Evaluation of Heavy-tailed Mutation Operator on Maximum Flow Test Generation Problem

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

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

ABSTRACT

The general recommendation for the mutation rate in standard-bit mutation is $1/n$, which gives asymptotically optimal expected optimization times for several simple test problems. Recently, Doerr et al. have shown that such mutation rate is not ideal, and is far from optimal for multimodal problems. They proposed the heavy-tailed mutation operator $\mathtt{fmut}_\beta$ which significantly improves performance of the (1+1) evolutionary algorithm on Jump problem and yields similar speed-ups for the vertex cover problem in bipartite graphs and the matching problem in general graphs.

We evaluate the $\mathtt{fmut}_\beta$ mutation operator on the problem of hard test generation for the maximum flow algorithms. Experiments show that the $\mathtt{fmut}_\beta$ mutation operator greatly increases performance of the (1+1) evolutionary algorithm. It also achieves performance improvement, although less drastic, on a simple population based algorithm, but hinders performance of a crossover based genetic algorithm.

CCS CONCEPTS

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

KEYWORDS

evolutionary algorithms, mutation operators, maximum flow

ACM Reference format:
Vladimir Mironovich and Maxim Buzdalov. 2017. Evaluation of Heavy-tailed Mutation Operator on Maximum Flow Test Generation Problem. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 4 pages.
DOI: http://dx.doi.org/10.1145/3067695.3082507

1 INTRODUCTION

Mutation is one of the most basic variation operators in evolutionary algorithms. In general it means a mild modification of a single parent individual. The classic example of a mutation operator called standard-bit mutation is used with a bit-string individual representation. Standard-bit mutation flips each bit of a parent bit-string independently with a certain mutation rate p_n. The general recommendation for genetic algorithms using a bit-string representation of length n is to take $1/n$ as the mutation rate. This maximizes the probability that the Hamming distance between parent and child is one and gives asymptotically optimal expected optimization times for several simple evolutionary algorithms on classic test problems [3, 14].

In the recent paper [7] Doerr et al. show that the general $1/n$ recommendation may be over-fitted to these test problems. Authors of the paper suggest that this traditional choice of the mutation rate is not ideal and one should rather choose mutation rate which maximizes rate of the largest required long-distance jump in the search space. From evaluating the $\mathrm{Jump}_{m,n}$ function authors conclude that no one-size-fits-all mutation rate exists, while finding a good mutation rate requires a deep knowledge of the fitness landscape.

In order to solve this problem Doerr et al. propose a new mutation operator called $\mathtt{fmut}_\beta$. It uses a dynamic mutation rate chosen according to a heavy-tailed distribution. The $\mathtt{fmut}_\beta$ mutation operator chooses a number α according to a power-law distribution D_n^β with (negative) exponent $\beta > 1$ and creates offspring via standard-bit mutation with a mutation rate of α/n. This operator optimizes any Jump function in a time different from optimum in only a small polynomial factor, and yields similar speed-ups for the vertex cover problem in bipartite graphs and the matching problem in general graphs. In particular runtimes for the (1+1) evolutionary algorithms employing the $\mathtt{fmut}_\beta$ mutation operator are better than ones of the classic (1+1) evolutionary algorithm by a factor of 2000.

The authors of the $\mathtt{fmut}_\beta$ mutation operator suggested to further evaluate it on more practical problems. In this paper, we evaluate the $\mathtt{fmut}_\beta$ mutation operator on the problem of hard test generation for the maximum flow algorithms, as we have successfully applied evolutionary algorithms to this problem before [4, 12, 13]. We slightly modify the $\mathtt{fmut}_\beta$ mutation operator to work with flow network representation, and evaluate its impact on performance of the (1+1) evolutionary algorithm and the population based genetic algorithm.

Results of our experiments support the claim that the $\mathtt{fmut}_\beta$ mutation operator improves performance of the (1+1) evolutionary algorithm. Moreover, it achieves performance improvement, although less drastic, even for the population based algorithms that do not use crossover. Finally, our results show that $\mathtt{fmut}_\beta$ mutation operator conflicts with crossover operator (specifically –

GECCO '17 Companion, Berlin, Germany

978-1-4503-4939-0/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3067695.3082507

uniform crossover), and actually hinders performance of the genetic algorithm when using both.

2 PROBLEM DESCRIPTION

The maximum flow problem is a well-known problem in the graph theory [5]. It is formulated as follows: given an oriented graph $G = (V, E)$ with integer capacities c_i for each edge and two specific vertices designated as the source s and the sink t one has to find *a maximum flow*, defined as a set of numbers f_i such that:

- for all edges, f_i is a non-negative integer less than or equal to the capacity of that edge c_i,
- for each vertex except s and t, the sum of f_i for the outgoing edges is equal to the sum of f_i of incoming edges,
- for the vertex s the sum of f_i for the outgoing edges minus the sum of f_i for incoming edges is maximum possible.

Maximum flow problem solutions' performance is highly input sensitive, thus it is typically hard to find hard tests for them. There are several ways to construct such tests via more conventional means [1, 11, 15], but they rely heavily on the knowledge of the tested algorithm. The optimization problem we consider is the generation of hard problem instances (or hard tests for short) for the maximum flow algorithms. A hard problem instance maximizes the running time of an solution algorithm.

There are many solution algorithms for the maximum flow problem described in the literature. Most of these algorithms are able to solve randomly generated problem instances quite quickly, but the best known upper bounds suggest that for these algorithms hard problem instances exist.

Some of the most known algorithms are:

- the Ford-Fulkerson algorithm [9], with running time $O(V \cdot E \cdot C_{\max})$, where $C_{\max}$ is the maximum capacity of an edge;
- the Edmonds-Karp algorithm [8], with running time $O(V \cdot E^2)$;
- the Dinic algorithm [6] with running time $O(V^2 \cdot E)$ which can be refined to $O(E \cdot \min(E^{1/2}, V^{2/3}))$ for unit capacities;
- the improved shortest path algorithm [2], with running time $O(V^2 \cdot E)$;
- the push-relabel algorithm [10], with running time $O(V^2 \cdot E)$.

From these algorithms, we consider only two: the Dinic algorithm and the improved shortest path algorithm (ISP). These two algorithms have been used previously [4, 12, 13], as they are efficient and have different behavior on similar problem instances.

A maximum flow problem instance, which is also an individual of an evolutionary algorithm, is a graph represented as the adjacency matrix M: a square $|V|^2$ matrix, where each element $M_{i,j}$ represents the capacity of an edge connecting vertex i to vertex j. Our previous experiments suggest that acyclic graphs are harder [4], thus for every edge it holds that $i < j$ and M is an upper triangular matrix.

The fitness function is the number of edges visited during the finding of the maximum flow by the solution algorithm. Such function is roughly proportional to the running time of the maximum flow algorithm, and was shown to be one of the most efficient in terms of fixed budget optimization results [4].

3 ALGORITHMS

The long standing recommendation for the mutation rate in a simple evolutionary algorithm for discrete optimization problem is $1/n$, where n is the length of individual. For the (1+1) evolutionary algorithm the standard-bit mutation with mutation rate of $1/n$ was shown to be the unique best mutation for the class of all pseudo-Boolean linear functions [14].

For the maximum flow problem instance represented as the adjacency matrix M as a standard mutation operator we previously used something similar to the common standard-bit mutation. With probability $1/n$, where $n = |E|$, the number of cells in M, it replaces each value $M_{i,j}$ in the upper triangular matrix with a random value bounded by the capacity limit C.

Unfortunately, as shown in the paper [7], the recommended mutation rate of $1/n$ is not the optimal one for $Jump_{m,n}$ function, or multimodal fitness landscapes in general. One of the ways to solve this problem would be to use a problem specific mutation rate, e.g. authors show that for $Jump_{m,n}$ the best mutation rate is m/n. However, in this case the number of flipped bits is heavily concentrated around m, and even a small deviation from the optimal mutation rate leads to significant increase in runtime.

Algorithm 1: The heavy-tailed mutation operator $\texttt{fmut}_\beta$ for bounded integer values.

```
Input: x ∈ [0, C − 1]^|E|
Output: y ∈ [0, C − 1]^|E| obtained from applying standard
    mutation to x with mutation rate α/|E|, where α is chosen
    randomly according to D^β_{|E|/2}
y ← x;
Choose α ∈ [1..|E|/2] randomly according to D^β_{|E|/2};
for j = 1 to |E| do
    if random([0, 1]) · |E| ≤ α then
        y_j ← random([0, C − 1]);
return y
```

The $\texttt{fmut}_\beta$ mutation operator proposed in [7] is designed to overcome the negative effect of strong concentration of the binomial distribution, while being structurally close to established way of performing mutation. We slightly modify it to work on the adjacency matrix representation of the graph instead of bit-strings. Our modified version of the $\texttt{fmut}_\beta$ mutation operator uses standard mutation with a mutation rate $\alpha/|E|$, where $\alpha \in [1..|E|/2]$ is chosen randomly on each iteration according to the power-law distribution with (negative) exponent $\beta > 1$, denoted as $D^{\beta}_{|E|/2}$. Thus, it replaces each value $M_{i,j}$ in the upper triangular matrix with a random value bounded by the capacity limit C with probability of $\alpha/|E|$. The pseudocode for this operator is given in Algorithm 1.

We evaluate performance of the $\texttt{fmut}_\beta$ mutation operator on two optimization algorithms: the (1+1) evolutionary algorithm and the population based genetic algorithm.

The (1+1) evolutionary algorithm is the most simple evolutionary algorithm. It starts with a random search point and on each iteration creates an offspring from the parent via mutation, replacing

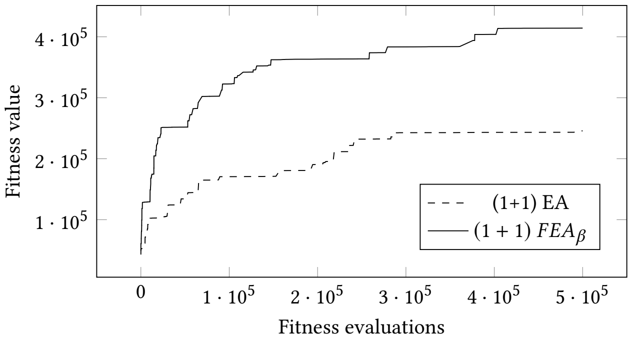

Figure 1: Comparison of $(1 + 1)$ EA and $(1 + 1)$ FEA_β, Dinic fitness function

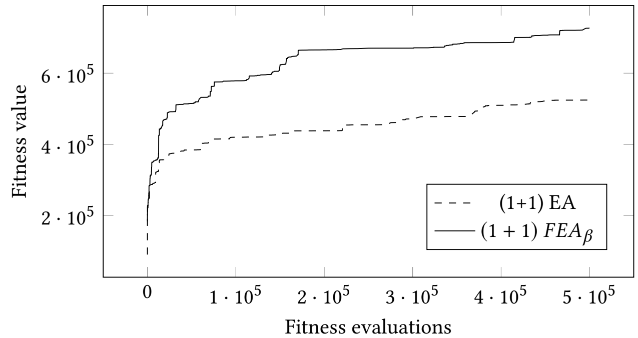

Figure 2: Comparison of $(1+1)$ EA and $(1+1)$ FEA_β, ISP fitness function

the parent unless the child has inferior fitness. We compare two variants of the (1+1) evolutionary algorithm. First variant uses standard mutation with probability $1/|E|$, we denote it as the $(1 + 1)$ EA. The other variant uses the $\texttt{fmut}_\beta$ mutation operator. We denote it as the $(1 + 1)$ FEA_β, similarly to the original notation from the paper [7].

The genetic algorithm is similar to the one from [4]. It is a rather standard genetic algorithm which uses tournament selection to choose individuals for reproduction, then applies crossover and mutation to the selected individuals, and forms new generation using elitist selection. We test two different combinations of operators for the genetic algorithm in this paper to evaluate performance of $\texttt{fmut}_\beta$ mutation operator. First, we introduce the $\texttt{fmut}_\beta$ mutation operator into the genetic algorithm without applying crossover. This allows to understand how $\texttt{fmut}_\beta$ mutation operator influences the performance of a simple population based genetic algorithm. We denote this pair as GA and FGA_β. Secondly, we introduce the $\texttt{fmut}_\beta$ mutation operator into the original version of the genetic algorithm, which uses uniform crossover. This brought up interesting results, as results show that the $\texttt{fmut}_\beta$ mutation operator conflicts with uniform crossover and actually hinders performance of the algorithm. We denote this pair of algorithms as $GA + UF$ and $FGA_\beta + UF$.

4 EXPERIMENTS AND RESULTS

In our experimental setup, as in previous papers [12, 13], the maximum number of vertices is set to 100 and the number of edges is set to 4950. Maximum capacity of an edge C is set to 8192, to keep in line with previous experiments. Generation sizes for the population based genetic algorithm is 100 individuals, with 70 child individuals generated on each iteration. For the $\texttt{fmut}_\beta$ mutation operator β is set to 1.5 and the power-law distribution implementation is taken from Apache Commons Math library. Initial population for each algorithm consists of the randomly generated upper triangular matrices, with values from range $[0, C - 1]$. For each tested combination of an optimization algorithms and maximum flow algorithm, at least 50 runs were performed with the computational budget of 500 000 evaluations.

Our first experimental run considered the $(1 + 1)$ EA and the $(1 + 1)$ FEA_β algorithms. The minimum, maximum, average and median fitness values are presented in Table 1 for both Dinic and improved shortest path algorithms. As can be seen from results, the $(1 + 1)$ FEA_β significantly outperforms the $(1+1)$ EA on both maximum flow solution algorithms. Plots for median runs in Figure 1 and Figure 2 show that the $(1 + 1)$ FEA_β is faster and is less prone to stagnation.

In our second experimental run we tried to apply the $\texttt{fmut}_\beta$ operator to the population based genetic algorithm. The minimum, maximum, average and median fitness values rounded to the nearest integer are presented in Table 2 for the Dinic algorithm and Table 3 for the improved shortest path algorithm. The impact of

Table 1: Performance of the (1+1) EA and (1+1) FEA$_\beta$, Dinic and ISP based fitness values, measured in 10^5 visited edges

	Dinic		ISP	
	(1+1) EA	(1+1) FEA$_\beta$	(1+1) EA	(1+1) FEA$_\beta$
MIN	0.90	2.38	2.11	3.88
MED	2.46	4.10	5.24	7.27
AVG	2.62	4.15	5.39	7.19
MAX	5.72	7.44	9.01	9.36

Table 2: Performance of the population based algorithms, Dinic based fitness values, measured in 10^5 visited edges

	GA	FGA$_\beta$	GA+UF	FGA$_\beta$+UF
MIN	1.34	1.87	3.50	2.68
MED	3.54	4.24	6.00	4.26
AVG	3.46	4.21	5.78	4.22
MAX	6.72	6.48	7.73	5.28

Table 3: Performance of the population based algorithms, ISP based fitness values, measured in 10^5 visited edges

	GA	FGA$_\beta$	GA+UF	FGA$_\beta$+UF
MIN	3.92	4.24	6.68	5.98
MED	5.69	6.26	7.98	7.23
AVG	5.74	6.25	7.92	7.18
MAX	8.22	8.66	9.02	8.11

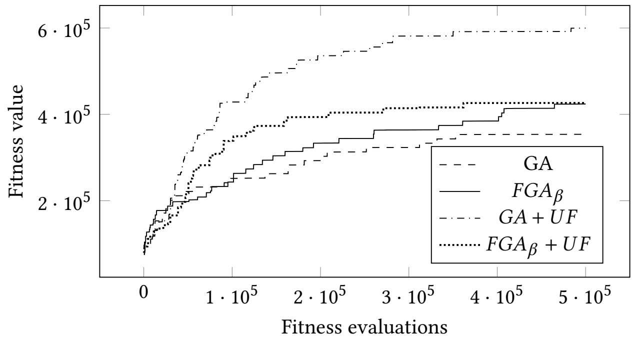

Figure 3: Comparison of genetic algorithms, Dinic fitness function

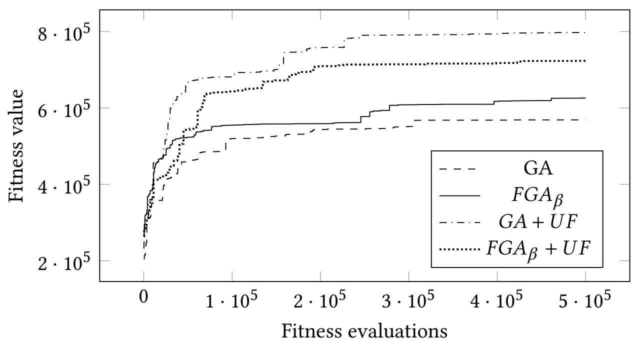

Figure 4: Comparison of genetic algorithms, ISP fitness function

Table 4: Performance change summary

Solution	Algorithm	Average Performance Change
Dinic	(1+1) EA	+58.31%
	GA w/o crossover	+21.79%
	GA w/ crossover	−26.97%
ISP	(1+1) EA	+33.39%
	GA w/o crossover	+8.89%
	GA w/ crossover	−9.33%

the $\mathtt{fmut}_\beta$ operator one the population algorithms is quite interesting. As can be seen from the results for algorithms that do not use crossover (GA and FGA_β) – the $\mathtt{fmut}_\beta$ operator does improve performance of the simple population based algorithm, although less significantly than in case of (1+1) evolutionary algorithm. On the other hand, the $\mathtt{fmut}_\beta$ mutation operator actually hinders performance of the algorithm when employed together with the uniform crossover operator – see results for $GA+UF$ and $FGA_\beta+UF$. Plots for median runs in Figure 3 and Figure 4 show that the FGA_β algorithm has almost no performance improvement from application of the uniform crossover, while the genetic algorithm with standard mutation and uniform crossover significantly outperforms other combinations of operators.

Summary of the impact of the $\mathtt{fmut}_\beta$ mutation operator on the average performance of an optimization algorithm is presented in Table 4. For each pair "standard algorithm - $\mathtt{fmut}_\beta$ counterpart" represented in the table the Wilcoxon rank sum test implemented in R programming language was performed. For all pairs the p value is significantly less than 0.05.

5 CONCLUSION

We presented an experimental evaluation of the heavy-tailed mutation operator $\mathtt{fmut}_\beta$ on the maximum flow test generation problem.

The experimental results augmented with basic statistical analysis show that the $\mathtt{fmut}_\beta$ mutation operator improves performance of the (1+1) evolutionary algorithm and the simple population based genetic algorithm compared to their standard mutation counterparts. Unfortunately, the $\mathtt{fmut}_\beta$ mutation operator seems to conflict with the uniform crossover operator, hindering performance of the genetic algorithm. Future work may be aimed to explain this phenomenon, as well as to further evaluate the $\mathtt{fmut}_\beta$ mutation operator not only on other optimization problems, but with other algorithms as well as evolutionary operators.

The source code for experiments is published at GitHub[1]. This work was financially supported by the Government of Russian Federation, Grant 074-U01.

REFERENCES

[1] 2017. DIMACS. Test Generators for the Maximum Flow Problem. (2017). http://www.informatik.uni-trier.de/~naeher/Professur/research/generators/maxflow/
[2] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
[3] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. 2010. Optimal Fixed and Adaptive Mutation Rates for the LeadingOnes Problem. In *Parallel Problem Solving from Nature – PPSN XI.* Number 6238 in Lecture Notes in Computer Science. Springer, 1–10.
[4] Maxim Buzdalov and Anatoly Shalyto. 2015. Hard Test Generation for Augmenting Path Maximum Flow Algorithms using Genetic Algorithms: Revisited. In *Proceedings of IEEE Congress on Evolutionary Computation.* 2121–2128.
[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 2001. *Introduction to Algorithms, 2nd Ed.* MIT Press, Cambridge, Massachusetts.
[6] E. A. Dinic. 1970. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.* 11, 5 (1970), 1277–1280.
[7] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. 2017. Fast Genetic Algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference.* DOI: http://dx.doi.org/10.1145/3071178.3071301 Full version available at http://arxiv.org/abs/1703.03334.
[8] Jack Edmonds and Richard M. Karp. 1972. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM* 19, 2 (1972), 248–262.
[9] L. R. Ford Jr. and D. R. Fulkerson. 1956. Maximal flow through a network. *Canadian Journal of Mathematics* 8 (1956), 399–404.
[10] A V Goldberg and R E Tarjan. 1986. A New Approach to the Maximum Flow Problem. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing.* ACM, New York, NY, USA, 136–146.
[11] Donald Goldfarb and Michael D. Grigoriadis. 1988. A computational comparison of the Dinic and network simplex methods for maximum flow. *Annals of Operations Research* 13, 1 (1988), 81–123.
[12] Vladimir Mironovich and Maxim Buzdalov. 2015. Hard Test Generation for Maximum Flow Algorithms with the Fast Crossover-Based Evolutionary Algorithm. In *Proceedings of Genetic and Evolutionary Computation Conference Companion.* 1229–1232.
[13] Vladimir Mironovich and Maxim Buzdalov. 2016. Comparative Study of Representations in the Maximum Flow Test Generation Problem. In *Proceedings of 22nd International Conference on Soft Computing MENDEL 2016.* Czech Republic, 67–72.
[14] Carsten Witt. 2013. Tight Bounds on the Optimization Time of a Randomized Search Heuristic on Linear Functions. *Combinatorics, Probability and Computing* 22, 2 (2013), 294–318.
[15] Norman Zadeh. 1972. Theoretical Efficiency of the Edmonds-Karp Algorithm for Computing Maximal Flows. *J. ACM* 19, 1 (1972), 184–192.

[1] https://github.com/vmironovich/papers/tree/master/one-ll