Benchmarking discrete optimization heuristics: from building a sound experimental environment to algorithm configuration

Ye, F.

Chapter 7

Dynamic Algorithm Selection

We analyze in this chapter how well existing benchmark data can be used for the selection of suitable algorithm combinations. Precisely, we investigate using dynamic crossover probability for the $(\mu + \lambda)$ GA and study "one-shot" dynamic algorithm selection (dynAS) policies based on the results of the benchmarking study presented in Section 5.2. The study in this chapter highlights the research topics for the future work of dynAS, namely automatic detection of alternating timing and "warm-start" strategies for adjusting parameters.

7.1 Background

It is well known that different algorithms or different instantiations of the same algorithm are best suited for different problems and even for different stages of the optimization process. Automated algorithm selection [95] as well as dynamic parameter selection [92] are therefore intensively studied meta-optimization problems in EC. However, the former has a strong requirement on being able to run different algorithms (or algorithm configurations) prior to making a decision which algorithm to apply to the problem at hand. Parameter control and related concepts (including hyper-heuristics, adaptive operator control, etc.), in contrast, assume that the selection has to be made on the fly, without leveraging existing data from previous or related runs. With the rise of artificial intelligence methods, EC is currently facing a paradigm shift, in that we aim to actively exploit existing performance data to select which algorithms to apply, and how to possibly adjust them during the run. We are, however, still far from achieving a fully automated informed online selection.

We study in this chapter how well we can predict from existing performance data which algorithm instances to combine for a given problem at hand. While we do allow for switching between different algorithms, the decision when to switch has to be made prior to the run, and depends, in our case, on the solution quality of the evaluated solution candidates. More precisely, we use the benchmarking data in Section 5.2 as starting point to investigate, for each of the 25 individual problems, how well we can predict which single-switch algorithm combinations would show good performance. For some functions we easily obtain algorithm combinations that outperform the best static algorithms. For other functions the results are rather mixed. On three functions, none of the 100 tested single-switch algorithm combinations was able to outperform the best static solver. The prediction quality of the approach suggested in [153] varies a lot between the different functions. While for LeadingOnes, for example, the performance predictions are rather accurate, large discrepancies between predicted and actual performance can be observed for more complex function. In particular for multi-modal functions the approach can get trapped by a first algorithm that is very efficient in converging to a local optimum from which the second algorithm cannot escape easily.

7.2 Dynamic Crossover Probability Selection: A Study Case on LeadingOnes

Based on our finding in [170] (see Section 4.3) that, on LeadingOnes, the optimal crossover probability of Algorithm 8 is dynamic along the problem dimension and population size, we start in this section with an investigation of using dynamic crossover probabilities for the $(\mu + \lambda)$ GA.

To obtain the *"optimal"* crossover probability at different stages of the optimization process, we test the $(10 + 10)$ GA using standard bit mutation with $p = 1/n$ and uniform crossover with different $p_c \in \{0.1k \mid k \in [9]\}$. Algorithms run at the stages of fitness value $f \in [s, s + 5], s \in \{5i \mid i \in [19]\}$ on 100-dimensional LeadingOnes. Practically, we initialize the population of the GAs with all the individual's fitness values equal to s, and the algorithms terminate once a solution with $f(x) \geq s + 5$ is found.

Figure 7.1 plots function evaluations used by the GAs at each stage. It shows that the GA with $p_c = 0$ uses the least function evaluations at the early stages $s \leq 40$, but other GAs with $p_c > 0$ use less function evaluations as s is increasing. Therefore, we

expect to improve the performance of the $(\mu+\lambda)$ GA by using the *"optimal"* crossover probabilities at all the stages of the optimization process. Figure 7.2 plots the fixed-target ERTs of GAs with static p_c and dynamic ones. The dynamic policy selects the corresponding best p_c at each stage. Practically, when the GA finds a solution with $s_1 \leq f(x) < s_2, s_2 = s_1 + 5, s \in \{5i \mid i \in [19]\}$, the p_c will be adjusted by the corresponding best value in Figure 7.1.

Figure 7.2 shows that the GA with dynamic p_c outperforms other GAs along the entire optimization process. The dynamic policy successfully hits the optimum $f(x) = 100$ using the smallest ERT $7,194$, while ERT of the best runner-up (the GA with $p_c = 0.2$) is $7,661$. This corresponds to a 6% improvement of the dynamic GA over the best static one. This performance empirically proves that the GA can benefit from dynamic crossover probability, and it displays a successful case of applying the dynAS for the GA. However, the dynAS problem is not usually coming with the ideal condition that candidate algorithms differ by only one parameter. Therefore, we are working on the GAs with more combinations of parameters in the next section.

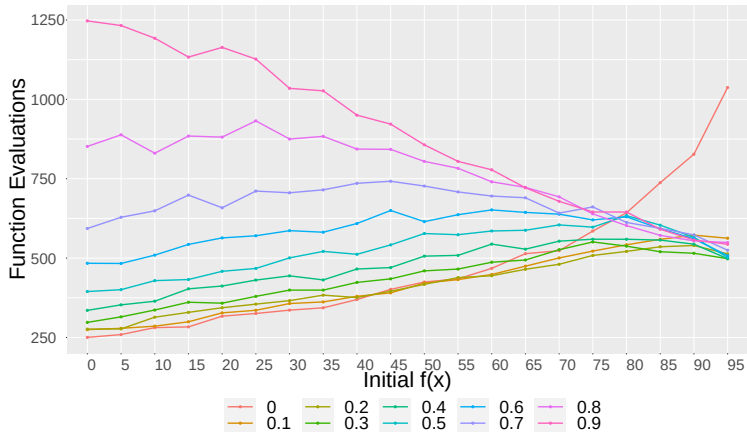

Figure 7.1: Average number of function evaluations needed by different $(10+10)$ GAs to find a solution y with $f(y) \geq s+5$ on the 100-dimensional LeadingOnes function when all ten points in the initial population are uniformly chosen from the set of points x that satisfy $f(x) = s$, for $s \in \{5i \mid i \in [19]\}$. The GAs differ only in the crossover probability $p_c \in \{0.1k \mid k \in [9]\}$ (different lines). Results are averaged of $1,000$ independent runs. The connecting lines are only meant to help visual interpretation, the data points are only at the values 0, 5, 10, ..., 95.

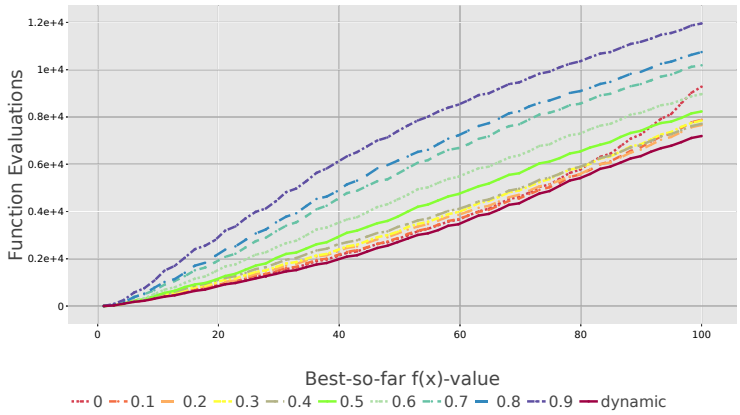

Figure 7.2: Fixed-target ERTs of GAs on 100-dimensional LeadingOnes. The legend presents values of p_c, and the *dynamic* method adjusts its p_c to the *optimal* value at each target $f(x) = s, s \in \{5i \mid i \in [19]\}$, based on the result in Figure 7.1. Results are average of 100 independent runs.

7.3 Dynamic Algorithm Section for the PBO Problems

Since the LeadingOnes case shows significant improvement by using dynamic crossover probabilities, which is a particular case of the dynAS, we study the behavior of the dynAS on a broader range of problems and GAs. We take as input the benchmarking data from Section 5.2, which comprise detailed performance records for 80 genetic algorithms on the 25 functions provided by IOHprofiler. We focus on ERT as performance measure. Detailed data can be found in [166].

Following the approach suggested in [153] we compute a "theoretical" ERT value for all combinations (A_1, A_2, ϕ_s), where A_1 is the first algorithm, A_2 the second, and ϕ_s the target value at which we switch from A_1 to A_2. To this end, we simply compute $\mathrm{ERT}(A_1, P, \phi_s) + \mathrm{ERT}(A_2, P, \phi_f) - \mathrm{ERT}(A_2, P, \phi_s)$, where all these ERT values are based on the performance records provided in Section 5.2. In total, we consider 42 possible switching points ϕ_s, which we select within the interval $[\phi_m, \phi_f]$ between the smallest fitness value ϕ_m of the problem and the best found target ϕ_f according to Table 5.2. We consider evenly spaced targets, for the original and for the log-scaled interval, respectively. For each problem, we consider only algorithms that hit the final target value with probability at least 80% according to the data from Section 5.2. Using this approach, we select for each problem the 100 best combinations (A_1, A_2, ϕ_s)

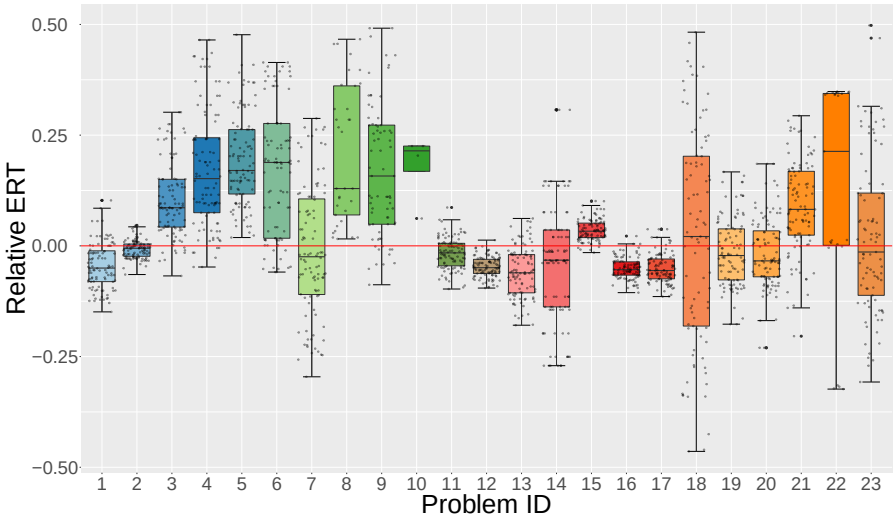

Figure 7.3: Relative ERT values of 100 1-switch combinations (A_1, A_2, ϕ_s) $(dERT)$ for 23 out of 25 IOHprofiler problems in dimension $d = 100$, compared to the ERT of the best static GAs according to the results in Section 5.2 $(sERT)$. Each black dot represents one ERT value. The relative deviation is calculated by $(dERT - sERT)/sERT$ so that negative values (below the red line) correspond to an advantage of the dynamic combination over the best static algorithm. We only display values between -0.5 and 0.5 so that the results of F24-F25 are missing here with values larger than 1. All ERT values are based on 100 independent runs.

and we then run the combination 100 independent times on the problem that they have been selected for.

In Figure 7.3 we compare the so-obtained ERT values with the best ERT value reported in Section 5.2, which we refer to as the *best static algorithm* (BSA). For combinations (A_1, A_2, ϕ_s) for which the parent population sizes μ_1 of A_1 is larger than the parent population size μ_2 of A_2 we selected the best μ_2 points to initialize the parent population of A_2. Where $\mu_1 < \mu_2$, the new parent population comprises all μ_1 points, as additional $\lfloor \mu_2/2 \rfloor - \mu_1$ copies of the best points, and $\lceil \mu_2/2 \rceil$ randomly added individuals.

For some of the problems (e.g., F1, F2, F7, F11-14, F16-23)), the ERT of several combinations (A_1, A_2, ϕ_s) outperform that of the BSA. For other functions, and in particular for F10, F24, and F25, none of the combinations (A_1, A_2, ϕ_s) is able to outperform the BSA.

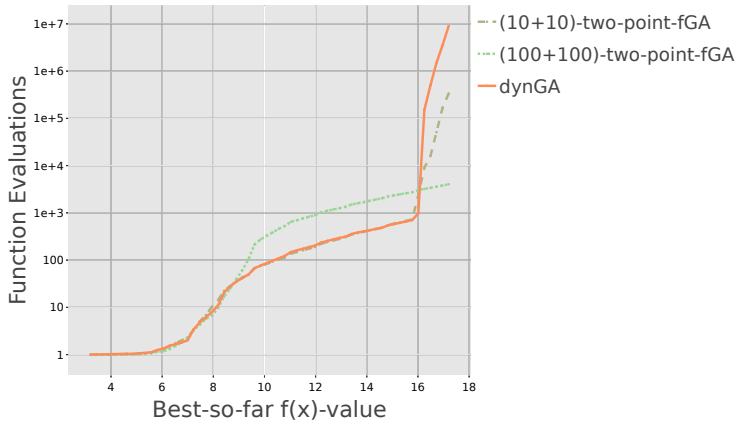

Figure 7.4: Fixed-target ERTs of GAs on F24 in dimension $d = 100$. The dynGA switches from the $(10 + 10)$-two-point-fGA to the $(100 + 100)$-two-point-fGA at the target $f(x) = 15.81$. Results are from 100 independent runs.

Local Optima are Deceptive An intuitive explanation for the cases where dynAS fails is that this is caused by local optima. Recall that in the computation of the predicted ERT, the contribution of A_1 to the predicted ERT is decided by its ERT hitting the target $f(x) = \phi_s$. However, using the ERT as the cost metric, we can not obtain information to estimate whether the algorithm is trapped or around a local optimum.

Figure 7.4 plots the fixed-target result of the best tested dynamic genetic algorithm (dynGA) on F24, which uses a $(10 + 10)$-two-point-fGA at first and the switches to a $(100 + 100)$-two-point-fGA. By using a small population size 10 initially, the dynGA converges to the switching point ($f(x) = 15.81$) fast, but it is trapped there and could not follow the original trend of the $(100+100)$ GA later. We do not solve this problem here, but it is interesting to spot this issue for future work.

7.4 Summary

We have investigated in this chapter possibilities to leverage existing benchmark data to derive switch-once dynamic algorithm selection policies. While for some cases the "theoretical" approach suggested in [153] could indeed predict combinations that outperformed the best static solver, the results are less positive for others. One obstacle that hinders an accurate performance prediction are local optima: when the first al-

gorithm is very good at converging to a local optimum, it is likely to be chosen as A_1. It is then important, however, to continue the search with an algorithm that has a good enough exploration power to escape the local optimum. This ability, however, seems hard to infer from the pure performance profiles, and may require a "human in the loop".

Going forward, our long-term goal is the automated detection of situations in which switching from one algorithm to another can be beneficial. To this end, we will further investigate efficient strategies to *warm-start* the algorithms by actively using the information accumulated thus far. In the here-presented study, we have used ERT values as performance measure and as indicator to select which algorithm combinations to execute. In future work we will consider other performance measures, and in particular those that measure the anytime performance of the algorithms.