In Proceedings of the IEEE Symposium Series in Computational Intelligence 2017. IEEE.

How to Select a Winner in Evolutionary Optimization?

Risto Miikkulainen
Sentient Technologies
The University of Texas at Austin

Hormoz Shahrzad
Sentient Technologies
San Francisco, CA

Nigel Duffy
Sentient Technologies
San Francisco, CA

Phil Long
Sentient Technologies
San Francisco, CA

***Abstract*—In many evolutionary optimization domains evaluations are noisy. The candidates are tested on a number of randomly drawn samples, such as different games played, different physical simulations, or different user interactions. As a result, selecting the winner is a multiple hypothesis problem: The candidate that evaluated the best most likely received a lucky selection of samples, and will not perform as well in the future. This paper proposes a technique for selecting the winner and estimating its true performance based on the smoothness assumption: Candidates that are similar perform similarly. Estimated fitness is replaced by the average fitness of candidate's neighbors, making the selection and estimation more reliable. Simulated experiments in the multiplexer domain show that this technique is reliable, making it likely that the true winner is selected and its future performance is accurately estimated.** [1]

I. Introduction

In evolutionary computation applications, the goal is usually to deliver a winner, i.e. the best candidate found during the evolution. This candidate is then employed in the application, and its good performance is expected to continue. For instance, when evolving web interfaces that convert casual browsers to paying customers [1], the result is a web page that converts at the expected rate.

In most such applications, the performance of the candidate is measured during evolution through sampling [2], [3]. The candidate is shown a relatively small number of examples, or it is subjected to a small number of simulated situations, to see how well it performs. A web interface, for instance, is tested on a 1000 or so users. The performance is thus an average of these samples, i.e. a statistical estimate.

Such estimation becomes a problem when a winner needs to be selected. For example, let's say that there are 100 web interface candidates with the same true conversion rate of 4%, and let's assume the standard deviation in the estimated rate is 1%. Then roughly 2% of the time one of these candidates will have an estimated conversion rate of 6%—that is there are two candidates that seem to convert at 50% better than they actually do. With thousands of such candidates generated and tested during evolution, it is very likely that a suboptimal one will be selected as a winner, and its performance will be overestimated.

This problem is a severe instance of the multiple hypothesis problem in statistics, and the standard solution to it is Bonferroni correction [4]. It essentially reduces the confidence by a factor of $1/N$ if N candidates are being tested. Given the large number of candidates, Bonferroni correction is not helpful in evolutionary computation. However, an important aspect of evolution is that the fitness landscape is not likely to be random. A fundamental assumption in evolution is that small changes in the candidates result in small changes in fitness—that is, although the fitness landscape can be strongly nonlinear globally, it changes smoothly locally. Indeed, mutation would not work if it did not.

This paper builds on this smoothness assumption to develop an approach for selecting a winner and estimating its performance more reliably. The main idea is to calculate the average fitness in a neighborhood of the candidate, and use the neighborhood fitness instead of the candidate's own estimated fitness to select the winner and to report its performance. Through a permutation test, confidence in this estimate, the selection, and improvement over control can be measured.

This idea is evaluated in a series of experiments in the 11-Multiplexer domain. Because the true fitness of each candidate in this domain is known, these experiments allow evaluating the viability of the approach and its variations thoroughly. Neighborhood fitness is found to be reliable way to improve the winner selection and the estimate of future performance. It is therefore a promising method for improving performance of evolutionary optimization in practical applications.

II. The Neighborhood Fitness Method

The problem can be specified as follows: Given that many candidates are produced during an evolutionary run, and their performance is estimated through sampling, there is a serious multiple hypothesis problem: The candidate that seems to perform the best may have simply gotten lucky in the sampling, and as a result, will perform poorly in the future.

A solution to this problem consists of solving three subproblems:

1) Decide which one is the best candidate
2) Estimate its performance
3) Confirm that its performance is statistically significantly better that that of other candidates (including the control)

There are potentially several ways to solve these problems, including age-layering [5] and performing successive A/B tests on the best candidates [6]. The approach pursued in this paper is similar to nonparametric functional estimation [7]. It is practical in that it can be implemented as a postprocessing step to evolution: It does not affect evolution itself and it

[1] Nigel Duffy is currently at EY and Phil Long at Google.

does not require additional sampling beyond that done during evolution. This is an important advantage because in practice, evolution may have to be terminated at any time based on customer request, and a good candidate needs to be delivered without further testing. In practice such termination often happens early, before evolution has fully converged, when candidate fitnesses still vary significantly, amplifying the need for reliable selection and estimation.

The first two problems can be solved by relying on the smoothness assumption, that is, that similar candidates have similar performance. Evolutionary computation in general relies on this assumption: small mutations, i.e. local search, are assumed to result in small changes in performance. With smoothness, it is then possible to estimate the performance of a candidate by averaging the performance of all candidates in its neighborhood:

$$f_{\mathrm{N},x} = 1/k \sum_{i=1}^{k} f_{\mathrm{C},i} \tag{1}$$

where $f_{\mathrm{N},x}$ is the *neighborhood fitness* of candidate x and $f_{\mathrm{C},i}$ is the *candidate fitness*, i.e. performance estimated through sampling, of candidate i among the k candidates in its neighborhood.

Even if the candidate itself may have gotten lucky, i.e. the candidate fitness is overestimating its performance, its neighborhood is likely to contain both lucky and unlucky candidates. This neighborhood fitness, therefore, is more reliable and can be used as a more realistic estimate of the candidate's performance, and thereby to identify good candidates. The definition of a neighborhood can be based on k-nearest neighbors, as in this paper, or it can be based on a distance metric in genotype space. Since evolution converges the population around good candidates, it is likely that there is a large enough number of candidates in good neighborhoods. Thus, the neighborhood fitness can be used to select good candidates and estimate their future performance better than candidate fitness.

Note that a similar idea is commonly used in signal processing: A measurement is replaced with a weighted average of measurements in its neighborhood. This process is called "filtering" because it can be shown to provably remove noise under some conditions [8].

This approach solves the first two problems. The third problem, i.e. to confirm that the best candidate is statistically significantly better than other candidates, including the control candidate, can be solved in two steps. In the first step, the probability that the performance of the best candidate B is due to luck is estimated. That is, a permutation test [9] will be run by repeating the following steps M times:

1) Permute the samples assigned to candidates, i.e. assign +/- labels in the same proportions as seen during evolution;
2) Calculate the neighborhood fitness of each candidate;
3) Find the candidate with the best neighborhood fitness in this permutation;
4) If it is equal or better than that of candidate B, increment a counter s.

The probability p_1 that performance as good as that of candidate B could have been observed by chance, given the multiple hypotheses, is then $p_1 = s/M$.

The second step tests whether the performance of B is significantly better than that of control. Control is one specific candidate, C, and it is usually tested more than the others, so its candidate fitness can be used to estimate its performance (instead of neighborhood fitness). However, its performance is still a random variable. To estimate the probability that the performance observed for candidate B is better than that of C, two binomial distributions with the observed means are compared. This test will yield the probability p_2 that the distributions are the same. The combined probability, i.e. that candidate B is actually better than C, is then $(1-p_1)(1-p_2)$.

The experiments in this paper will focus on evaluating the viability of the neighborhood fitness approach experimentally, by comparing winner selection and its performance estimation based on candidate and neighborhood fitnesses to known and estimated true fitnesses. The p-value calculations are left for future work.

III. Evaluation in the 11-Multiplexer Domain

The neighborhood fitness method is evaluated in the domain of evolving a set of rules to implement an 11-multiplexer function. In this domain, fitness can be estimated by a small number of samples from the domain, but also the true fitness of every candidate can be calculated efficiently. It is therefore possible to evaluate whether neighborhood fitness allows estimating true fitness more accurately, and whether it allows selecting candidates with better true fitness, than the usual candidate fitness.

Multiplexer functions have long been used to evaluate machine-learning methods because they are difficult to learn but easy to check. In general, the input to the multiplexer function consists of u address bits A_v and 2^u data bits D_v, i.e. it is a string of length $u + 2^u$ of the form $A_{u-1}...A_1A_0D_{2^u-1}...D_1D_0$. The value of the multiplexer function is the value (0 or 1) of the particular data bit that is singled out by the u address bits. For example, for the 11-Multiplexer, where $u = 3$, if the three address bits $A_2A_1A_0$ are 110, then the multiplexer singles out data bit number 6 (i.e. D_6) to be its output.

A Boolean function with $u+2^u$ arguments has 2^{u+2^u} rows in its truth table. Thus, the sample space for the Boolean multiplexer is of size 2^{u+2^u}. When $u = 3$, the search space is of size $2^{2^{11}} = 2^{2048} \approx 10^{616}$. However, since evolution can also generate redundant expressions that are all logically equal, the real size of the search space can be much larger, depending on the representation.

A. Representation

In prior work on evolving solutions to the 11-Multiplexer problem [10], a rule-based representation was used where each

candidate specifies a set of rules of the type

$$< \text{rule} > ::= < \text{conditions} > \rightarrow < \text{action} > . \quad (2)$$

The conditions specify values on the bit string and the action identifies the index of the bit whose value is then output. For instance, the following rule outputs the value of data bit 6 when the first three bits are 110:

$$< A_0 = 0 \ \& \ A_1 = 1 \ \& \ !A_2 = 0 > \rightarrow \ D_6. \quad (3)$$

These rules are evolved through the usual mutation and crossover operators of genetic programming [11].

In the experiments in this paper, however, this representation is modified somewhat to make it more similar to evolutionary applications such as web interface design. The length of the chromosome in such applications is fixed, which makes it possible to measure distances between chromosomes. Accordingly, the multiplexer is represented in this paper by a fixed set of eight rules, where each rule specifies whether the corresponding data bit should be output. Further, each rule consists of exactly three elements that together specify the condition, such as $B_0 = 0, B_1 = 1, B_2 = 0$. Unlike before, where only address bits appeared in the conditions, B_i can be any one of the address or data bits. The search space is thus larger, compensating for the smaller search space due to the fixed size of the genome. The elements in the condition are ordered from lowest to highest, i.e. the order does not matter. If multiple rules fire for a given example, the lowest of the corresponding data bits is output.

The genomes are linearized so that they consist of 3*8=24 elements. Crossover point can then be placed on any element boundary, and mutation can change any of the elements by identifying a different bit and its value. Note that there is no mutation that changes only the right-hand side of the element. It turns out that in the multiplexer domain that often results in a larger change in fitness than changing both the bit and its value, making the landscape less smooth and different from applications like web-interface design.

To compute the size of the search space, note that each rule can consist of $\binom{11}{3}$ possible bits and 2^3 right hand sides, for a total of

$$(11!/(3!*(11-3)!))*8 = ((11*10*9*8!)/(3*2*8!)*8) = 1320 \quad (4)$$

different conditions. With eight rules, the size of the search space is

$$1320^8 = (1.32 * 10^3)^8 \approx 9.22 * 10^{24}, \quad (5)$$

Although this is significantly smaller than the 10^{616} for the original representation, it is approximately nine times larger than the number of stars in the known universe. With this representation, a distance between two genomes can be calculated simply as the number of elements in the eight rules that differ, i.e. it ranges within 0..24.

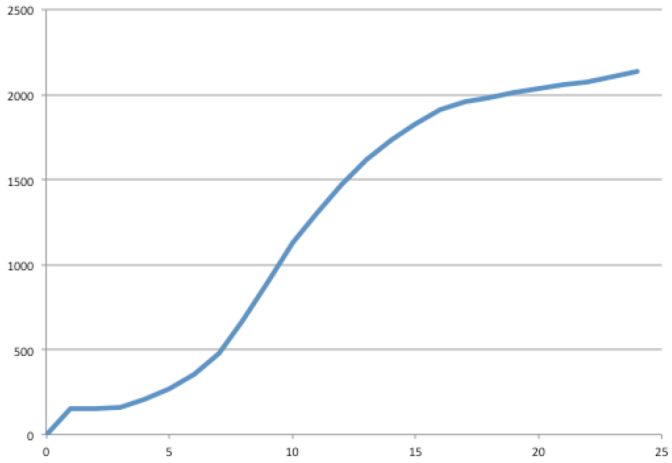

Fig. 1. Fitness difference as a function of genetic distance in the multiplexer domain. The distance of the 250 candidates with the highest estimated fitness was measured to all 4000 candidates in the population at convergence (generation 70), as shown in the x-axis. It varies from 0 to 24, with each segment in the genetic encoding adding one unit of difference. The difference in fitness, in the scale of -4096..2048, were averaged at different distances and plotted in the y-axis. Other experimental configurations lead to similar results. The curve has an S-shape, with an inflection point around distance 7, suggesting that fitness values in close neighborhoods of the best candidate change relatively smoothly, making the neighborhood averaging approach possible.

B. Fitness Calculations

The 11-multiplexer was chosen as a test domain because it is both easy to estimate the fitness and calculate the true fitness of a candidate. True fitness is obtained by evaluating the candidate systematically with all 2048 different cases. Each correct sample increases the fitness by one, and incorrect reduces it by two. Therefore, true fitness ranges within -4096..2048.

Fitness is estimated by creating random strings of 11 bits and counting how often the candidate returns the correct bit. That number is then scaled to the range -4096..2048 by multiplying it with 2048 / number of samples. Multiplexer has regularities such that if a candidate is correct on a large number of samples, such as 1024 or 512, it is likely to be correct on all 2048. Therefore, in the experiments a small number of samples, such as 32, is used to create a noisy evaluation that models real-world sampling well.

In the experiments, smoothness and overestimation in the 11-multiplexer domain are first tested as a sanity check. Whether the method improves accuracy and winner selection is then evaluated, and variations of the method with different population and sampling sizes are measured. There is no control candidate in this domain, so the significance testing is not meaningful. Significance will be measured in the second domain on web-interface design.

C. Sanity Checks: Smoothness and Overestimation

The first step is to estimate whether the fitness landscape of the multiplexer domain is smooth enough for the neighborhood fitness idea to work. The fitness difference is expected to increase smoothly and gradually with distance, and preferably

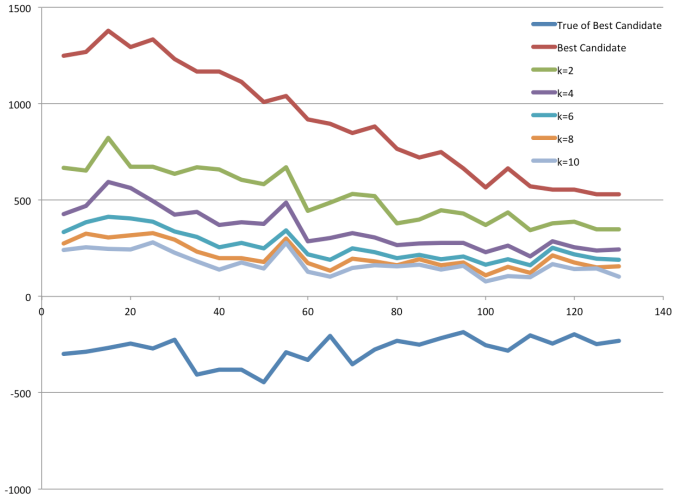

Fig. 2. Best candidate fitness, true fitness of that candidate, and best neighborhood fitness for different k throughout evolution. Generations are shown in the x-axis and difference (in true fitness) from the candidate with the best true fitness in the entire population at each generation is shown in the y-axis. Data was averaged over 10 runs until the fastest runs converged. Each evolved a population of 4000 evaluated with 32 samples. "Best Candidate" is the best candidate fitness found at each generation and "True of Best Candidate" is the true fitness of that candidate. The plots "k=2..k=10" show the best neighborhood fitnesses in the population for each different neighborhood size k (where the neighborhood includes the candidate itself and its k nearest neighbors). The candidate fitness grossly overestimates the true fitness; neighborhood fitness reduces the error by 1/2 to 2/3, providing a better estimate of future performance.

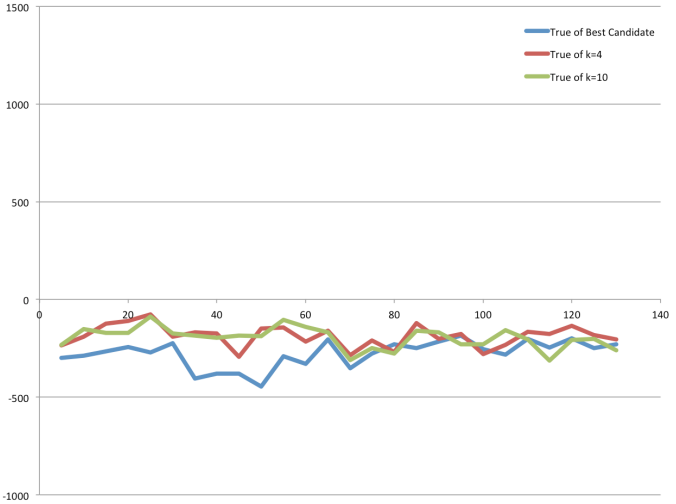

Fig. 3. The true fitnesses of the candidate with the best candidate fitness, and the candidates with the best neighborhood fitnesses with $k = 4$ and $k = 10$ throughout evolution for the same runs as in Figure 2. Up to generation 60, the neighborhood fitness can be used to identify candidates whose true fitness is significantly better than the true fitness of the candidate with the best estimated fitness. In applications where evolution may have to return candidates early, neighborhood fitness thus provides significant improvements in future performance.

slowly in short distances. This is indeed the case, as can be seen in Figure 1. The curve has an S-shape, with the inflection point at around distance 7. This is interesting because the multiplexer function is not particularly smooth: small changes in the rules, such as flipping the right side of a condition from 0 to 1 or vice versa, can have a large effect on behavior. However, the mutations and distance measurement based on a fixed set of elements and rules makes this domain smoother, and therefore a good surrogate for applications such as web interface design.

The next check is whether the candidate fitness indeed overestimates the true fitness, as expected. As can be seen in Figure 2, this is indeed the case. The candidate fitness is a gross overestimate, especially early in evolution. Towards the end, when the candidates' true fitnessess are closer to the optimum, there is less room to overestimate and the effect is smaller.

D. Improvements: Future Performance and Its Estimate

The first main question, then, is whether neighborhood fitness helps. It is indeed much more accurate than candidate fitness in measuring the true fitness, reducing the error by a 1/2 to 2/3. A whole range of k values works well; larger neighborhoods reduce the error slightly more, but that effect tapers off at around $k - 10$. The accuracy is also robust and similar throughout evolution. Thus, neighborhood fitness provides a reliably improved estimate of true fitness, and therefore future performance.

The second main question is whether the neighborhood fitness makes it possible to select the best candidate more reliably than candidate fitness. Again, this is indeed the case, as can be seen from Figure 3. Early on in evolution, when the estimates are least accurate, the true fitness of the candidate with best neighborhood fitness is significantly better than the true fitness of the candidate with best candidate fitness. This advantage lasts until about 60 generations, when candidates are better and the candidate fitness thus more reliable (cf. Figure 2).

Larger neighborhoods (e.g. $k = 10$) seem to be more reliable than small neighborhoods (e.g. $k = 4$) but more data is necessary to draw that conclusion with statistical significance. Eventually when k becomes large enough, the neighborhood fitness is likely to measure something different than the fitness of the candidate at the center of the neighborhood, which eventually will overwhelm the benefit of making a more accurate measurement of that quantity. However, within the range observed in these experiments, the improvement is robust to different values of k, and the expected true fitness of the neighborhood best constant throughout the run. Neighborhood fitness is thus a reliable way to select the winner; in applications like web interface design, where solutions need to be found early in evolution, it offers significant improvement over candidate fitness, identifying candidates that perform better.

Of course, in any individual run, what matters more than average performance of the neighborhood fitness is its actual performance on that run. Figure 4 shows a data from Figures 2 and 3 for one example run with $k = 10$. The plots are more noisy, but an important aspects of the method becomes visible: About half the time, the neighborhood fitness identifies the

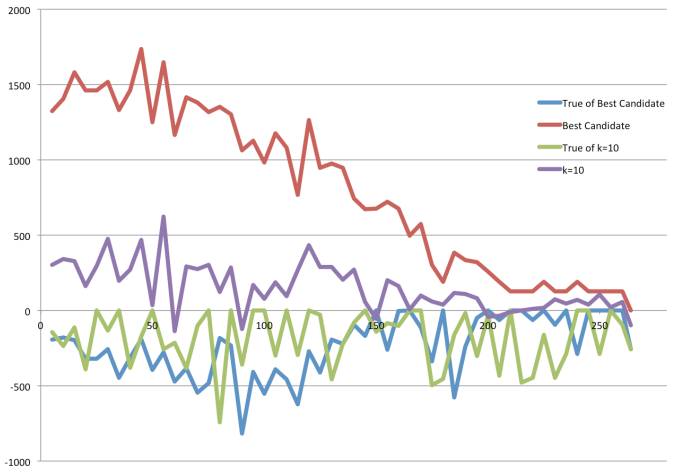

Fig. 4. best candidate fitness, best neighborhood fitness, and corresponding true fitnesses throughout a single evolution run from Figures 2 and 3. The plots are more noisy but show the same trends. Importantly, the "True of k=10", i.e. the true fitness of the candidate with the best neighborhood fitness, reaches the 0-level about half the time in early evolution, and is almost always above the "True of Best Candidate." That is, it can be used to identify the best candidate in the entire population half the time, and other times a candidate that is better than that identified by the best candidate fitness. It can therefore be used to improve evolution results reliably in practice.

true population champion, i.e. the candidate with the best true fitness in the entire population. In early evolution, it almost always allows picking a candidate that is better than the one picked based on candidate fitness. Therefore, the method can be used to improves performance not just statistically, but also reliably in actual applications.

E. Robustness to Method Variations

As has already been discussed, the method is robust to variations to some of the main parameters such as the neighborhood size k and length of the run. In a subsequent experiment described below, it also turns out to be relatively robust wrt. population size and density. On the other hand, with increased sampling the estimates become more accurate, and the advantage of neighborhood fitness reduces.

The first experiment evaluates whether more dense neighborhoods lead to better neighborhood estimates. That is, if the k nearest neighbors are more similar to the candidate, its neighborhood fitness is likely to be more accurate. While the simulations so far used a population of 4000 and mutation proportion of 60% (with the remaining 40% of offspring generated through crossover), a more dense neighborhood was created with population of 8000 and mutation proportion of 80%. However, there is little difference (Figure 5). A possible explanation is that even with a small population and more frequent crossover, evolution converges the population around the best candidates. A larger population and more mutation may increase the density overall, but not that much around the best candidates. Thus, evolution is naturally well suited for the neighborhood fitness approach, and robust against population size and genetic operators used.

The second experiment evaluates the effect of increased sampling on the neighborhood fitness method. In two setups, 64 and 128 samples were used to evaluate the fitness of each candidate during evolution, instead of 32. As can be seen from Figure 6, neighborhood fitness confers still a slight advantage with 64 samples, but not much with 128. Indeed, at that level of sampling, the estimate is very reliable in the multiplexer domain. That is, if a candidate gets 128 samples right, it very likely gets all 2048 cases right. Therefore, the neighborhood fitness method offers the largest advantage in domains where only partial sampling is possible, leading to unreliable estimates of fitness. This is true in many real-world applications where fitness evaluation is expensive, such as designing web-interfaces, robotic controllers, or game agents.

IV. Discussion and Future Work

The neighborhood fitness method offers the greatest advantage early in evolution, before the populations start to converge around optimal solutions. This property turns out to be important in practical applications where sampling resources are limited. For instance in web-interface optimization, the customer often wants to deploy a new page as soon as significant improvement is seen, instead of letting evolution run its course and find near-optimal solutions [1].

A compatible advantage is that the method can be applied at any point during evolution as a postprocessing step, and does not require further traffic for testing the best candidates. Evolution can therefore be terminated at any point when the customer so desires, and a winning candidate delivered immediately, with good confidence that it is indeed a good one and will perform as expected in the future.

In domains where more sampling is available, and the termination point is predictable, it might be possible to relax these requirements. More traffic could be directed to the best candidates for a while before the winner is selected, narrowing down the choice systematically. Implementing such a method and comparing it with neighborhood fitness is a most interesting direction of future work.

V. Conclusion

This paper shows how a winner in evolutionary optimization can be selected more reliably and its performance estimated more accurately based on the average fitness in its neighborhood rather than its own fitness. This method assumes that the fitness landscape is locally smooth and good candidates have close neighbors in the population, which is likely to be the case in many domains. The method can be applied as a postprocessing step at any point in evolution, making it possible to deploy it in practical applications without modifying evolution or sampling. It offers the greatest advantage early in evolution, which is useful in many applications where resources are limited. It is therefore a promising method for improving performance of evolutionary optimization in practical applications.

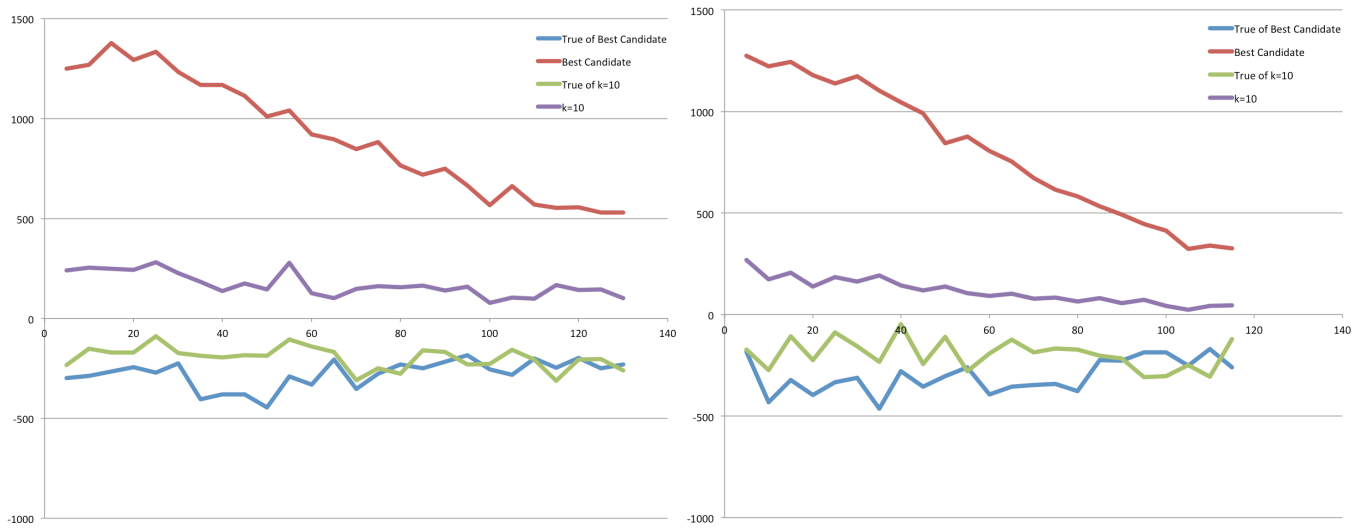

Fig. 5. Best candidate fitness, best neighborhood fitness, and corresponding true fitnesses with populations of different densities. On the left is the average over 10 runs of a population with 4000 candidates and mutation proportion of 60%; on the right, an average of 10 runs of a population of 8000 and mutation of 80%. The differences are negligible, suggesting that the evolutionary algorithm already maintains dense populations around the best candidates.

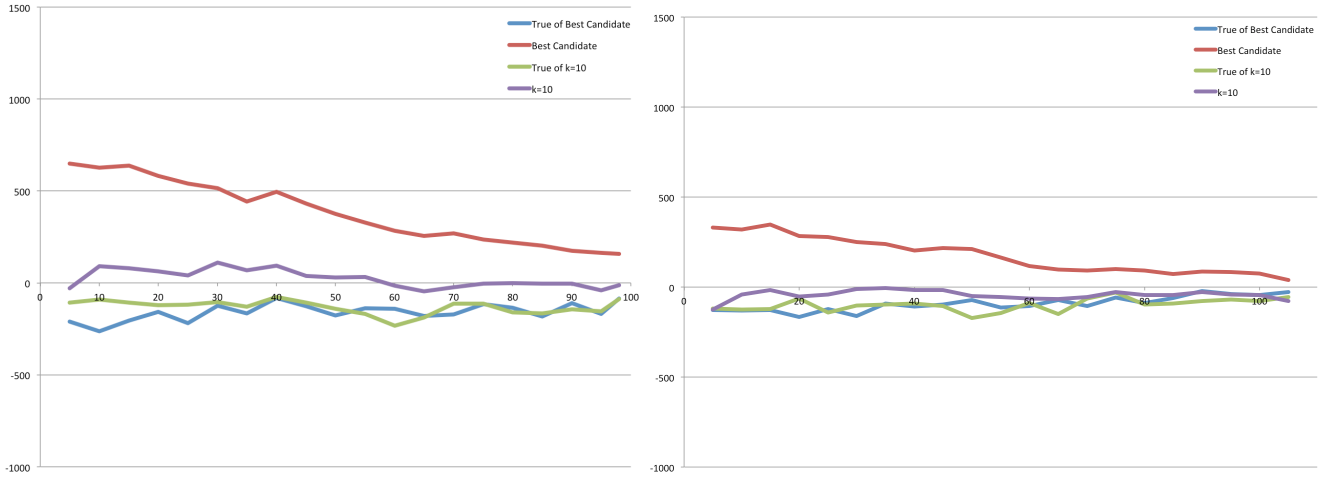

Fig. 6. Best candidate fitness, best neighborhood fitness, and corresponding true fitnesses with different amount of sampling. On the left is the average over 10 runs where each candidate was evaluated with 64 samples during evolution, and on the right, the average over 10 runs where 128 samples were used. Neighborhood fitness allows making better estimates and selecting better winners with 64 samples, but that advantage is smaller with 128 samples, which is very reliable in the multiplexer domain. The method is thus best suited for domains where fitness evaluations are expensive and not comprehensive, as in many real-world domains.

References

[1] R. Miikkulainen, N. Iscoe, A. Shagrin, R. Cordell, S. Nazari, C. Schoolland, M. Brundage, J. Epstein, R. Dean, and G. Lamba, "Conversion rate optimization through evolutionary computation," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2017)*. New York, NY: ACM, 2017.

[2] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments—a survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, pp. 303–317, 2005.

[3] Y. Martínez, L. Trujillo, E. Naredo, and P. Legrand, "A comparison of fitness-case sampling methods for symbolic regression with genetic programming," in *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*, A.-A. Tantar, E. Tantar, J.-Q. Sun, W. Zhang, Q. Ding, O. Schütze, M. Emmerich, P. Legrand, P. Del Moral, and C. A. Coello Coello, Eds. Berlin: Springer, 2014, pp. 201–212.

[4] R. G. Miller, *Simultaneous Statistical Inference*. Berlin: Springer, 1966.

[5] H. Shahrzad, B. Hodjat, and R. Miikkulainen, "Estimating the advantage of age-layering in evolutionary algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016)*. New York, NY: ACM, 2016.

[6] R. Kohavi and R. Longbotham, "Online controlled experiments and A/B tests," in *Encyclopedia of Machine Learning and Data Mining*, C. Sammut and G. I. Webb, Eds. New York: Springer, 2016.

[7] B. L. S. P. Rao, *Nonparametric functional estimation*. Academic Press, 2014.

[8] L. R. Rabiner and B. Gold, *Theory and application of digital signal processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[9] D. S. Collingridge, "A primer on quantitized data analysis and permutation testing," *Journal of Mixed Methods Research*, vol. 7, p. 7995, 2013.

[10] H. Shahrzad and B. Hodjat, "Tackling the Boolean multiplexer function using a highly distributed genetic programming system," in *Genetic Programming Theory and Practice XII*, R. Riolo, W. P. Worzel, and M. Kotanchek, Eds. New York: Springer, 2015, pp. 167–179.

[11] F. J. Berlanga, A. Rivera, M. J. del Jesús, and F. Herrera, "Gp-coach: Genetic programming-based learning of compact and accurate fuzzy rule-based classification systems for high-dimensional problems," *Information Sciences*, vol. 180, no. 8, pp. 1183–1200, 2010.