# The influence of fitness caching on modern evolutionary methods and fair computation load measurement

Michal W. Przewozniczek
Department of Computational Intelligence
Wroclaw University of Science and Technology
Wroclaw, Poland
michal.przewozniczek@pwr.edu.pl

Marcin M. Komarnicki
Department of Computational Intelligence
Wroclaw University of Science and Technology
Wroclaw, Poland
marcin.komarnicki@pwr.edu.pl

## ABSTRACT

Any evolutionary method may store the fitness values for the genotypes it has already rated. Any time the fitness is to be computed, the check may be made if the fitness for the same genotype was not computed earlier. If so, then instead of re-evaluating the same genotype, the stored value from the repository may be returned. Such technique will be denoted as fitness caching. It is easy to implement in any evolutionary method, and it minimizes the number of fitness function evaluations (FFE), which is desirable. Despite its simplicity fitness caching may significantly affect the computation load spent on fitness computation. Moreover, it may cause that the FFE will not be a reliable computation load measure.

## CCS CONCEPTS

•**Computing methodologies → Artificial intelligence;**

## KEYWORDS

Fitness Function Evaluations Number Minimization, Genetic Algorithms, Computation Load Measurement

**ACM Reference format:**
Michal W. Przewozniczek and Marcin M. Komarnicki. 2018. The influence of fitness caching on modern evolutionary methods and fair computation load measurement. In *Proceedings of Genetic and Evolutionary Computation Conference Companion, Kyoto, Japan, July 15–19, 2018 (GECCO '18 Companion),* 2 pages.
DOI: 10.1145/3205651.3205788

## 1 INTRODUCTION

The computation load used by Evolutionary Algorithm (EA) is spent on two components - the computation of fitness value and other method activities. It is frequent to assume that the overall computation load used by EA is linearly dependent on Fitness Function Evaluations number (FFE). Such assumption is not correct for a general case, the examples with detailed justification may be found in [5, 6]. Nevertheless, the expenses on fitness value computation are frequently the main computational cost of any EA. Their minimization shall be profitable for every evolutionary method. Fitness caching is one of the techniques that allow reaching this objective by storing the information about fitness values of already rated genotypes. If the fitness for the same genotype is to be re-computed, the value from the repository is returned. Thus, the FFE is minimized, and the overall computation load used by a method may (but does not have to) decrease.

We consider two fitness caching techniques - the *population fitness caching* and the *brutal fitness caching*. When the population fitness caching is used, before computing the fitness, the method checks if the population does not contain an individual with the same genotype. If so then the fitness of an individual is returned. When the brutal fitness caching is used, every genotype is stored with fitness value that refers to it. Before computing the value of any genotype, the method checks if the genotype is not already a known one. If so then the stored fitness value is returned.

## 2 THE RESULTS

The objective of the presented research is to show what may (but does not have to) happen when fitness caching is used. For instance, we show that an evolutionary method may compute fitness rarely, or may not require even a single fitness function evaluation during hours of computation. Thus, FFE may not be a reasonable computation load measure when fitness caching limits the value of FFE per iteration is zero or is close to zero.

The methods, taken into consideration were LTGA, P3, DSMGA-II, and DSMGA-IIe [1–3]. All methods were coded in C++ and were joined in one programmistic project. All the experiments were executed on PowerEdge R430 Dell server Intel Xeon E5-2670 2.3 GHz 64GB RAM with Windows 2012 Server 64-bit installed. We use the concatenations of standard and bimodal deceptive functions. The same or similar problems were used in [1–4]. The full repository with detailed results and source codes may be downloaded from https://github.com/przewooz/ffe_cache.

Below, we present the behavior of methods which are stuck at the beginning of the run. All methods except P3 that is parameterless were using the population of size 1000. The computation time was on 10 minutes. Each experiment was repeated 100 times. The first problem was a single standard order-16 deceptive function. For such problem the population of 1000 individuals is usually too small to find a global optimum (built only from '1's). Therefore, all methods get almost immediately stuck (usually after the first iteration). DSMGA-II, DSMGA-IIe, and LTGA were able to find an optimal solution only in a small part of the runs. These runs were ignored since they contained only a single method iteration and were not interesting. All other results are presented in Table 1.

As shown in Table 1, in almost all iterations of DSMGA-II and DSMGA-IIe have not used even a single FFE. The reason is as follows. At the beginning of the method, each individual is optimized with


*GECCO '18 Companion, Kyoto, Japan*
 978-1-4503-5764-7/18/07…$15.00
DOI: 10.1145/3205651.3205788

**Table 1: FFE and cache - the results for a single order-16 bimodal deceptive function and a single order-16 bimodal deceptive function**

| | Standard deceptive | | | Bimodal deceptive | | |
|---|---|---|---|---|---|---|
| | FFE cache | Pop. cache | No FFE iter. | FFE cache | Pop. cache | No FFE iter. |
| **DSMGA-II** | | | | | | |
| **mean** | 46.29% | 0.00% | 99.98% | 99.83% | 47.85% | 99.97% |
| **min** | 45.14% | 0.00% | 99.97% | 99.58% | 24.48% | 99.97% |
| **max** | 47.24% | 0.00% | 99.98% | 99.89% | 65.58% | 99.98% |
| **DSMGA-IIe** | | | | | | |
| **mean** | 46.06% | 0.00% | 99.95% | 99.70% | 48.65% | 99.98% |
| **min** | 44.85% | 0.00% | 0.00% | 99.52% | 33.15% | 99.96% |
| **max** | 47.47% | 0.01% | 99.96% | 99.80% | 54.63% | 99.99% |
| **LTGA** | | | | | | |
| **mean** | 99.98% | 85.27% | 74.92% | 99.99% | 39.22% | 87.02% |
| **min** | 99.98% | 84.83% | 63.25% | 99.98% | 37.86% | 80.03% |
| **max** | 99.98% | 85.57% | 91.90% | 99.99% | 39.56% | 92.65% |
| **P3** | | | | | | |
| **mean** | 89.28% | 9.32% | 7.98% | 74.08% | 38.56% | 0.00% |
| **min** | 85.43% | 0.00% | 5.85% | 61.08% | 28.53% | 0.00% |
| **max** | 93.86% | 30.26% | 31.69% | 97.93% | 57.17% | 74.05% |

the First Improvement Hill Climber (FIHC) [2]. Therefore, after FIHC, genotypes of all individuals contain only '0's and no linkage is detected (all individuals are the same). Thus, after the first iteration, FFE is not computed any more and the percentage of successful population cache uses is zero.

LTGA does not employ FIHC, so during the first iteration individuals are optimized by the evolutionary process and reach the state in which they are built only from '0's. After this iteration, neither fitness is computed, nor cache is used because all individuals are the same (built only from '0's). Thus, after the first iteration, no new genotypes are found, and no fitness needs to be computed. In P3 new individuals are added during its run. Thus, the number of iterations without FFE computation is significantly lower. However, the amount of cached FFE remains high - over 85% in all runs.

The second considered test case is a single bimodal order-16 deceptive function. Similarly as in the previous case, for such a problem the population of 1000 individuals is usually too small to find the optimal solution (built only from '0's or only from '1's). Therefore, all methods are caught in the local optima containing 8 '0's and 8 '1's. However, since the number of local optima is $\binom{8}{8/2}$, the population is not homogeneous as in the previous case. Similarly, DSMGA-II, DSMGA-IIe, and LTGA were usually unable to find the global optimum. The runs in which they succeeded were ignored because they were not interesting. The percentage of iterations without any fitness evaluation is high for DSMGA-II and DSMGA-IIe. It shows that if any of these two methods is stuck then it is hardly capable of searching for new solutions. The similar observation applies to LTGA. For P3 the above observation is not true. Since P3 dynamically adds new individuals during the method run, it is more likely to jump out from the local optimum. Nevertheless, the percentage of cached FFE is high for all considered methods - close to 100% for DSMGA-II, DSMGA-IIe, LTGA, and no less than 61% for P3 in all runs. The percentage of population cache use was similar for all of the considered methods. Usually, it was about 40% (LTGA and P3) or 50% (DSMGA-II and DSMGA-IIe). It is an important observation since population cache is easier to use and requires significantly less memory.

For the two considered test cases for the evolutionary methods are almost immediately stuck. For the first test case it is possible that from some moment method does not generate any fitness computation requests at all. For the second test case all methods except P3, are unable to generate any new genotypes. In such situation, the use of fitness caching significantly reduces the necessary FFE. For DSMGA-II and DSMGA-IIe the percentage of method iterations without any new fitness computation is almost 100%, for LTGA it is over 80%. Thus, it is allowed to state that fitness caching significantly optimizes the FFE used by these three methods. However, for both of the considered test cases, after the first few iterations DSMGA-II, DSMGA-IIe, and LTGA are stuck and do not require any FFE. Thus, FFE is not a reliable stop condition in these cases. Similar results were obtained for test problems using up to 2000 bits, due to the space limitations they can not be presented here.

## 3 CONCLUSIONS

In this paper, we present that a small technical issue, namely the fitness caching, may significantly affect the behavior of evolutionary methods. The main conclusions are as follows. The source code quality may significantly affect the number of FFE used by a method - if the method uses fitness caching then, the FFE may be significantly lower than without it. In general, FFE shall not be used as a stop condition without justification. For instance, FFE may be unfair if fitness caching is used. When FFE is used as a computation load measure, and fitness caching is employed, then it seems reasonable to use time as an additional stop condition, to assure that the method at least will stop after being stuck.

## ACKNOWLEDGMENTS

This work was supported by the Polish National Science Centre (NCN) under Grant 2015/19/D/ST6/03115.

## REFERENCES

[1] Ping-Lin Chen, Chun-Jen Peng, Chang-Yi Lu, and Tian-Li Yu. 2017. Two-edge Graphical Linkage Model for DSMGA-II. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*. ACM, 745–752.

[2] Brian W. Goldman and William F. Punch. 2014. Parameter-less Population Pyramid. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO '14)*. ACM, 785–792.

[3] Shih-Huan Hsu and Tian-Li Yu. 2015. Optimization by Pairwise Linkage Detection, Incremental Linkage Set, and Restricted / Back Mixing: DSMGA-II. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO '15)*. ACM, 519–526.

[4] Marcin M. Komarnicki and Michal W. Przewozniczek. 2017. Parameter-less Population Pyramid with Feedback. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17)*. ACM, 109–110.

[5] Halina Kwasnicka and Michal Przewozniczek. 2011. Multi Population Pattern Searching Algorithm: A New Evolutionary Method Based on the Idea of Messy Genetic Algorithm. *IEEE Trans. Evolutionary Computation* 15 (2011), 715–734.

[6] M. W. Przewozniczek. 2017. Problem Encoding Allowing Cheap Fitness Computation of Mutated Individuals. In *2017 IEEE Congress on Evolutionary Computation (CEC)*. 308–316.