Population-Based Black-Box Optimization for Biological Sequence Design: Supplementary Material

Christof Angermueller [1] **David Belanger** [1] **Andreea Gane** [1] **Zelda Mariet** [1] **David Dohan** [1] **Kevin Murphy** [1] **Lucy Colwell** [1 2] **D Sculley** [1]

A. In-Silico Design problems

problem	Num inst.	Num rounds	Batch size	Vocab size	Seq length	Init size
TfBind8	12	10	100	4	8	0
TfBind10	2	10	100	4	10	0
RandomMLP	16	10	100	20	20/40	0
RandomRNN	12	10	100	20	20/40	0
PfamHMM	24	10	500	20	50-100	500
ProteinDistance	24	6	500	20	50-100	0
PDBIsing	10	10	500	20	20/50	0
UTR	1	10	1000	4	50	0

Table 1: Information about benchmark problems, including the number of instances per problem, the number of optimization rounds, batch size, vocabulary size, sequence length, and the number of initial samples. RandomMLP and RandomRNN considers sequences of length 20 or 40. The sequence length of PfamHMM and ProteinDistance varies between 50 and 100. PfamHMM provides methods with 500 initial samples. All other problems do not provide initial samples.

Table 1 summarizes the considered optimization problems. For each problem, we construct multiple instances by varying the protein target (PDBIsing, PfamHMM, TfBind), or the neural network architecture and random seed for weight initialization (RandomMLP/RandomRNN). We provide additional details per problem in the following sub-sections.

A.1. TfBind

The optimization goal is to produce length-8 DNA sequences that maximize the binding affinity towards a particular transcription factor. We use the following transcription factor to create 12 optimization problems: CRX_REF_R1, CRX_R90W_R1, NR1H4_REF_R1, NR1H4_C144R_R1, HOXD13_REF_R1, HOXD13_Q325R_R1, GFI1B_REF_R1, FOXC1_REF_R1, PAX4_REF_R1, PAX4_REF_R2, POU6F2_REF_R1, and SIX6_REF_R1.

We min-max normalize the binding affinity values for each transcription factor target to the zero-one interval.

TfBind10 differs from TfBind8 in that sequences are of length 10 and only two transcription factors (Cbf1 and Pho4) are available, which were characterized experimentally (Le et al., 2018).

A.2. Random Neural Network

The goal is to optimize the scalar output of a randomly initialize neural network. Optimization proceeds over 10 rounds with a batch size 100. We construct different instances by varying the sequence length (20 or 40), the vocabulary size (4 or 20), the random seed (0 or 13), and the architecture of the network (described in the following).

RandomMLP considers networks with a varying number of convolutional and fully connected layers. Networks have either no convolutional layer, or one layer with 128 units, a kernel width of 13, and a stride size of 1. The number of fully connected layers is either one (128 hidden units) or three (128, 256, 512 hidden units). We use a linear activation function for the output layer and a relu activation function for all other layers.

RandomRNN considers networks with one (128 hidden units), two (128, 256 hidden units), or three (128, 256, 512 hidden units) LSTM layers.

A.3. PfamHMM

Sequences annotated by Pfam within each family have variable length and the likelihood under the HMM can be evaluated for arbitrary-length sequences. For simplicity, however, here we consider optimization over fixed-length sequences. The length is chosen as the median length of unaligned sequence domains that belong to the Pfam-full sequence alignment for the corresponding family.

The initial dataset is obtained by (1) selecting all sequences from Pfam-full that belong to the given family and have the chose length, (2) evaluating their likelihood under the

[1]Google Research [2]University of Cambridge. Correspondence to: Christof Angermueller <christofa@google.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

HMM, and (3) sampling 500 sequences with a likelihood below the 50th percentile.

We selected families relatively short sequences. Future work might consider longer sequences while optimizing only a subset of positions.

A.4. ProteinDistance

The ProteinDistance problem tasks methods to find sequences with a high cosine similarity in the embedding space to a chosen target sequence. We used the network from Bileschi et al. (2019) to obtain embeddings of proposed sequences. This network was trained to classify the Pfam family of a protein sequence, and its embeddings have been shown to capture broad protein features that are useful for fewshot learning. We construct optimization problems by choosing different Pfam HMM seed sequences as the target sequence, and used the same Pfam families that we used the PfamHMM problem.

A.5. PDBIsingModel

We employ $f(\mathbf{x}) = \sum_i \phi_i(x_i) + \beta \sum_{ij} C_{ij}\phi(x_i, x_j)$, where x_i refers to the character in the i-th position of sequence x. C_{ij} is a binary *contact map* dictating which positions are in contact and $\phi(x_i, x_j)$ is a position-independent *coupling block*. We construct optimization problems by choosing 10 different proteins from the Protein Data Bank (Berman et al., 2003). $C_{ij} = 1$ if the $C\alpha$ atoms of the amino acid residues at positions i and j are separated by less than 8 Angstroms in the protein's 3D structure. The coupling block is based on global co-occurence probabilities of contacting residues (Miyazawa & Jernigan, 1996). The local terms $\phi_i(x_i)$ are set using the sequence of amino acids for the true PDB protein. The score for setting x_i to a specified value is given by the log Blosum substitution probability between that value and the corresponding value in the PDB sequence. Finally, β is chosen heuristically such that the local terms do not dominate the objective too much.

B. Optimization Methods

B.1. Model-Based Optimization

We follow (Angermueller et al., 2020) for building the regressor model. We optimize the hyper-parameters of diverse regressor models by randomized search, and evaluate model performance by explained variance score estimated by five-fold cross validation. We select all models with a score ≥ 0.4 and build an ensemble by averaging their predictions. We consider the following scikit-learn (Pedregosa et al., 2011) regressor classes and hyper-parameters:

- BayesianRidge: alpha_1, alpha_2, lambda_1, lamdba_2
- RandomForestRegressor: max_depth, max_features, n_estimators
- LassoRegressor: alpha
- GaussianProcessRegressor: kernel (RBF, RationalQuadratic, Matern) and kernel parameters

We use the posterior mean acquisition function, which performed as good or better as the upper confidence bound, expected improvement, or probability of improvement. We optimize the acquisition function using evolutionary search for 500 rounds with a batch size of 25. We construct the next batch by selecting the top n unique and novel sequences with the highest acquisition function value.

B.2. Latent-Space Model-Based Optimization

An alternative approach to discrete optimization is to train an encoder-decoder model that enables mapping discrete sequences to and from continuous vectors and moving the optimization process into the continuous space (Gómez-Bombarelli et al., 2018; Kusner et al., 2017; Roeder et al., 2018; Killoran et al., 2017; Cao et al., 2019; Luo et al., 2018; Gupta & Kundaje, 2019). The approach has been introduced for problems with a large amount of (unlabeled) initial data in a single-round optimization set up, which we extend to multi-round optimization. In each round, we use all the observed sequence-reward pairs to jointly train a variational auto-encoder (VAE) (Kingma & Welling, 2014) and a neural network regressor from latent embeddings to the corresponding rewards. Jointly training the regressor and VAE encourages organizing latent representations by reward scores (Gómez-Bombarelli et al., 2018). Once the encoder-decoder model has been fixed, similar to the previous work, we train a new regressor from scratch on the embedding-reward pairs, and use it to score new sequences during continuous optimization.

We train the VAE by maximizing the variational lower-bound (Kingma & Welling, 2014). We anneal the KL divergence and the neural network regressor loss over time. We standard scale regressor target labels, and use the mean-squared error as traning objective. The encoder and decoder architectures match those in the DbAs-VAE; the regressor is a fully-connected network with a single hidden layer. We consider the following hyper-parameters (actual values in the parentheses): latent space size (50), learning rate (0.006), weight of the regressor loss term (2 * sequence length), sigmoid annealing slope (1.), batch size (20), number of epochs (60), hidden sizes for the encoder (128), decoder (128), and regressor (50).

For training a new regressor on the embedding-reward pairs, we use the ensemble-based approach described Section B.1.

We perform optimization in the latent space using the cross-entropy method with a diagonal multivariate Gaussian generative model. We perform weighted maximum likelihood

on high scoring samples, re-weighted by the likelihood ratio of a given prior and the current generative model (CbAs) (Brookes et al., 2019). We use as prior a diagonal multivariate Gaussian generative model fit on the training data of the VAE and regressor. The re-weighting approach encourages the optimization trajectory to remain close to the prior, which is desirable as the latent regressor performance is expected to degrade as we move away from training samples.

We run the cross-entropy method for a fixed number of iterations. The proposed batch is obtained by decoding the best-scoring unique samples (according to the regressor score) across multiple runs of the cross-entropy method with different random seeds. We tune the following hyper-parameters (actual values in the parentheses): the number of cross-entropy method instances (25), the number iterations per instance (20), the number of samples drawn per iteration (100), and the number high-scoring samples to extract (10). We initialize the cross-entropy method with the embeddings of the highest scoring 25 sequences observed during the optimization. Finally, we note that, as optimization is being performed in the continuous space, one could use a differentiable regressor and gradient-based optimization.

B.3. Evolution

Evolution generates a new child sequence by selecting two parents from the population of $(\mathbf{x}, f(\mathbf{x}))$ pairs, recombining them, and mutating them. At each optimization rounds, at repeats these steps iteratively to generate a batch of child sequences. Following (Real et al., 2019), samples are no longer considered during parent selection after a fixed number optimization rounds to prevent elite samples from dominating the population ("death by old age"). Two parents are chosen for each child via tournament selection, which involves taking the best of T samples from the alive population. The chosen parent sequences A and B are recombined by copying the sequences from left to right beginning with a pointer on parent A. At each position, the pointer has some probability (we used 0.1) of switching to reading the other parent. After crossover, we mutate the child by changing each position to a different value with a fixed mutation probability (we used 0.01).

B.4. DbAs-VAE

We follow (Brookes & Listgarten, 2018) and use a fully connected MLP encoder and decoder with one hidden layer and 64 units, 32 latent vectors, which is trained with a learning rate of 0.01 for 60 epochs and a batch size of 20. We use a quantile-cutoff of 0.85.

B.5. DbAs-RNN

Same as DbAs-VAE, except that the generative model is a LSTM with one hidden layer and 128 hidden units.

B.6. FBGAN

Following (Gupta & Zou, 2018), we use a Wasserstein GAN as generative model inside cross-entropy optimization. Unlike using a fixed threshold for selecting sequences, we use a quantile cutoff since it performed better in our experiments on diverse problems. We tune the quantile cutoff, learning rate, batch size, discriminator and generator training epochs, the gradient penalty weight, the Gumble softmax temperature, and the number of latent variables of the generator.

B.7. P3BO

We use the population size of 15, and note that similar performance can be achieved with a smaller population size greater than 5 (see Figure 11). We use MBO, DbAs-VAE, DbAs-RNN, Evolution, and SMW as constituent algorithm classes (Section 6.1). We sample hyper-parameters of these algorithm classes from following distributions:

SMW This method is hyper-parameter free.

Evolution

- crossover_probability: uniform(interval(0.1, 0.3))
- mutation_probability: uniform(interval(0.05, 0.2))

DbAs-VAE

- quantile: uniform(0.825, 0.975)
- learning_rate: loguniform(interval(0.008, 0.012))
- num_vae_units: uniform(categorical(64, 128))

DbAs-RNN

- quantile: uniform(0.9, 0.975)
- learning_rate: loguniform(interval(0.0005, 0.012))
- num_lstm_units: uniform(categorical(64, 128))

MBO

- acquisition_function(uniform(categorical([PosteriorMean, UCB])))
- ucb_scale_factor: uniform(interval(0.5, 1.2))
- regressor: uniform(categorical(['Ensemble', 'BayesianRidge']))

Here, 'Ensemble' refers to the ensemble model described in B.1 and 'BayesianRidge' to the BayesianRidge regressor of the scikit-learn package.

C. Diversity metrics

Finding not only one but multiple diverse high reward sequences is desirable to increase the chance that some of the

found sequences satisfy downstream screening criteria that are not captured by the primary optimization objective, e.g. stability or viscosity. For quantifying diversity, we devised the following metrics:

Mean pairwise Hamming distance We compute the mean pairwise Hamming distance of sequence in a batch. We also considered the Edit distance but found it to be too slow to compute for problems with large batch sizes.

Mean entropy For a given batch of sequences, we compute the entropy over characters at each position, and then average over positions. Since we found this metric to be highly correlated with the mean pairwise Hamming distance ($R^2 = 0.99$), we only show results for the mean pairwise Hamming distance.

Reward vs. Hamming distance We plot the mean reward of sequences in a batch. vs. the mean pairwise Hamming distance, which shows the trade-off of these two metrics.

Number of high-reward clusters For problems with known maximum we first select all sequences proposed so far with a reward above a threshold, e.g $0.8 \cdot \max f(x)$. We then cluster the selected sequences using hierarchical complete linkage clustering, and count the number of resulting clusters with a minimum length-normalized Hamming distance if 0.5. This metric can be only maximized by finding both well separated and high reward sequences.

Fraction of optima found This metric can be only computed for enumerative optimization problems, in our case TfBind8. In a pre-computation step, we first select all possible sequences with a reward ≥ 0.9 (the maximum reward is 1.0, Section A.1). We then cluster selected sequences into distinct clusters with a minimum edit distance of 3 using hierarchical complete linkage clustering. We define the sequence with the highest reward of each cluster as one optimum. We count an optimum as found if the exact same sequence is proposed. We treat the forward and reverse sequence as two distinct optima. This metric differs from the *Number of high-reward clusters* metric in two points: 1) the number of clusters is normalized by the total number of clusters, and 2) the edit distance is used for clustering sequences to assign sequences that contain similar motifs at different positions to the same cluster.

D. Additional results

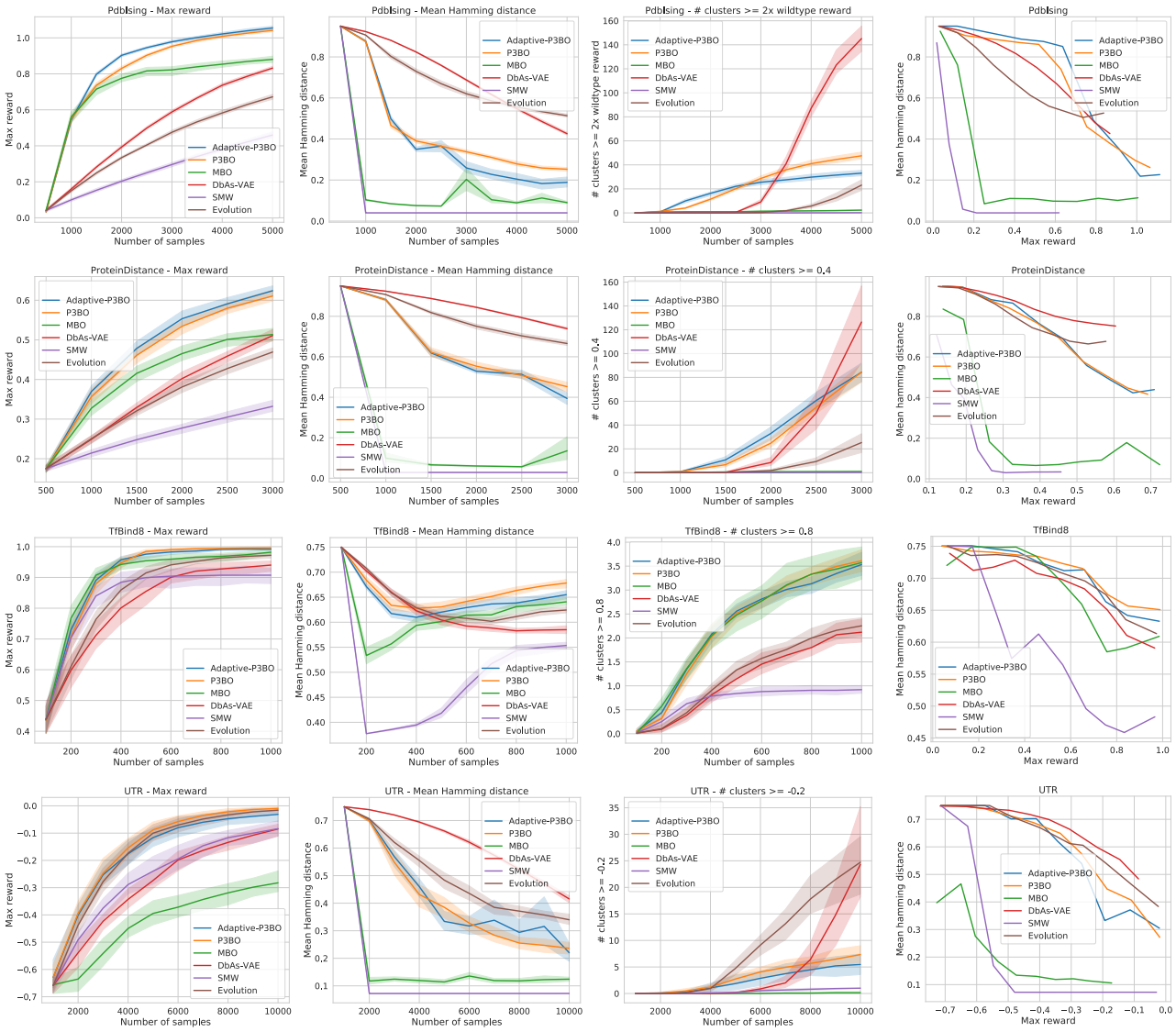


Figure 1: Visualization of alternative diversity (Section C) for the PdbIsing, ProteinDistance, TfBind8, and UTR problem. We find that DbAs-VAE and Evolution generate more diverse sequences than P3BO, albeit with a lower reward. P3BO finds significantly more diverse and higher reward sequences than both MBO and SMW, and provides overall a good trade-off between finding high reward and diverse sequences.

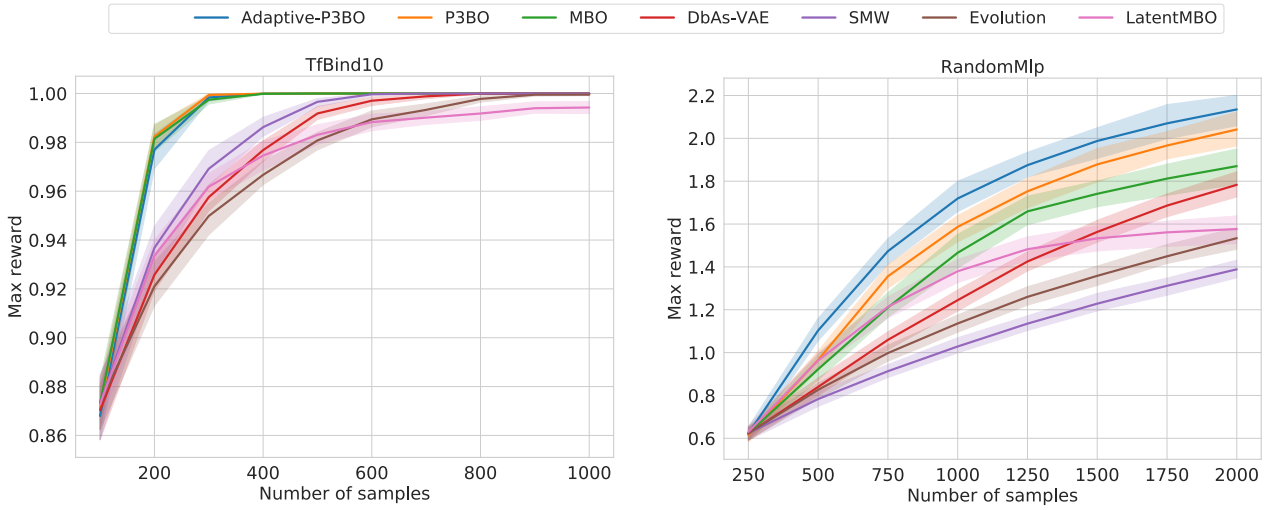


Figure 2: Performance curves for the RandomMLP and TfBind10 problem, which were not shown in the main text due to space limitations. Lines show the average over all targets available for each problem, while the shaded areas indicate the 95% boostrap confidence-intervals.

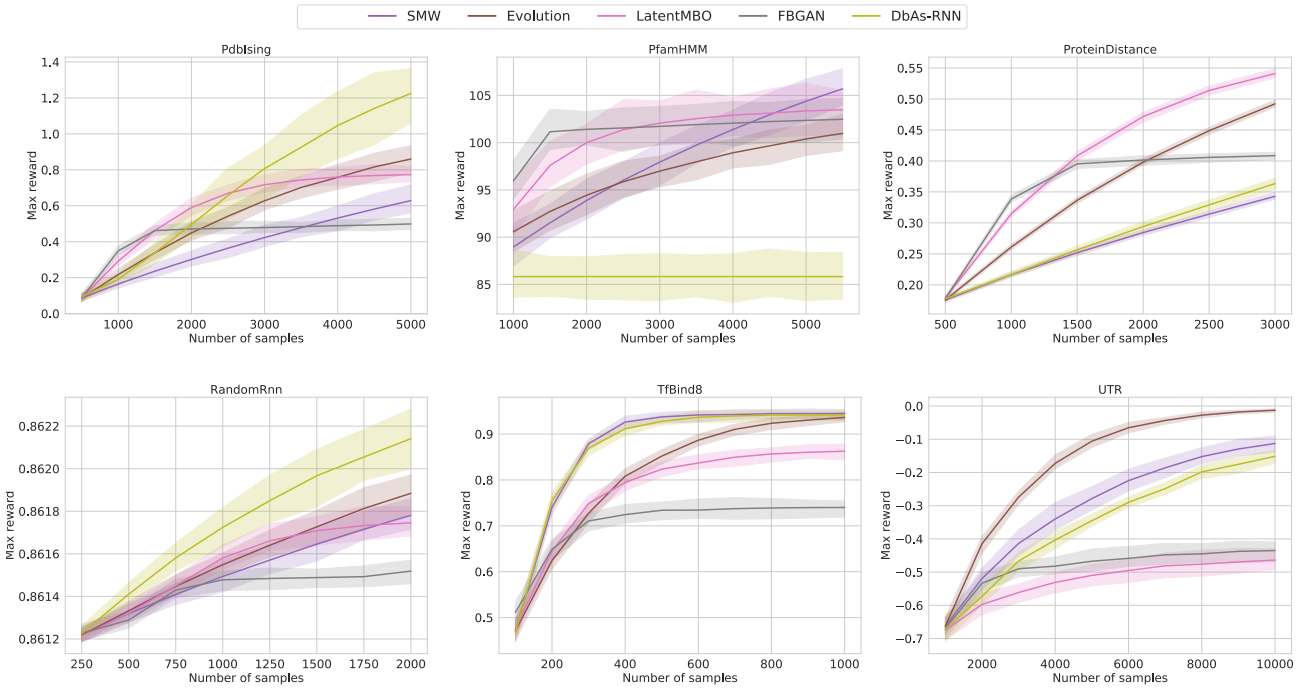


Figure 3: Performances curves for additional baselines (FBGAN and DbAs-RNN), which were not shown in the main text due to space limitations. Lines show the average over all instances per optimization problem, while the shaded areas indicate the 95% bootstrap confidence-intervals.

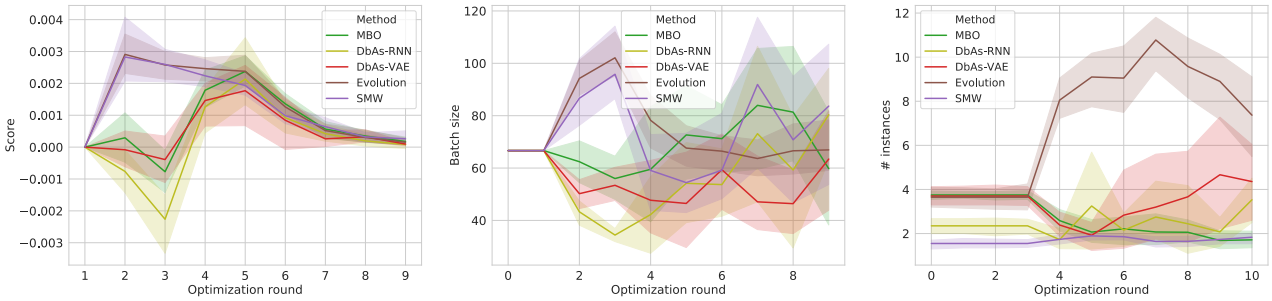
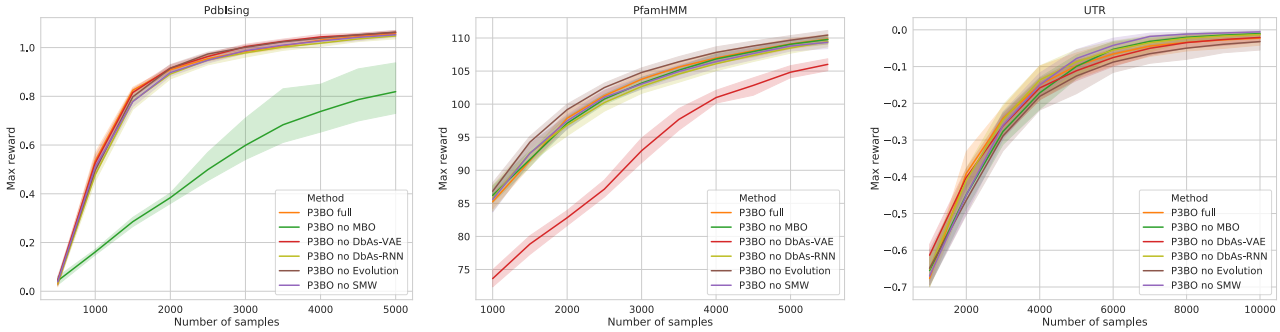
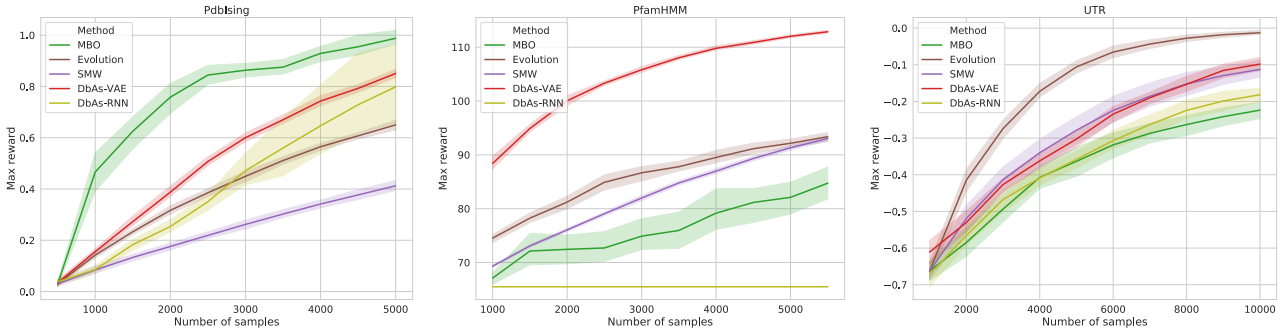


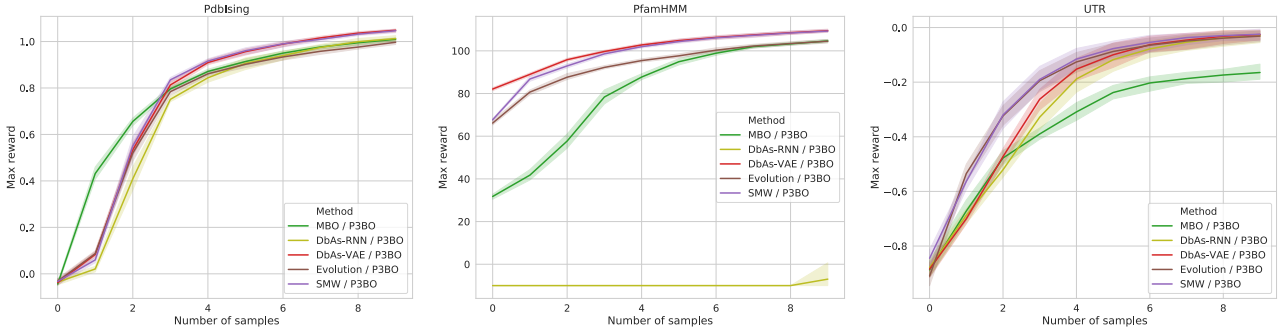
Figure 4: Insights into Adaptive-P3BO applied to the UTR problem. Shown are the credit score (left), the number of sequences sampled (middle), and the number of instances (right) per algorithm class over time. Since *Evolution* has the highest credit score (relative improvement) for early rounds, more sequences are sampled from *Evolution* (middle), and Adaptive-P3BO increases the number of *Evolution* instances from 4 to 11 (the total population size is 15). The adaptation starts after three warm-up rounds used to reliably estimate the credit score of algorithms.



(a) Performance of P3BO when removing individual algorithms.

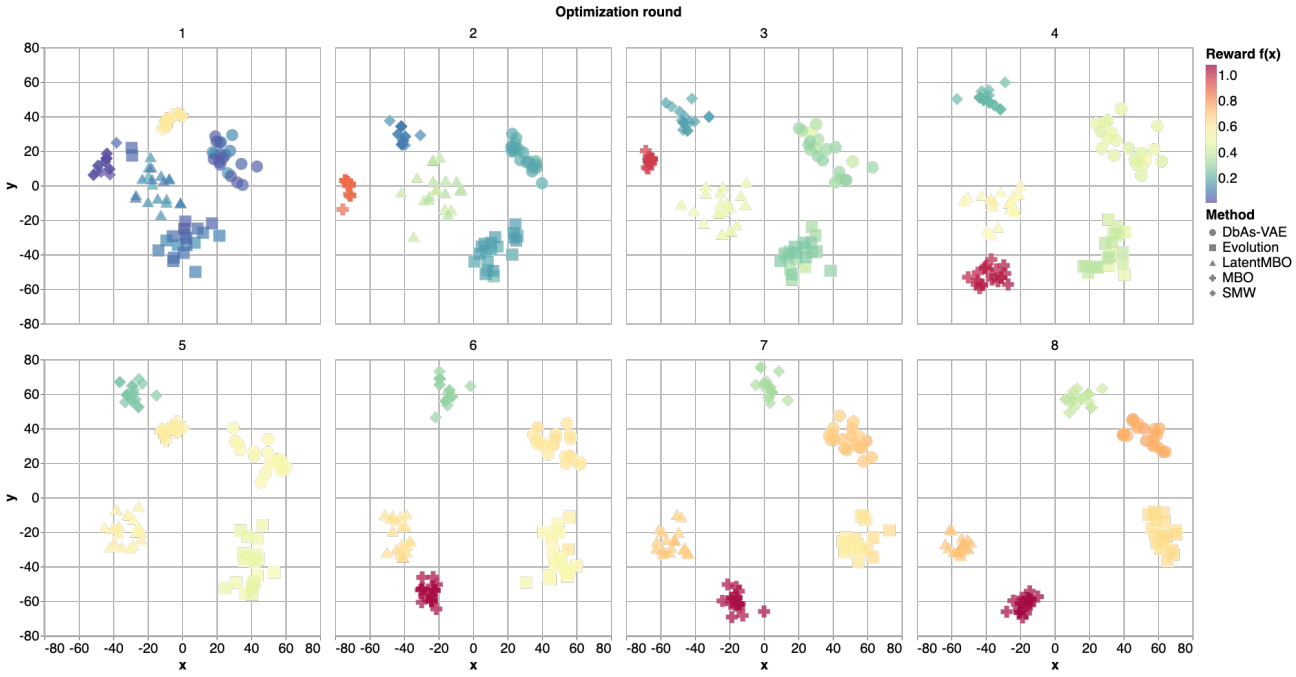


(b) Performances of methods when used stand-alone without data sharing.

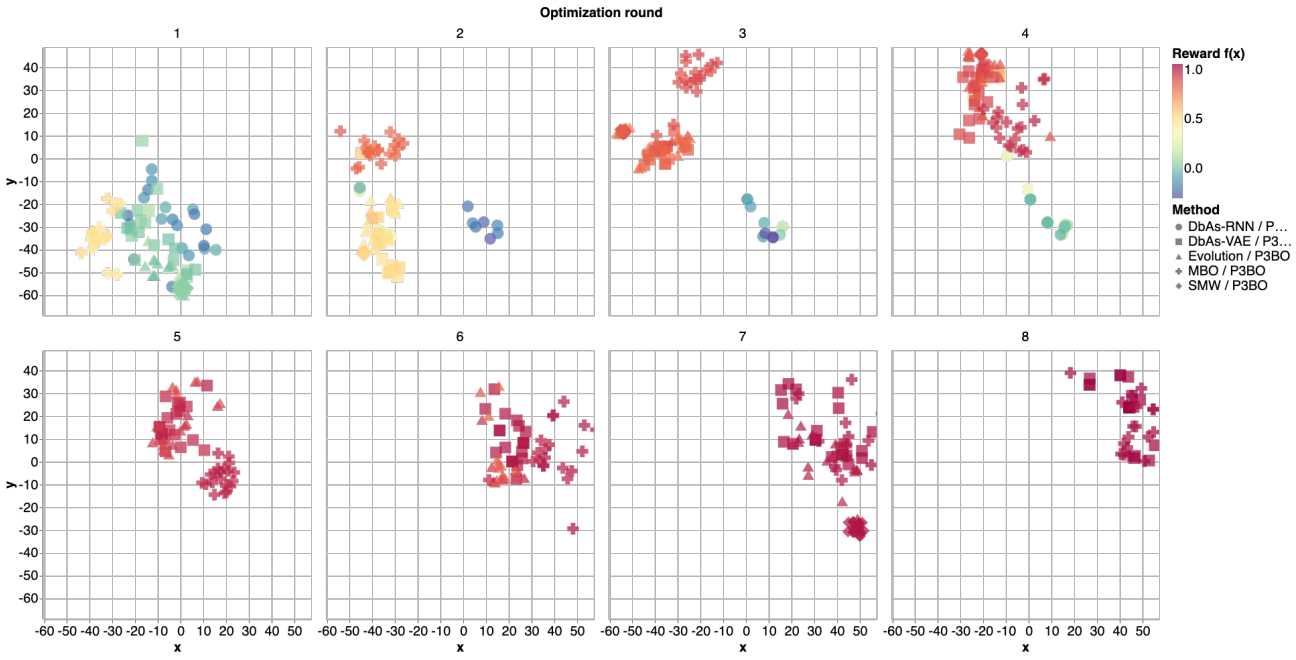


(c) Performances of methods when used inside P3BO with data sharing.

Figure 5: Performance of P3BO on the PdbIsing, PfamHMM, and UTR problem when removing individual algorithms from its population. The top row compares P3BO with all algorithms (P3BO full) to variants with one algorithm removed. The middle row shows the performance of algorithms when used stand-alone without sharing data (samples $(x, f(x))$)a, and the bottom row when used inside P3BO with data sharing. Removing MBO from the population of P3BO results in a performance drop on PdbIsing (top row) due to the good performance of MBO on that problem (middle row). In contrast, DbAs-VAE is the best performing algorithm on PfamHMM, which results in a performance drop when removing it. Sharing samples acquired by one algorithm with all other algorithms in the population (bottom row) results in a higher performance of individual algorithms than without sharing samples (middle row). For example, SMW benefits from the high-reward sequences found by MBO on PdbIsing in early rounds (middle row, left plot) and thereby manages to find sequences with a higher reward than MBO in following rounds (bottow row, left plot).



(a) t-SNE of sequences without data sharing.



(b) t-SNE of sequences with data sharing.

Figure 6: t-SNE of sequences proposed by different algorithms with and without sharing samples $(x, f(x))$ between algorithms. The shape of each point (sequence) corresponds to the algorithm that proposed the sequence x and the color to the reward $f(x)$. Without sharing samples, methods propose distinct, well separated, sequences, and only MBO finds high reward sequences quickly. By sharing samples, methods benefit from the high reward sequences discovered by MBO in early rounds and propose similar sequences in subsequent rounds. Results are shown for PdbIsing problem with PDB structure 1KDQ.

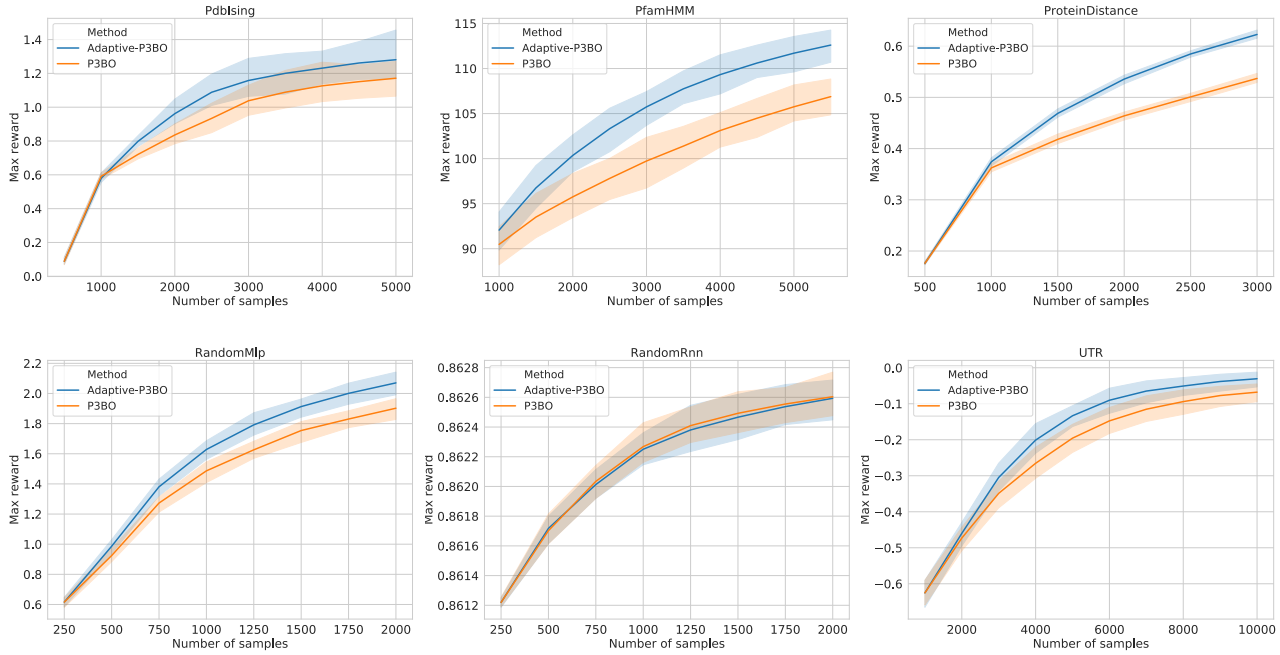


Figure 7: Performance comparison of P3BO and Adaptive-P3BO when starting with a poorly initialized population of algorithms. By adapting the population online, Adaptive-P3BO can recover from the initial population, resulting in a clear performance improvement.

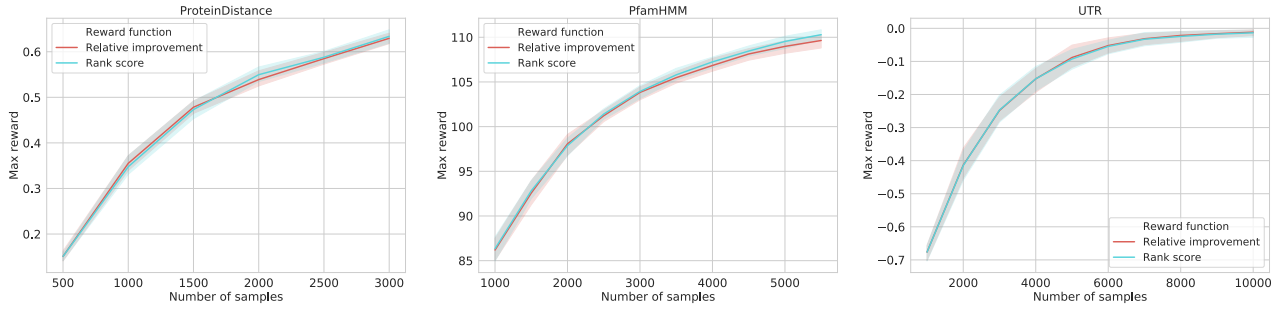


Figure 8: Comparison of the relative improvement reward function described in Section 3.2 with the rank-based reward function as proposed by Fialho et al. (2010). Both approaches perform similarly across problems.

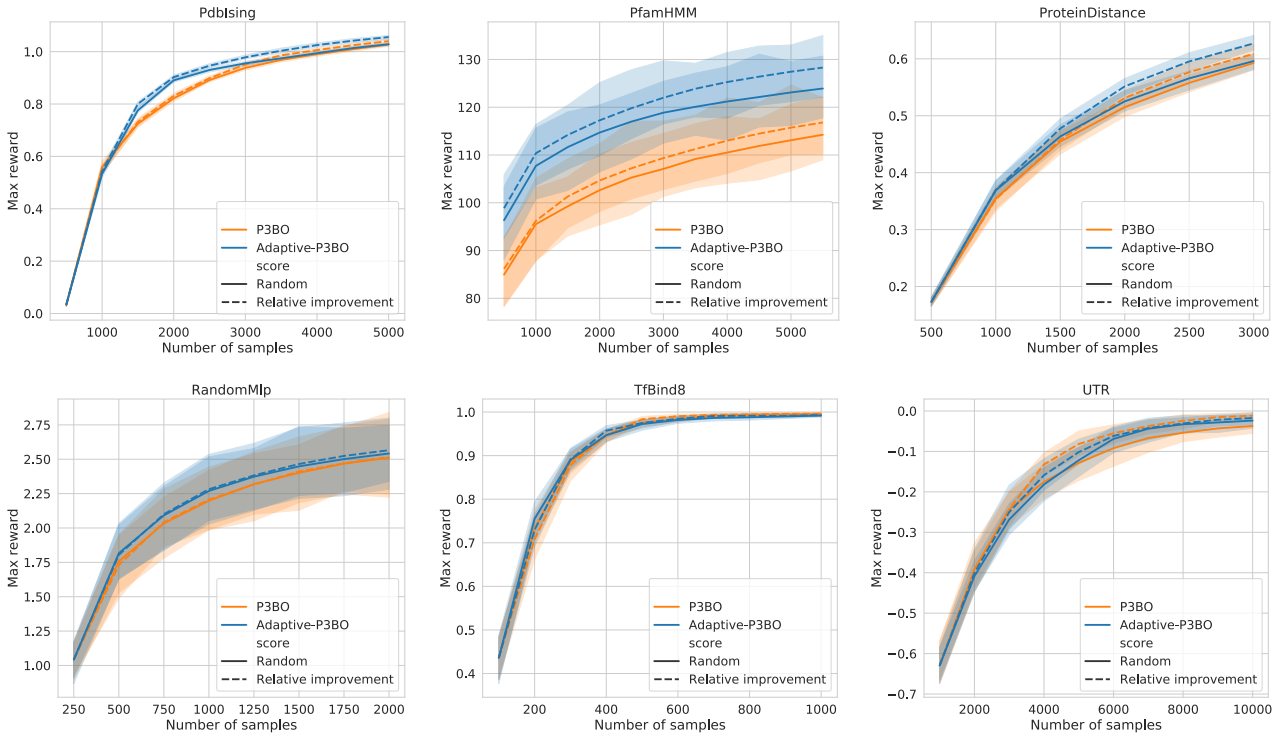


Figure 9: Performance of P3BO and Adaptive-P3BO when using the relative improvement scoring function described in Section 3.2 (dashed line) vs. scoring methods randomly (solid line).

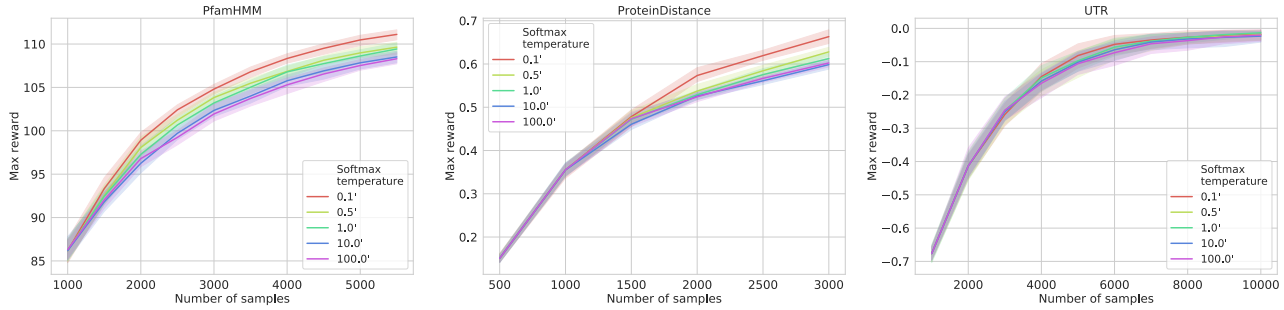


Figure 10: Sensitivity of P3BO to the softmax temperature τ for computing selection probabilities (Section 3.2) on the PfamHMM, ProteinDistance, and UTR problem. Shows that scaling the number of sequences sampled from algorithms in the population proportional to their credit score ($\tau < 10$) is better than sampling sequences uniformly ($\tau \geq 10$).

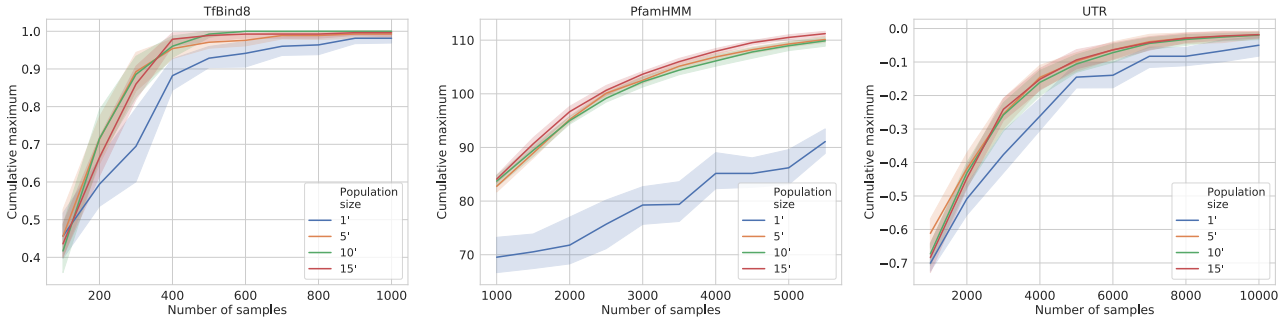


Figure 11: Cumulative maximum of P3BO for different population sizes N on three optimization problems. While a population size of 15 is best on average, similar performances can be achieved with smaller populations. Using only one algorithm ($N = 1$) results in a clear performance decrease. For this analysis, the initial population was sampled randomly from a pool of algorithms as described in Section 6.2, subject to sampling at least one instance per algorithm class.

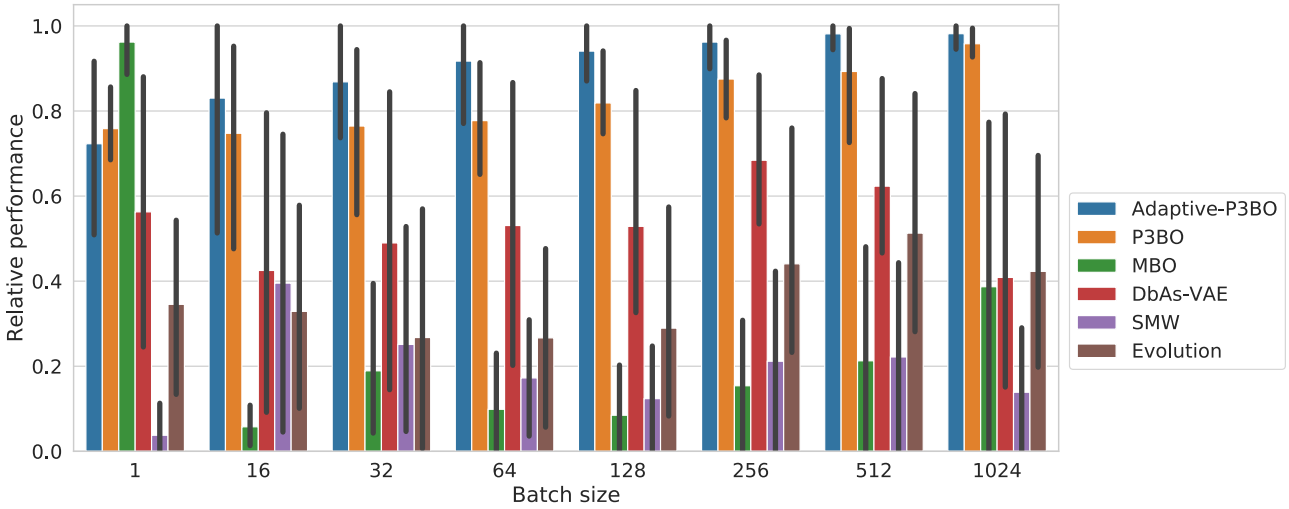


Figure 12: Relative performance of methods depending on the number of samples per optimization round (batch size). The y-axis corresponds to the area under the cumulative max reward curve, min-max normalized across methods. Error bars show the variation across five optimization problems (one instance of the ProteinDistance, PfamHMM, UTR, and RandomMLP problem). Shows that P3BO is applicable to optimization settings with various batch sizes, including non-batched optimization.

References

Angermueller, C., Dohan, D., Belanger, D., Deshpande, R., Murphy, K., and Colwell, L. Model-based reinforcement learning for biological sequence design. In *ICLR*, 2020.

Berman, H. M., Bourne, P. E., Westbrook, J., and Zardecki, C. The protein data bank. In *Protein Structure*. CRC Press, 2003.

Bileschi, M. L., Belanger, D., Bryant, D. H., Sanderson, T., Carter, B., Sculley, D., DePristo, M. A., and Colwell, L. J. Using deep learning to annotate the protein universe. *bioRxiv*, 2019.

Brookes, D. H. and Listgarten, J. Design by adaptive sampling. *arXiv preprint arXiv:1810.03714*, 2018.

Brookes, D. H., Park, H., and Listgarten, J. Conditioning by adaptive sampling for robust design. *arXiv preprint arXiv:1901.10060*, 2019.

Cao, S., Wang, X., and Kitani, K. M. Learnable embedding space for efficient neural architecture compression. *arXiv preprint arXiv:1902.00383*, 2019.

Fialho, ., Schoenauer, M., and Sebag, M. Fitness-AUC bandit adaptive strategy selection vs. the probability matching one within differential evolution: an empirical comparison on the bbob-2010 noiseless testbed. In *Proceedings of the 12th annual conference comp on Genetic and evolutionary computation - GECCO '10*, pp. 1535, Portland, Oregon, USA, 2010. ACM Press. ISBN 978-1-4503-0073-5. doi: 10.1145/1830761.1830770. URL http://portal.acm.org/citation.cfm?doid=1830761.1830770.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2), 2018.

Gupta, A. and Kundaje, A. Targeted optimization of regulatory dna sequences with neural editing architectures. *bioRxiv*, pp. 714402, 2019.

Gupta, A. and Zou, J. Feedback GAN (FBGAN) for DNA: A novel feedback-loop architecture for optimizing protein functions. *arXiv preprint arXiv:1804.01694*, 2018.

Killoran, N., Lee, L. J., Delong, A., Duvenaud, D., and Frey, B. J. Generating and designing dna with deep generative models. *arXiv preprint arXiv:1712.06148*, 2017.

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017.

Le, D. D., Shimko, T. C., Aditham, A. K., Keys, A. M., Longwell, S. A., Orenstein, Y., and Fordyce, P. M. Comprehensive, high-resolution binding energy landscapes reveal context dependencies of transcription factor binding. *Proceedings of the National Academy of Sciences*, 115(16):E3702–E3711, 2018.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T.-Y. Neural architecture optimization. In *Advances in neural information processing systems*, pp. 7816–7827, 2018.

Miyazawa, S. and Jernigan, R. L. Residue–residue potentials with a favorable contact pair term and an unfavorable high packing density term, for simulation and threading. *Journal of molecular biology*, 256(3), 1996.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2019.

Roeder, G., Killoran, N., Grathwohl, W., and Duvenaud, D. Design motifs for probabilistic generative design. 2018.