Model Based Quality Diversity Optimization

Lennart Schneider

Master's thesis supervised by

Florian Pfisterer **Martin Binder** **Prof. Dr. Bernd Bischl**

Ludwig-Maximilians-Universität München

Submitted on May 20th, 2021

Department of Statistics

Ludwig-Maximilians-Universität München

Summary

Quality diversity optimization algorithms generate a set of high-performing yet behaviorally diverse solutions. Typically, diversity is defined via pre-specified so-called behavioral niches using so-called feature functions that represent high-level behavioral characteristics of their domain. Classical quality diversity optimization algorithms are based on the principles of evolutionary algorithms. However, Kent and Branke (2020) recently proposed a model based quality diversity algorithm, BOP-Elites, that is embedded within the Bayesian optimization framework. Here, both the objective function and feature functions are treated as black-box functions and each function is modeled using a probabilistic surrogate model. While the seminal work of Kent and Branke (2020) shows promising results, only a very limited type of optimization problem has been considered: Quality diversity optimization of an objective function and a single feature function defined on a one-dimensional continuous domain (which is then actually discretized) assuming pairwise disjoint niches. In this thesis, BOP-Elites is extended to overlapping niches, higher-dimensional domains, multiple feature functions and mixed domains. Moreover, results of a novel application are presented, using BOP-Elites for finding a set of high-performing yet resource-related diverse neural architectures for image classification.

Contents

List of Figures

List of Tables

1 Introduction

Optimization is an important tool in science and practice allowing for efficient use of resources. In classical single-objective optimization, the goal is to find the minimum of an objective function over a feasible set. Mathematically speaking, a (single-objective) optimization problem consists of a vector of variables $\boldsymbol{x} \in \mathcal{X}$ (also called unknowns or parameters), an objective function f of $\boldsymbol{x}$ ($f : \mathcal{X} \to \mathcal{Y}$) which should be minimized or maximized, and potential constraint functions that define certain equalities and inequalities that $\boldsymbol{x}$ must satisfy (Nocedal & Wright, 2006, Chapter 1). Such an optimization problem can then be formulated as:

$$\min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}) \text{ subject to } c_i(\boldsymbol{x}) = 0, i \in \mathcal{E},\ c_j(\boldsymbol{x}) \geq 0, j \in \mathcal{I},$$

where $\mathcal{E}$ and $\mathcal{I}$ are sets of indices for equality and inequality constraints (Nocedal & Wright, 2006, Chapter 1). Oftentimes, optimization problems are continuous, i.e., $\mathcal{X} \subseteq \mathbb{R}^n$, but this must not be the case, e.g., in discrete optimization the domain is given by the set of integers. A solution to the optimization problem is typically denoted as[1]:

$$\boldsymbol{x}^* \coloneqq \arg\min_{\boldsymbol{x} \in \mathcal{X}'} f(\boldsymbol{x}),$$

where $\mathcal{X}' \subseteq \mathcal{X}$ denotes the feasible set, i.e., $\mathcal{X}'$ contains all $\boldsymbol{x}$ that satisfy all constraints.

Typically, optimization problems can be classified via the following properties (Nocedal & Wright, 2006, Chapter 1): **1.** The domain of f (e.g., continuous vs. discrete optimization), **2.** the codomain of f (i.e., single- vs. multi-objective optimization), **3.** the constraints (constrained vs. unconstrained optimization, where the latter refers to the set of equality and inequality constraints being the empty set), **4.** whether a local or global solution should be found (where a local solution refers to a point with an objective value smaller than all other feasible nearby points and a global solution refers to a point with the lowest objective value among all feasible points) **5.** whether f is deterministic or stochastic and **6.** whether the domain of f is a convex set and f is a convex function.

[1] if finding the maximum of f instead of the minimum is desired, $f(\boldsymbol{x})$ is replaced by $-f(\boldsymbol{x})$

In the context of machine learning (ML) a central optimization problem is given by the tuning of hyperparameters (hyperparameter optimization, HPO) of a learner, or so called inducer, I (Hutter, Kotthoff, & Vanschoren, 2018, Chapter 1): Given d hyperparameters, the hyperparameter configuration space is given as $\boldsymbol{\Lambda} = \Lambda_1 \times \ldots \times \Lambda_d$, where Λ_i is the (continuous, discrete or categorical) domain of the i-th hyperparameter. Moreover, I with its hyperparameters instantiated to $\boldsymbol{\lambda}$ is denoted as $I_{\boldsymbol{\lambda}}$. Given a dataset $\mathcal{D}$, the goal is to find:

$$\arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \mathbb{E}_{(D_{\text{train}}, D_{\text{valid}}) \sim \mathcal{D}} \left[\boldsymbol{V}(L, I_{\boldsymbol{\lambda}}, D_{\text{train}}, D_{\text{valid}}) \right], \tag{1.1}$$

where $\boldsymbol{V}(L, I_{\boldsymbol{\lambda}}, D_{\text{train}}, D_{\text{valid}})$ measures the loss of a model generated by the inducer I with hyperparameters $\boldsymbol{\lambda}$ on training data D_{train} and evaluated on validation data D_{valid}, e.g., this can be the cross validation error for a user-given loss function (such as the mean squared error). Note that in practice the expectation above is approximated due to only finite data $D \sim \mathcal{D}$ being accessible. The domain of HPO problems is typically a mixed domain (i.e., hyperparameter sets often contain continuous, discrete and categorical hyperparameters). Regarding the number of objectives, both single- and multi-objective HPO is common (in the case of multiple objectives, one is interested in the trade-off between two or more objectives, e.g., performance and resource usage or multiple loss functions, see, e.g., Horn and Bischl 2016; Igel 2005). HPO problems can either be constrained or unconstrained (often simple box constraint apply), usually a global solution is desired and appropriate optimization algorithms are used and HPO problems are typically non-convex. Finally, another particularity of HPO problems is given by the so-called search space: It is common that not all regions of the domain are of interest and therefore a subspace of the domain is defined via for example box constraints on continuous hyperparameters or collapsing levels of categorical hyperparameters. This subspace then formally constitutes a feasible set which in the context of HPO is referred to as the search space.

Another important property of HPO problems has not been mentioned so far: HPO problems are almost exclusively so-called black-box problems, i.e., the objective is a so-called black-box function that, provided an input returns an output, but the inner workings are not analytically available, which results in the need for gradient free optimization (Audet & Hare, 2017, Chapter 1)[2]. Moreover, the evaluation of the objective function of HPO problems at a given configuration of hyperparameters is typically very costly, e.g., in the case of neural architecture search (NAS, see, e.g., Hutter et al. 2018, Chapter 3) performed by standard training and validation of architectures on data, this

[2]for an example of a non-black-box HPO problem, see Maclaurin, Duvenaud, and Adams (2015)

takes up to several GPU days (where a GPU day is loosely speaking the computation time on a single GPU).

Classical black-box optimization algorithms are given by grid search and random search: In grid search, a finite set of values for each hyperparameter is specified and the Cartesian product of these sets are evaluated, whereas in random search, configurations sampled (uniformly) at random are evaluated (Hutter et al., 2018, Chapter 1). Other popular algorithms are given by so-called population-based methods, such as genetic algorithms (GAs) and evolutionary algorithms (EAs, Hutter et al. 2018, Chapter 1). For example, the well-known CMA-ES algorithm (Hansen, 2016) samples configurations from a multivariate Gaussian distribution with its mean and covariance being updated in each generation based on the performance of the population's individuals. Finally, another popular class of black-box optimization algorithms is given by the Bayesian optimization framework (Hutter et al., 2018, Chapter 1). In Bayesian optimization (historically introduced by Močkus 1975), a probabilistic surrogate model is iteratively fitted to all observations made so far. A (comparably cheap to evaluate) acquisition function then determines the utility of different configurations, i.e., candidates that should be evaluated next, using the predictive distribution of the surrogate model balancing exploration and exploitation. A brief formal introduction to Bayesian optimization is given in Section 2.2.

So far, the goal of solving a (single-objective) optimization problem has been formulated as finding *the* (global) minimum of the objective function. A substantial challenge in the optimization of black-box functions are local minima, i.e., points that have an objective value smaller than all other feasible *nearby* points (for a formal definition, see Nocedal and Wright 2006, Chapter 2). For example, most evolutionary algorithms rely on the heuristic that random changes to good solutions will lead to better solutions but for highly deceptive optimization problems, this may be insufficient in order to find the global minimum because low-performing valleys need to be crossed to find the global optimum, or even just a better local optimum (Floreano & Mattiussi, 2008, Chapter 3.8) and the domain of the objective function will not be explored sufficiently. Many modern evolutionary algorithms therefore encourage diversity via iteratively increasing mutation rates when the performance of intermediate solutions no longer improves (e.g., Clune et al. 2008) or explicitly select points for diversity (e.g., Lehman and Stanley 2011a; Stanley and Miikkulainen 2002). Here, a distinction has to be made between *genetic* diversity, i.e., diversity of the points with respect to their domain values and *behavioral* diversity, i.e., diversity with respect to a function of the domain. For example, consider a population of robots with genotypes described by the length, shape and weight of their

components (which formally define the domain, $\mathcal{X}$), where we are interested in minimizing an objective function $f : \mathcal{X} \to \mathbb{R}_{>0}$ (e.g., speed) depending on the choice of different components. *Genetic* diversity refers to diversity with respect to the length, shape and weight of these components directly whereas *behavioral* diversity refers to a function of these, e.g., different choices of components result in a different size ($g_1 : \mathcal{X} \to \mathbb{R}_{>0}$), weight ($g_2 : \mathcal{X} \to \mathbb{R}_{>0}$) and energy consumption ($g_3 : \mathcal{X} \to \mathbb{R}_{>0}$) of the robot.

This idea of promoting diversity within optimization algorithms has emerged to a sub-field of optimization that has been termed **Quality Diversity** optimization (Cully, Mouret, & Doncieux, 2019). Pioneer work is given by the Novelty Search (Lehman & Stanley, 2011a, 2011b) algorithm, where the goal is no longer to improve performance but simply select points only for diversity in the behavior space (also called feature space). Novelty Search relies on a distance metric and aims at producing as many different behaviors as possible. Together with the idea of maintaining a collection of high-performing individuals (e.g., Cully and Mouret 2016), this led to the development of the first actual (evolutionary) quality diversity optimization algorithm, the MAP-Elites algorithm (Mouret & Clune, 2015). In a nutshell, MAP-Elites finds high-performing yet behaviorally diverse solutions (with respect to pre-specified niches defined via feature functions of interest), by maintaining a set of intermediate solutions for each so-called behavioral niche and generating new solutions via random selection of an elite (which is the best intermediate solution for a niche found so far) that is further varied via mutation and crossover with the other elites. As as example, MAP-Elites will search for the fastest robot that is small, light and energy efficient; the fastest robot that is tall, light and energy efficient; the fastest robot that is tall, heavy and energy efficient, etc. (Mouret & Clune, 2015). So far, quality diversity optimization algorithms have been successfully used to, for example, create repertoires of behaviors of robots that allow for adaption to damage (Cully, Clune, Tarapore, & Mouret, 2015), to design diverse aerodynamic shapes (Gaier, Asteroth, & Mouret, 2018), or to optimize workforce scheduling and routing problems (Urquhart & Hart, 2018). The general idea of why quality diversity optimization should be used instead of, e.g., sequentially solving multiple constrained optimization problems for each niche, lies in solving the set of problems simultaneously is expected to be faster as it is likely that high-performing solutions for neighboring niches will be close and therefore sharing information should be beneficial (Chatzilygeroudis, Cully, Vassiliades, & Mouret, 2020; Mouret & Clune, 2015; Nguyen, Yosinski, & Clune, 2015). Moreover, solving independent constrained optimization problems would be especially wasteful in the context of black-box optimization (Chatzilygeroudis et al., 2020; Kent & Branke, 2020).

As outlined so far, quality diversity optimization algorithms are typically based on the principles of evolutionary algorithms. However, Kent and Branke (2020) recently proposed a model based quality diversity optimization algorithm, BOP-Elites, that is embedded within the Bayesian optimization framework. In BOP-Elites, not only the objective function but also each feature function is modeled using a probabilistic surrogate model (where both the objective function and all feature functions are treated as black-box functions). New candidates are proposed by maximizing the expected joint improvement of elites, selecting points that have a high probability of belonging in niches where the expected improvement over the current elite is high. While the work of Kent and Branke (2020) on model based quality diversity optimization shows promising results, a limitation is that only a single type of optimization problem has been considered: Quality diversity optimization of an objective function and a single feature function defined on a one-dimensional continuous domain (which is then actually discretized) assuming pairwise disjoint niches.

The goal of this thesis is to extend the seminal work of Kent and Branke (2020) on model based quality diversity optimization. In the following Chapter 2, the general quality diversity optimization problem is introduced formally (Section 2.1) and a differentiation with respect to multi-model, multi-task, and multi-objective optimization is given. Following a brief introduction to Bayesian optimization and some typical surrogate models (Section 2.2), model based quality diversity optimization is introduced in the form of the BOP-Elites algorithm (Section 2.3). In Chapter 3, simulation studies are presented investigating the performance of the BOP-Elites algorithm extended to inter alia: overlapping niches, higher-dimensional domains, multiple feature functions and mixed domains. Chapter 4 then presents the results of an application study where BOP-Elites is used to find a set of high-performing yet resource-related diverse neural architectures for image classification. Finally, Chapter 5 concludes with a general discussion and outlook.

2 Theoretical Background

2.1 Quality Diversity Optimization

This formal introduction of quality diversity optimization follows the main outline of the framework introduced in Mouret and Clune (2015) and Kent and Branke (2020). The goal of a quality diversity optimization algorithm is to find a set of high-performing, yet behaviorally diverse solutions. Performance is characterized with respect to an objective function f:

$$\begin{aligned} f : \mathcal{X} &\to \mathcal{Y} \\ \boldsymbol{x} &\mapsto y, \end{aligned}$$

whereas behavior is characterized with respect to feature functions $g_i, i = 1, \ldots, k$:

$$\begin{aligned} g_i : \mathcal{X} &\to \mathcal{Z}_i \\ \boldsymbol{x} &\mapsto z_i. \end{aligned}$$

Typically, $\mathcal{Y} \subseteq \mathbb{R}$ (but principally multi-objective quality diversity optimization could also be done) and $\forall i = 1, \ldots, k : \mathcal{Z}_i \subseteq \mathbb{R}$. Diversity is defined via so-called behavioral niches $N_j \subseteq \mathcal{X}, j = 1, \ldots, c$, which are sets of points characterized via niche-specific boundaries $\boldsymbol{b}_{ij} = \left[b_{\text{lower}_{ij}}, b_{\text{upper}_{ij}}\right) \subseteq \mathcal{Z}_i$ on the image of the feature functions g_i. A point $\boldsymbol{x}$ belongs to niche N_j if its values with respect to the feature functions lie between the respective boundaries, i.e.:

$$\begin{aligned} \boldsymbol{x} \in N_j &\iff \forall i = 1, \ldots, k : g_i(\boldsymbol{x}) \in \boldsymbol{b}_{ij} \\ &\iff \left(b_{\text{lower}_{1j}} \leq g_1(\boldsymbol{x}) < b_{\text{upper}_{1j}}\right) \wedge \ldots \wedge \left(b_{\text{lower}_{kj}} \leq g_k(\boldsymbol{x}) < b_{\text{upper}_{kj}}\right) \end{aligned} \tag{2.1}$$

The goal of a quality diversity optimization algorithm can then be summarized. For each niche N_j find the point that minimizes the objective function f:

$$\boldsymbol{x}_j^* \coloneqq \arg\min_{\boldsymbol{x} \in N_j} f(\boldsymbol{x})$$

That is, obtain a set of solutions $\mathcal{S} \coloneqq \left\{\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_c^*\right\}$ that are diverse with respect to the feature functions, but yet high-performing. Throughout this thesis, it is assumed that $\mathcal{Y} \subseteq \mathbb{R}$ and $\forall i = 1, \ldots, k : \mathcal{Z}_i \subseteq \mathbb{R}$. Without loss of generality, it is assumed that the objective function is to be minimized. Moreover, both the objective function as well as all feature functions are assumed to be deterministic.

Before giving a brief formal introduction to the Bayesian optimization framework, a differentiation of quality diversity optimization to multi-modal, multi-task and multi-objective optimization is provided. Multi-modal optimization algorithms aim to return multiple solutions that correspond to local minima (Chatzilygeroudis et al., 2020). Formally, multi-modal optimization algorithms return a set of solutions:

$$\left\{\boldsymbol{x}_j^* \in \mathcal{X}' : f(\boldsymbol{x}_j^*) < f(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \mathcal{X}', d(\boldsymbol{x}, \boldsymbol{x}_j^*) < \epsilon, \epsilon > 0\right\},$$

where d is a distance function and $\mathcal{X}'$ denotes the feasible set ($\mathcal{X}' \subseteq \mathcal{X}$). Compared to quality diversity optimization, there are no feature functions that define behavioral niches but multi-modal optimization will result in genetic diversity, i.e., diversity of the points with respect to their domain values. Additionally, quality diversity optimization algorithms may return far more solutions than local minima exist.

Multi-task optimization algorithms are typically defined for objective functions that are parameterized by a task descriptor (Chatzilygeroudis et al., 2020):

$$\begin{aligned} f : \mathcal{X} \times \mathcal{T} &\to \mathcal{Y} \\ (\boldsymbol{x}, \boldsymbol{\tau})^T &\mapsto y. \end{aligned}$$

Here, task descriptors could be, e.g., the morphology of a robot. Multi-task optimization algorithms then aim to return a set of solutions where each solution is given by

$$\boldsymbol{x}_{\boldsymbol{\tau}}^* \coloneqq \arg\min_{\boldsymbol{x} \in \mathcal{X}'} f\left((\boldsymbol{x}, \boldsymbol{\tau})^T\right).$$

Recently, MAP-Elites has been extended to multi-task optimization problems where the task $\boldsymbol{\tau}$ is selected in the neighborhood of the parents using the standard MAP-Elites

algorithm resulting in a more global search by incorporating all tasks simultaneously (Mouret & Maguire, 2020).

Multi-objective optimization involves multiple objective functions (for a brief introduction, see, e.g., Konak, Coit, and Smith 2006):

$$f_i : \mathcal{X} \to \mathcal{Y}_i, i = 1, \dots k, k \geq 2$$

where the vector of objective functions is denoted as $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \dots, f_1(\boldsymbol{x}))^T$. The goal of multi-objective optimization algorithms states as:

$$\min_{\boldsymbol{x} \in \mathcal{X}'} \boldsymbol{f}(\boldsymbol{x}),$$

where $\mathcal{X}'$ denotes the feasible set ($\mathcal{X}' \subseteq \mathcal{X}$). Note that all objective functions should be minimized simultaneously. However, as typically no single solution minimizing all objective functions simultaneously exists, attention is given to so-called Pareto optimal solutions that cannot be improved in any objective without degrading at least one other objective. Mathematically, $\boldsymbol{x}_1 \in \mathcal{X}'$ is said to Pareto dominate another $\boldsymbol{x}_2 \in \mathcal{X}'$ if:

$$\forall i = 1, \dots, k : f_i(\boldsymbol{x}_1) \leq f_i(\boldsymbol{x}_2) \wedge \exists j \in 1, \dots, k : f_j(\boldsymbol{x}_1) < f_j(\boldsymbol{x}_2).$$

A solution is then called Pareto optimal if there exists no other solution that dominates it and the set of Pareto optimal solutions is called the Pareto front. Intuitively, multi-objective optimization has a completely different goal in mind compared to quality diversity optimization, i.e., there are multiple objective functions but neither feature functions nor niches. While typically a set of Pareto optimal solutions is returned, these solutions must not necessarily be (behaviorally) diverse but simply reflect different trade-offs with respect to the objectives. However, by introducing a binary objective function for each niche N_j

$$\begin{aligned} f_{N_j} : \mathcal{X} &\to \{0, 1\} \\ \boldsymbol{x} &\mapsto \begin{cases} 0 \text{ if } \boldsymbol{x} \in N_j, \\ 1 \text{ else,} \end{cases} \end{aligned}$$

quality diversity optimization problems could potentially be framed as multi-objective optimization problems (although this approach should scale relatively poorly in the number of niches and it has not been considered in the literature so far). In Table

Optimization	Objective Output	Diversity	Particularity
Multi-Modal	scalar	genetic	
Multi-Task	scalar	via task descriptors	task descriptors
Multi-Objective	vector valued	trade-off in objectives	
Quality Diversity	(so far) scalar	behavioral	feature functions & niches

Table 2.1. Differences and similarities of multi-modal, multi-task, multi-objective and quality diversity optimization.

2.1, differences and similarities of multi-modal, multi-task, multi-objective and quality diversity optimization is summarized.

2.2 Bayesian Optimization

Bayesian optimization is a powerful tool for the optimization of black-box functions that has gained great popularity in the past years. The Bayesian optimization framework has two main ingredients (see, e.g., Hutter et al. 2018, Chapter 1 or Shahriari, Swersky, Wang, Adams, and de Freitas 2016): First, a probabilistic surrogate model that captures beliefs about the behavior of the unknown objective function via a prior distribution while the posterior distribution represents the updates beliefs. More formally, a surrogate model $\mathcal{M}$ provides a probabilistic interpretation of the function it models, where possible explanations for the function are seen as draws $f^l \sim \mathcal{P}(f|\mathcal{D})$[1]. Typically, $\mathcal{M}$ dictates the parameters $\boldsymbol{\theta}$ of a distribution over the function's behavior at any point $\boldsymbol{x}$. By optimizing the models' (hyper)parameters $\boldsymbol{\xi}$, a belief is than formed as $\mathcal{P}(y|\boldsymbol{x}, \mathcal{D}) = \mathcal{P}(y; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are specified by the surrogate model $\mathcal{M}$ evaluated at $\boldsymbol{x}$, i.e., $\mathcal{M}(\boldsymbol{x}; \boldsymbol{\xi}) = \boldsymbol{\theta}$. Note that the formulation above is seen in function space view. Secondly, a so-called acquisition function that leverages the uncertainty in the posterior proposing new candidate points for evaluating - ideally finding a good trade-off between exploration and exploitation. Compared to the evaluation of the expensive black-box function, acquisition functions are generally very cheap to compute and can be optimized thoroughly. Seeing Bayesian optimization fully modular, one can also identify a third ingredient: The optimizer that optimizes the acquisition function (see, e.g., Wilson, Hutter, and Deisenroth 2018).

A popular choice for the surrogate model is given by Gaussian processes (Rasmussen & Williams, 2006). A Gaussian process $\mathcal{G}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$ is a collection of random vari-

[1] where $\mathcal{D}$ is the data used to fit the surrogate model

ables, any finite number of which have consistent Gaussian distributions. A Gaussian process is completely specified via its mean $m(\boldsymbol{x})$ (in the context of Bayesian optimization typically assumed to be constant) and covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$. In the case of a deterministic objective function, the mean $\mu(\cdot)$ and variance predictions $\sigma^2(\cdot)$ are calculated as:

$$\mu(\boldsymbol{x}) = \mathbf{k}_*^T \mathbf{K}^{-1} \boldsymbol{y}$$
$$\sigma^2(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{x}) - \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{k}_*,$$

where $\mathbf{k}_*$ is the vector of covariances between $\boldsymbol{x}$ and all previous observations, $\mathbf{K}$ is the covariance matrix of all previous observations and $\boldsymbol{y}$ are the observed objective function values. Following the surrogate model notation established earlier, one can denote the distribution of the surrogate model prediction y for point $\boldsymbol{x}$ as $\mathcal{P}(y; \boldsymbol{\theta}) = \mathcal{N}\left(\mu(\boldsymbol{x}), \sigma^2(\boldsymbol{x})\right)$, where $\left(\mu(\boldsymbol{x}), \sigma^2(\boldsymbol{x})\right)^T$ constitute $\boldsymbol{\theta}$. Classical choices for the covariance function are the Gaussian (also known as squared exponential or radial basis function kernel) or Mátern 5/2 kernel (Rasmussen & Williams, 2006, Chapter 4).

As a surrogate model, Gaussian processes have nice properties such as well-calibrated uncertainty estimates and closed-form computability of the predictive distribution. However, downsides, such as poor scalability to high dimensions and cubical scaling in the number of data points, resulted in other machine learning models being used as surrogate models (Hutter et al., 2018, Chapter 1), however, note that not all models fulfill the function space view of a surrogate model as described above, but simply return a mean and variance prediction. For example, deep neural networks are very flexible and scalable models. A standard feed-forward neural network can be used as a surrogate model either by using a Bayesian neural network (Springenberg, Klein, Falkner, & Hutter, 2016) where a posterior distribution is inferred over network weights or by using an ensemble of neural networks with different random weight initializations and training set orders (White, Neiswanger, & Savani, 2019). As another example, random forests (Breiman, 2001) have gained popularity as a surrogate model (Hutter, Hoos, & Leyton-Brown, 2011) due to their natural ability to handle mixed domains with dependencies, e.g., in the case of HPO and categorical hyperparameters, Gaussian processes either require preprocessing of the input (Jenatton, Archambeau, González, & Seeger, 2017) or transformations of the covariance function (Garrido-Merchán & Hernández-Lobato, 2020).

A popular acquisition function is given by the expected improvement (EI) (Jones,

Schonlau, & Welch, 1998):

$$\alpha_{\text{EI}}(\boldsymbol{x}) \coloneqq \mathbb{E}_y[\mathbb{I}(\boldsymbol{x})] = \mathbb{E}_y\left[\max\left(f_{\min} - y, 0\right)\right].$$

Here, $f_{\min}$ is the best observed value so far and y is the surrogate model prediction for point $\boldsymbol{x}$. If the posterior distribution of the surrogate model follows a normal distribution, the EI can be calculated in closed form:

$$\mathbb{E}_y[\mathbb{I}(\boldsymbol{x})] = \left(f_{\min} - \mu(\boldsymbol{x})\right)\Phi\left(\frac{f_{\min} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right) + \sigma(\boldsymbol{x})\phi\left(\frac{f_{\min} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right).$$

Here, $\mu(\boldsymbol{x})$ and $\sigma(\boldsymbol{x})$ are the mean and standard deviation prediction of the surrogate model, and $\phi(\cdot)$ and $\Phi(\cdot)$ are the standard normal density and distribution function.

The acquisition function optimizer solves the inner optimization problem in each iteration given an acquisition function $\alpha(\cdot)$:

$$\boldsymbol{x}^\star \coloneqq \arg\max_{\boldsymbol{x}\in\mathcal{X}} \alpha(\boldsymbol{x}).$$

Effectively, the acquisition function optimizer proposes the point with the largest acquisition value for evaluation (in the case of sequential Bayesian optimization and single-point proposal which will be followed here). Popular algorithms for optimizing the acquisition function are given by the Nelder-Mead (Nelder & Mead, 1965) or L-BFGS-B (Byrd, Lu, Nocedal, & Zhu, 1995) algorithm (in the case of a continuous domain and for L-BFGS-B additionally gradients being available) or a simple random search in the case of a mixed domain.

In summary, seeing Bayesian optimization as a fully modular framework one can identify the following components: **1.** A probabilistic surrogate model that models the objective function providing a mean and variance prediction, **2.** an acquisition function on which basis new points for evaluation are proposed and **3.** an optimizer to optimize the acquisition function, effectively proposing the new points.

2.3 BOP-Elites

In the seminal work of Kent and Branke (2020), quality diversity optimization has been united with Bayesian optimization in an algorithm that they termed Bayesian Optimization of Elites (BOP-Elites). Kent and Branke (2020) consider a setting where besides an objective function to be optimized, every solution is characterized by categorizing their

feature values into niches (as introduced in Section 2.1). If a solution's feature values are computationally expensive to compute or linked to the objective function evaluation, it is sensible to treat the feature functions as black-box functions themselves. Uniting this idea with the Bayesian optimization framework results in not only one surrogate model being used to model the objective function, but multiple surrogate models being used additionally modeling the feature functions. In principle, one could aim at jointly modeling all feature values using a single multi-output surrogate model. In practice, using multiple surrogate models and treating the feature functions independently is more reasonable, as for example standard GPs are not capable of multi-output modeling. In the BOP-Elites algorithm, new points for evaluation should be proposed by considering joint information, i.e, the predicted objective function value but also the predicted feature function value and the resulting probability of new points falling into the predefined niches. It should be noted that Kent and Branke (2020) explicitly introduce the algorithm for $\mathcal{X} \subseteq \mathbb{R}$, $k = 1$ (one-dimensional continuous domain of the objective function f and single feature function g_1) and pairwise disjoint niches. In the following, these limitations are relaxed, i.e., $\mathcal{X}$ can be a high-dimensional mixed space, there can be more than a single feature function and niches must not necessarily be pairwise disjoint, i.e., it must not hold that $\forall j, j' \in 1, \ldots, c : j \neq j' \implies N_j \cap N_{j'} = \emptyset$. As acquisition function, Kent and Branke (2020) introduce the expected joint improvement of elites (EJIE) that measures the expected improvement to the ensemble problem of identifying the best solution in every niche:

$$\alpha_{\text{EJIE}}(\boldsymbol{x}) \coloneqq \mathbb{E}_{\boldsymbol{z}}\left[\mathbb{E}_y\left[\mathbb{I}_{\boldsymbol{z}}(\boldsymbol{x})\right]\right] = \sum_{j=1}^{c} \mathcal{P}(\boldsymbol{x} \in N_j|\mathcal{D})\mathbb{E}_y\left[\mathbb{I}_{\boldsymbol{z}}(\boldsymbol{x})\right]. \tag{2.2}$$

Here, the outer expectation is over $\boldsymbol{z} = (z_1, \ldots, z_k)^T$ denoting the vector of surrogate model predictions for the feature functions, $\mathcal{P}(\boldsymbol{x} \in N_j|\mathcal{D})$ is the (posterior) probability of $\boldsymbol{x}$ falling into niche N_j, and $\mathbb{E}_y\left[\mathbb{I}_{\boldsymbol{z}}(\boldsymbol{x})\right]$ is the expected improvement with respect to niche N_j derived on the basis of $\boldsymbol{z}$ (Equation 2.1):

$$\mathbb{E}_y\left[\mathbb{I}_{\boldsymbol{z}}(\boldsymbol{x})\right] = \mathbb{E}_y\left[\max\left(f_{\min_{\mathrm{N_j}}} - y, 0\right)\right],$$

where $f_{\min_{N_j}}$ is the best observed value in niche N_j so far and y is the surrogate model prediction for point $\boldsymbol{x}$.

The probability of a point $\boldsymbol{x}$ belonging to niche N_j generally (see Equation 2.1) states

as:

$$\mathcal{P}(\boldsymbol{x} \in N_j|\mathcal{D}) = \mathcal{P}(\boldsymbol{z} \leq \boldsymbol{b}_{\text{upper}_{.j}}) - \mathcal{P}(\boldsymbol{z} \leq \boldsymbol{b}_{\text{lower}_{.j}})$$

where $\boldsymbol{z} = (z_1, \ldots, z_k)^T$ again denotes the vector of surrogate model predictions for point $\boldsymbol{x}$ and $\boldsymbol{b}_{\text{upper}_{.j}} = (b_{\text{upper}_{1j}}, \ldots, b_{\text{upper}_{kj}})^T$ and $\boldsymbol{b}_{\text{lower}_{.j}}$ is defined analogously.

As already outlined above, in practice it is sensible to use multiple surrogates to model each feature function independently. Furthermore, assuming the posterior distribution of the surrogates modeling the feature functions each to follow a normal distribution with a mean prediction and standard deviation prediction of $\nu_i(\boldsymbol{x})$ and $\tau_i(\boldsymbol{x})$, $\mathcal{P}(\boldsymbol{x} \in N_j|\mathcal{D})$ then simplifies to:

$$\mathcal{P}(\boldsymbol{x} \in N_j|\mathcal{D}) = \prod_{i=1}^{k} \left[\Phi\left(\frac{\nu_i(\boldsymbol{x}) - b_{\text{upper}_{ij}}}{\tau_i(\boldsymbol{x})} \right) - \Phi\left(\frac{\nu_i(\boldsymbol{x}) - b_{\text{lower}_{ij}}}{\tau_i(\boldsymbol{x})} \right) \right],$$

where $\Phi(\cdot)$ again denotes the standard normal distribution function.

The full BOP-Elites algorithm can be summarized in pseudocode as in Algorithm 1.

Algorithm 1: BOP-Elites

Result: $\mathcal{S} = \left\{\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_c^*\right\}$ (set of solutions)
$\mathcal{X}, f, g_i, i = 1, \ldots, k, N_j$ with $\boldsymbol{b}_{ij}, j = 1, \ldots, c, \mathcal{D}_{\text{design}}, N_{\text{total}}$
$\mathcal{D} \leftarrow \mathcal{D}_{\text{design}}$
for $n \leftarrow 1$ **to** N_{total} **do**
  Fit surrogate models to current data $\mathcal{D}$
  $\boldsymbol{x}^\star \leftarrow \arg\max_{\boldsymbol{x} \in \mathcal{X}} \alpha_{\text{EJIE}}(\boldsymbol{x})$
  Evaluate $y \leftarrow f(\boldsymbol{x}^\star), \forall i = 1, \ldots, k : z_i \leftarrow g(\boldsymbol{x}^\star)$
  if $\boldsymbol{x}^\star \in N_j \wedge y < f(\boldsymbol{x}_j^*)$ **then**
    $\boldsymbol{x}_j^* \leftarrow \boldsymbol{x}^\star$
  end
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\boldsymbol{x}^\star, y, z_1, \ldots, z_k)\}$
end

Kent and Branke (2020) then present the results of a small-scale simulation study, investigating the performance of the BOP-Elites algorithm in the scenario of the true objective function and feature function each being the mean prediction of a GP with a Gaussian kernel fitted to some initial data $\mathcal{D}_f = \left\{\left(x^{(i)}, y^{(i)} \sim U(0, 20)\right)\right\}$, $\mathcal{D}_{g_1} = \left\{\left(x^{(i)}, z_1^{(i)} \sim U(0, 20)\right)\right\}$. They compare their "ensemble" BOP-Elites algorithm to two other variants that conduct separate searches in each niche further varying whether joint surrogate models are fitted: In the "sequential" BOP-Elites algorithm, niches are

optimized in a round-robin fashion for a fixed number of iterations and for each niche N_j, the expected expected improvement with respect to niche N_j weighted with the probability of $\boldsymbol{x}$ falling into Niche N_j is considered as the acquisition function:

$$\alpha_{\mathrm{EIN}_j}(\boldsymbol{x}) \coloneqq \mathcal{P}(\boldsymbol{x} \in N_j | \mathcal{D}) \mathbb{E}_y \left[\mathbb{I}_{\boldsymbol{z}}(\boldsymbol{x}) \right]. \tag{2.3}$$

Still, the surrogates modeling the objective and feature function are trained relying on all data observed so far over all runs. Contrary to this, the "independent" BOP-Elites algorithm only trains the surrogate models on the data observed so far for each separate niche run (and uses independent surrogate models for every niche). Comparing the "ensemble" BOP-Elites algorithm to the "sequential"‘ and "independent" BOP-Elites algorithms allows for the investigation of the following questions: **1.** How much does the BOP-Elites algorithm profit from a simultaneous search in all niches ("ensemble" vs. "sequential")? **2.** How much does the BOP-Elites algorithm profit from building joint surrogate models ("sequential" vs. "independent").

The following section presents results of several small-scale simulation studies. Simulation 1 is a conceptual replication of the findings of Kent and Branke (2020). In Simulation 2.1, the limitation of pairwise disjoint niches is relaxed and in 2.2, the dimensionality of the domain of f and g_1 is increased, whereas in Simulation 2.3, the limitation of using only a single feature function g_1 is relaxed and in Simulation 2.4, a mixed domain of f and g_1 is considered. Finally, in Simulation 3 the performance of different acquisition function optimizers is examined.

3 Simulation Studies

The following simulations studies replicate (Simulation 1) and extend the findings of Kent and Branke (2020) regarding the BOP-Elites algorithm in the following ways: **1.** The limitation of pairwise disjoint niches is relaxed (Simulation 2.1), **2.** the dimensionality of the domain of the objective function and feature function is increased (Simulation 2.2), **3.** the number of feature functions is increased (Simulation 2.3), **4.** a mixed domain of the objective function and feature function is considered (Simulation 2.4) and **5.** the performance of different acquisition function optimizers is examined (Simulation 3).

In all simulations, the objective function f is to be minimized. As an evaluation metric, the total error is used:

$$\mathrm{TE}(\boldsymbol{x}_1^*, \dots \boldsymbol{x}_c^*) \coloneqq -\sum_{j=1}^{c} \big(f(\boldsymbol{x}_j^*) - f(\hat{\boldsymbol{x}}_j)\big),$$

where $f(\boldsymbol{x}_j^*)$ is the "true" function minimum for niche N_j obtained by exhaustive search over the domain performed using an equidistant grid of dimension $\lfloor 10^{6\frac{1}{d_{\mathrm{cont}}}} \rfloor d_{\mathrm{cont}}$, where d_{cont} is the dimensionality of the domain (in the case of a mixed domain, d_{cont} is the dimensionality with respect to the continuous part and the grid is expanded by crossing with all other possible values for integer, categorical and boolean subdomains) and $f(\hat{\boldsymbol{x}}_j)$ is the best solution for niche N_j found (if no solution has been found for a niche, this value is set to $f_{\max}$ obtained via the same exhaustive search procedure as described above). No measure regarding the diversity of the solutions or the coverage of the feature space is used because in the case of only few discrete niches, this is not very informative (Chatzilygeroudis et al., 2020) in the sense that all algorithms typically find a solution for every niche. The "ensemble" BOP-Elites algorithm will be abbreviated as `EJIE` (Equation 2.2), the "sequential" one as `EIN` (Equation 2.3) and the "independent" one as `IND` (Equation 2.3 but independent surrogate models for each niche). The abbreviation `RS` refers to a simple random search serving as a baseline (if not stated otherwise, 1000 points are drawn uniformly at random, evaluated and the niches the points belong to are determined post hoc). For additional computational details, please see Section 6.2.

3.1 Simulation 1: A Conceptual Replication of Kent & Branke (2020)

This simulation serves as a conceptual replication of the simulation of Kent and Branke (2020). The domain of f: $x \mapsto y$ and g_1: $x \mapsto z_1$ is given by $[0, 10]$. $f(\cdot)$ and $g_1(\cdot)$ are given by the mean prediction of GPs with Gaussian kernels fitted to initial data $\mathcal{D}_f = \left\{\left(x^{(i)}, y^{(i)} \sim U(0, 20)\right)\right\}_{i=1}^{11}$ and $\mathcal{D}_{g_1} = \left\{\left(x^{(i)}, z_1^{(i)} \sim U(0, 20)\right)\right\}_{i=1}^{11}$, with $(x^{(1)}, \ldots, x^{(11)})^T = (0, \ldots, 10)^T$. Five niches are defined via the following boundaries on $\mathcal{Z}_1$: $\boldsymbol{b}_{11} = [0, 4), \boldsymbol{b}_{12} = [4, 8), \boldsymbol{b}_{13} = [8, 12), \boldsymbol{b}_{14} = [12, 16), \boldsymbol{b}_{15} = [16, 20)$. An example problem is visualized in Figure 3.1 showing the objective function value y and feature function value z_1 depending on x. In the left plot, the abscissa is colored with respect to the niche x belongs to (colored in the same colors) which is derived based on the feature function values z_1 on the right. The goal is to find the minimum objective function value for each niche (visualized by the colored points).

For both GPs, length-scales are constrained to $[0.5, 2]$ (isotropic) and hyperparameters are optimized via the L-BFGS-B algorithm (Byrd et al., 1995) using ten randomly initialized starting values in a multistart setting and selecting the hyperparameter solutions of the best run. As surrogate models for $f(\cdot)$ and $g_1(\cdot)$, GPs with Gaussian kernels are used with the same technical specifications as the true functions. As an optimizer for the acquisition function a Nelder-Mead simplex algorithm (Nelder & Mead, 1965) with support for box constraints (Box, 1965) is used, allowing for up to 100 function evaluations (terminating earlier, if the relative change in the optimization parameters is less than $1e-6$ with respect to the L_1 norm). 5 points are sampled uniformly at random and used as the initial design points and all algorithms are run for a total of 35 iterations. In total, 100 simulation runs are performed (note that each run constitutes a slightly different optimization problem due to the initial data that is used to fit the true functions being sampled). Results are given in Figure 3.2. The "ensemble" BOP-Elites (`EJIE`) algorithm strongly outperforms its competitors reaching a mean total error of around 1 after the 20th iteration. The "sequential" BOP-Elites (`EIN`) algorithm outperforms the "independent" (`IND`) one indicating that building joint surrogate models results in a performance boost.

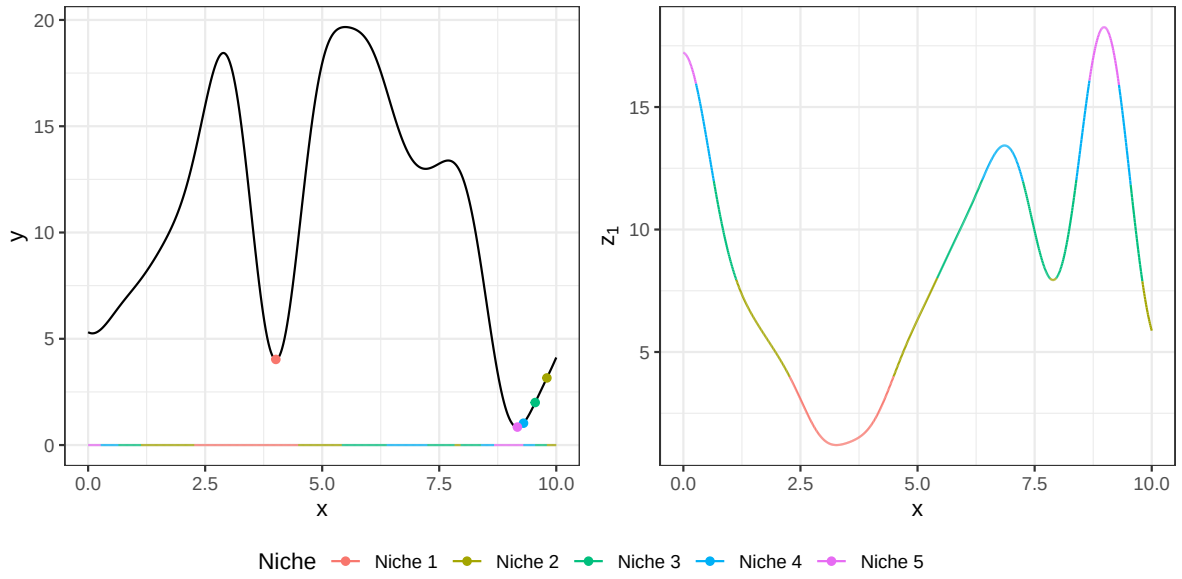


Figure 3.1. Simulation 1: Example problem.

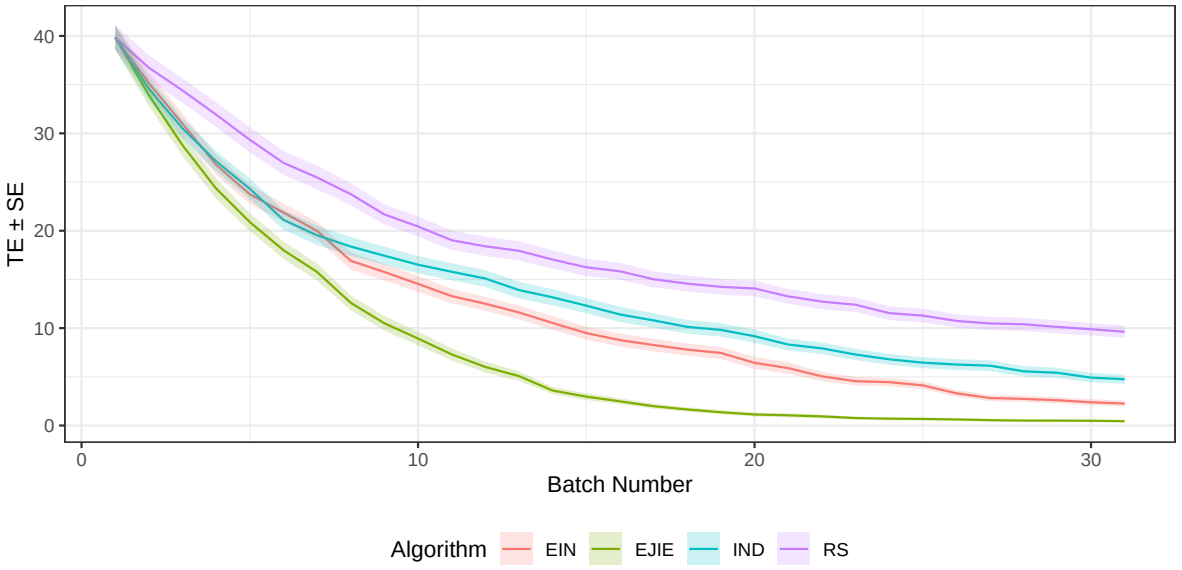


Figure 3.2. Simulation 1: Mean total error. Ribbons represent standard errors. 100 simulation runs.

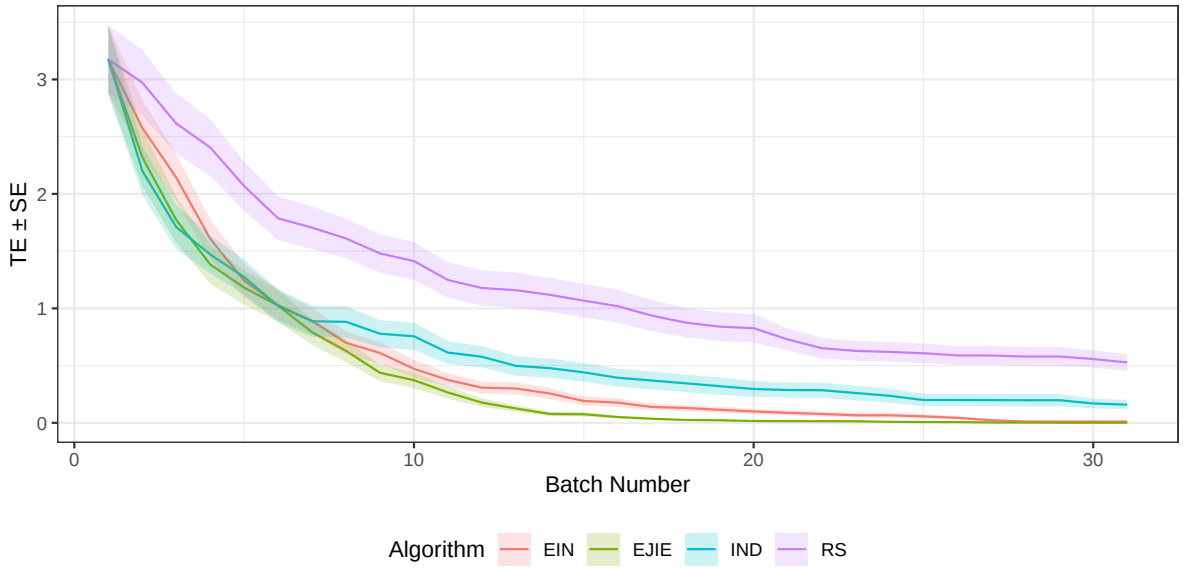


Figure 3.3. Simulation 2.1: Overlapping niches. Mean total error. Ribbons represent standard errors. 100 simulation runs.

3.2 Simulation 2: Extending Kent & Branke (2020)

All simulations presented in this subsection relax or extend some assumptions made by Kent and Branke (2020), i.e., they consider overlapping niches, higher-dimensional domains, multiple feature functions, and mixed domains.

3.2.1 Simulation 2.1: Overlapping Niches

The domain of $f\colon x \mapsto y$ and $g_1\colon x \mapsto z_1$ is given by $[0, 10]$. $f(\cdot)$ and $g_1(\cdot)$ are given as in Simulation 1. Five niches are defined via the following boundaries on $\mathcal{Z}_1$: $\boldsymbol{b}_{11} = [0, 4), \boldsymbol{b}_{12} = [0, 8), \boldsymbol{b}_{13} = [0, 12), \boldsymbol{b}_{14} = [0, 16), \boldsymbol{b}_{15} = [0, 20)$, i.e., they are nested in each other. Surrogates are defined as in Simulation 1 and and all other technical details are the same as in Simulation 1. 5 points are sampled uniformly at random and used as the initial design points and all algorithms are run for a total of 35 iterations. Results are given in Figure 3.3 based on 100 simulation runs. Again, the "ensemble" BOP-Elites (`EJIE`) algorithm outperforms its competitors reaching a total error of close to zero at around the 13th iteration.

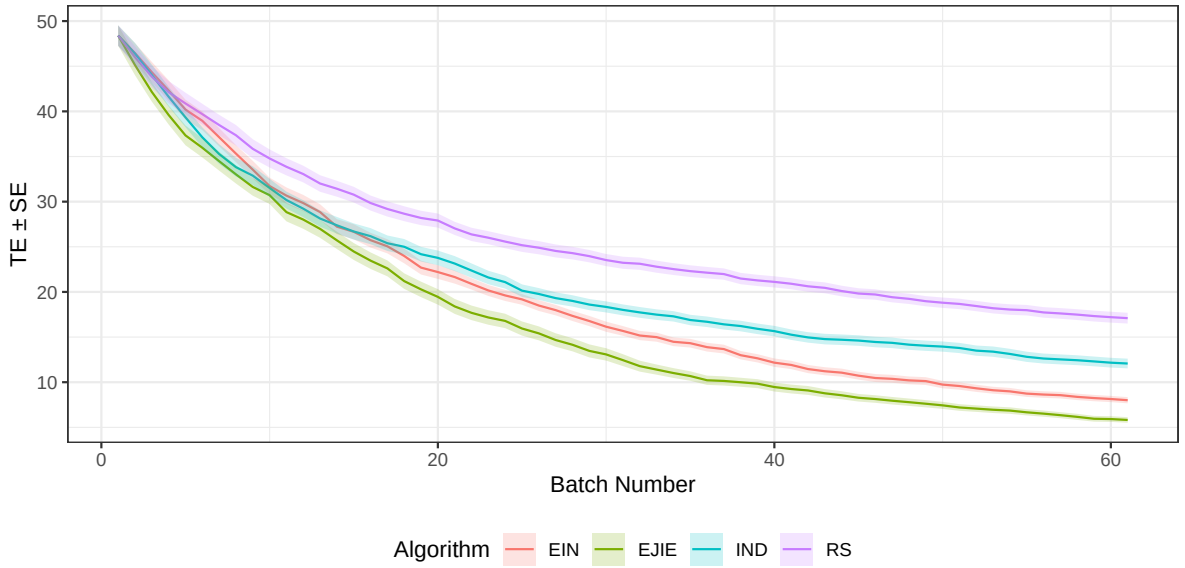


Figure 3.4. Simulation 2.2: Two-dimensional search space. Mean total error. Ribbons represent standard errors. 100 simulation runs.

3.2.2 Simulation 2.2: 2-Dimensional Domain

The domain of f: $\boldsymbol{x} \mapsto y$ and g_1: $\boldsymbol{x} \mapsto z_1$ is given by $[0, 10] \times [0, 10]$. $f(\cdot)$ and $g_1(\cdot)$ are given by the mean prediction of GPs with Gaussian kernels fitted to initial data $\mathcal{D}_f = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)} \sim U(0, 20)\right)\right\}_{i=1}^{121}$, $\mathcal{D}_{g_1} = \left\{\left(\boldsymbol{x}^{(i)}, z^{(i)} \sim U(0, 20)\right)\right\}_{i=1}^{121}$ with $\boldsymbol{x}^{(i)} = \boldsymbol{X}_{i.}$, where $\boldsymbol{X}$ is a matrix of dimension 121×2 where rows are constituted by the Cartesian product $\{0, \ldots, 10\} \times \{0, \ldots, 10\}$, e.g., $\boldsymbol{X}_{1.} = (0, 0)$. Niches and surrogate models are defined as in Simulation 1 (that is, niches are again pairwise disjoint) and all other technical details are the same as in Simulation 1. 10 points are sampled uniformly at random and used as the initial design points and all algorithms are run for a total of 70 iterations. Results are given in Figure 3.4 based on 100 simulation runs. The "ensemble" BOP-Elites (`EJIE`) algorithm again outperforms it competitors although convergence to a lower total error would naturally require more iterations.

3.2.3 Simulation 2.3: Two Feature Functions

The domain of f: $x \mapsto y$, g_1: $x \mapsto z_1$ and g_2: $x \mapsto z_2$ is given by $[0, 10]$. $f(\cdot)$, $g_1(\cdot)$ and $g_2(\cdot)$ are given by the mean prediction of GPs fitted to initial data $\mathcal{D}_f = \left\{\left(x^{(i)}, y^{(i)} \sim U(0, 20)\right)\right\}_{i=1}^{11}$, $\mathcal{D}_{g_1} = \left\{\left(x^{(i)}, z_1^{(i)} \sim U(0, 20)\right)\right\}_{i=1}^{11}$ and $\mathcal{D}_{g_2} = \left\{\left(x^{(i)}, z_2^{(i)} \sim U(0, 20)\right)\right\}_{i=1}^{11}$, with $(x^{(1)}, \ldots, x^{(11)})^T = (0, \ldots, 10)^T$. Four niches are de-

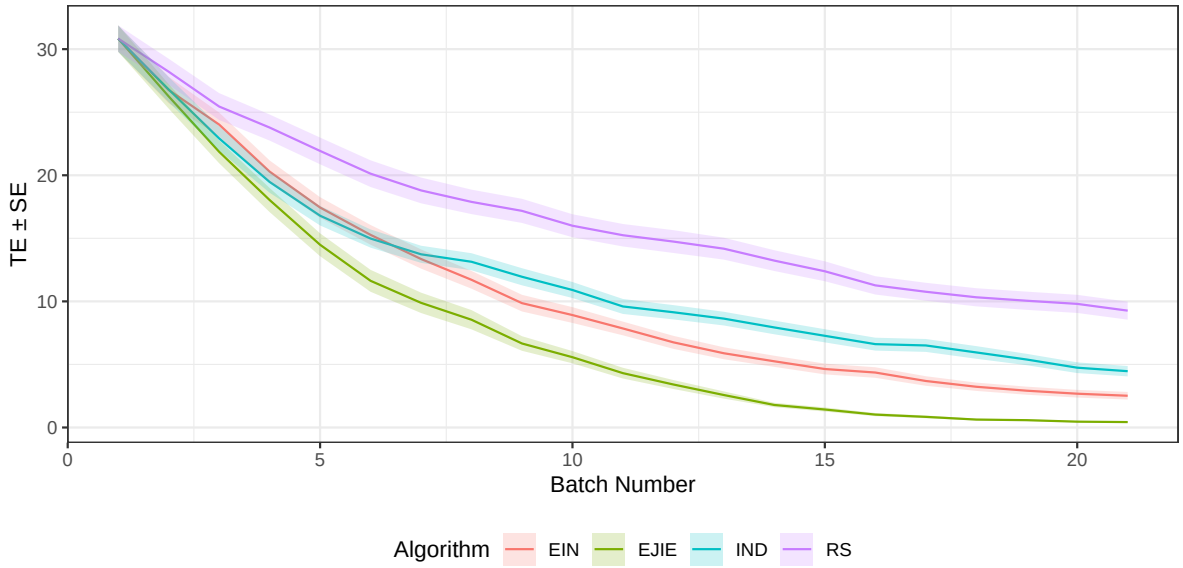


Figure 3.5. Simulation 2.3: Two feature functions. Mean total error. Ribbons represent standard errors. 100 simulation runs.

fined via the following boundaries on $\mathcal{Z}_1$ and respectively $\mathcal{Z}_2$: $\boldsymbol{b}_{11} = [0, 10), \boldsymbol{b}_{21} = [0, 10), \boldsymbol{b}_{12} = [0, 10), \boldsymbol{b}_{22} = [10, 20), \boldsymbol{b}_{13} = [10, 20), \boldsymbol{b}_{23} = [0, 10], \boldsymbol{b}_{14} = [10, 20), \boldsymbol{b}_{24} = [10, 20)$[1]. Surrogates are defined as in Simulation 1 (where a similar GP is used to model $g_2(\cdot)$), and all other technical details are the same as in Simulation 1. 5 points are sampled uniformly at random and used as the initial design points and all algorithms are run for a total of 25 iterations. Results are given in Figure 3.5 based on 100 simulation runs. Compared to the results of Simulation 1 (Figure 3.2), convergence is slightly slower, although the "ensemble" BOP-Elites (`EJIE`) manages to yield a total error lower than 1 at around the 16th iteration and outperforms its competitors.

3.2.4 Simulation 2.4: Mixed Domain

The domain of $f\colon \boldsymbol{x} \mapsto y$ and $g_1\colon \boldsymbol{x} \mapsto z_1$ is given by $[0, 10] \times \{\text{"a"}, \text{"b"}, \text{"c"}\} \times \{\text{FALSE}, \text{TRUE}\}$. $f(\cdot)$ is given by the mean prediction of a regression tree (Breiman, Friedman, Stone, & Olshen, 1984) fitted to initial data $\mathcal{D}_f = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{66}$ with $\boldsymbol{x}^{(i)} = \boldsymbol{X}_{i.}$, where $\boldsymbol{X}$ is a matrix of dimension 66×3 where rows are constituted by the Cartesian product $\{0, \ldots, 10\} \times \{\text{"a"}, \text{"b"}, \text{"c"}\} \times \{\text{FALSE}, \text{TRUE}\}$, e.g., $\boldsymbol{X}_{1.} = (0, \text{"a"}, \text{FALSE})$.

[1] this e.g., results in the first niche being defined as $\{x \in [0, 10] : 0 \leq g_1(x) < 10 \ \wedge \ 0 \leq g_2(x) < 10\}$

Here, $y^{(i)}$ is constructed as follows:

$$y^{(i)} \sim \begin{cases} U(0,20) \ \ + 10 \cdot \mathbb{1}_{\text{TRUE}}(x_3) \text{ if } x_2 = \text{“a”} \\ U(10,30) + 10 \cdot \mathbb{1}_{\text{TRUE}}(x_3) \text{ if } x_2 = \text{“b”} \\ U(20,40) + 10 \cdot \mathbb{1}_{\text{TRUE}}(x_3) \text{ if } x_2 = \text{“c”}. \end{cases}$$

$g_1(\cdot)$ is given by the mean prediction of a GP with a Gaussian kernel fitted to initial data $\mathcal{D}_{g_1} = \left\{\left(x_1^{(i)}, z^{(i)} \sim U(0,20)\right)\right\}_{i=1}^{11}$, i.e., only x_1 is relevant for g_1, where $(x_1^{(1)}, \ldots, x_1^{(11)})^T = (0, \ldots, 10)^T$. Niches are defined as in Simulation 1. As a surrogate model for $f(\cdot)$ and $g_1(\cdot)$ either a random forest or a GP with preprocessing (converting Booleans to integers and one-hot encoding of categorical variables) is used. All other technical details are the same as in Simulation 1. 15 points are sampled uniformly at random and used as the initial design points and all algorithms are run for a total of 75 iterations. Results are given in Figure 3.6 based on 100 simulation runs. Comparing the scenario of using GP (`GP`) surrogate models to using random forests (`RF`) shows that the GP surrogate models strongly outperform the random forests - at least with respect to the “ensemble” BOP-Elites (`EJIE`) algorithm. This is somewhat surprising, as the preprocessing of the GP (one-hot encoding of categorical variables) essentially results in a five-dimensional space being modeled (compared to the original three-dimensional space). Looking at the GP surrogate models and the “sequential” (`EIN`) and “independent” (`IND`) BOP-Elites algorithms, a clear step pattern is visible that emerges due to the niches being optimized sequentially. This pattern is not visible when looking at the random forest surrogate models, indicating that (naturally) a random forest cannot model a mean prediction of a GP as a true function (for the feature function) as well as a GP itself. In hindsight, this shows that the simulation design is somewhat biased against the random forest surrogate models. Nevertheless, the “ensemble” (`EJIE`) BOP-Elites algorithm using random forests surrogate models still outperforms random search (`RS`), especially with respect to its speed of improvement of the total error.

3.3 Simulation 3: Comparing Acquisition Function Optimizers

In this simulation, different acquisition function optimizers are compared for solving the inner optimization problem of the Bayesian optimization framework within the (“ensem-

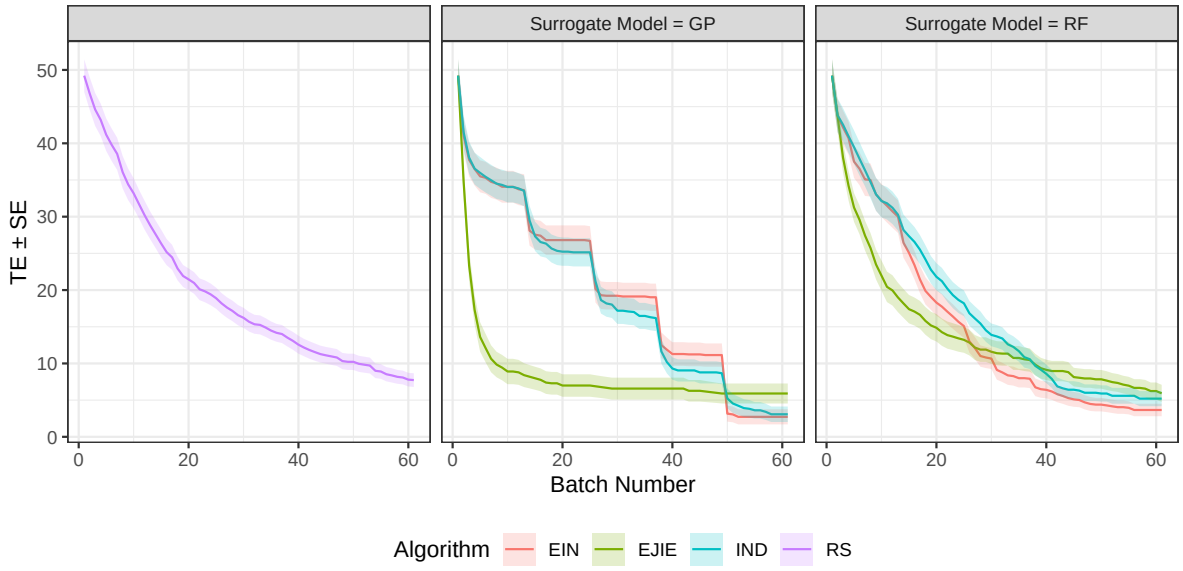


Figure 3.6. Simulation 2.3: Mixed domain. Mean total error. Ribbons represent standard errors. 100 simulation runs.

ble") BOP-Elites algorithm:

$$\boldsymbol{x}^\star \leftarrow \arg\max_{\boldsymbol{x}\in\mathcal{X}} \alpha_{\text{EJIE}}(\boldsymbol{x}) \quad \text{(line 6 in Algorithm 1)}$$

Comparing acquisition function optimizer is motivated by the finding in the quality diversity literature that simultaneous search in all niches aids the generation of better solution for each niche (Chatzilygeroudis et al., 2020; Mouret & Clune, 2015; Nguyen et al., 2015). To investigate this hypothesis, the following acquisition function optimizers are compared: **1.** A Nelder-Mead simplex algorithm as already used in the simulations before (`NM`), **2.** a simple random search where points are drawn uniformly at random (`RS`) and the best point is proposed, **3.** a small-scale GA with a $(10 + 5)$ evolution strategy (`MIES`), mutating continuous parameters by adding an independent standard normally distributed term, performing uniform crossover with a probability of 0.5, where parents are selected at random and the top individuals are selected based on the best fitness value, **4.** a similar small-scale GA that starts with including the current best solutions for each niche in the initial population (`MIES_warm`). Comparing the performance of `MIES` to `MIES_warm` allows for an investigation of the hypothesis, that the benefit of simultaneous search in all niches can be transferred to some extent to the model based quality diversity framework. All acquisition function optimizers are allowed to use up to 100 function evaluations (where the Nelder-Mead algorithm can terminate earlier as

already outlined in the previous simulations). To allow for meaningful conclusions, a fifth acquisition function optimizer is added as a baseline: Random search with 10^5 function evaluations, i.e., 10^5 points are drawn uniformly at random and the best one is proposed (`RS+`).

The simulation design is given as follows: The domain of $f\colon \boldsymbol{x} \mapsto y$ and $g_1\colon \boldsymbol{x} \mapsto z_1$ is given by $[0, 10]^d$, where d varies from 1 to 4. $f(\cdot)$ is given by the d-dimensional Shekel function constructed with 10 local minima (Surjanovic & Bingham, 2013) and $g_1(\cdot)$ is given by the mean prediction of a GP with a Gaussian kernel fitted to initial data $\mathcal{D}_{g_1} = \left\{\left(x_1^{(i)}, z^{(i)} \sim U(0, 20)\right)\right\}_{i=1}^{11}$, i.e., only x_1 is relevant for g_1, where $(x_1^{(1)}, \ldots, x_1^{(11)})^T = (0, \ldots, 10)^T$. This design represents a more realistic and difficult quality diversity optimization problem. Niches and surrogate models are defined as in Simulation 1 and all other technical details are the same as in Simulation 1. The "ensemble" BOP-Elites algorithm (`EJIE`) is used and only the acquisition function optimizer is varied. $5d$ points are sampled uniformly at random and used as the initial design points and all algorithms are run for a total of $35d$ iterations.

Figure 3.7 shows the mean total error for the "ensemble" BOP-Elites algorithm with each different acquisition function optimizer split for the dimensionality d based on 100 simulation runs. In the scenario of $d = 1$, all acquisition function optimizers result in a similar overall performance, although Nelder-Mead (`NM`) tends to result in a somewhat slower convergence. Looking at the scenario of $d = 2$, the random search evaluating 10^5 random points (`RS+`) tends to outperform the other optimizers, but only by a small margin. `NM` further falls behind and the $(10 + 5)$ small-scale GA with a warm start based on the current best solutions for each niche (`MIES_warm`) stronger outperforms both the standard random search (`RS`) and the $(10 + 5)$ small-scale GA without a warm start (`MIES`). This trend consolidates in the scenarios of $d = 3$ and $d = 4$, where `MIES_warm` results in a more substantial performance boost compared to `MIES`.

To investigate whether the different acquisition function optimizers actually find better solutions with respect to the inner optimization problem or whether the solutions simply yield a better improvement regardless of the quality of solving the inner optimization problem, the mean expected joint improvement of elites and the actual improvement after evaluation was calculated. Here, the BO loop always relies on the `RS+` acquisition function optimizer, i.e., the next point to be evaluated is always chosen based on the best solution provided by `RS+` and the values for the different acquisition function optimizers have to be interpreted in a "what if" scenario, i.e., what improvement would the point proposed by `MIES_warm` have yielded if the BO loop had followed the `MIES_warm`

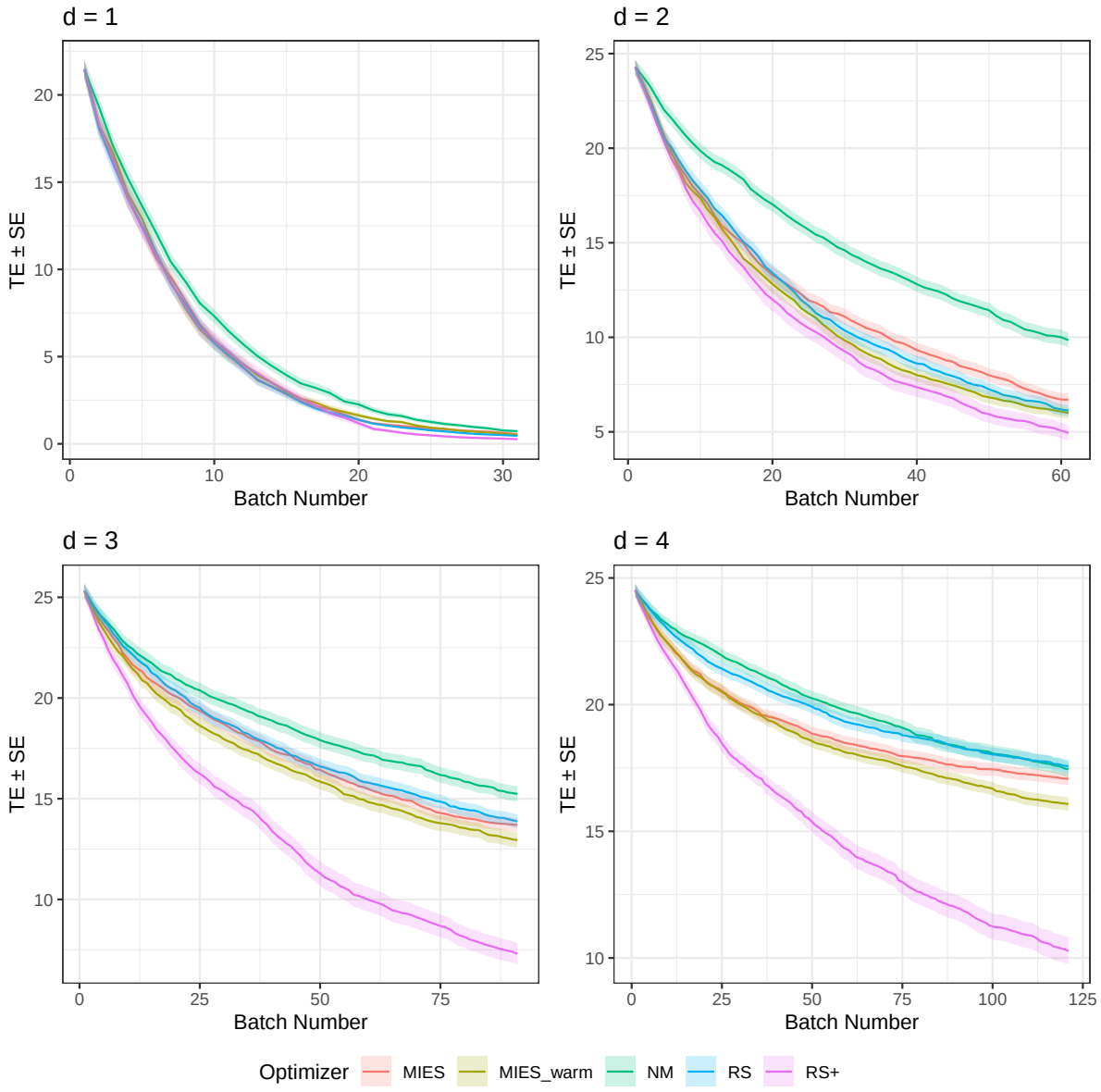


Figure 3.7. Simulation 3: Comparing acquisition function optimizers. Mean total error. Ribbons represent standard errors. 100 simulation runs.

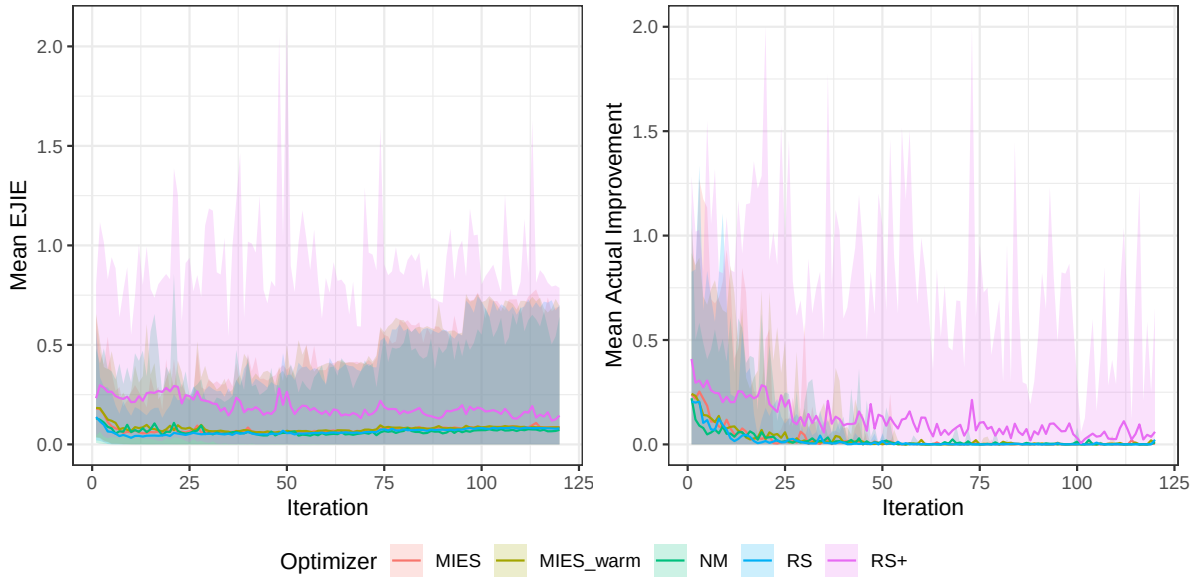


Figure 3.8. Simulation 3: Comparing acquisition function optimizers, $d = 4$. Mean EJIE / Mean actual improvement. Ribbons represent 2.5% and 97.5% quantiles. 100 simulation runs.

optimizer for the next iteration of the BO loop. The actual improvement is calculated as the improvement over the current best solution in niche N_j which is selected based on the niche the evaluated point would have belonged to conditional on the proposed point giving a lower objective function value (i.e., the actual improvement is bounded below by zero). Results are given in Figure 3.8 for the scenario of $d = 4$. `RS+` results in both higher EJIE and actual improvement, i.e., `RS+` solves the inner optimization problem better than the other optimizers and the evaluation of the solutions matches the expected improvement with respect to the actual improvement observed. The other three optimizer perform comparatively similar, although `MIES_warm` appears to have the edge with respect to the actual improvement whereas `NM` falls behind in solving the inner optimization problem.

3.4 Summary and Discussion

In Simulation 1, the results of Kent and Branke (2020) were conceptually replicated in the sense that the "ensemble" BOP-Elites algorithm strongly outperformed its competitors. Differences in results can be explained by the fact that Kent and Branke (2020) discretized the domain into 1000 equidistant points, whereas Simulation 1 treated the domain as naturally continuous. Throughout Simulation 2.1 to 2.3, the "ensemble" BOP-Elites algorithm again outperformed its competitors and performed well throughout the

extensions of overlapping niches, higher-dimensional domains and multiple feature functions.

In Simulation 2.1 the "sequential" and "independent" BOP-Elites algorithm show a performance close to the "ensemble" BOP-Elites algorithm which can be explained due to the overlapping niches, i.e., once a good solution is found in the first niche, this solution is also applicable to all other niches because they all sequentially overlap. Future simulations could therefore focus on different overlapping niches and also the scenario of points not belonging to any niche.

In Simulation 2.3 which is characterized by a three-dimensional mixed domain, GP surrogate models with preprocessing outperformed the random forests surrogate models. This can potentially be explained due to the difference in surrogate model performance with respect to the feature function, i.e., the true function being the mean prediction of a GP which a GP can naturally model very well resulting in more precise predicted probabilities of points belonging to niches (which could also explain the step patterns in the case of the "sequential" and "independent" BOP-Elites algorithms when combined with GP surrogate models). Future simulations could focus on further investigating the question whether a good surrogate model performance with respect to the feature function(s) is more important than good surrogate model performance with respect to the objective function, i.e., by not using a surrogate model at all for the feature function(s) but simply determining the niche a point belongs to by evaluating the feature function(s) in an oracle scenario.

Simulation 3 showed that the choice of the acquisition function optimizer does matter for the BOP-Elites algorithm. While conducting a random search with 10^5 acquisition function evaluations is in most scenarios not practicable, this did result in a strong performance boost, especially for an increased dimensionality of the domain of the objective function and feature function. This itself is interesting because one would argue that random search suffers from the curse of dimensionality, but this only highlights the fact that there is plenty of room for improvement regarding the choice of the acquisition function optimizer. In general, the Nelder-Mead acquisition function optimizer performed comparatively poorly, although this may have been the case due to the multitude of local minima in the acquisition function surface. Variants of Nelder-Mead optimizers may therefore benefit from a random restart procedure. Moreover, a comparison of a small-scale GA with the same small-scale GA which uses a warm start given by the current best solutions for each niche showed that the benefit of simultaneous search in all niches in quality diversity optimization potentially can be transferred to the model based setting by using a suitable acquisition function optimizer (that can exploit the information

given by the best solutions for each niche via, e.g., crossover). Future simulations should further investigate this potential.

In the following application section, the "ensemble" BOP-Elites algorithm is used to find a set of high-performing yet (resource-related) diverse neural architectures for image classification.

4 Application Study

Neural Architecture Search (NAS) promises to automatically find well performing architectures of deep neural networks that facilitate the learning of strong representations for a given dataset (Elsken, Metzen, & Hutter, 2019b). So far, NAS methods have outperformed manually designed architectures on tasks such as image classification (Real, Aggarwal, Huang, & Le, 2019; Zoph, Vasudevan, Shlens, & Le, 2018) or object detection (Zoph et al., 2018). Formally, NAS can be embedded within the HPO setting (see equation 1.1) with the particularity that other hyperparameters such as batch size or learning rate are typically considered fixed while tuning over the search space of architectures and no joint tuning is performed.

Following Elsken et al. (2019b), NAS can be categorized according to three dimensions: search space, search strategy and performance estimation strategy. The search space defines which architectures can be represented, while the search strategy details how to explore the search space (e.g., Bayesian optimization, evolutionary methods, reinforcement learning or gradient-based methods). The simplest performance estimation strategy is given by training the architecture on training data and evaluating its performance on validation data. However, due to the huge computational demand strategies like using lower fidelity estimates (e.g., Bello, Zoph, Vasudevan, and Le 2017) or weight sharing (e.g., Pham, Guan, Zoph, Le, and Dean 2018) have become popular.

Typically, NAS is considered a single-objective optimization problem (Hutter et al., 2018, Chapter 3) and the goal is to find an architecture that maximizes performance (e.g., validation accuracy). However, resource restrictions like number of FLOPS naturally arise when deploying networks on different hardware (e.g., in computer vision, where algorithms are being integrated and deployed on very heterogeneous small devices, see e.g., Xiong, Mehta, and Singh 2019). One possibility for handling resource restrictions is to incorporate them as additional objective functions in a multi-objective optimization problem (e.g., Elsken, Metzen, and Hutter 2019a). Another possibility is to incorporate them as constraints in a constrained optimization problem (e.g., Jin et al. 2019; Xiong et al. 2019). While the multi-objective approach allows for finding a set of (non-dominated) solutions along the Pareto front during a single optimization run, these

solutions must not necessarily be diverse and only reflect different trade-offs with respect to the different objectives. On the other hand, the constrained approach only allows for finding a single solution (that satisfies a given set of constraints) in a single optimization run. In this application study, the BOP-Elites algorithm is applied to NAS, aiming at finding a set of high-performing yet diverse architectures with respect to pre-defined (resource-related) feature functions in a single run. In principle, these resource functions could range from number of trainable parameters or number of FLOPS to predict time or memory usage during prediction. Note that while prior work applying quality diversity optimization algorithms to machine learning problems exists (e.g., Cazenille, Bredeche, and Halloy 2019; Costa, Lourenço, Correia, and Machado 2020; Parker-Holder, Pacchiano, Choromanski, and Roberts 2020; Stanley, Clune, Lehman, and Miikkulainen 2019), this application is the first of its kind to use a (model based) quality diversity optimization algorithm directly for NAS.

To facilitate the computational burden, the study was conducted using the NAS-Bench-301 benchmark (Siems et al., 2020). NAS-Bench-301 is a surrogate benchmark for neural architecture search. Contrary to other tabular benchmarks for NAS like the NAS-Bench-101 (Ying et al., 2019), or NAS-Bench-201 (Dong & Yang, 2020), NAS-Bench-301 relies on surrogate models that can be used for prediction instead of querying tables to get the validation accuracy or training time of the architecture in question. This overcomes the limitation of only a small number of architectures being searched that tabular benchmarks suffer (where all architectures in the search space must be evaluated exhaustively). NAS-Bench-301 covers the cell-based search space of DARTS (Liu, Simonyan, & Yang, 2019; Zoph et al., 2018) trained on the CIFAR-10 dataset (Krizhevsky, 2009), which contains more than 10^{18} possible architectures. Surrogates were constructed by evaluating architectures using the standard 40k, 10k, 10k split for train, validation and test set. For more details on NAS-Bench-301, see Siems et al. (2020).

The DARTS search space (Liu et al., 2019; Zoph et al., 2018) consist of so-called normal and reduction cells that are stacked to form a convolutional neural network. A cell is a directed acyclic graph (DAG) consisting of an ordered sequence of vertices (here nodes). Each node is a feature map and each directed edge is associated with an operation that transforms the input node. Each cell is assumed to have two input nodes and one output node, whereby the input nodes are defined as the cell outputs in the previous two layers and the output of the cell is obtained by a reduction operation to all intermediate nodes. Here, the four intermediate nodes add element-wise feature maps from two previous nodes in the cell. The directed edges represent one of the

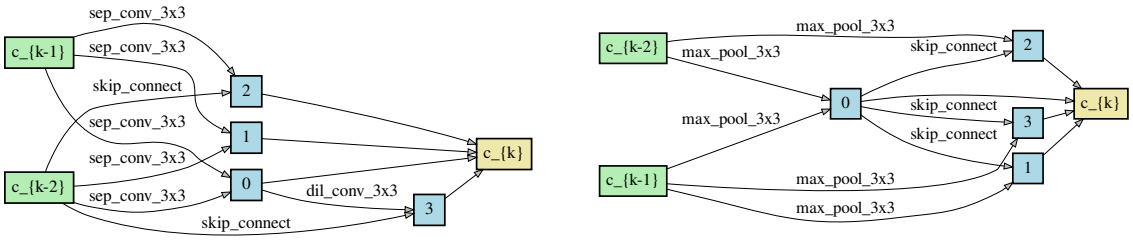


Figure 4.1. Example DARTS architecture. Normal cell on the left. Reduction cell on the right.

following operations: 3x3 and 5x5 separable convolutions, 3x3 and 5x5 dilated separable convolutions, 3x3 max pooling, 3x3 average pooling, identity, and zero (skipping the connection). All operations are of stride one (if applicable) and the convolved feature maps are padded. The ReLU-Conv-BN order is used for convolutional operations, and each separable convolution is applied twice. Cells located at 1/3 and 2/3 of the total depth of the network are reduction cells, in which all the operations adjacent to the input nodes are of stride two. An example architecture as described in Liu et al. (2019) is given in Figure 4.1, where c_{k-1} and c_{k-2} denote the input nodes and c_k denotes the output node. Additional details on the search space and its architectures can be found in Zoph et al. (2018), Liu et al. (2019) and Siems et al. (2020).

As a feature function, the number of trainable parameters was selected[1] and five overlapping niches were defined as: $[0, 2500000)$, $[0, 3000000)$, $[0, 3500000)$, $[0, 4000000)$, $[0, \infty)$. The reason for constructing overlapping niches lies in the context of resource restrictions, since it is more sensible to assume that a device that can handle up to, e.g., a fixed number of FLOPS (or here parameters) can also handle a lower number of FLOPS. The right boundaries were selected to closely reflect the following quantiles of the empirical cumulative distribution function of the around 60000 architectures used to train the NAS-Bench-301 surrogate models: $0.15, 0.45, 0.75, 0.9, 1$ (i.e., of all architectures used to train the surrogate models, 75% had less than 3500000 trainable parameters).

As outlined in Section 2, the BOP-Elites algorithm is embedded within the Bayesian optimization framework. As competitors, BANANAS (White et al., 2019) and a simple random search (as NAS method, `Random`) were selected. BANANAS is also embedded within the Bayesian optimization framework and uses an ensemble of feed-forward neural networks as surrogate model (where each network is initialized using different random weights and trained using a random set order) and encodes architectures using a trun-

[1] which can be interpreted as a proxy related to, e.g., memory usage or number of FLOPS during predict time

cated path encoding (for every path, i.e., every possible ordering of nodes, a binary feature is generated, indicating whether the DAG contains all directed edges along this path but only those paths are included that are "likely" to occur when randomly sampling edges in the DAG subject to a maximum edge constraint). BANANAS then uses independent Thompson sampling (White et al., 2019) as the acquisition function which is optimized using a mutation algorithm (the best architecture observed so far is selected and mutated in 100 different ways by changing a single operation or edge randomly). According to White et al. (2019), BANANAS "achieves state-of-the-art performance on NAS search spaces". As neither BANANAS nor random search are aware of the feature function and niches, niches were derived post hoc by determining the number of trainable parameters for each evaluated architecture.

The BOP-Elites algorithm was configured as follows: As surrogate models, random forests with some preprocessing were used (imputing missing categorical values with a new level ".missing"). Note that no transformation of the architectures was carried out (i.e., representing the DAG via an adjacency matrix, see, e.g., White, Neiswanger, Nolen, and Savani 2020), because a random forest can naturally handle the representation of an architecture via interdependent categorical parameters (edges between nodes and their operation) which is provided by NAS-Bench-301 in the form of a ConfigSpace (Lindauer et al., 2019) containing 34 categorical parameters with 24 dependencies. More precisely, in this natural tabular encoding, architectures are represented by enumerating all nodes and potential edges and introducing categorical hyperparameters for each operation along each potential edge, where the nodes serving as input of each intermediate node are again defined as categorical hyperparameters and operations on a certain edge can only be specified if this edge is actually present in the DAG (Siems et al., 2020). As an acquisition function the EJIE was used (see Equation 2.2). The EJIE was optimized using a mutation algorithm similar to the one BANANAS uses but adapted for the niches setting. More specifically, the best architectures observed in each niche so far are selected and a single operation or edge is mutated randomly in each architecture. Disjoint pairs of parents are then selected at random and a single operation or edge that is currently not mutated is selected randomly and crossover of this operation or edge is performed with a probability of 0.5. This procedure is then repeated for the resulting children until only a single child is left. The complete GA is repeated 100 times resulting in 100 candidate architectures. More details on and a justification for the configuration of the BOP-Elites algorithm is given in an ablation study in the appendix (see Section 6.1 in the appendix). Each algorithm starts with evaluating 10 architectures drawn uniformly at random which are used as the initial design points and all algorithms are run

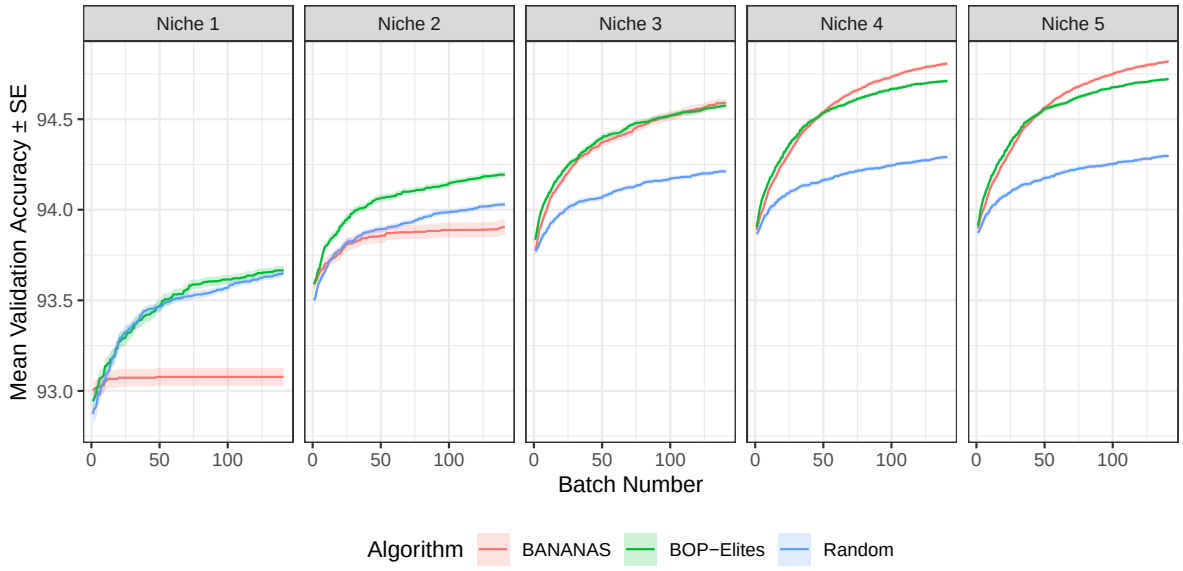


Figure 4.2. BOP-Elites on NAS-Bench-301. Mean validation accuracy. Ribbons represent standard errors. 100 replications.

for a total of 150 iterations (which would correspond to roughly 10 GPU days). Note that by default BANANAS only updates its surrogate model every 10 iterations (due to the computational cost to train the ensemble of feed-forward neural networks). In the application study presented here, BANANAS was configured to update its surrogate model every iteration to allow for a meaningful comparison to the BOP-Elites algorithm. Results are based on 100 replications. Computational details are given in Section 6.2.

Figure 4.2 shows the mean validation accuracy over the 100 iterations separate for each niche. Generally, BANANAS improves its performance with respect to niches 3 to 5 while BOP-Elites shows a more uniform distributed performance improvement over all niches. Random search results in comparably good performance with respect to niches 1 and 2 but falls behind in the other niches. All in all, this is the expected behavior: BANANAS is not aware of the niches and architectures with a higher number of trainable parameters typically also yield better performance. Contrary to this, BOP-Elites balances high performance with diversity with respect to the niches and is able to find well performing architectures in every niche. Figure 4.3 additionally visualizes the mean validation accuracy over all five niches over the 100 iterations, showing more clearly that BOP-Elites is able to find well performing solutions over all niches.

This application study is the first of its kind to use a (model based) quality diversity optimization algorithm for NAS. On the NAS-Bench-301 surrogate benchmark, BOP-Elites is able to find a set of high-performing yet diverse neural architectures of the

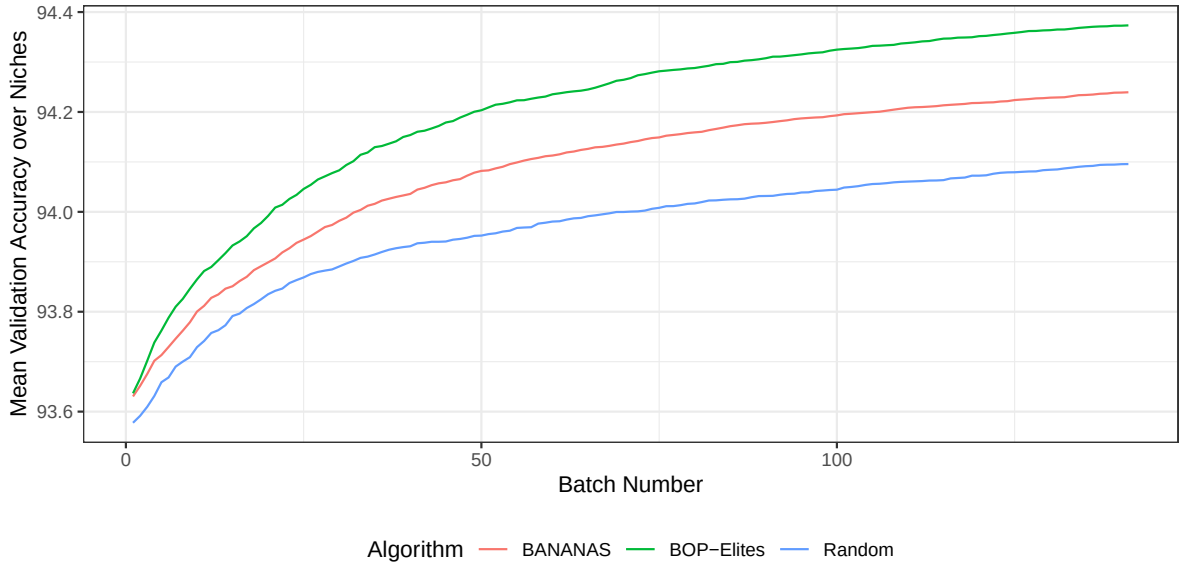


Figure 4.3. BOP-Elites on NAS-Bench-301. Mean validation accuracy over all niches. 100 replications.

DARTS search space with respect to a pre-defined resource-related feature function and thereupon derived niches. The overall quality of the solutions outperforms that of BANANAS, a state-of-the-art model based NAS algorithm. However, as seen in the ablation study, fine-tuning of the configuration of BOP-Elites (choice of surrogate model and acquisition function optimizer) is needed to achieve good performance although most of the performance difference stems from the choice of the acquisition function optimizer. Similarly, BANANAS appears to mostly rely on its acquisition function optimizer to be able to yield good performance on NAS-Bench-301. This is interesting as the choice of acquisition function optimizer is typically not examined in detail when configuring model based algorithms. Future work should extend the results presented here to other benchmarks and search spaces but also real applications including other feature functions such as number of FLOPS, memory usage or energy usage during predict time.

5 General Discussion and Outlook

The goal of quality diversity optimization is to find a set of high-performing, yet diverse solutions. In the setting described here, so called feature functions constitute so called behavioral niches and the goal is to find a high-performing solution for every niche. In the model based quality diversity setting, both the objective function and all feature functions are treated as black-box functions naturally calling for algorithms being embedded within the Bayesian optimization framework. Seminal work is given by Kent and Branke (2020) introducing the BOP-Elites algorithm which relies on probabilistic surrogate models for both the objective function and feature function(s) and balances exploration and exploitation over niches by considering a novel acquisition function called the expected joint improvement of elites.

In this thesis, BOP-Elites has been extended and investigated in the scenarios of overlapping niches (i.e., not necessarily pairwise-disjoint), higher-dimensional domains, multiple feature functions, as well as mixed domains. Throughout all extensions, BOP-Elites yielded promising results. However, two findings are of central importance: First, it is unclear whether a good performance of the surrogate(s) modeling the feature function(s) should be of higher interest than the performance of the surrogate modeling the objective function. Looking at the simulation scenario of a mixed domain, results indicate that the former might be of higher importance, i.e., if BOP-Elites cannot determine the probability of a point belonging to a niche with high accuracy, the expected joint improvement may result in non-optimal points being proposed leading to overall less improvement as if the improvement of the point is under or overestimated. First, future work should systematically investigate the effect of the performance of the different surrogate models on the overall performance of BOP-Elites. Secondly, results regarding the choice of acquisition function optimizer suggest that it might be possible to transfer the benefit of simultaneous search in all niches, observed in the quality diversity literature (Chatzilygeroudis et al., 2020; Mouret & Clune, 2015; Nguyen et al., 2015), to the model based setting by using GAs with crossover that incorporate the best solution found for each niche so far in a warm start setting (i.e., by including these points in the initial population). Future work should examine this finding more comprehensively.

Looking at the current methodological state of BOP-Elites, several extensions could be of central interest: First, BOP-Elites should be extended to handle a noisy objective and/or feature function(s). Moreover, looking at determining the probability of a point belonging to a niche based on the surrogate models' predictions modeling the feature functions, it could be fruitful to relax the assumption of independent feature functions. For example, multi-output Gaussian processes could be used to jointly model all feature functions and by exploiting the correlations between them could provide better predictions (Bonilla, Chai, & Williams, 2008; Dai, Álvarez, & Lawrence, 2017; Moreno-Muñoz, Artés, & Álvarez, 2018), allowing for a more precise derivation of the probability of a point belonging to a niche. Moreover, multi-point proposal variants could be developed which might exploit the existence of diverse niches for their proposals. In this context, the derivation of novel acquisition functions would be useful. For example, entropy based acquisition functions like entropy search (Hennig & Schuler, 2012), predictive entropy search (Hernández-Lobato, Hoffman, & Ghahramani, 2014) or max-value entropy search (Wang & Jegelka, 2017) could be adapted to the BOP-Elites setting.

On the NAS-Bench-301 surrogate benchmark, BOP-Elites was able to find a set of high-performing yet diverse neural architectures of the DARTS search space with respect to a pre-defined resource-related feature function and thereupon derived niches. This is a novel application in the sense that no other (model based) quality diversity optimization algorithm has yet been applied to NAS directly. Overall, BOP-Elites showed good performance surpassing state-of-the-art NAS algorithms. In ablation studies, focus was given to the configuration of BOP-Elites showing that especially the choice of the acquisition function optimizer is of high importance for NAS problems (at least for NAS-Bench-301). Future work should generalize this finding to other NAS benchmarks. Moreover, the capability of BOP-Elites to solve resource restricted NAS problems should be examined in real applications including other feature functions such as number of FLOPS, memory usage or energy usage during predict time.

6 Appendix

6.1 NAS-Bench-301 Ablation Study

6.1.1 BANANAS Configurations

In this section, results of an ablation study on the configuration of BANANAS (White et al., 2019) and BOP-Elites (Kent & Branke, 2020) on NAS-Bench-301 are presented. As introduced in Section 4, BANANAS is embedded within the Bayesian optimization framework and uses an ensemble of feed-forward neural networks as a surrogate model combined with path encoding of architectures. As an acquisition function, BANANAS uses independent Thompson sampling which is optimized using a mutation algorithm where the best architecture observed so far is selected and mutated in 100 different ways by changing a single operation or edge randomly. As BOP-Elites initially performed poorly on NAS-Bench-301, an ablation study on the configuration of BANANAS on NAS-Bench-301 was conducted, aiming to answer the following question: Which configuration of which components (surrogate model, path encoding, acquisition function, acquisition function optimizer) leads to good performance of BANANAS on NAS-Bench-301.

The general design is given as follows: Regarding the surrogate model and path encoding, either a random forest (`RF`) with or without path encoding (`Paths`)[1] or an ensemble of feed-forward neural networks (`NN`) was used. The acquisition function was varied between independent Thomson sampling (`ITS`) and expected improvement (`EI`) and the acquisition function optimizer was varied between the mutation algorithm described above (`Mut`) and a simple random search where 1000 architectures are drawn uniformly at random (`RS`). These variations were crossed in a full factorial design where applicable. Ten architectures were sampled uniformly at random and evaluated as the initial design points and all algorithms were run for 100 iterations.

Results are given in Figure 6.1 where the first facet shows the performance of the orig-

[1] in the case of no path encoding with some preprocessing where missing categorical values were imputed with a new level ".missing"

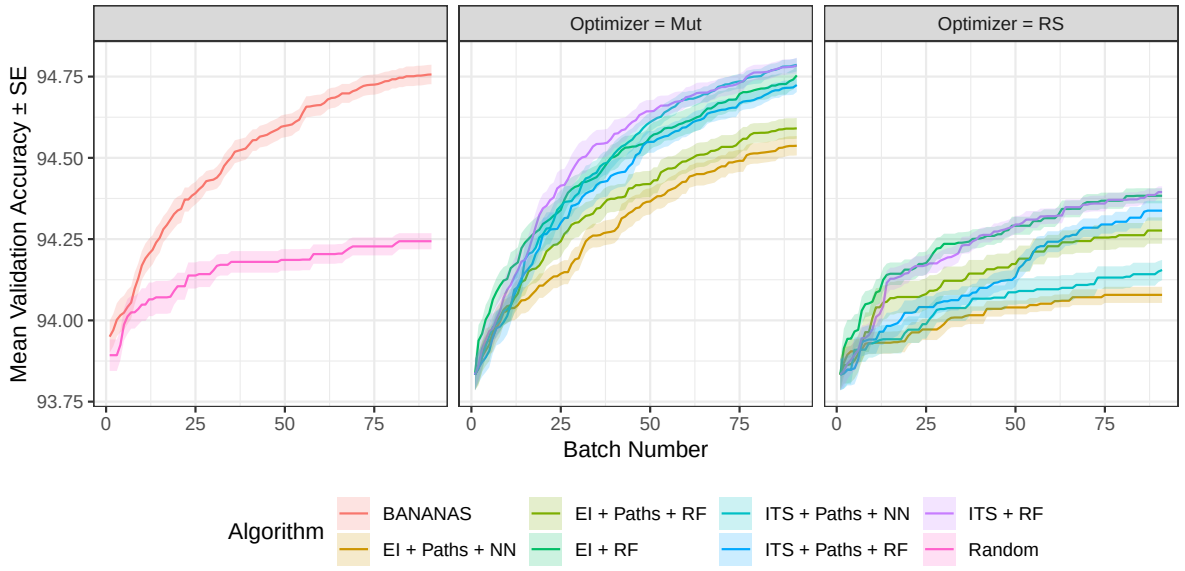


Figure 6.1. Different BANANAS configurations on NAS-Bench-301. Mean validation accuracy. Left facet: BANANAS and `Random`. Middle facet: Acquisition Function Optimizer `Mut`. Right facet: Acquisition Function Optimizer `RS`. Ribbons represent standard errors. 20 replications.

inal BANANAS configuration and random search (as NAS method, `Random`) whereas the second and third facet show the performance of the configurations listed above using `Mut` or `RS` as the acquisition function optimizer. Note that `ITS + Paths + NN + Mut` by design is a reimplementation of BANANAS. Summarizing the results, several effects can be identified: Using independent Thompson sampling instead of expected improvement only benefits configurations that rely on path encoding and especially those that use mutation as the acquisition function optimizer. In general, using mutation as acquisition function optimizer always results in a strong performance boost compared to random search as acquisition function optimizer. Notably, BANANAS' novel ensemble of feed-forward neural networks together with path encoding only performs well if combined with mutation as acquisition function optimizer and is otherwise outperformed by `Random`. Moreover, the very simple configuration of a random forest as a surrogate model, with no path encoding together with expected improvement that is optimized using mutation performs similarly to the default BANANAS configuration. Table 6.1 presents results of a four-way ANOVA on the final performance of the algorithms outlined above (excluding BANANAS and `Random`) with respect to the factors surrogate candidate, architecture encoding, acquisition function, and acquisition function optimizer. The acquisition function optimizer is by far the most important determinant of final performance.

	Sum Sq	Df	F value	Pr(>F)
Surrogate Candidate	0.35	1	18.58	0.0000
Architecture Encoding	0.37	1	20.11	0.0000
Acquisition Function	0.53	1	28.21	0.0000
Acq. F. Optimizer	10.84	1	582.31	0.0000
Residuals	4.38	235		

Table 6.1. Different BANANAS configurations on NAS-Bench-301. Results of a four-way ANOVA on the factors surrogate candidate, architecture encoding, acquisition function, and acquisition function optimizer. Type II sums of squares.

Another small scale ablation study was conducted investigating the performance difference of BANANAS with respect to the acquisition function optimizers. Based on the random forest surrogate model with no path encoding and expected improvement as the acquisition function, three different acquisition function optimizers were compared: Random search where 10^5 architectures are drawn uniformly at random (`RS+`), random search where 1000 architectures are drawn uniformly at random (`RS`) and the mutation algorithm (`Mut`) described above. Ten architectures were sampled uniformly at randomly and evaluated as the initial design points and all algorithms were run for 100 iterations. Results are given in Figure 6.2. As can be seen, `Mut` strongly outperforms even the `RS+` optimizer.

To investigate whether the different acquisition function optimizers actually find better solutions with respect to the inner optimization problem or whether the solutions simply yield a better improvement regardless of the quality of solving the inner optimization problem, the mean expected improvement and the actual improvement after evaluation were calculated similarly as in Simulation 3. Here, the BO loop always relies on the `RS+` acquisition function optimizer, i.e., the next architecture to be evaluated is always chosen based on the best solution provided by `RS+` and the values for the different acquisition function optimizers have to be interpreted in a "what if" scenario, i.e., what improvement would the architecture proposed by `Mut` have yielded if the BO loop had followed the `Mut` optimizer for the next iteration of the BO loop. Ten architectures were sampled uniformly at random and evaluated as the initial design points and all algorithms were run for 100 iterations. Results are given in Figure 6.3. `Mut` results in both higher mean EI and mean actual improvement, i.e., `Mut` solves the inner optimization problem better than the other optimizers and the evaluation of the solutions matches the expected improvement with respect to the actual improvement observed. Notably,

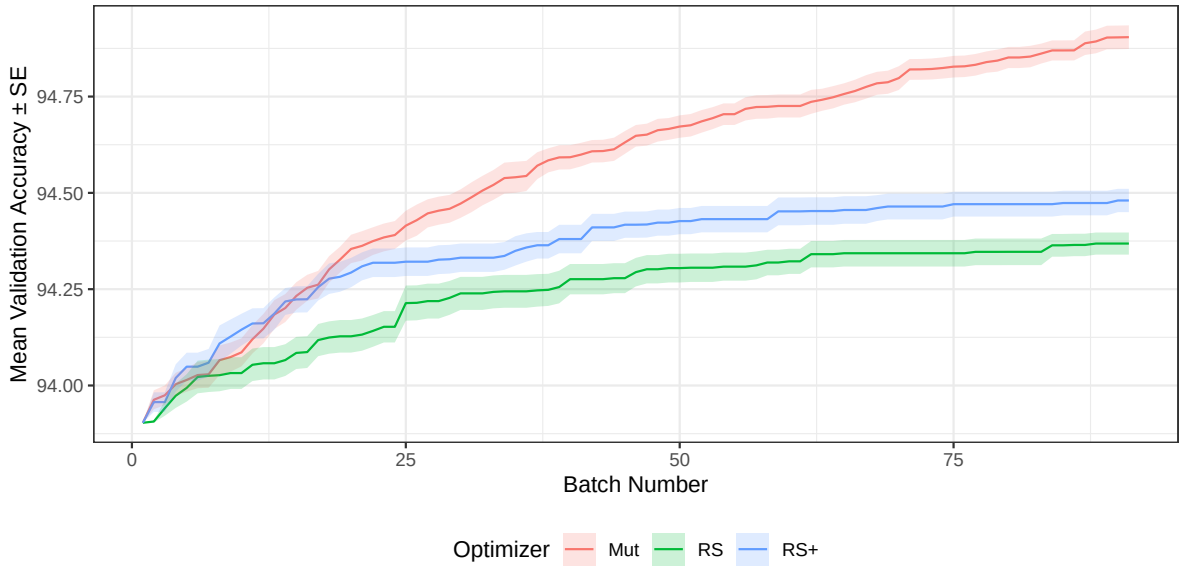


Figure 6.2. Random forest surrogate model and expected improvement on NAS-Bench-301. Different acquisition function optimizers. Mean validation accuracy. Ribbons represent standard errors. 20 replications.

`Mut` even strongly outperforms `RS+`, which is quite surprising given the fact that all that `Mut` does is to mutate a random single operation or edge of the best architecture found so far. This result hints at the fact that the DARTS search space may profit strongly from local search variants (e.g., White, Nolen, and Savani 2020)

6.1.2 BOP-Elites Configurations

Based on the results of the ablation study above, BOP-Elites was configured using random forests as a surrogate model (with some preprocessing where missing categorical values were imputed with a new level ".missing") due to its simplicity, lower computational effort but nevertheless good performance. The last configurable component missing for the configuration of BOP-Elites on NAS-Bench-301 is given by the acquisition function optimizer. Therefore, another small scale experiment was conducted varying the acquisition function optimizer of the BOP-Elites algorithm. Five different acquisition function optimizers were compared, namely: **1.** Random search (`RS`) where 1000 architectures are drawn uniformly at random, **2** the mutation algorithm used by BANANAS (`MUT`) adapted for the niche setting by sampling the niche for which the best architecture observed so far should be mutated, **3.** `Mut` where for every other iteration (if $n \equiv 0 \pmod 2$) `RS` is used for the optimization (`Mut Interleave`), **4.** an own custom GA adapted for the niches setting (`Mut/Cross`) and **5.** `Mut/Cross` where for every other

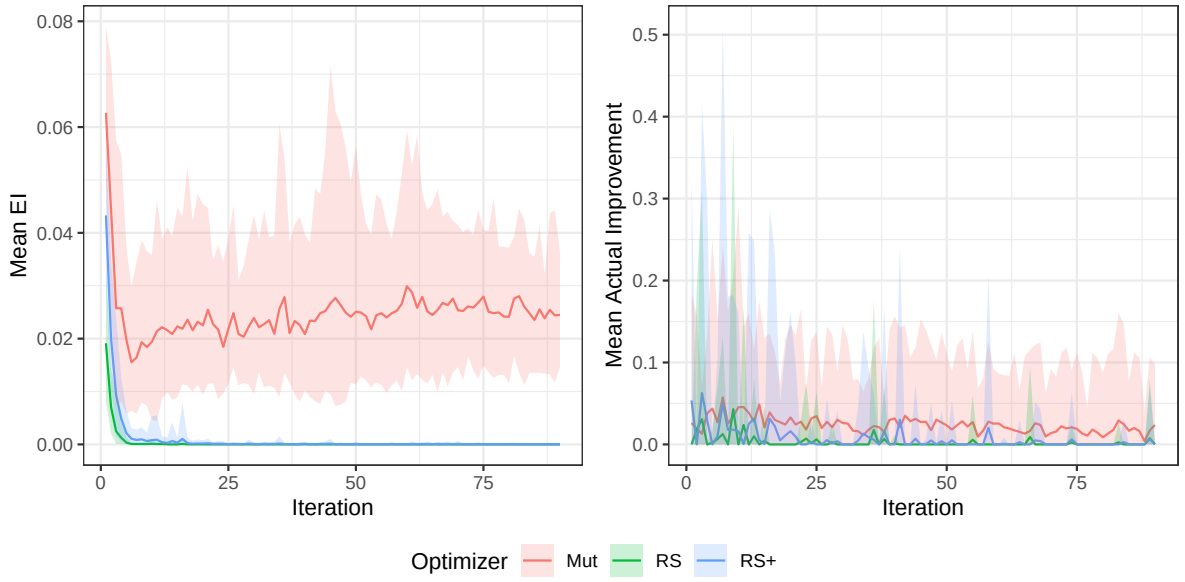


Figure 6.3. Random forest surrogate model and expected improvement on NAS-Bench-301. Different acquisition function optimizers. Mean EI / Mean actual improvement. Ribbons represent 2.5% and 97.5% quantiles. 20 replications.

iteration (if $n \equiv 0 \pmod 2$) `RS` is used for the optimization (`Mut/Cross Interleave`). `Mut/Cross` works as the following: The best architectures observed in each niche so far are selected and a single operation or edge is mutated randomly in each architecture. Disjoint pairs of parents are then selected at random and a single operation or edge that is currently not mutated is selected randomly and crossover of this operation or edge is performed with a probability of 0.5. This procedure is then repeated for the resulting children until only a single child is left and the complete GA is repeated 100 times resulting in 100 candidate architectures. Ten architectures were sampled randomly and evaluated as the initial design points and the BOP-Elites algorithms was run for 150 iterations with the varying acquisition function optimizers listed above. Results are given in Figure 6.4. Overall, results differ strongly depending on the choice of the acquisition function optimizer. In general, `RS` and the random search interleaving variants (`Mut Interleave` and `Mut/Cross Interleave`) perform well for the lower niches (especially niche 1). In these niches, both `Mut` and `Mut/Cross` fail to lead to substantial performance increase and stagnate similarly to BANANAS that is completely unaware of the feature function and niches (see, e.g., Figure 4.2). In niches 3 to 5, the custom GAs outperform the other optimizers and the crossover variation `Mut/Cross` manages to yield the highest performance for niches 4 and 5. In order to have good performance with respect to all niches it appears that interleaving random search as the acquisition function optimizer

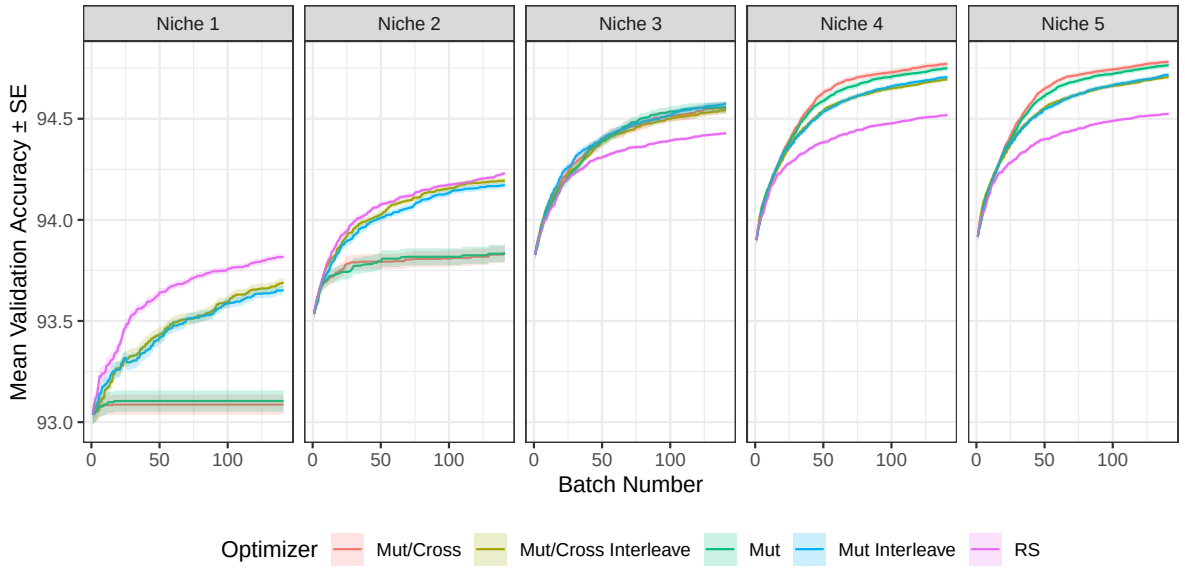


Figure 6.4. Configuration of BOP-Elites on NAS-Bench-301. Different acquisition function optimizers. Mean validation accuracy. Ribbons represent standard errors. 100 replications.

(as done in `Mut Interleave` and `Mut/Cross Interleave`) is a must-have. A potential explanation for the bad performance of `Mut` and `Mut/Cross` with respect to the lower niches could be that these optimizers are biased for architectures with a higher number of trainable parameters because these tend to yield a higher improvement in performance, combined with an imprecise surrogate model prediction with respect to the feature function (therefore deriving the probability of an architecture belonging to a certain niche could be inexact and the expected joint improvement of elites would then be determined to a larger extent by the performance gain with respect to niches that cover a wider range).

6.2 Computational Details

The BOP-Elites algorithm was implemented in R (R Core Team, 2020) within the `mlr3` ecosystem relying on `mlr3mbo` (version 0.0.0.9999, Richter et al. 2021) and `bbotk` (version 0.3.0.9999, Becker, Richter, Lang, Bischl, and Binder 2021). Up-to date forks are available at `https://github.com/sumny/mlr3mbo` and `https://github.com/sumny/bbotk`. Preprocessing pipelines were built using `mlr3pipelines` (version 0.3.0, Binder et al. 2020). Gaussian processes were used as implemented in the `mlr3extralearners` (Sonabend & Schratz, 2020) package (version 0.1.1) wrapping `DiceKriging::km` (version 1.5.8; Roustant, Ginsbourger, and Deville 2012). Here, default hyperparameter val-

ues were set except for `nugget.stability = 1e-8`. Random forests were used as implemented in the `mlr3extralearners` package wrapping `ranger::ranger` (version 0.12.1; Wright and Ziegler 2017) with the following hyperparameter values: `num.trees = 500`, `se.method = "jack"`, `respect.unordered.factors = "order"`, and additionally `min.node.size = 1` for the simulation studies. Regression trees were used as implemented in `mlr3` wrapping `rpart::rpart` (version 4.1-15; Therneau and Atkinson 2019) with no hyperparameters changes except for `minbucket = 5`. Regarding acquisition function optimizers, the Nelder-Mead implementation (`NLOPT_LN_NELDERMEAD`) of `NLopt` (Johnson, 2021) was used as implemented in `bbotk` wrapping `nloptr::nloptr` (version 1.2.2.2); random search was used as implemented in `bbotk` and `mlr3mbo`; genetic algorithms were built using `miesmuschel` (Binder, 2021) version 0.0.0-9000. Additional packages being used are: `mlr3` (version 0.11.0; Lang et al. 2019), `mlr3misc` (version 0.7.0; Lang and Schratz 2021), `paradox` (version 0.7.1; Lang and Schratz 2021), `R6` (version 2.5.0; Chang 2020).

Python (Van Rossum & Drake Jr, 1995) 3.8.7 was used via the `reticulate` package (version 1.18; Ushey, Allaire, and Tang 2020) within R. For NAS-Bench-301, `nasbench301` version 0.2 (Siems et al., 2020) was used relying on the `xgb_v1.0` surrogate model (deterministic) for the validation accuracy. The number of trainable model parameters of the architectures was determined by building the respective convolutional neural network using `DARTS` (Liu et al., 2019) and looping over all cells and layers. Prior to fitting the surrogate model, the number of trainable parameters was log-transformed. The feed-forward ensemble of neural networks and path encoding as used by BANANAS was directly adopted as implemented in `naszilla` (version 1.0; White, Neiswanger, et al. 2020). BANANAS and random search (as a NAS method) were run using `naszilla` employing the same `nasbench301` setup as described above under Python 3.6.12 (due to different module requirements). BANANAS was configured to update its surrogate model after each iteration (instead of the default 10 iterations). Additional Python modules required by `nasbench301`, `DARTS` or `naszilla` are not explicitly listed here.

Figures were created using the `ggplot2` (Wickham, 2016) and `ggpubr` (Kassambara, 2020) packages (version 3.3.3 and 0.4.0). All computations were performed on 2 Intel© Xeon© E5-2650 v2 @ 2.60GHz CPUs each with 16 threads using R 4.0.3 under Ubuntu 20.04.1 LTS. Parallelization in R was done via the `future` (Bengtsson, 2020) and `future.apply` (Bengtsson, 2020) packages (version 1.21.0 and 1.7.0) on top of the internal parallelization of the `data.table` (Dowle & Srinivasan, 2021) package (version 1.14.0). Numerical values were rounded based on the IEC 60559 standard.

6.2.1 NAS Best Practices Checklist

Here, answers are given to applicable questions of the NAS best practices checklist (version 1.0), see Lindauer and Hutter (2019).

- for all NAS methods NAS-Bench-301 (`nasbench301` version 0.2) was used relying on the `xgb_v1.0` surrogate model (deterministic) for the validation accuracy
- all computations were run on the same hardware (2 Intel© Xeon© E5-2650 v2 @ 2.60GHz CPUs)
- ablation studies on the configuration of BANANAS and BOP-Elites on NAS-Bench-301 were run and are reported in the appendix in Section 6.1
- the same evaluation protocol was used for all methods (for BANANAS and random search niches were derived post hoc if needed)
- performance was compared with respect to the number of architecture evaluations
- random search was included as a NAS method
- multiple runs (either 20 or 100) were conducted; reproducibility with respect to the BOP-Elites algorithm implemented in R is given due to an initial random seed being set; regarding `naszilla`, no seed can be explicitly set

References

Audet, C., & Hare, W. (2017). *Derivative-free and blackbox optimization*. New York: Springer.

Becker, M., Richter, J., Lang, M., Bischl, B., & Binder, M. (2021). bbotk: Black-box optimization toolkit [Computer software manual]. (https://bbotk.mlr-org.com, https://github.com/mlr-org/bbotk)

Bello, I., Zoph, B., Vasudevan, V., & Le, Q. V. (2017). Neural optimizer search with reinforcement learning. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (pp. 459–468).

Bengtsson, H. (2020). *A unifying framework for parallel and distributed processing in R using futures*. arXiv:2008.00553 [cs.DG]. (Online; accessed 10 May 2021)

Binder, M. (2021). miesmuschel: Mixed integer evolutionary strategies [Computer software manual]. Retrieved from https://github.com/mlr-org/miesmuschel (R package version 0.0.0-9000)

Binder, M., Pfisterer, F., Schneider, L., Bischl, B., Lang, M., & Dandl, S. (2020). mlr3pipelines: Preprocessing operators and pipelines for 'mlr3' [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=mlr3pipelines (R package version 0.3.0)

Bonilla, E. V., Chai, K., & Williams, C. (2008). Multi-task Gaussian process prediction. In J. Platt, D. Koller, Y. Singer, & S. Roweis (Eds.), *Proceedings of the 20th international conference on neural information processing systems* (pp. 153–160).

Box, M. J. (1965). A new method of constrained optimization and a comparison with other methods. *The Computer Journal*, *8*(1), 42–52.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. Boca Raton: CRC Press.

Byrd, R. H., Lu, P., Nocedal, J., & Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, *16*(5), 1190–1208.

Cazenille, L., Bredeche, N., & Halloy, J. (2019). Automatic calibration of artificial neural

networks for zebrafish collective behaviours using a quality diversity algorithm. In U. Martinez-Hernandez et al. (Eds.), *Conference on biomimetic and biohybrid systems* (pp. 38–50).

Chang, W. (2020). R6: Encapsulated classes with reference semantics [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=R6 (R package version 2.5.0)

Chatzilygeroudis, K., Cully, A., Vassiliades, V., & Mouret, J.-B. (2020). *Quality-diversity optimization: A novel branch of stochastic optimization.* arXiv:2012.04322 [cs.NE]. (Online; accessed 05 May 2021)

Clune, J., Misevic, D., Ofria, C., Lenski, R. E., Elena, S. F., & Sanjuán, R. (2008). Natural selection fails to optimize mutation rates for long-term adaptation on rugged fitness landscapes. *PLoS Computational Biology*, *4*(9), e1000187.

Costa, V., Lourenço, N., Correia, J., & Machado, P. (2020). Exploring the evolution of GANs through quality diversity. In *Proceedings of the 2020 genetic and evolutionary computation conference.*

Cully, A., Clune, J., Tarapore, D., & Mouret, J.-B. (2015). Robots that can adapt like animals. *Nature*, *521*(7553), 503–507.

Cully, A., & Mouret, J.-B. (2016). Evolving a behavioral repertoire for a walking robot. *Evolutionary Computation*, *24*(1), 59–88.

Cully, A., Mouret, J.-B., & Doncieux, S. (2019). *Quality-diversity optimisation algorithms.* https://quality-diversity.github.io/. (Online; accessed 07 April 2021)

Dai, Z., Álvarez, M., & Lawrence, N. (2017). Efficient modeling of latent information in supervised learning using Gaussian processes. In I. Guyon et al. (Eds.), *Proceedings of the 31st international conference on neural information processing systems* (pp. 5137–5145).

Dong, X., & Yang, Y. (2020). *Nas-bench-201: Extending the scope of reproducible neural architecture search.* arXiv:2001.00326 [cs.CV]. (Online; accessed 30 April 2021)

Dowle, M., & Srinivasan, A. (2021). data.table: Extension of 'data.frame' [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=data.table (R package version 1.14.0)

Elsken, T., Metzen, J. H., & Hutter, F. (2019a). Efficient multi-objective neural architecture search via Lamarckian evolution. In *Proceedings of the international conference on learning representations.*

Elsken, T., Metzen, J. H., & Hutter, F. (2019b). Neural architecture search: A survey. *Journal of Machine Learning Research*, *20*(55), 1–21.

Floreano, D., & Mattiussi, C. (2008). *Bio-inspired artificial intelligence: Theories,*

methods, and technologies. Cambridge: MIT press.

Gaier, A., Asteroth, A., & Mouret, J.-B. (2018). Data-efficient design exploration through surrogate-assisted illumination. *Evolutionary Computation*, *26*(3), 381–410.

Garrido-Merchán, E. C., & Hernández-Lobato, D. (2020). Dealing with categorical and integer-valued variables in Bayesian optimization with Gaussian processes. *Neurocomputing*, *380*, 20–35.

Hansen, N. (2016). *The CMA evolution strategy: A tutorial.* arXiv:1604.00772v1 [cs.LG]. (Online; accessed 07 April 2021)

Hennig, P., & Schuler, C. J. (2012). Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, *13*(6).

Hernández-Lobato, J. M., Hoffman, M. W., & Ghahramani, Z. (2014). Predictive entropy search for efficient global optimization of black-box functions. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, & K. Q. Weinberger (Eds.), *Proceedings of the 27th international conference on neural information processing systems* (pp. 918–926).

Horn, D., & Bischl, B. (2016). Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In A. Likas (Ed.), *2016 IEEE symposium series on computational intelligence (SSCI)* (pp. 1–8).

Hutter, F., Hoos, H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In C. Coello (Ed.), *Proceedings of the fifth international conference on learning and intelligent optimization* (pp. 507–523).

Hutter, F., Kotthoff, L., & Vanschoren, J. (Eds.). (2018). *Automated machine learning: Methods, systems, challenges*. New York: Springer. (In press, available at http://automl.org/book)

Igel, C. (2005). Multi-objective model selection for support vector machines. In C. Coello, A. Aguirre, & E. Zitzler (Eds.), *Evolutionary multi-criterion optimization* (pp. 534–546).

Jenatton, R., Archambeau, C., González, J., & Seeger, M. (2017). Bayesian optimization with tree-structured dependencies. In D. Precup & Y. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (pp. 1655–1664).

Jin, X., Wang, J., Slocum, J., Yang, M. H., Dai, S., Yan, S., & Feng, J. (2019). *RC-DARTS: resource constrained differentiable architecture search.* arXiv:1912.12814 [cs.CV]. (Online; accessed 30 April 2021)

Johnson, S. G. (2021). *The NLopt nonlinear-optimization package.* http://github.com/stevengj/nlopt.

Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, *13*(4), 455–492.

Kassambara, A. (2020). ggpubr: 'ggplot2' based publication ready plots [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=ggpubr (R package version 0.4.0)

Kent, P., & Branke, J. (2020). *BOP-Elites, a Bayesian optimisation algorithm for quality-diversity search.* arXiv:2005.04320 [math.OC]. (Online; accessed 07 April 2021)

Konak, A., Coit, D. W., & Smith, A. E. (2006). Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, *91*(9), 992–1007.

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images* (Tech. Rep.). University of Toronto.

Lang, M., Binder, M., Richter, J., Schratz, P., Pfisterer, F., Coors, S., ... Bischl, B. (2019, dec). mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*.

Lang, M., & Schratz, P. (2021). mlr3misc: Helper functions for 'mlr3' [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=mlr3misc (R package version 0.7.0)

Lehman, J., & Stanley, K. O. (2011a). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, *19*(2), 189–223.

Lehman, J., & Stanley, K. O. (2011b). Evolving a diversity of virtual creatures through novelty search and local competition. In N. Krasnogor (Ed.), *Proceedings of the 13th annual conference on genetic and evolutionary computation* (pp. 211–218).

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Marben, J., Müller, P., & Hutter, F. (2019). *BOAH: A tool suite for multi-fidelity Bayesian optimization & analysis of hyperparameters.* arXiv:1908.06756 [cs.LG]. (Online; accessed 04 May 2021)

Lindauer, M., & Hutter, F. (2019). *Best practices for scientific research on neural architecture search.* arXiv:1909.02453 [cs.LG]. (Online; accessed 10 May 2021)

Liu, H., Simonyan, K., & Yang, Y. (2019). DARTS: Differentiable architecture search. In *Proceedings of the international conference on learning representations.*

Maclaurin, D., Duvenaud, D., & Adams, R. (2015). Gradient-based hyperparameter optimization through reversible learning. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning* (pp. 2113–2122).

Močkus, J. (1975). On Bayesian methods for seeking the extremum. In *Optimization*

techniques IFIP technical conference (pp. 400–404).

Moreno-Muñoz, P., Artés, A., & Álvarez, M. (2018). Heterogeneous multi-output Gaussian process prediction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Proceedings of the 32nd international conference on neural information processing systems* (p. 6712—6721).

Mouret, J.-B., & Clune, J. (2015). *Illuminating search spaces by mapping elites.* arXiv:1504.04909 [cs.AI]. (Online; accessed 07 April 2021)

Mouret, J.-B., & Maguire, G. (2020). Quality diversity for multi-task optimization. In *Proceedings of the 2020 annual conference on genetic and evolutionary computation* (pp. 121–129).

Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, *7*(4), 308–313.

Nguyen, A. M., Yosinski, J., & Clune, J. (2015). Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation* (pp. 959–966).

Nocedal, J., & Wright, S. (2006). *Numerical optimization.* New York: Springer.

Parker-Holder, J., Pacchiano, A., Choromanski, K., & Roberts, S. (2020). *Effective diversity in population based reinforcement learning.* arXiv:2002.00632 [cs.LG]. (Online; accessed 07 May 2021)

Pham, H., Guan, M., Zoph, B., Le, Q., & Dean, J. (2018). Efficient neural architecture search via parameters sharing. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (pp. 4095–4104).

R Core Team. (2020). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from https://www.R-project.org/

Rasmussen, C. E., & Williams, C. (2006). *Gaussian processes for machine learning.* Cambridge: MIT Press.

Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI conference on artificial intelligence* (pp. 4780–4789).

Richter, J., Becker, M., Lang, M., Bischl, B., Binder, M., & Moosbauer, J. (2021). mlr3mbo: Flexible Bayesian optimization in R [Computer software manual]. (https://mlr3mbo.mlr-org.com, https://github.com/mlr-org/mlr3mbo)

Roustant, O., Ginsbourger, D., & Deville, Y. (2012). DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling

and optimization. *Journal of Statistical Software*, *51*(1), 1–55.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & de Freitas, N. (2016). Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, *104*(1), 148-175.

Siems, J., Zimmer, L., Zela, A., Lukasik, J., Keuper, M., & Hutter, F. (2020). *NAS-Bench-301 and the case for surrogate benchmarks for neural architecture search.* arXiv:2008.09777 [cs.LG]. (Online; accessed 30 April 2021)

Sonabend, R., & Schratz, P. (2020). mlr3extralearners: Extra learners for mlr3 [Computer software manual]. (R package version 0.1.1)

Springenberg, J. T., Klein, A., Falkner, S., & Hutter, F. (2016). Bayesian optimization with robust Bayesian neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Proceedings of the 30th international conference on neural information processing systems* (pp. 4141–4149).

Stanley, K. O., Clune, J., Lehman, J., & Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, *1*(1), 24–35.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, *10*(2), 99–127.

Surjanovic, S., & Bingham, D. (2013). *Virtual library of simulation experiments: Test functions and datasets.* https://www.sfu.ca/~ssurjano/shekel.html. (Online; accessed 05 May 2021)

Therneau, T., & Atkinson, B. (2019). rpart: Recursive partitioning and regression trees [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=rpart (R package version 4.1-15)

Urquhart, N., & Hart, E. (2018). Optimisation and illumination of a real-world workforce scheduling and routing application (WSRP) via Map-Elites. In A. Auger, C. M. Fonseca, N. Lourenço, M. P., L. Paquete, & D. Whitley (Eds.), *International conference on parallel problem solving from nature* (pp. 488–499).

Ushey, K., Allaire, J., & Tang, Y. (2020). reticulate: Interface to 'python' [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=reticulate (R package version 1.18)

Van Rossum, G., & Drake Jr, F. L. (1995). *Python reference manual*. Amsterdam: Centrum voor Wiskunde en Informatica Amsterdam.

Wang, Z., & Jegelka, S. (2017). Max-value entropy search for efficient Bayesian optimization. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (pp. 3627–3635).

White, C., Neiswanger, W., Nolen, S., & Savani, Y. (2020). A study on encodings

for neural architecture search. In *Proceedings of the 34th conference on neural information processing systems.*

White, C., Neiswanger, W., & Savani, Y. (2019). *BANANAS: Bayesian optimization with neural architectures for neural architecture search.* arXiv:1910.11858 [cs.LG]. (Online; accessed 21 April 2021)

White, C., Nolen, S., & Savani, Y. (2020). Local search is state of the art for neural architecture search benchmarks. In *ICML workshop on automatic machine learning.*

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis.* New York: Springer. Retrieved from https://ggplot2.tidyverse.org

Wilson, J., Hutter, F., & Deisenroth, M. (2018). Maximizing acquisition functions for Bayesian optimization. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, & C.-B. N. (Eds.), *Proceedings of the 32nd international conference on neural information processing systems* (pp. 9906–9917).

Wright, M. N., & Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, *77*(1), 1–17.

Xiong, Y., Mehta, R., & Singh, V. (2019). *Resource constrained neural network architecture search: Will a submodularity assumption help?* arXiv:1904.03786 [cs.CV]. (Online; accessed 30 April 2021)

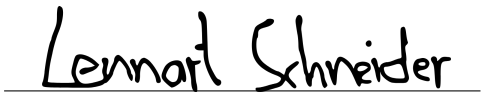
Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., & Hutter, F. (2019). NAS-Bench-101: Towards reproducible neural architecture search. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (pp. 7105–7114).

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8697–8710).

Declaration of Authorship

I hereby declare that I wrote this master's thesis by myself without auxiliary means or sources other than those indicated. All parts taken from published and unpublished scripts are indicated as such. This master's thesis has not been previously presented as an examination paper in this or any other form.

Munich, May 20th, 2021,



(Lennart Schneider)