From benchmarking optimization heuristics to dynamic algorithm configuration

Vermetten, D.L.

Chapter 6

Testing Generalizability: MA-BBOB

Algorithm complementarity can be exploited in various ways, as illustrated in previous chapters. In this thesis, most of our experimentation has been using ELA features to represent the problem landscape and use this information to choose the most promising algorithm (combination) to run. The ability of ELA to differentiate between BBOB functions indeed suggests that these features are a useful representation for the algorithm selection problem which is confirmed by several studies on algorithm selection for continuous optimization heuristics which use BBOB as their benchmark suite [126, 124, 135, 18, 138, 174]. However, a key challenge with using BBOB for this type of algorithm selection lies in the evaluation of the results. One method is a leave-one-function-out technique [180], which uses 23 functions for training and the remaining one for testing. This approach tends to show poor performance since each problem has been designed to represent different high-level challenges for the optimization algorithm. As such, another technique of cross-validation by splitting function instances is commonly used [126]. However, this is likely to overfit and overestimate the performance of the selector, since the instances of different problems are inherently very similar. Thus, overfitting to biases of the instance design is an often overlooked risk [135].

One potential way in which this bias can be reduced is by creating new, larger sets of benchmark problems (e.g. using genetic programming to fill the instance space [172, 149]), or by creating a problem generator (e.g., the GNBG generator [263] or the

W-model in pseudo-Boolean optimization [257]). Such problem generators can then create arbitrarily many benchmark functions, to be used in the common train/test or cross-validation mechanisms from the machine learning community [188].

This chapter is a shortened version of the journal paper [249], which accumulates and extends work presented at the GECCO [251] and the AutoML [250] conferences. Our focus is on describing the Many-Affine BBOB function generator (MA-BBOB). To construct new functions, we create affine combinations between existing BBOB functions, building on the work of Dietrich and Mersmann [58]. Our generator is a generalization of their approach, designed to create unbiased combinations of problems where the contribution of the components can be smoothly varied. We highlight the core design choices made in the construction of MA-BBOB in Section 6.1 and illustrate their impact on the types of problems which can be created.

The parameterization of the MA-BBOB generator allows us to investigate controlled, potentially small differences in functions from both the ELA and algorithm performance perspectives. This is illustrated in Section 6.2 by investigating the addition of global structure (sphere function) to all other BBOB problems, as well as transitioning between pairs of BBOB problems.

Finally, in Section 6.3 we make use of a set of 1 000 functions generated with MA-BBOB to illustrate an algorithm selection scenario, and show that the generalization from BBOB to MA-BBOB fails to meet expectations. A comparison of the ELA-based algorithm selection approach with an artificial baseline using the weights of the affine combinations indicates that there is room to improve on the current ELA-based setup, especially when trying to generalize from BBOB to MA-BBOB.

6.1 Many-Affine BBOB

6.1.1 Pairwise Affine Combinations

To create affine combinations between two BBOB functions, we use a slightly modified version of the procedure proposed in [58]. Specifically, we define the combination C as follows:

$$
\begin{aligned}
C(F_{1,I_1}, F_{2,I_2}, \alpha)(x) = 10^X, \text{ with} \\
X = \Big(\alpha \log_{10}\left(F_{1,I_1}(x) - F_{1,I_1}(O_{1,I_1})\right) + \\
(1-\alpha)\log_{10}\left(F_{2,I_2}(x - O_{1,I_1} + O_{2,I_2}) - F_{2,I_2}(O_{2,I_2})\right)\Big)
\end{aligned}
$$

Here, F_1, I_1, F_2, and I_2 are the two base functions and their instance numbers, as defined in BBOB [96]. O_{1,I_1} and O_{2,I_2} represent the location of the optimum of functions F_{1,I_1} and F_{2,I_2} respectively. The transformation to x when evaluating F_{2,I_2} is performed to make sure the location of the optimum is at O_{1,I_1}. As opposed to the original definition, we subtract the optimal values before aggregating and take a logarithmic mean between the problems. This way, we can use consistent values for α across problems, without having to perform the entropy-based selection performed in [58]. It has the additional benefit of ensuring the objective value of the optimal solution is always 0, so the comparison of performance across instances and problems is simplified. In Figure 6.1, we illustrate the change in the landscape for the combination of F21 and F1, for different values of α.

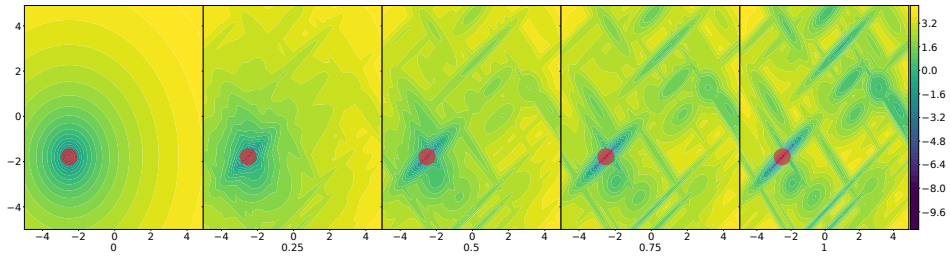


Figure 6.1: Evolution of the landscape (log-scaled function-values) of the affine combination between F21 ($\alpha = 1$) and F1 ($\alpha = 0$), instance 1 for both functions, for varying α in 0.25 increments. The red circle highlights the location of the global optimum.

6.1.2 Combining Multiple BBOB Functions

We extend the pairwise affine combinations from Section 6.1.1 to create a function generator which uses affine combinations of multiple BBOB functions. In particular, our generator is defined as follows:

$$\textit{MA-BBOB}(\vec{W}, \vec{I}, \vec{X}_{\texttt{opt}})(x) = R^{-1}\Big(\sum_{i=1}^{24} W_i \cdot R_i\big(F_{i,I_i}(x - \vec{X}_{\texttt{opt}} + O_{i,I_i}) - F_{i,I_i}(O_{i,I_i})\big)\Big)$$

Here, $\vec{W}$ and $\vec{I}$ are 24-dimensional vectors containing the weight and instance identifiers respectively, and $\vec{X}_{\texttt{opt}}$ is the location of the optimum, which we generate uniformly at random in the domain $[-5, 5]^d$. Finally, R_i and R^{-1} are rescaling functions, defined in Section 6.1.2. We highlight the motivation behind each of these design choices in the following subsections.

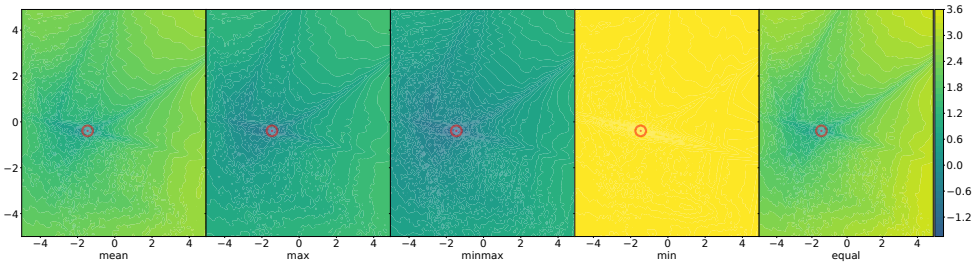


Figure 6.2: Log-scaled fitness values of an example of a single many-affine function with 5 different ways of scaling. The first 4 are taking the mean, max, $(\max + \min)/2$ and min of 50 000 random samples to create the scale factor, while the fifth ('equal') option does not make use of this scaling.

Scaling of Function Values

While the geometric weighted average used in Section 6.1.1 between component functions reduces the impact of small differences in scale, some BBOB problems vary by orders of magnitude, which can still cause one function to dominate the combined landscape. To address this, we add a rescaling function to the MA-BBOB definition, which transforms the log-precision on each component function into approximately $[0, 1]$ before the transformation. This is done by capping the log-precision at -8, adding 8 so the minimum is at 0 and dividing by a *scale factor* S_i. This procedure aims to make the target precision of 10^2 similarly easy to achieve on all component problems. We thus get the following scaling functions:

$$R_i(x) = \frac{\max(\log_{10}(x), -8) + 8}{S_i}$$
$$R^{-1}(x) = 10^{(10 \cdot x) - 8}$$

To determine practical scale factors, we collect a set of 50 000 random samples and evaluate them. We then aggregate the resulting function values (transformed to log-precision) in several ways: min, mean, max, $(\max + \min)/2$. In Figure 6.2, we show the differences between these methods for a selected problem in $2d$. Somewhat subjectively, we select the $(\max + \min)/2$ scaling as the technique to use for the MA-BBOB generator. To ensure we don't have to repeat this sampling procedure each time we instantiate the problem in a new dimensionality, we investigate the relation between dimensionality and the chosen scale factor calculation. This is visualized in Figure 6.3, where we see that, with an exception for the smallest dimensionalities, the

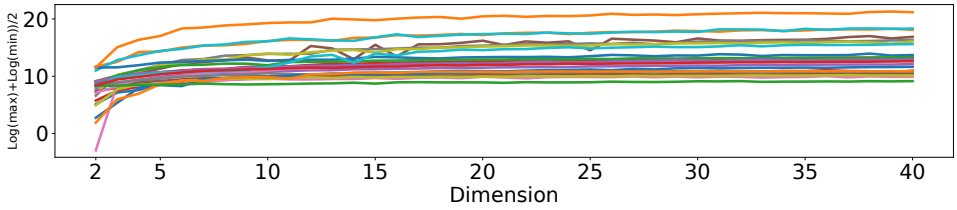


Figure 6.3: Evolution of the log-scaled $(\max + \min)/2$ scaling factor, relative to the problem dimensionality. The values are based on 50 000 samples. Each line corresponds to one of the 24 BBOB functions.

Function ID	1	2	3	4	5	6	7	8	9	10	11	12
Scale Factor	11.0	17.5	12.3	12.6	11.5	15.3	12.1	15.3	15.2	17.4	13.4	20.4
Function ID	13	14	15	16	17	18	19	20	21	22	23	24
Scale Factor	12.9	10.4	12.3	10.3	9.8	10.6	10.0	14.7	10.7	10.8	9.0	12.1

Table 6.1: Final scale factors used to generate MA-BBOB problems.

values remain quite stable. Because of this, we make use of a static scale factor rather than defining one for each dimensionality individually. The final factors used are calculated as a rounded median of the values from Figure 6.3, and shown in Table 6.1.

Instance Creation

Another design choice we made was to place the optimum of the combined function uniformly in the domain ($[-5, 5]^d$). This differs from the earlier versions used for pairwise combinations of BBOB functions [58, 251], where the optimum of one of the component functions was re-used. However, the biases in the original BBOB instance generation procedure would then be transferred into the combinations as well [150]. Since our function generator does not have to guarantee the preservation of global function properties, we take the risk of moving parts of the regions of interest outside the domain to have a less biased location of the global optimum. Figure 6.4 shows how a $2d$-function changes when moving the optimum location.

Sampling Random Functions

To allow for the usage of MA-BBOB as a function generator, we need to create a default setting to generate useful weight-vectors. This could be done uniformly at random (given a normalization step). However, in this way, the weight for every component is likely to be non-zero, so most functions contribute to the final combination, erasing

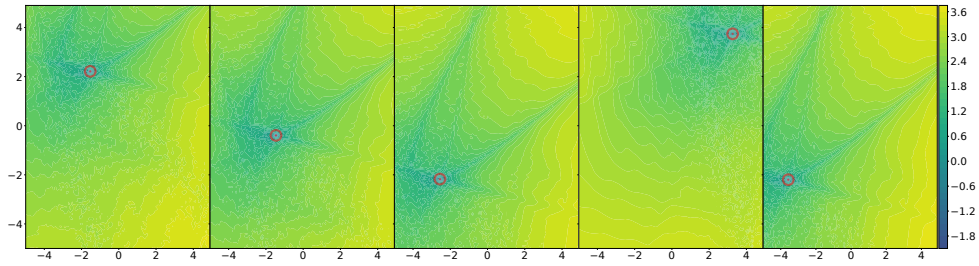


Figure 6.4: Log-scaled fitness values of an example of a single many-affine function with changed location of optimum.

the possibility of generating unimodal problems since some multimodality will always be included from some of the multimodal component functions.

To address this issue, we adapt the sampling technique to combine fewer component functions on average. Our approach is based on a threshold value to determine which functions contribute to the problem. The procedure for generating weights is thus as follows: (1) Generate initial weights uniformly at random, (2) adapt the threshold to be the minimum of the user-specified threshold and the third-highest weight, (3) this threshold is subtracted from the weights, all negative values are set to 0. The second step is to ensure that at least two problems always contribute to the new problem. We decide to set the default value at $T = 0.85$, such that on average 3.6 (i.e., 15% of 24) problems will have a non-zero weight.

6.2 Pairwise Affine Combinations

For the first analysis of the MA-BBOB functions, we limit ourselves to the combination of pairs of functions. This allows a more low-level investigation into the transition of both landscape features and algorithm performance.

6.2.1 Setup

For the algorithm performance-based analysis, we make use of a portfolio of five algorithms. Of these, three are accessed through the Nevergrad framework [200]:

- Differential Evolution (DE) [219]
- Constrained Optimization By Linear Approximation (Cobyla) [191]

- Diagonal Covariance Matrix Adaptation Evolution Strategy (dCMA-ES) [98]

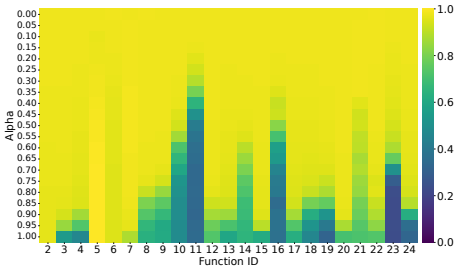
The remaining algorithms are two modular algorithm families: modular Differential Evolution [240] (modDE) and modular CMA-ES [51] (modCMA). All algorithms, including the modular ones, use default parameter settings. Each run we perform has a budget of $2\,000d$, where d is the dimensionality of the problem. We perform 50 independent runs per function. For the pairwise function combinations, we stick to the terminology introduced in Section 6.1.1 for easier comparison with the previous results in [251].

In the remainder of this section, we set $I_2 = 1$. As such, when discussing the *instance* of a pairwise affine combination $C(F_1, I_1, F_2, I_2, \alpha)$, we are referring to I_1. Note that we also introduce the uniform sampling of the optima for these experiments, following the description in Section 6.1.2. For our performance measure, we make use of the normalized area over the convergence curve (AOCC), to be maximized. The AOCC is an anytime performance measure, which is equivalent to the area under the cumulative distribution curve (AUC) given infinite targets for the construction of the ECDF. This measure is thus slightly more precise than the AUC, and can easily be computed online. To remain consistent with the performance measures used in our previous work, and analysis of results on BBOB in general, we use 10^2 and 10^{-8} as the bounds for our function values, and perform a log-scaling before calculating the AOCC. We thus calculate the normalized AOCC of a single run as follows:

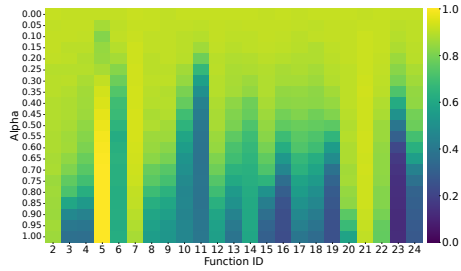
$$AOCC(\vec{y}) = \frac{1}{B}\sum_{i=1}^{B} 1 - \frac{\min(\max((\log_{10}(y_i), -8), 2) + 8}{10}$$

where $\vec{y}$ is the sequence of best-so-far function values reached, B is the budget of the run. To obtain the AOCC over multiple runs, we simply take the average.

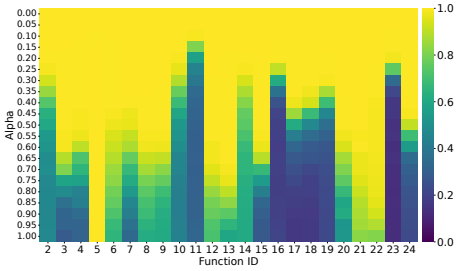
For the landscape analysis, we make use of the pFlacco [192] package to calculate the ELA features. We use a sample size of $1\,000d$ points, sampled using a Sobol' sequence. We note this large sample size is used to remove some of the inherent variability in the ELA features, even though practical applications usually rely on much smaller budgets. To be consistent with our previous work [250], we don't include features which require additional samples and remove all features which lead to NaN-values or remain static for all functions, resulting in a set of 44 features.



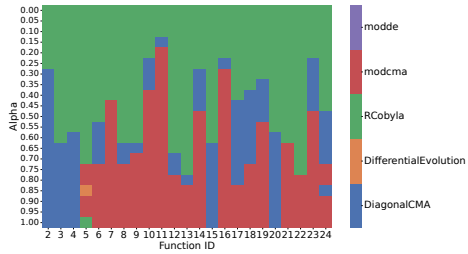
(a) AOCC values for Diagonal CMA-ES.



(b) AOCC values for Differential Evolution.



(c) AOCC values for Cobyla.



(d) Best performing algorithm from the portfolio, based on AOCC.

Figure 6.5: Normalized area under the ECDF curve of three selected algorithms (a-c) and best-ranking algorithm from the full portfolio (d) for each combination of the BBOB-function (x-axis) with a sphere model, for given values of α (y-axis) AOCC is calculated after 10 000 function evaluations, based on 50 runs on 50 instances (and location of optimum). Note that $\alpha = 0$ corresponds to the sphere function.

6.2.2 Adding Global Structure

For the first set of experiments, we make use of affine combinations where we combine each function with F1: the sphere model (as the F_2 function in the combination). As can be seen in Figure 6.1, adding a sphere model to another function creates an additional global structure that can guide the optimization toward the global optimum. As such, these kinds of combinations might allow us to investigate the influence of an added global structure on the performance of optimization algorithms. While to some extent this can already be investigated by comparing results on the function groups of the original BBOB with different levels of global structure, the affine function combinations allow for a much more fine-grained investigation.

In Figure 6.5a, we can see that the performance of CMA-ES does indeed seem to move smoothly between the sphere and the function with which it is combined. It is

however interesting to note the differences in speed at which this transition occurs. For example, while the final performance on functions 3 and 10 seems similar, the transition speed differs significantly. This seems to indicate that for F10, the addition of some global structure has a relatively weak influence on the challenges of this landscape from the perspective of the CMA-ES, while even small amounts of global structure significantly simplify the landscape of F3.

We can perform a similar analysis on other optimization algorithms. In Figures 6.5b and 6.5c, we show the same heatmap as Figure 6.5a, but for Differential Evolution and Cobyla, respectively. It is clear from these heatmaps that the performance of DE is more variable than that of CMA-ES, while Cobyla's performance drops off much more quickly. The overall trendlines for DE do seem to be somewhat similar to those seen for diagonal CMA-ES: the transition points between high and low AOCC in Figure 6.5b are comparable to those seen in Figure 6.5a. There are however still some differences in behavior, especially relative to Cobyla. These differences then lead to the question of whether there exist transition points in ranking between algorithms as well. Specifically, if one algorithm performs well for $\alpha = 0$ but gets overtaken as $\alpha \to 1$, exploring this change in ranking would give further insight into the relative strengths and weaknesses of the considered algorithms.

To study the impact on the relative ranking of algorithms, we make use of the full portfolio of 5 algorithms and rank them based on AOCC on each affine function combination. We then visualize the top ranking algorithm on each setting in Figure 6.5d. From this figure, we can see that Cobyla deals well with the sphere model, managing to outperform the other algorithms when the weighting of the sphere is relatively high. Then, after a certain threshold, the CMA-ES variants consistently outperform the rest of the portfolio, with dCMA taking over when Cobyla is no longer preferred. However, as α increases further, and the influence of the sphere model diminishes, an interesting pattern seems to occur. For several problems, there is a second transition point to modCMA, indicating that the differences in default parameterizations between the used libraries have a large impact on the algorithms' behavior. One significant factor is related to the initial stepsize, which is smaller for dCMA, and thus might lead to it becoming more easily stuck in local optima when the global structure is not as strong.

In order to better understand what the transitions in algorithm ranking look like, we can zoom in on one of the functions and plot the distribution of AOCC for all values of α. This is done in Figure 6.6, where we look at the combination between F10 and the sphere model. In this figure, we observe that Cobyla is very effective at optimizing the sphere and the combinations with low α. However, when α increases,

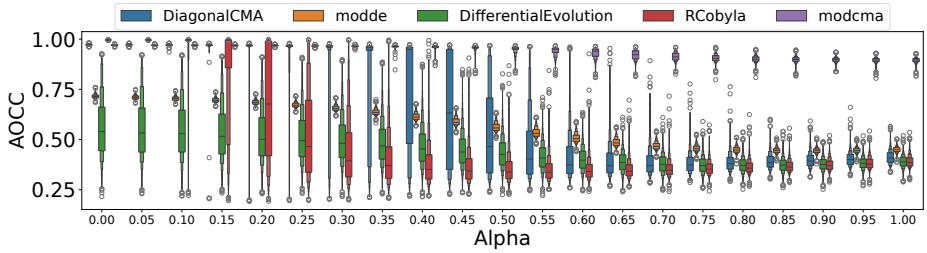


Figure 6.6: Distribution of AOCC values for 5 algorithms on the affine combinations between F10 ($\alpha = 1$) and F1 ($\alpha = 0$), for selected values of α.

Cobyla quickly starts to fail, while, for example, DiagonalCMA still manages to solve most instances at $\alpha = 0.25$ with similar AOCC. As α increases further, the modCMA's performance remains stable, showing only a minor drop in performance relative to the one seen in dCMA.

6.2.3 Impact on ELA Features

In addition to the performance perspective, we can also look at what happens to the landscape feature of the BBOB functions as we add increasingly more influence from the sphere function. Since we measure 44 different ELA features, our analysis of the impact is rather more high-level than the algorithm performance viewpoint, as we first aim to capture the overall stability of the features for increasing α values. This is measured as the sum of absolute differences in feature mean for consecutive α's, which is plotted in Figure 6.7. From this figure, we can see that the mean of most features remains quite stable, with a few notable exceptions. In particular, functions 16 and 23 show many feature changes from the sphere, which matches observations from e.g. [203, 216].

In addition to the differences between functions, it is also clear to see that features don't all behave in the same way. Since Figure 6.7 only shows absolute changes in mean, the deviation between instances might also play a role. To analyze this in some more detail, we select a single function (F10) and look at the evolution of both the feature mean and its standard deviation in Figure 6.8. From this figure, we can see that the mean of each feature seems to transition rather smoothly between the two component functions, in a similar way to the performance plots in Section 6.2.2. However, we should note that the standard deviation of many features is relatively

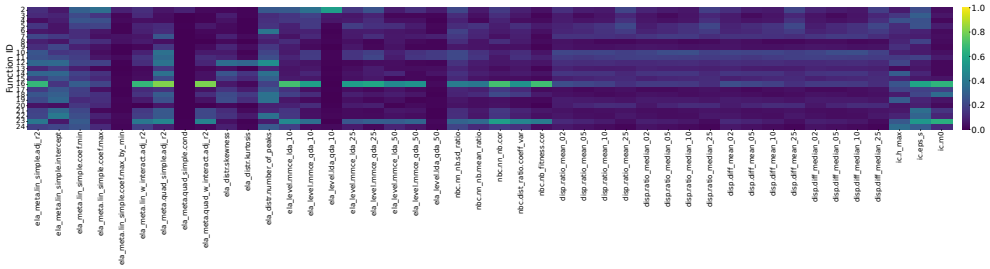


Figure 6.7: Total changes in each ELA feature when transitioning from the sphere to the function indicated in the row. Each cell represents the sum of differences in mean between pairs of consecutive values of α, so high values indicate a large total change in mean, while low values indicate features which remain stable throughout the transition.

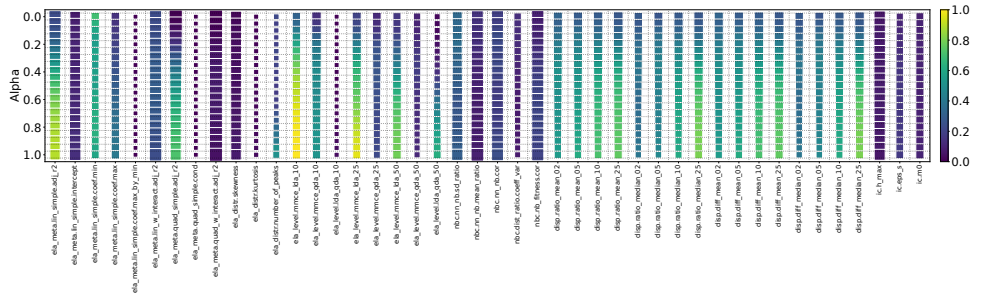


Figure 6.8: Evolution of ELA features with changing between Sphere ($\alpha = 0$) and F10 ($\alpha = 1$). The color indicates the mean of the feature over the 50 instances (lighter = larger), while the size indicates the variance (larger = higher variance).

large for each α value.

6.2.4 Impact of Optimum Location and the Instance

As can be seen from the relatively large variance in both ELA features and algorithm performance, the instance and location of the optimum can have a major impact on both the landscape and the corresponding algorithm behaviour. In particular, the way in which we defined an instance in our setup is not necessarily equivalent to the common interpretation, e.g., from BBOB. Since we allow the optimum of a component function to be moved anywhere in the domain, this can lead to large parts of the original function no longer being reachable. This is the main reason why some BBOB functions have very restricted distributions for their optima, which can be seen by

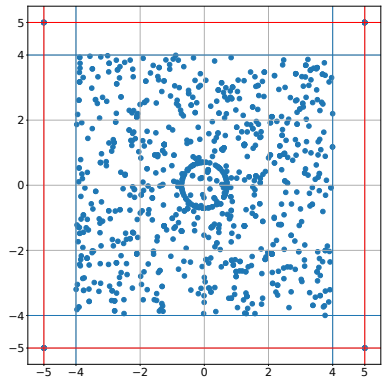


Figure 6.9: Location of optima of the 24 2d BBOB functions (1000 random instances). The red lines mark the commonly used box-constraints of $[-5, 5]^d$.

analyzing the overall distribution of optima across all BBOB functions, visualized in Figure 6.9 and previously observed in e.g.[150].

To analyse how much the algorithms in our portfolio are influenced by the choice of instance and location of optimum, we determine the relative impact of different instances for each function (F_1 and α value). This is done by first averaging the performance across all 50 runs on each instance. By dividing this by the total variance present across all runs on all instances of that function, we obtain a relative measure of 'stability' across instances, which is visualized in Figure 6.10. This figure shows that some algorithms are inherently more impacted by the instance/location of the optimum (modDE), while, for example, for Cobyla, the variance increases with increasing α, which suggests that it is very stable on the sphere problem, but becomes much more impacted by variations in the landscape when more non-sphere influence is added.

To further analyse the impact of this increased flexibility in terms of function generation, we perform an experiment involving two versions of the original BBOB functions. The first is the data from Section 6.2.2 where $\alpha \in \{0, 1\}$, which corresponds to data for 50 instances of each function, with each of them having its optimum moved to a random point in the domain. The second set of instances are created by taking the same instances of the BBOB functions, but not shifting their optimum (the rescaling from Section 6.1.2 is still applied). This allows us to compare the influence of moving the optimum on the landscape of the resulting problem. In Figure 6.11, we compare the distribution of all features on these two versions of BBOB function 23.

Finally, we can take an aggregated view of the features, and project the 44 dimen-

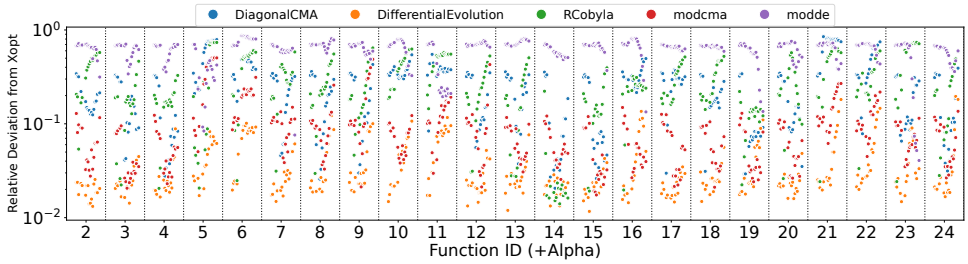


Figure 6.10: Relative deviation in AOCC caused by the change of the global optimum for all combinations of BBOB functions with the sphere model (calculated as deviation per instance divided by deviation across all instances). The x-axis indicates changing function ID, and within each function ID the transition goes from $\alpha = 0$ to $\alpha = 1$ (left to right).

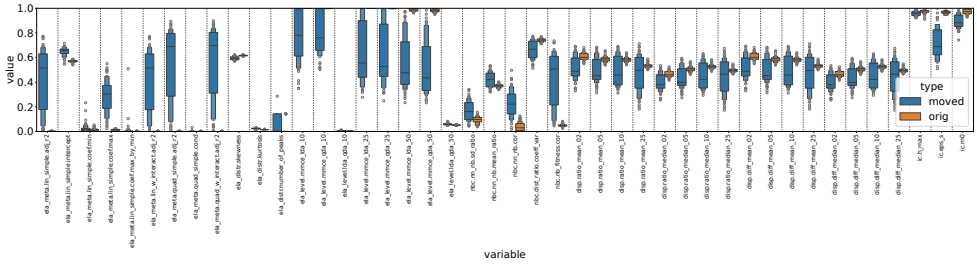
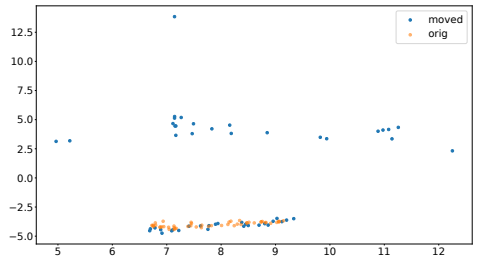
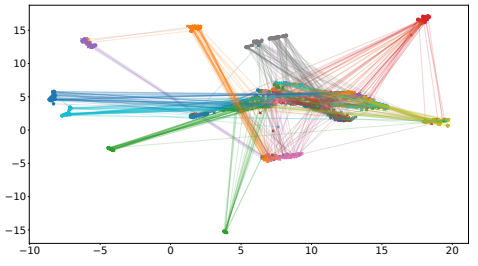


Figure 6.11: Distribution of normalized ELA features for the BBOB instance creation procedure and the same instance moved to have an optimum location uniformly in the domain, for F23.



(a) Projection of the moved and original version of F18 (50 instances).



(b) Projection of all BBOB functions connecting the original (circles) and moved (crosses) versions of each function.

Figure 6.12: UMAP projection trained on original BBOB, then used to plot both the moved and original functions in 2D.

sional space into two dimensions using PCA on the original BBOB versions. Then, we can plot the moved versions of the function into the same space, and observe the differences. The result of this projection is shown in Figure 6.12, where we see that many functions are moved much closer to the center of the projected space. This suggests that some of the 'unique' feature combinations present in the original BBOB functions are being lost when moving their optimum. This happens because large parts of the function are moved outside of the domain, and replaced by parts which were originally located outside the bounds. For some functions, these components are exponentially increasing, leading to a large part of the space which is dominated by these artifacts, which is represented in the ELA-features.

6.2.5 Pairwise Combinations

While combining functions with a sphere model can be viewed as adding global structure to a problem, combinations between other functions can provide interesting insights into the transition points between different types of problems. To illustrate the kinds of insights that can be gained from these combinations, we select a subset of 5 functions and collect performance data on each combination with the same 21 α values (with both orderings of the function). We show the performance in terms of normalized AOCC of diagonal CMA-ES on these function combinations in Figure 6.13. Note that for $\alpha = 1$, we are using the function specified in the column label, while for $\alpha = 0$ we have the function specified in the row label.

When comparing this figure to its equivalent from the GECCO paper [251], it is

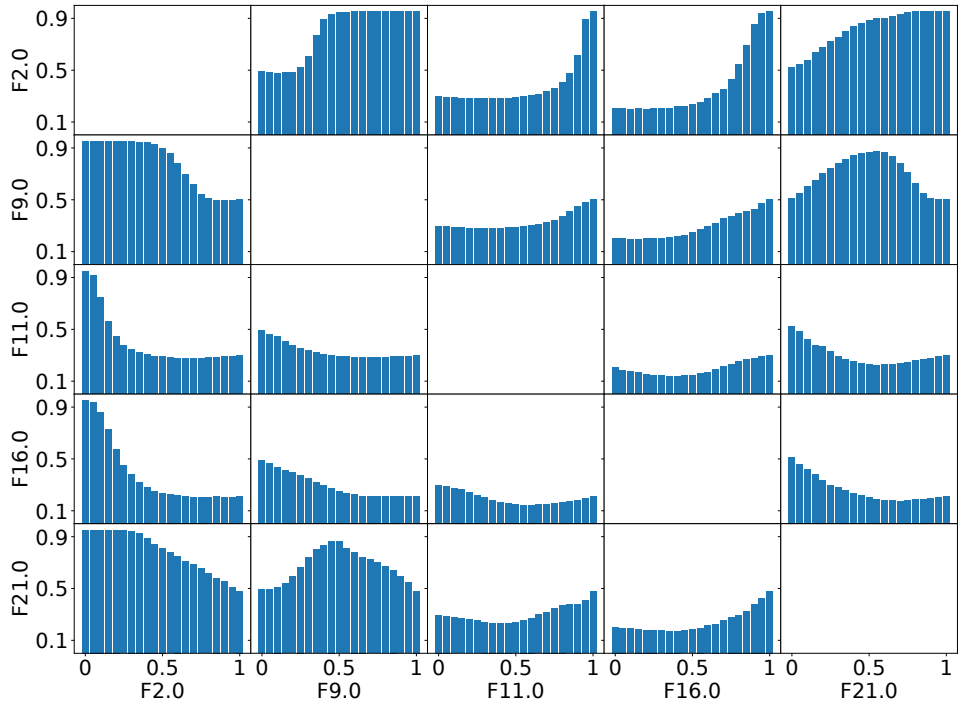


Figure 6.13: Normalized area over the convergence curve for Diagonal CMA-ES on each of the affine combinations between the selected BBOB problems. Each facet corresponds to the combination of the row and column function, with the x-axis indicating the used α. AOCC values are calculated based on 50 runs on 25 instances, with a budget of 10 000 function evaluations.

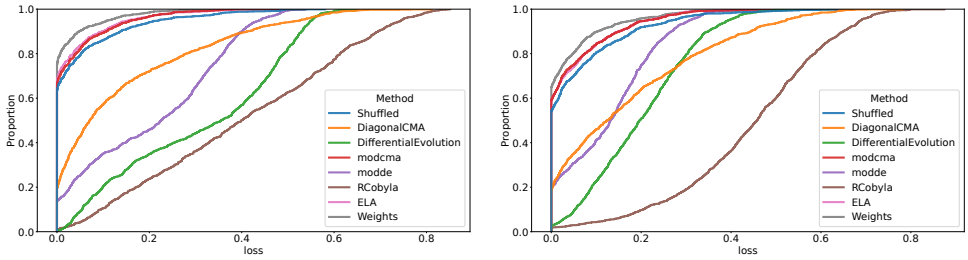
important to note the fact that Figure 6.13 is almost fully symmetric around the diagonal, which was not the case in the GECCO paper. Even though we might expect (F_1, F_2, α) to be similar to $(F_2, F_1, 1-\alpha)$, this was not the case when the location of the global optimum was selected as the optimum of one of the component functions, as different BBOB functions can have significantly different distributions of potential global optima [150]. This is in large part the reason why we enabled the MA-BBOB generator to sample the optimum uniformly at random in the domain. While Section 6.2.4 showed that this can potentially move interesting parts of some component functions outside the domain, we view this as a worthwhile tradeoff to achieve fully unbiased global optima.

From Figure 6.13, we can also see that the transition of performance between the two extreme α values is mostly smooth. While there are some rather quick changes, e.g., for the transition between F2 and F11, these seem to be the exception rather than the rule. Particularly interesting are the settings where the performance of affine combinations between two functions proves to be much easier or harder than the functions which are being combined. For example, this is the case for the combinations of F21 and F9.

6.3 Combining Multiple Functions: Testing Generalizability

For our final set of experiments, we make use of a set of 1000 functions generated using the setup described in Section 6.1.2. This data is taken directly from [250], and contains both ELA and performance data (for the same set of algorithms described in Section 6.2.1, but using the original AUC measure instead of the AOCC). In [250] we analyzed this data to understand the MA-BBOB instance generation procedure, with the goal of generating a wide set of benchmark problems on which algorithm selection and other automated machine learning techniques can be tested.

In this experiment, we take the perspective of algorithm selection and train a random forest model to predict the best algorithm to use for each function, based either on the ELA features of the problem or the weights of the component functions. We can then compare the loss in terms of AUC relative to the virtual best solver (VBS) for both of these models, in different training contexts. We can either use the common cross-validation setup, or attempt to test for generalization ability based only on the original BBOB functions. In Figure 6.14a we show the cumulative loss



(a) AUC loss for 5 dimensional functions. **(b)** AUC loss for 2 dimensional functions.

Figure 6.14: Cumulative loss (AUC) for different models: cross-validation (mixture of BBOB + MA-BBOB generated combinations) based on weights and ELA, and each of the single-algorithm models.

for the cross-validation setup on the 5-dimensional functions. From this, we can see that the ELA-based selector performs worse than the one based on the weights. This confirms the previous observation that the ELA features might not be sufficiently representative to accurately represent the problems in a way which is relevant for ranking optimization algorithms.

In order to better estimate how much the structure of the ELA features helps the prediction, we can add in a naive baseline. This is created by shuffling the labels (best ranked algorithm) of all samples before training. This shuffled model is in essence just a selector based on the frequency of labels in the training data, and the difference between this version and the original ELA-based selector shows how much the structure of the ELA-features helps improve the predictions. The results for the cross-validation setup in 2D are shown in Figure 6.14b, where we see that the benefit over most of the individual algorithms is inherent to the selected portfolio, since the shuffled model outperforms all algorithms except modCMA. This suggests that our algorithm portfolio is severely unbalanced.

When looking at the generalization task, this imbalance is exacerbated further, since for MA-BBOB the modCMA is ranked first on an even larger fraction of functions than on BBOB, as shown in Figure 6.15. In combination with the added challenge of transferring to a new suite, this leads to our algorithm selection models being outperformed by the modCMA, which is the Single Best Solver (SBS) in this case, as illustrated in Figure 6.16. While the algorithm portfolio is partly responsible for this shortcoming, the generalization ability does not significantly improve when removing the modCMA from our portfolio. This suggests that training on the original BBOB

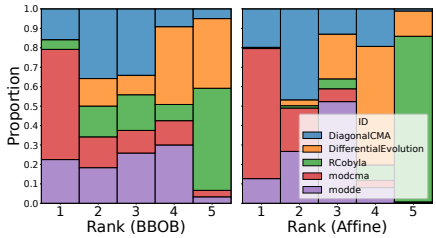


Figure 6.15: Distribution of ranks based on per-function AUC after 10 000 evaluations.

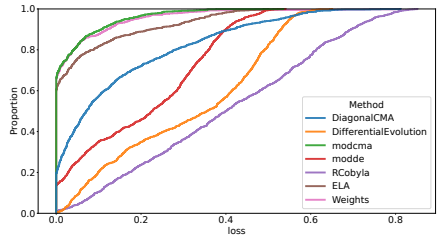


Figure 6.16: Cumulative loss (AUC) on the 5 dimensional MA-BBOB problems for models trained on the BBOB functions, and each of the single-algorithm models.

instances does not sufficiently represent the challenges faced in the MA-BBOB suite. An important aspect of the challenge of this transfer is the location of the optima, as discussed in Section 6.2.4.