d_i is the distance of block i from an edge in the given instance. The value range of *SE* is consistent with the range of the original evaluation method; 0–6, allowing for direct comparison between the canonical 6×6 grid and larger instance sizes.

A score near 0 would indicate that the agent has made no progress towards moving the blocks to the edge of the environment, or in some cases moved blocks closer to the centre in a counterintuitive manner. A score of 6 would indicate a state where all of the blocks in the instance have been successfully moved to the edges of the environment by the agent. At the end of each generation the agents use their resultant *SE* value as their fitness score.

4 Self-adaptive Crossover Operator

For a TP instance of size $n = 6$, an agent at the standard level of difficulty, $\mathrm{D} = 1$, is allowed $m = 80$ movement operations [10]. For linear GP these operations are encoded as a genome containing m alleles, with each allele corresponding to one of the three possible agent actions outlined in Sect. 3.

For each individual genome, the aggregate number of *move forward one square* ($\mathrm{A_F}$), *turn left* ($\mathrm{A_L}$) and *turn right* ($\mathrm{A_R}$) alleles are counted, these values make up the genome composition. It is important to note that this composition of the genome does not take into consideration the sequential order of the alleles, but only the aggregate number of each type of allele present. *We hypothesise that for each Tartarus instance there exist optimal compositions of agent actions,*

which, when used to seed future individuals, will likely lead to an increase in solution performance.

As the composition of the individual genome is made up of three primary components, it can be viewed on a ternary plot in order to visualise the magnitude of the components present in the composition. A population of 1000 individuals were generated, corresponding to 697 unique genome compositions. The population was executed across 100 different TP instances of size $n = 6$, and the resultant fitness scores averaged.

Analysis of the data showed there to be a clear divide in the average fitness scores between individuals who have an approximately equal composition, from the central region of the ternary plot, and those individuals with an uneven composition, who lie on the periphery. 80% of the compositions fall within the central region; here the variation in average fitness scores is low, with values ranging from 3.3–3.75, as shown in Fig. 3.

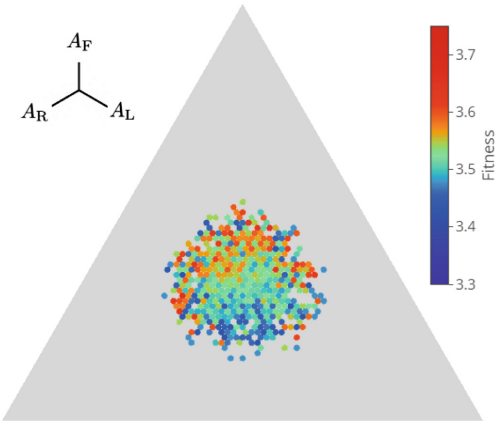
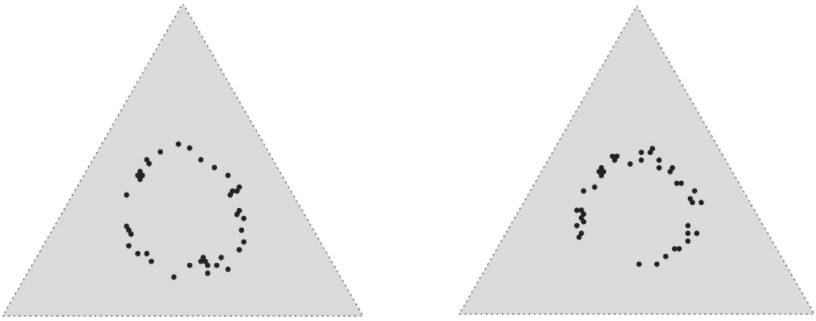


Fig. 3. The central 80% of compositions

However, for individuals who have an uneven composition, who fall outside of this central region; the variation in average fitness scores is high, with values ranging from 2.6–4.6. This is highlighted most clearly in Fig. 4, showing the bottom 10% and the top 10% of individual compositions in terms of averaged fitness score. It can be seen that the top 10% and bottom 10% of compositions exist in two defined bands surrounding the central region.

Upon further investigation, it was found that increasing the number of *move forward* instructions in the genome, relative to number of *turn left* and *turn right* instructions, leads to a noticeable increase in fitness score. This can be seen most notably in Fig. 3; there is a defined change in fitness scores between the compositions in the uppermost section of the plot, with higher A_F, and the compositions in the lower section of the plot, with lower A_F.



(a) Bottom 10% of Compositions (b) Top 10% of Compositions

Fig. 4. Top and bottom 10% of compositions

This is expected behaviour, it is intuitive that compositions containing a high proportion of *turn left* or *turn right* instructions would simply spin around and not move far from the initial grid location, therefore having a lower score. In a similar manner, compositions containing a lower but approximately equal number of *turn left* and *turn right* instructions, the impact of these would effectively be cancelled out, resulting in a lower score.

We postulated that it would be possible to use this information to design a self-adaptive crossover bias in order to exploit the changes in expected fitness for different areas of the composition space. This would allow for the introduction of bias in the generation of new individuals by favouring offspring with certain compositions. As it is the output of the chosen crossover operator that is affected, the process of generating new individuals, the proposed self-adaptations can be incorporated and utilised alongside any traditional crossover approach.

In order to do this, the crossover operator was parameterised at the individual level. Each individual was assigned a random target A_F value T'_g during initialisation, in the range $A_F = \frac{2}{5}m - \frac{4}{5}m$, from where the value can adapt during evolution. The process of adapting the target value is divided into two stages. In the first stage, the '*how*' stage, the target value T'_g is updated at the end of generation g, during the evaluation step, according to the performance of the individual in comparison to previous evaluations:

$$T'_g = \begin{cases} T_g & \textbf{if} \quad F_g > F_{g\text{-}1} \\ T_g + R_g & \textbf{if} \quad F_g \leq F_{g\text{-}1}, \end{cases} \tag{2}$$

where T_g is the current target value, F_g and $F_{g\text{-}1}$ are the current and previous fitness scores of the individual and R_g is a uniformly distributed random value in the interval:

$$\left[-\frac{A_{F\ g}}{T_g}, \frac{A_{F\ g}}{T_g} \right],$$

where $A_{F\,g}$ is the current A_F value in generation g. In the second stage, the '*when*' stage, the probability of triggering the self-adaptation and implementing the new target value T'_g into the crossover parameters of the individual is calculated:

$$P(T'_g) = \frac{T_g}{G \cdot B \cdot T'_g}, \tag{3}$$

where G is the number of generations without an improvement in the fitness score of the individual and B is the number of blocks present in the instance.

The probability $P(T'_g)$ is influenced by both the number of generations G since the actions of the individual led to an improvement in fitness score and the change between the target values T_g and T'_g. As G increases or the difference between T_g and T'_g increases, the chance that the self-adaptation will be triggered becomes greater. If the self-adaptation is triggered, at the start of the next generation, T_{g+1} will be initialised with the current value T'_g.

A population of 100 individuals was generated, each with a genome containing a random mixture of $m = 142$ alleles. These individuals were tested on 100 instances of size $n = 8$. The target A_F values T chosen by the individual at each generation g were averaged. As shown in Fig. 5, over time, the target values chosen by the individuals within the population stabilise and converge to a small range of values. Figure 5 also shows the maximum and minimum T values within the population, over generations, until they converge.

It can be seen that by generation 18 the target values of all the individuals within the population have converged to approximately $A_F = 95$, for an instance of size $n = 8$. This indicates that allowing for the self-adaptation of the target value T leads to the creation of a crossover operator favouring individuals with compositions with close to optimal A_F values. From Fig. 5 we can conclude that an A_F value close to the optimal value is found.

The utilisation of the proposed self-adaptive crossover bias leads to an increase in both the *overall solution performance* and the *rate of solution improvement* in the Tartarus Problem. In Fig. 6 the performance of the self-adaptive crossover bias, averaged over 20 different TP instances of size $n = 8$, is plotted against the performance using standard canonical crossover. The range in fitness values present in the population at each generation is shown by the shaded areas, with the average score shown as solid lines.

The occurrences of self-adaptations being triggered within a population plotted against the changes in maximum fitness score, on a generation by generation basis is shown in Fig. 7. The plot shows that there is a strong correlation between the occurrence of self-adaptations within the population and an increase in the maximum fitness score achieved. We can conclude that the mechanisms by which the self-adaptation is calculated and triggered are effective, improving the performance of individuals in the population through the modification and manipulation of evolutionary pressures.

Between generation 11 and generation 12, 32% of the individuals in the population triggered self-adaptations of their target A_F value T_n. This led to an increase of 0.5 in the maximum fitness score of the population, bringing it

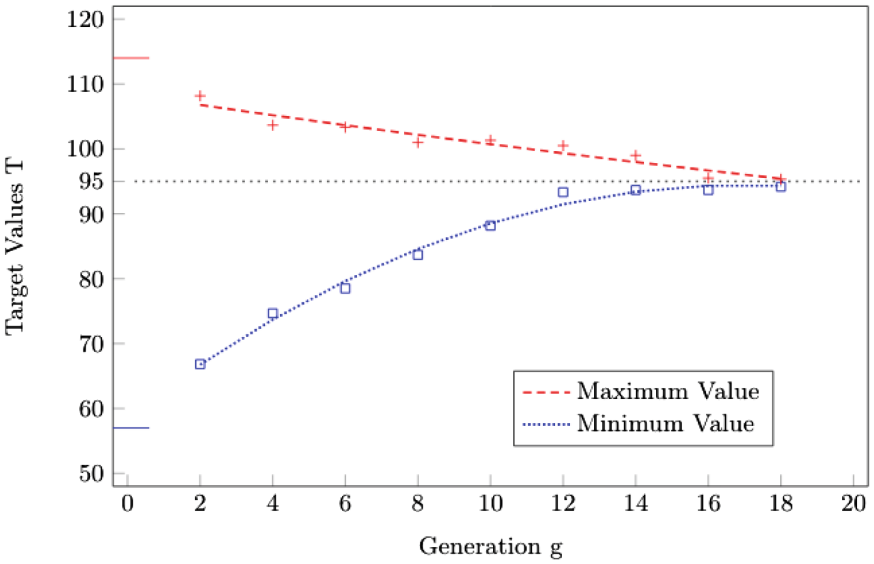


Fig. 5. Convergence of target A_F value T within the population

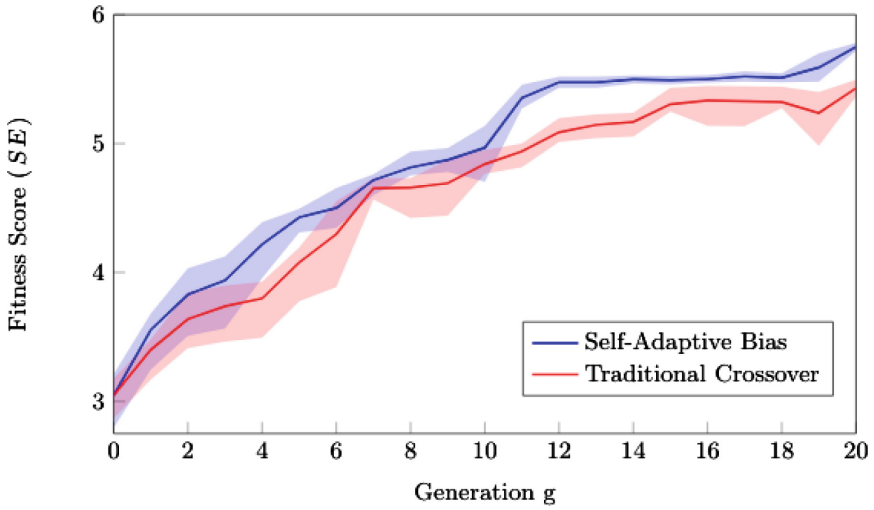


Fig. 6. Comparison between self-adaptive bias and traditional crossover

from 4.5 to 5.0. This is a substantial increase in the maximum fitness score of the population, a direct consequence of the self-adaptations carried out by the individuals.

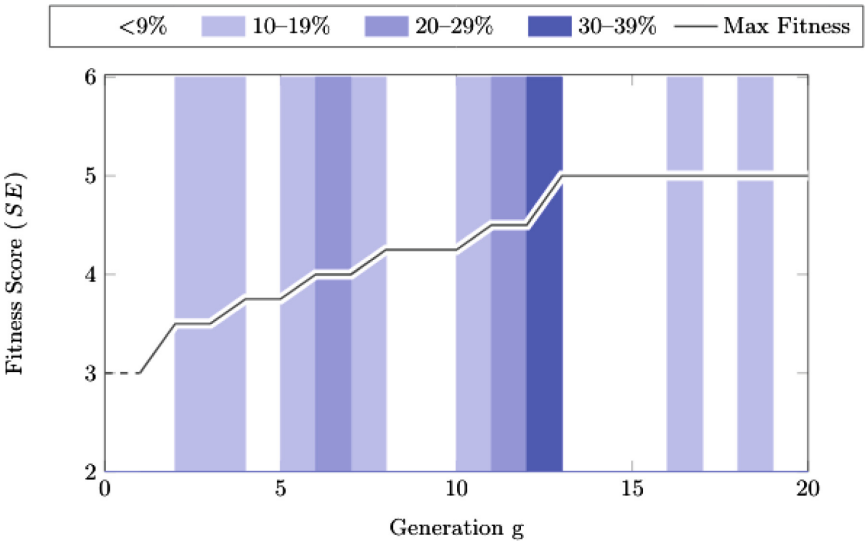


Fig. 7. Occurrences of self-adaptation and the maximum fitness score.

5 Conclusion

In this paper we outlined a novel approach to introducing *self-adaptation* into a crossover operator bias at the *individual level*.

The self-adaptation is triggered by the individual *as and when* required on a continual basis, rather than according to a pre-defined schedule or episodic time interval. The introduction of bias into the crossover operator, favouring offspring with certain compositions leads to convergence to solutions with higher average fitness scores.

We demonstrated that the individuals within the population were able to converge on a target parameter to be used by the crossover operator bias. This crossover bias was successfully utilised in order to generate solutions with higher average fitness score, when compared to solutions utilising traditional crossover operators.

The next step is to concentrate on testing the robustness of the proposed self-adaptation mechanism. Work will be conducted to test the applicability of the mechanism on other benchmark problems in order to ensure that it is generalisable and flexible. The long-term aim is to adapt and improve the self-adaptive mechanism so that it may be used on real world problems and applications.

References

1. Eiben, A.E., Michalewicz, Z., Schoenauer, M., Smith, J.E.: Parameter control in evolutionary algorithms. In: Lobo, F.G., Lima, C.F., Michalewicz, Z. (eds.) Parameter Setting in Evolutionary Algorithms. Studies in Computational Intelligence, vol. 54, pp. 19–46. Springer, Berlin (2007). https://doi.org/10.1007/978-3-540-69432-8_2
2. Kirkpatrick, S., Gelatt, C., Vecchi, M.: Optimization by simulated annealing. Science **220**, 671–680 (1983)
3. Qin, A.K., Suganthan, P.N.: Self-adaptive differential evolution algorithm for numerical optimization. In: Proceedings of the 2005 IEEE Congress on Evolutionary Computation, vol. 2, pp. 1785–1791. IEEE (2005)
4. Hesser, J., Männer, R.: Towards an optimal mutation probability for genetic algorithms. In: Schwefel, H.-P., Männer, R. (eds.) PPSN 1990. LNCS, vol. 496, pp. 23–32. Springer, Heidelberg (1991). https://doi.org/10.1007/BFb0029727
5. Hansen, N, Ostermeier, A., Gawelczyk, A.: On the adaptation of arbitrary normal mutation distributions in evolution strategies: the generating set adaptation. In: Eshelman, L.J. (ed.) Proceedings of the 6th International Conference on Genetic Algorithms, ICGA 1995, pp. 57–64. Morgan Kaufmann (1995)
6. Hinterding, R., Michalewicz, Z., Peachey, T.C.: Self-adaptive genetic algorithm for numeric functions. In: Voigt, H.-M., Ebeling, W., Rechenberg, I., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 420–429. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61723-X_1006
7. Bäck, T.: The interaction of mutation rate, selection and self-adaptation within a genetic algorithm. In: Proceedings of the 2nd Conference on Parallel Problem Solving from Nature, PPSN II, pp. 85–94 (1992)
8. Dang, D.-C., Lehre, P.K.: Self-adaptation of mutation rates in non-elitist populations. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 803–813. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_75
9. Teller, A.: The evolution of mental models. In: Kinnear Jr, K.E. (ed.) Advances in Genetic Programming, pp. 199–217 (1994)
10. Griffiths, T.D., Ekárt, A.: Improving the Tartarus problem as a benchmark in genetic programming. In: McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., García-Sánchez, P. (eds.) EuroGP 2017. LNCS, vol. 10196, pp. 278–293. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55696-3_18
11. White, D.R., et al.: Better GP benchmarks: community survey results and proposals. Genet. Program. Evolvable Mach. **14**(1), 3–29 (2013)
12. McDermott, J., et al.: Genetic programming needs better benchmarks. In: Soule, T., et al. (eds.) Proceedings of the 14th International Conference on Genetic and Evolutionary Computation, GECCO 2012, pp. 791–798 (2012)
13. Koza, J.R.: Scalable learning in genetic programming using automatic function definition. In: Kinnear Jr, K.E. (ed.) Advances in Genetic Programming, pp. 99–117 (1994)