Fitness diversification in the service of fitness optimization: a comparison study

Kamil Basiukajc Maciej Komosinski Konrad Miazga

Poznan University of Technology
Institute of Computing Science
Poznan, Poland
maciej.komosinski@cs.put.poznan.pl

Abstract

Blindly chasing after fitness is not the best strategy for optimization of hard problems, as it usually leads to premature convergence and getting stuck in low-quality local optima. Several techniques such as niching or quality–diversity algorithms have been established that aim to alleviate the selective pressure present in evolutionary algorithms and to allow for greater exploration. Yet another group of methods which can be used for that purpose are *fitness diversity* methods. In this work we compare the standard single-population evolution against three fitness diversity methods: fitness uniform selection scheme (FUSS), fitness uniform deletion scheme (FUDS), and convection selection (ConvSel). We compare these methods on both mathematical and evolutionary design benchmarks over multiple parametrizations. We find that given the same computation time, fitness diversity methods regularly surpass the performance of the standard single-population evolutionary algorithm.

Keywords: evolutionary algorithms, fitness diversity, fitness uniform selection scheme, fitness uniform deletion scheme, convection selection

1 Introduction

Despite their versatility, evolutionary algorithms (EAs) [18] are – in their most basic form – often not powerful enough to find satisfying solutions in notoriously challenging domains such as evolutionary design [1, 9, 4]. In order to overcome their limitations, EAs often employ mechanisms such as niching or fitness sharing [16] which help the algorithm preserve diversity of solutions and avoid premature convergence to an unsatisfying local optimum. While not necessarily developed with that purpose in mind, *quality–diversity* algorithms [15] are in many cases also a great tool for solving difficult, multimodal, sparse and non-convex problems [2, 10].

Fitness diversity algorithms relieve the selective pressure present in EAs by searching for solutions with different fitness values simultaneously. While usually these algorithms still contain some form of selective pressure guiding the search towards better solutions, solutions do not compete with all the other solutions, but only with these of similar quality. Population structure found in these algorithms provides a scaffolding for simultaneous improvement of solutions of all fitness ranges. This approach facilitates crossing of fitness valleys and continual discovery of new local optima. The general philosophy of fitness diversity algorithms draws from many social structures found both in human cultures and ecological systems [5].

There are several algorithms which could be classified as *fitness diversity* algorithms, such as fitness uniform selection scheme (FUSS) [7, 8], fitness uniform deletion scheme (FUDS) [14, 8],

The final version of this paper appeared in *Genetic and Evolutionary Computation Conference Companion (GECCO '22)*, Boston, USA, pp. 471–474, 2022. http://dx.doi.org/10.1145/3520304.3528949

convection selection (ConvSel) [13, 11], and hierarchical fair competition (HFC) [5, 6]. So far, however, there are no papers providing a thorough comparison of fitness-diversity algorithms. As the first step towards creating such a comparison, in this work we focus on three of these methods: FUSS, FUDS and ConvSel. We compare the performance of these techniques against the standard single-population EA with tournament selection. As a testbed for this comparison we have selected three mathematical benchmarks and one evolutionary design problem of optimizing simulated three-dimensional agents to move fast in a flat environment.

2 Algorithms

Although they achieve their goal by different means, all the fitness diversity algorithms support evolution of solutions at all fitness levels. Because of this, they concentrate less on best solutions, which may lead to subpar performance on easier problems. On the other hand, they are an especially good fit for difficult, deceptive problems which may require traversing fitness valleys and continuously discovering new solution without converging to a single local optimum.

2.1 Fitness uniform selection scheme (FUSS)

Fitness uniform selection scheme (FUSS) [7, 8] abandons any direct mechanism of pressure to select more fit solutions. Instead, it draws a value from a uniform distribution $U(f_{min}, f_{max})$ where f_{min} and f_{max} are respectively the lowest and the highest fitness present in a current population, and then returns a solution the fitness of which is the closest to that value.

In a case where solutions of some specific fitness value are difficult to find due to their rarity, FUSS will sample the already found solution of a similar fitness more often. This way, even though there is no implicit selective pressure towards better solutions, they will be sampled more often assuming that better quality solutions are harder to find than low quality solutions, which is the case for most optimization problems [8].

2.2 Fitness uniform deletion scheme (FUDS)

While the fitness uniform deletion scheme (FUDS) [14, 8] aims to equalize the density of solutions of different fitness values similarly to FUSS, it does so not through selection, but rather through deletion. First, it divides the range of possible fitness values into a number of subintervals of equal length. Then, during evolution, every new solution is assigned to one of these subintervals. Whenever a solution needs to be removed from a population, it is a random solution from the subinterval which is currently the most occupied. This way, only the solutions from the more crowded fitness ranges are removed, and so the number of solutions within each subinterval should tend to be roughly equal. Due to FUDS being a *deletion* scheme, it still requires some selection scheme to be used.

2.3 Convection Selection (ConvSel)

Convection selection [13, 11] is a method of selecting solutions to new subpopulations in an island-model algorithm [17]. It is a multi-population technique, where each subpopulation is assigned solutions from a specific fitness range. In this paper, we use the *EqualWidth* policy [11] of the fitness range assignment, which divides the population into M subpopulations, each of which is assigned a disjunctive fitness range of equal width, i.e., the i-th population is assigned the range $\left(f_{min} + (i-1) \cdot \frac{f_{max}-f_{min}}{M}, f_{min} + i \cdot \frac{f_{max}-f_{min}}{M}\right]$. f_{min} and f_{max} are extreme fitness values in the current population, and these values are included in the lowest/highest range. Every $N \cdot R$ evaluations (where N is the size of the full population, and R is a scaling parameter) solutions migrate: all the subpopulations merge, the ranges of subpopulations are recalculated according to the fitness of the best and the worst solution in the population, and solutions are redistributed to their new subpopulations. The subpopulations can be evolved sequentially or in parallel. Although right after

the division the sizes of subpopulations may differ, in the following generations ConvSel aims to keep the sizes of all subpopulations equal to their target size. Between migrations of solutions, all the subpopulations undergo a standard evolution, where any selection scheme can be used.

The name of convection selection comes from a convection-like effect that is visible in solutions processed by this algorithm across subpopulations, where solutions of all fitness levels can improve over time, no matter if they are historically underperforming or they were only recently broken by an unfortunate mutation. This mechanism of "fixing" broken solutions can help evolution cross fitness valleys and find new, better local optima.

3 Experiments

In the experiments, four algorithms were compared: a single population standard EA (*StdEA*), an EA with the fitness uniform selection scheme (*FUSS*), an EA with the fitness uniform deletion scheme (*FUDS*) and a multi-population EA with convection selection (*ConvSel*). All algorithms used steady-state evolution with a tournament selection (except for FUSS) and a random deletion (except for FUDS and FUSS, with FUSS using FUDS as the deletion scheme [8]).

3.1 Experimental setup

Parameter name	StdEA	FUSS	FUDS	ConvSel	Values
population size	✓	✓	✓	✓	[100, 200, 500]
tournament size	✓		✓	✓	[3, 5, 7]
crossover prob.	✓	✓	✓	✓	[0.5, 0.75]
mutation prob.	✓	✓	✓	✓	[0.25, 0.5]
M				✓	[5, 10]
R				✓	[10, 25]
α (math only)	✓	✓	✓	✓	[0.2, 0.4]

Table 1: The values of parameters used in the experiments. For each parameter, the methods using it are marked. Parameter α was used only in mathematical benchmarks.

FUDS requires an estimation of upper and lower bounds of the obtainable fitness values. For the mathematical benchmark functions, their theoretical maximum and minimum values were known in advance. For the problem of velocity optimization, these bounds were determined based on the results of the experiments involving the remaining methods.

For every combination of the algorithm, its parameter values (Table 1) and the fitness function, 30 (for the mathematical benchmarks) or 10 (for evolutionary design) independent evolutionary runs were conducted. Then, for every combination of the parameters, the mean and the standard deviation were calculated over all the runs. To ensure a fair, parameter-agnostic comparison of the methods, at each moment during evolution, only the parametrizations yielding the best average results at that moment were compared between the algorithms. The evolutionary runs were terminated after evaluating $5 \cdot 10^4$ solutions in the case of mathematical benchmarks, or after $7 \cdot 10^5$ solutions for the evolutionary design problem.

3.2 Mathematical fitness functions

The minimization problems used were Drop-wave function (Dwf), Schaffer function N. 2 (SchfN.2), and Schaffer function N. 4 (SchfN.4).

The solutions in the mathematical benchmarks are represented as vectors of real numbers, with the initial populations starting with random solutions drawn uniformly from the ranges of variables pre-specified for each benchmark separately.

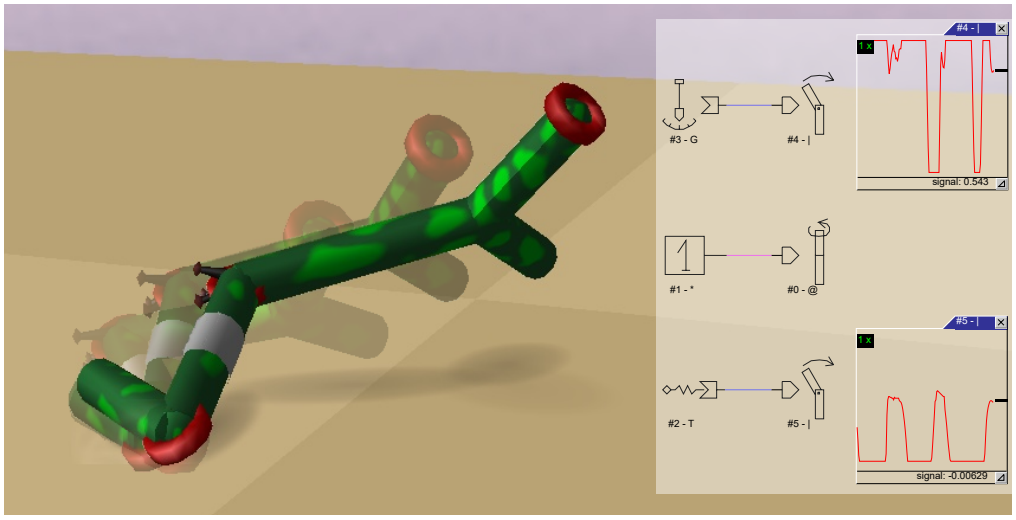

Figure 1: An example of a jumping agent encoded by the genotype `LRX[@, 1:8.594] Q(LLLLRcLRcX[*][T]LLqfX[G] [|, p:0.902, -1:12.765]FX[|, -3:0.934], X)`. This sample agent was evolved to maximize velocity with a small penalty for the length of the genotype to minimize redundancy.

Gaussian mutation was used with a standard deviation set for every variable as a fraction (1%) of its allowed range. The crossover operator used was implemented in the DEAP library [3] as the cxBlend() function. The α parameter defines the extent to which new values of variables can be drawn on both sides of the values of parents, with $\alpha > 0$ allowing for sampling new values beyond the range of the parent values.

3.3 Evolutionary design fitness function

To test and compare the performance of fitness diversification techniques on a more diverse set of fitness functions, apart from continuous mathematical benchmarks we also included an optimization problem of a different nature: an evolutionary design task. In this case, the optimization concerned three-dimensional structures equipped with sensors, actuators and a neural network. The goal was to optimize agents that move as fast as possible in a simulated, flat environment. The agents were built of elastic rods and their structure was encoded by a string of symbols. This optimization problem is very hard as it includes both a combinatorial component (discrete parts of the body) and a continuous component (weights in the neural network and neuron properties). Both physical structure and its control system are highly interdependent, and even when treated independently, each of these components is responsible for multiple local optima in the fitness landscape.

The sequence of symbols in the genotype encoded both the body and the neural network according to the *f1* genetic encoding in the Framsticks simulator (`http://www.framsticks.com/a/al_genotype`). Each neuron, sensor, and effector was encoded separately as a substring in square brackets. An example of an agent in simulation along with its genotype is shown in Fig. 1.

Mutation changed individual aspects of the genotype. The mutation operator was designed to be neutral, i.e. to avoid introducing biases; probabilities of adding and removing a given element were equal. Two-point crossing over was used, but while the cut points in the first parent were chosen randomly (avoiding cutting inside neuron descriptions), in the second parent the cut points were chosen proportionally, so if similar parents were crossed over, their corresponding parts would be exchanged during crossover. In a specific case where identical parents were crossed over, this operation would not introduce any changes.

To estimate the velocity of an agent (i.e., fitness), the agent was simulated for 10 000 simulation steps. However, before this period, a stabilization phase occurred where the agent was simulated until the displacement of its center of gravity was lower than 0.01 during consecutive 100 simulation

	Dwf		SchfN.2		SchfN.4	
Method	Mean	Std	Mean	Std	Mean	Std
StdEA	−0.9889	0.011	0.0000	0.000	0.2927	0.000
FUSS	−0.9920	0.013	0.0019	0.000	0.2944	0.000
FUDS	−0.9986	0.001	0.0000	0.000	0.2926	0.000
ConvSel	−0.9991	0.003	0.0000	0.000	0.2926	0.000
Min	**−1.0000**	**n/a**	**0.0000**	**n/a**	**0.2926**	**n/a**

Table 2: Comparison of best average values obtained by fitness diversity methods. The last row contains the global minimum of each function.

steps. This phase was introduced to avoid initial, passive movements of the body of an agent influencing their fitness.

The simulation and genetic operations were performed by the Framsticks engine [12] which was available as a native C++ Linux library (*.so file). The engine was controlled by python scripts responsible for evolution and fitness diversification techniques (`http://www.framsticks.com/trac/framsticks/browser/framspy`).

4 Results

4.1 Mathematical benchmarks

The results of the algorithms for mathematical benchmarks are summarized in Table 2. All the methods were able to find solutions with fitness close to the optimal values. Given the parameter grid specified in Table 1, ConvSel and FUDS were found to be significantly better than standard EA ($p < 0.0001$) and FUSS ($p < 0.01$) on Dwf problem. For SchfN.2 all algorithms were shown to be significantly better than FUSS ($p < 0.0001$), and for the SchfN.4, both ConvSel ($p < 0.01$) and FUDS ($p < 0.001$) were superior to FUSS. Out of all the compared methods, FUSS consistently performed the worst, which is likely to be related to its lack of a direct selective pressure, and therefore its inability to find the exact optimum. The traditional approach (StdEA) was the second worst, with two methods (FUDS and ConvSel) outperforming it even on the simple mathematical benchmarks that were considered here.

4.2 Evolutionary design

The results for the evolutionary design problem of evolving fast creatures are shown in Table 3. All tested fitness diversity methods managed to find better solutions than the traditional approach (StdEA), with FUSS and ConvSel exceptionally outperforming the other methods. While FUSS was unable to pinpoint the exact global optimum in low-dimensional mathematical benchmarks, convection selection performs well for both types of tasks, which proves the versatility of this algorithm. However, ConvSel has the highest number of parameters, which may need to be tuned for the problem at hand. Additionally, in this evolutionary design benchmark, the results of ConvSel have the highest standard deviation of all the methods (across repetitions within the best parametrization). Unfortunately, given only 10 runs per parametrization, there were no statistically significant differences between the algorithms.

StdEA		FUSS		FUDS		ConvSel	
Mean	Std	Mean	Std	Mean	Std	Mean	Std
0.0143	0.0147	0.0219	0.0141	0.0177	0.009	0.0216	0.0198

Table 3: Comparison of best average values obtained by fitness diversity methods for agent velocity maximization problem.

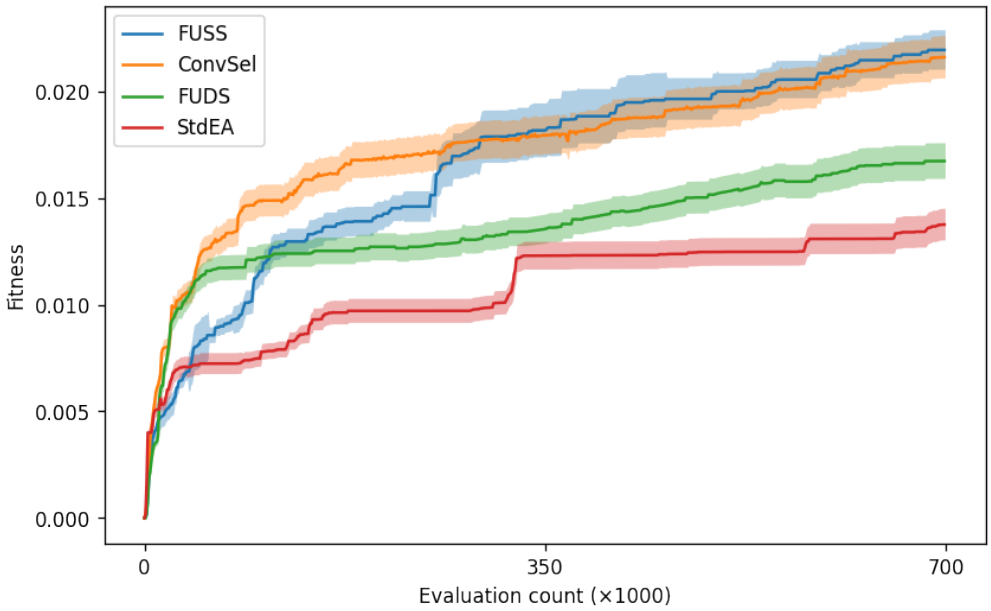

Figure 2: Comparison of the performance of fitness diversity methods on the agent velocity maximization problem. Each series shows the highest average fitness value obtainable for an algorithm for all of the tested sets of parameter values. For clarity, the band around each series represents one twentieth of the standard deviation.

Fig. 2 shows the best average fitness values obtained by each method within the tested parameter grid, independently at each step of the evolutionary run. The band around each series represents one twentieth of standard deviation. The traditional approach (StdEA) improves mainly through short bursts of rapid improvements, while fitness diversity algorithms show continuous improvement throughout the evolution. FUDS quickly ends the initial phase of quick improvement and sets then on a steady pace of improvement comparable with the other fitness diversity methods, which continues until the termination of the algorithm. The performance of FUSS and ConvSel is very similar, with both algorithms finding new better solutions at a steady rate.

5 Summary and future work

In this paper we have compared the performance of three fitness diversity methods: fitness uniform selection scheme (FUSS), fitness uniform deletion scheme (FUDS), and convection selection (ConvSel). We have shown that for a difficult problem of evolutionary design, all tested fitness diversity methods outperform single-population steady-state EA with tournament selection. Simultaneously, some of the fitness diversity methods such as FUDS and ConvSel can compete with traditional approaches. These results encourage further examination of the fitness diversity methods – they may prove to be a good, but still poorly explored way of developing better-performing and efficient EAs.

In the future we want to extend our comparison to include more algorithms, such as hierarchical fair competition (HFC) and age-layered population structure (ALPS). We would also like to extend our testbed to include more fitness functions of different types, including more demanding mathematical benchmarks, a larger number of evolutionary design problems, genetic programming problems, and combinatorial problems such as the traveling salesman problem or the set-covering problem. Results of a bigger comparison may reveal the most important mechanisms present in fitness diversity methods, and the main characteristics of the problems for which these algorithms are best suited.

Acknowledgments

The research of KB and KM was supported by the Faculty of Computing, Poznan University of Technology, through the funds provided by the Ministry of Science and Higher Education, grant no. 0311/SBAD/0713. The research of MK was supported by the Polish Ministry of Education and Science, grant no. 0311/SBAD/0726.

References

[1] Peter J. Bentley. *Evolutionary Design by Computers*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1999.

[2] Konstantinos Chatzilygeroudis, Antoine Cully, Vassilis Vassiliades, and Jean-Baptiste Mouret. Quality-Diversity Optimization: a novel branch of stochastic optimization. In *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*, pages 109–135. Springer, 2021.

[3] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[4] Philip F. Hingston, Luigi C. Barone, and Zbigniew Michalewicz. *Design by evolution: advances in evolutionary design*. Springer, 2008.

[5] Jian Jun Hu and Erik D. Goodman. The hierarchical fair competition (HFC) model for parallel evolutionary algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02*, volume 1, pages 49–54. IEEE, 2002.

[6] Jianjun Hu, Erik Goodman, Kisung Seo, Zhun Fan, and Rondal Rosenberg. The hierarchical fair competition (HFC) framework for sustainable evolutionary algorithms. *Evolutionary Computation*, 13(2):241–277, 2005.

[7] Marcus Hutter. Fitness uniform selection to preserve genetic diversity. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02*, volume 1, pages 783–788. IEEE, 2002.

[8] Marcus Hutter and Shane Legg. Fitness uniform optimization. *IEEE Transactions on Evolutionary Computation*, 10(5):568–589, 2006.

[9] Rafal Kicinger, Tomasz Arciszewski, and Kenneth De Jong. Evolutionary computation and structural design: A survey of the state-of-the-art. *Computers & Structures*, 83(23):1943–1978, 2005.

[10] Adam Klejda, Maciej Komosinski, and Agnieszka Mensfelt. Diversification techniques and distance measures in evolutionary design of 3D structures. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22), July 9–13, 2022, Boston, USA*. ACM, 2022. doi:10.1145/3520304.3528948.

[11] Maciej Komosinski and Konrad Miazga. Tournament-based convection selection in evolutionary algorithms. *PPAM 2017 proceedings, Lecture Notes in Computer Science*, 10778:466–475, 2018. doi:10.1007/978-3-319-78054-2_44.

[12] Maciej Komosinski and Szymon Ulatowski. Framsticks: Creating and understanding complexity of life. In Maciej Komosinski and Andrew Adamatzky, editors, *Artificial Life Models in Software*, chapter 5, pages 107–148. Springer, London, 2nd edition, 2009. URL: http://www.springer.com/978-1-84882-284-9.

[13] Maciej Komosinski and Szymon Ulatowski. Multithreaded computing in evolutionary design and in artificial life simulations. *The Journal of Supercomputing*, 73(5):2214–2228, 2017. doi:10.1007/s11227-016-1923-4.

[14] Shane Legg and Marcus Hutter. Fitness uniform deletion: A simple way to preserve diversity. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1271–1278, 2005.

[15] Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

[16] Bruno Sareni and Laurent Krahenbuhl. Fitness sharing and niching methods revisited. *IEEE transactions on Evolutionary Computation*, 2(3):97–106, 1998.

[17] Zbigniew Skolicki. An analysis of island models in evolutionary computation. In *Proceedings of the 7th annual workshop on Genetic and evolutionary computation*, pages 386–389, 2005.

[18] Xinjie Yu and Mitsuo Gen. *Introduction to evolutionary algorithms*. Springer Science & Business Media, 2010.