Behavior-based Neuroevolutionary Training in Reinforcement Learning

Jörg Stork
TH Köln
Cologne, Germany
joergs.stork@th-koeln.de

Martin Zaefferer
TH Köln
Cologne, Germany
martin.zaefferer@th-koeln.de

Nils Eisler
TH Köln
Cologne, Germany
nils.eisler@th-koeln.de

Patrick Tichelmann
TH Köln
Cologne, Germany
patrick.tichelmann@th-koeln.de

Thomas Bartz-Beielstein
TH Köln
Cologne, Germany
thomas.bartz-beielstein@th-koeln.de

A. E. Eiben
Vrije Universitat Amsterdam
Amsterdam, Netherlands
a.e.eiben@vu.nl

ABSTRACT

In addition to their undisputed success in solving classical optimization problems, neuroevolutionary and population-based algorithms have become an alternative to standard reinforcement learning methods. However, evolutionary methods often lack the sample efficiency of standard value-based methods that leverage gathered state and value experience. If reinforcement learning for real-world problems with significant resource cost is considered, sample efficiency is essential. The enhancement of evolutionary algorithms with experience exploiting methods is thus desired and promises valuable insights. This work presents a hybrid algorithm that combines topology-changing neuroevolutionary optimization with value-based reinforcement learning. We illustrate how the behavior of policies can be used to create distance and loss functions, which benefit from stored experiences and calculated state values. They allow us to model behavior and perform a directed search in the behavior space by gradient-free evolutionary algorithms and surrogate-based optimization. For this purpose, we consolidate different methods to generate and optimize agent policies, creating a diverse population. We exemplify the performance of our algorithm on standard benchmarks and a purpose-built real-world problem. Our results indicate that combining methods can enhance the sample efficiency and learning speed for evolutionary approaches.

CCS CONCEPTS

• **Theory of computation** → **Evolutionary algorithms**; • **Computing methodologies** → **Neural networks**; **Reinforcement learning**.

KEYWORDS

Neuroevolution, Reinforcement Learning, Neural Networks, Evolutionary Algorithms, Surrogate Optimization

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463171

ACM Reference Format:
Jörg Stork, Martin Zaefferer, Nils Eisler, Patrick Tichelmann, Thomas Bartz-Beielstein, and A. E. Eiben. 2021. Behavior-based Neuroevolutionary Training in Reinforcement Learning. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449726.3463171

1 INTRODUCTION

In the last years, neuroevolution (NE) and population-based methods have shown to be a valuable and scalable alternative to classic approaches in reinforcement learning (RL), as they have shown promising performance on several problems [11, 13, 15, 23]. In particular, they are computationally efficient if the problem itself is fast to evaluate and a large number of parallel evaluations are possible. Evolutionary algorithms (EAs) applied to RL often rely on an episode-to-episode fitness evaluation, considering an RL episode's final cumulative reward for selection and updating. Further, they do not take advantage of the details of individual behavioral interactions and do not leverage from the gathered experiences of state information. Thus, they often require a significant amount of fitness evaluations to evolve well-performing agents. In particular, if artificial neural networks (ANNs) with changing topologies are considered, which employ a large search space [28]. The resulting low sample efficiency is a challenge in real-world problems due to their high costs, as each action can have a considerable duration.

This paper's primary goal is to combine a topology-changing neuroevolutionary algorithm with behavior-based search components to improve the sample efficiency.

Standard value-based and actor-critic policy gradient methods exhaust behavioral information for their updates and are remarkably successful in many different domains [30]. The combination of evolutionary methods with value-based methods has recently become an active area of research. Recent methods include hybrid approaches, which collect experiences by an evolutionary part, then apply value-based learning, such as policy-gradients, to selected population members [14, 15]. The use of experience-based mutation operators, e.g., which perform gradient updates, is also promising to improve evolutionary methods [6].

Another approach for improving the sample efficiency is to replace the actual problem function with a surrogate, as employed in surrogate model-based optimization (SMBO). The implementation

of surrogates in neuroevolutionary algorithms for reinforcement learning has been of increased interest [8, 28]. The difficulty in modeling agents is the definition of an adequate distance between their policies. In the case of changing topologies, the definition of genotypic differences (i.e., differences of the encoding) is algorithm-dependent and not necessarily helpful [8, 27]. One method is to utilize behavior-based distances, in this context defined as the actions of an agent taken in pre-defined states. Behavioral embeddings are also used to maintain diversity in a population, which is essential to avoid overfitting to certain behaviors [21].

In this paper, we seize these ideas and implement a hybrid framework for *behavior-based neuroevolutionary training* (BNET), which combines the evolution of a population of agents with topology-changing neuroevolution, employed by *cartesian genetic programming* (CGP) [16, 31]. Our contributions are as follows:

(1) We introduce the new algorithm where the population's agents are optimized in parallel by fitness-based search, behavior-based search, and surrogate-based search. As a critical challenge in RL is exploration, our implementation is focused on maintaining a diverse set of agents, each contributing to the fitness search and sharing experiences. Different approaches to leverage from a shared experience set exist [25]; we employ a selection method to create an archive of experiences from high-performing yet diverse policies.

(2) We define a set of new advantage-weighted behavior loss functions to conduct the behavior-based search, leveraging on principles from standard actor-critic policy algorithms and behavior-based distances [27, 30]. In combination, they are part of a gradient-free learning process for the agent policies. For the surrogate-based search, we employ the *surrogate model-based optimization for neuroevolution* (SMB-NE) algorithm, first introduced in [28].

(3) We define a method for robust candidate selection to increase learning stability. Our fitness-based search focuses on keeping a stable elitist solution, referred to as champion, which is of central importance considering real-world environments. As these environments frequently provide stochastic rewards, e.g., caused by random starting conditions, we employ a robust selection method, which compares mean estimates of their actual performance and reduces the probability of choosing an inferior candidate.

(4) We implement a prototype of the framework and test it on common synthetic benchmark problems against standard value-based RL methods to prove our concept. We also analyze the performance of each search method in BNET. Finally, we introduce a new adaptable real-world problem featuring a robot-controlled maze environment as a benchmark for RL algorithms.

2 METHODS

2.1 Reinforcement Learning and Value-based Methods

In RL, we train agents' policies $\pi_n, n = 1, ..., N$ to solve instances of an environment: for each environment step t the agent observes an environment state s_t and the policy decides, which action a_t is conducted. The agent receives a reward $r(s_t, a_t, s_{t+1})$ for each observed state during its learning episode. This leads to a *trajectory* with T steps. The cumulative reward $R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$ at the end of an episode, discounted by factor γ, is typically utilized as a target function of the policy optimization process. We refer to R_t of a full episode as *fitness* of a policy.

Advantage-based actor-critic methods [30] have a policy actor and estimate the *advantage* of an action a_t in state s_t by a critic. The critic $V_\varphi(s_t)$, with φ being its parameters, approximates the estimated *value* of a state s_t, given by the value function $V(s) = \mathbb{E}_\pi\{R_t | s_t = s\}$. For a full-episode learner, i.e., considering the complete per-state reward information R_t at the end of an RL episode, a typical approach is to compute the advantage by a Monte-Carlo (MC) estimate $A_\varphi(s_t, a_t) = R_t(s_t, a_t) - V_\varphi(s_t)$, where $V_\varphi(s_t)$ is approximated by the critic. Further, the actor is updated with help of a gradient function $\nabla_\theta J(\theta) = \sum_t \nabla_\theta log \pi_\theta(s_t, a_t) A_\varphi(s_t, a_t)$ Here θ are the ANN parameters, and π_θ is the policy associated with these parameters. We refer to the *experience* of an agent, considering the (s_t, a_t, r_{t+1}) triplets an agent observes during (multiple) interaction episodes in a RL environment.

2.2 Neuroevolution by Cartesian Genetic Programming

CGP [16] is a genetic programming method relying on grid-based encodings to realize graph representations. If applied to ANNs [31], it allows to create topologies with different transfer functions and free node-to-node connections, i.e., the generated topologies do not follow the typical layered structure. CGP, in its basic version, does not allow the direct use of back-propagation or gradients for ANN optimization, mainly due to the topology-changing neuroevolution process. Certain CGP implementations allow the utilization of gradient information [10]. However, these learn topologies and weights sequentially, not simultaneously. The genotype-behavior mapping is complicated as distances on the genotype-level are barely related to their distance in behavior space [27]. We optimize the CGP-ANNs with a gradient-free $(\mu + \lambda)$-EA with rank-based fitness selection [4] for the optimization of neuroevolution candidates, utilizing behavior-based loss functions, as described in Section 2.3.

2.3 Optimization in the Behavior Space

Each policy computes the probability $p(a_t|s_t)$ of taking an action a_t for an input state s_t. Formally, we denote *behavior* as the set of probabilities corresponding to K states $\mathrm{S} = \{s_1, ..., s_K\}$. Further, $\mathcal{B}$ is denoted as the *behavior space*, i.e., the set of all possible behaviors, with $\pi_n \in \mathcal{B}$ and $n = 1, ..., N$. In the behavior space, two agents can be directly compared on the state set S by calculating their mean *behavior distance* [27, 28], denoted by

$$\mathrm{d}_b(\pi, \pi', \mathrm{S}) = \frac{1}{T} \sum_{t=1}^{T} |\pi(\mathbf{s}_t) - \pi'(\mathbf{s}_t)| \tag{1}$$

For the case of experience-based policy optimization, we defined the *advantage-weighted behavior distance*:

$$\mathrm{d}_{wb}(\pi, \pi', \mathrm{S}, \mathrm{A}) = \left(\frac{\frac{1}{T^*} \sum_{t=1}^{T^*} |\pi(s_t) - \pi'(s_t)| \times A_\varphi(s_t, a_t)}{\frac{1}{T^*} \sum_{t=1}^{T^*} |A_\varphi(s_t, a_t)|} \right) \tag{2}$$

Here, S are sampled states from environment interactions with a trajectory of T^* time-steps, while A are the respective actions taken during this trajectory. Further, A_φ is the precomputed approximated advantage for each of the stored actions, see sec. 2.1. As a second

method, we utilize the *advantage-weighted cross-entropy*. The cross-entropy is defined by:

$$H(\pi, \pi', s_t) = -\sum_{i=1}^{I} \left[\pi_i(s_t) \ln\left(\pi'_i(s_t)\right)\right] \tag{3}$$

$\pi_i(s_t)$ is the i-th element of the probability output of the policy π given the input state s_t. The cross-entropy distance (d_c) is further weighted by the advantage estimation $A^+_\varphi(a_t, s_t)$ and computed over a set of stored states $\mathbf{S}$ and actions $\mathbf{A}$:

$$d_c(\pi, \pi', \mathbf{S}, \mathbf{A}) = \frac{1}{T^*} \sum_{t=1}^{T^*} \left[H(\pi, \pi', s_t) \times A^+_\varphi(s_t, a_t)\right] \tag{4}$$

Instead of the complete estimated advantage, we only consider the set of positive advantages:

$$A^+_\varphi(s_t, a_t) = \begin{cases} A_\varphi(a_t, s_t), & \text{if } A_\varphi(s_t, a_t) \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

The positive advantages drive the search towards estimated beneficial behavior, while negative advantages will not affect the weighted cross-entropy distance d_c. We define our loss function for NE-EA during the behavior-based optimization as the sum of distances (d_{wb} or d_c) between the target policy π and several stored reference behaviors π_m with $m = 1, ..., M$. Furthermore, their sampled trajectories with states $\mathbf{S}_m$ and actions $\mathbf{A}_m$ are:

$$L(\pi) = \sum_{m=1}^{M} \left[\, \mathrm{d}(\pi, \pi_m, \mathbf{S}_m, \mathbf{A}_m)\right]. \tag{6}$$

For the loss functions, the probability distribution of the stored behaviors is priorly adapted to maximize the performed action's probability. The optimization in the behavior space $\mathcal{B}$ is similar to imitation learning, i.e., fitting a network to replicate a stored behavior. However, instead of replicating a single reference behavior, we optimize the target policy to minimize the distance to an advantage-weighted set of multiple reference behaviors from different policies. The EA for the optimization is outlined in algorithm 2.1.

Algorithm 2.1: Neuroevolution EA

INPUT: Memory of M stored reference policies π^*_m, states and actions $\mathbf{S}^*_m, \mathbf{A}^*_m$;
optional: pre-defined candidates π
preset: *mutation rate, NE candidate parameters*
begin
  initialize new polices as candidates
  evaluate initial candidates with loss function $L(\pi)$
  select μ parents from initial candidates
  while *not* *termination-condition* **do**
    mutate parents to get λ offspring
    evaluate offspring with loss function $L(\pi)$
    select μ next iteration's parents with minimum loss from parents and offspring
    optional: *update mutation rate*
  end
end
OUTPUT: best found policies

2.4 Optimization by Behavior Surrogates

The definition of the behavior distance between policies also allows us to create approximation models, so-called surrogates, which predict a policy's fitness during optimization. The surrogates allow searching in the behavior space without any additional environment interactions, which are substituted by the surrogate. Surrogate model-based optimization (SMBO) is frequently applied for costly processes with high resource-demand for each function evaluation [5]. Our SMBO is based on the surrogate model-based optimization for the neuroevolution (SMB-NE) algorithm [28], which utilizes a Kriging [5] regression model. The model measures the similarity of samples by a kernel, utilizing distance and correlation matrices of observations. If applied to real-valued samples, the exponential kernel $\mathrm{k}(x, x') = \exp(-\theta||x - x'||_2)$ is a typical choice. The essential kernel parameter θ influences how fast the correlation decays to zero if the Euclidean distance between two samples $||x - x'||_2$ increases. In our work, we follow the idea of kernel-based models for combinatorial search spaces [20, 35]. We replace the Euclidean distance $||x - x'||_2$ by the *behavior distance*, resulting into the kernel:

$$\mathrm{k}(\pi, \pi', \mathrm{S}) = \exp(-\theta\, \mathrm{d}(\pi(\mathrm{S}), \pi'(\mathrm{S}))) \tag{7}$$

Here, one challenge is the appropriate definition of the state set S, as it has a considerable influence on the distance. We rely on a selection approach presented in [29], where both policies' stored states are combined for each pairwise distance calculation. If a new target network's fitness without stored states needs to be predicted, the reference policies' stored states are applied as reference input. The Kriging model parameters are fitted by maximum likelihood estimation and optimized with DIRECT-L [7], further utilizing the *nugget effect* [32]. Kriging combines relatively accurate mean predictions with the ability to provide uncertainty estimates of each prediction. The combination of mean and uncertainty are used to compute infill functions, which predict the desirability of a solution. A frequently applied infill-criterion is Expected Improvement (EI) [12, 19], which integrates the uncertainty estimate to explore new solutions, which may be farther away from observed solutions. In [22], the benefits and downsides of using EI for different optimization problems are discussed. For high-dimensional cases, it is recommended to use the predicted mean instead of EI since the increase in dimensionality leads to an inherent increase in uncertainty (see also: [33]). In the case of neuroevolution, the behavior space is high-dimensional. Thus, we use the predicted mean of the Kriging surrogate as an infill function. The R package `CEGO` [34, 35] is used to train the Kriging surrogates, while the NE-EA is utilized to optimize the target network.

3 BEHAVIOR-BASED NEUROEVOLUTIONARY TRAINING

In this section, we introduce our new algorithm for population-based neuroevolution for the training of RL policies (BNET). The population consists of CGP-ANNs, which are utilized as RL policies π. We refer to each network instance as a candidate. BNET is similar to a population-based evolutionary algorithm with elitism. Nevertheless, new candidates are not merely created employing a variation of previous candidates. Instead, new candidates are also

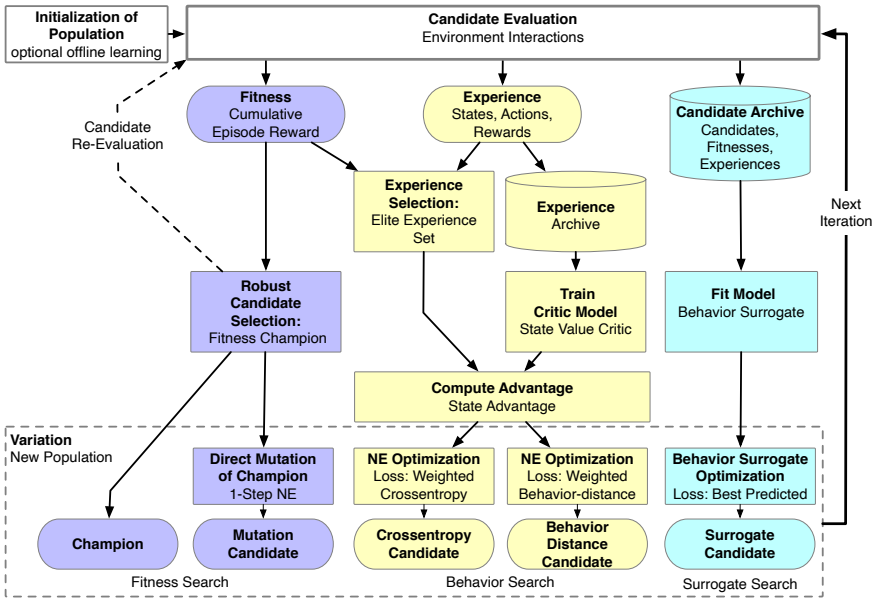

Figure 1: BNET algorithm cycle. Quadratic boxes are methods or algorithms, and rounded edges illustrate observed data. The candidate generation methods are based on extracted data from the environment interactions during the fitness evaluation: The fitness-based search selects a champion with the best overall fitness (violet). The behavior-based search utilizes a selected set of experiences, and advantage-critic model, and the defined behavior-distance and cross-entropy loss-functions to optimize networks and use as new candidates (yellow). Further, a surrogate model is fitted to all candidates' archives and generates a candidate in a surrogate-model-based search (light-blue).

generated by behavior learning and by exploiting behavior-based surrogates. The algorithm outlined in Fig. 1 is essentially divided into three parts, which employ different ways to search for new candidates: fitness-based, behavior-based, and surrogate-based. In summary, the fitness-based search selects a champion with the best overall fitness by robust selection and creates direct mutations of this champion. The behavior-based search utilizes a selected set of experiences, a value-based critic model, and the loss function from Eq. 6 to generate new candidates. Further, a surrogate model is fitted to an archive of all candidates and searched for new candidates.

3.1 Initialization and Evaluation

The first population is either initialized by NE or by optimizing a population to fit previously evaluated policies' behavior. We refer to the second method as *offline* initialization, as it does not require any new (online) environment interactions. If we initialize NE policies offline, the behavior search (as explained below) is applied, where a previously-stored experience set is required as a reference.

The candidates are then evaluated in the RL environment with two goals: First, evaluate the current population's policies to acquire their fitness; Second, to gather new environment experiences from these policies. The policies can be evaluated either fully deterministic, stochastic, or stochastic with exploration. The deterministic policy evaluation chooses the action with the highest probability, whereas the stochastic evaluation samples actions based on the probabilities. Deterministic policies have equal behavior if evaluated repeatedly, but the achieved fitness still may differ, e.g., if the environment itself is stochastic. Deterministic policies are most suitable for exploitation or testing. As stochastic policies depend on probabilities, they allow for a certain level of exploration. Additional exploration can be enforced by taking random actions or adding a random factor to any policy behavior.

3.2 Fitness Search and Robust Selection

The fitness search, visualized as the leftmost, violet path in Fig. 1, focuses on selecting a *fitness champion*, i.e., the candidate of the population with the best overall fitness. Moreover, a mutation candidate is generated by applying a small direct mutation to the champion's policy network and used as a second new candidate, the *mutated champion*. However, RL environments, such as control tasks, typically have different starting conditions or even stochastic state transitions, i.e., an action in a specific state may transition to different states in the next time step. Thus, the measured fitness will be noisy, and even a deterministic policy produces different results when evaluated repeatedly.

Therefore, we employ a robust fitness selection, including the re-evaluation of candidates and comparing their fitness distributions. The robust selection shall ensure that the champion is a reasonable estimate of the best policy discovered so far, even in the presence of noisy fitness. In the first iteration, the fitness champion, denoted as π^*, is selected as the candidate with the highest sampled fitness so far and re-evaluated in the second iteration. Its fitness is then set to the mean of all evaluations. Starting with the second and all future iterations, we use the concept of challengers: A challenger is a candidate π that has a one-time evaluated fitness better than the mean champion fitness, i.e., if $f(\pi) > \frac{1}{n^*} \sum_1^{n^*} f(\pi^*)$, where n^*

is the number of samples of champions performance $f(\pi^*)$. If a new population contains at least one challenger, this challenger is re-evaluated to get a more robust fitness estimate and the mean fitness values are utilized as a measure for selection. A challenger is accepted as champion, if $\frac{1}{n}\sum_1^n f(\pi) > \frac{1}{n^*}\sum_1^{n^*} f(\pi^*)$, where n is a parameter of the selection, either set to match n^* or a desired number of repeats r, by $n = \max(n^*, r)$. The value of r should be chosen according to the task and estimated noise of the environment. If a champion was re-evaluated less than r, it is also re-evaluated.

In case of multiple challengers, they are ordered by their fitness and sequentially evaluated against the champion. If a new champion is selected during the consecutive comparisons, the next challenger is only considered if it is still superior in fitness. The selected champion is then re-evaluated in the next iteration. The constant re-evaluation of the champion, if not changed, leads to a stable estimate of the actual fitness value.

3.3 Behavior Search

The behavior-based optimization of the neuroevolutionary policies builds upon principles from standard RL algorithms, such as *policy-gradients* and *actor critic*. The challenge in neuroevolution is the absence of any gradient information between NE candidates if a simultaneous topology optimization is performed. A neuroevolutionary behavior optimization thus requires metrics that allow the comparison of policies further to establish a search direction in the behavior space. We employ the behavior optimization with the *advantage-weighted behavior distance* from Eq. 2 or the *advantage-weighted cross-entropy* Eq. 4 and implement it in our loss function Eq. 6. This loss application requires a defined set of experiences, an advantage function, and an optimization algorithm. As a reference set, we collect and store a fixed-sized set of *elite experiences*. The set consists of experiences and the performance of single evaluations from different candidates or repeated evaluations of the same candidate. In the first iterations, the set grows until the maximum size of the set is reached. After the robust selection, the experience is replaced with those of higher fitness episodes in each consecutive iteration. However, the number of replacements in each iteration is limited. The limitation prevents that the candidate set gets dominated by the experiences of single candidates. A diverse set is thought to help avoid overfitting and increase the chance to learn better behavior. A diverse experience set is the first difference to classic imitation learning, where a single policy's behavior is adapted. The second difference is given by the advantage weighting of each action in the distances by Eq. 2 and 4. The required advantage function is predicted by a critic model, which is learned to the complete set of all discovered experiences. Also, the advantage-weighted cross-entropy considers only the actions with positive advantage, i.e., the policy is trained to replicate this behavior. Both loss metrics are employed in the NE-EA to generate new candidates.

3.4 Surrogate Search

Corresponding to section 2.4, the surrogate search is based on a Kriging model fitted to an archive of tested candidates with their connected mean fitness and experiences. The distance kernel in Eq. 7 is applied for modeling the relations between policy behaviors and their fitnesses, where the state set S of each comparison consists of the stored experience archive of the associated candidate pair. The fitted surrogate is then employed to predict the fitness of new candidates and utilized as loss-function $L(\pi) = -\hat{f}(\pi)$ during a NE-EA optimization process. The fitness values are adapted to ensure a minimization problem (i.e., negated in the typical reward maximization case).

4 EXPERIMENTS

The BNET framework is flexible, as the algorithm modules for generating candidates can be combined in several ways. For example, it can also be employed as a pure direct neuroevolution approach by refraining from using the behavior-based or surrogate-based search, or in contrast, as a pure behavior search algorithm. Thus, our experiments are two-fold: first, we tested different versions of the algorithm against a set of open-AI baselines algorithms on the problems CartPole-V0 and MountainCar-V0. For this experiment, the focus was to estimate overall performance, how beneficial the different proposed modules are, and how they affect the search quality. In a second experiment, we tested our algorithm against the same baselines on a newly designed real-world problem.

As comparison baselines, we employ *Advantage Actor Critic* (A2C) [17] and *Proximal Policy Optimization* (PPO) [26] from the *stable baselines* package [9]. Both do not require full episodes to learn (i.e., the algorithm is trained after x time steps, not necessarily full episodes), which might give it performance advantages over BNET. Our performance measure, particularly concerning the real-world environment, is the number of required time steps until a (stable) solution is found. For all experiments, we implement a prototype version of the BNET framework using R 3.6.3, an R-interface to the CGP-ANN Library by Turner [28] and *reticulate 1.14*, *Keras 2.3*, OpenAI *gym 0.18.0*, and *tensorflow 1.15.4* [1–3]. All simulated experiments were conducted on an HPC-Cluster. More than 50,000h of total computation time was spent during development and experimentation. The BNET prototype was not systematically tuned for optimal parameter settings due to the high computational effort. The used parameter setup is based on preliminary tests and CGP-ANN or SMB-NE related publications [28, 29]. The baseline algorithms parameter were also improved (from the stable baseline default settings) based on preliminary results for each problem at hand, e.g., the reward discount parameter gamma and the learning rate. One aspect of the BNET setup was kept equal for all tested problems: The CGP-ANNs have a maximum of 200 active nodes with arity ten and a function set including tanh, sigmoid, gaussian, softmax, step, and rectified linear units. The direct, behavior, and surrogate search's mutation rates were 1%,5% and 5%, respectively. The NE-EA uses a (20+2) population with 1000 iterations for the behavior search and (8+2), 500 for the surrogate. The critic network is a fully-connected feed-forward ANN with two layers and 128/64 tanh units, trained for 1000 steps in each iteration. The prototype code and all experimental results are available in an online repository: https://github.com/jstork/BNET-GECCO21.

4.1 Open AI Gym Benchmark Setup

For comparing the performance of BNET against different baselines, we chose two basic Open AI Gym standard benchmarks. They were explicitly chosen because of their different characteristics.

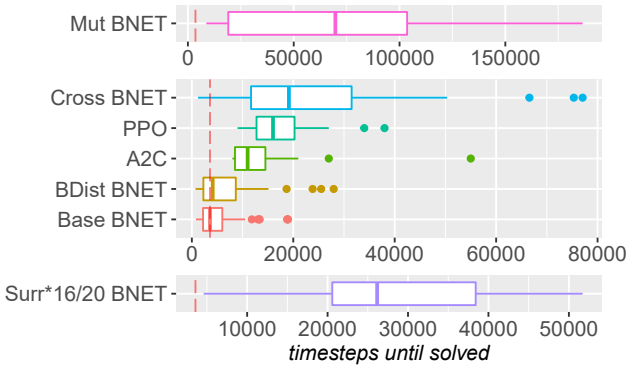

Figure 2: Results of the CartPole-v0 environment. Please mind the different scales for the surrogate and mutation variant. The surrogate variant was only able to finish in 16 out of 20 runs. The best results are achieved by generating all candidates (Base median=3576, red dashed line) or only the behavior distance candidate (BDist with median=4111).

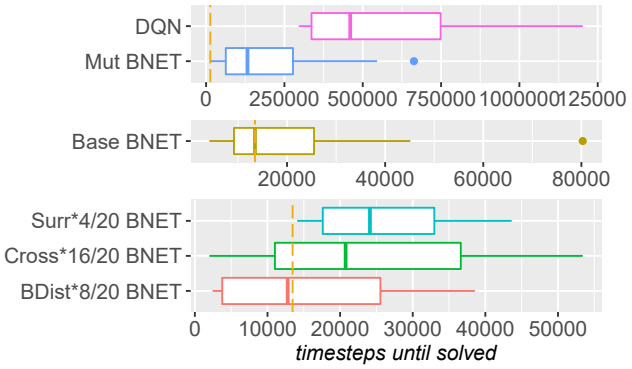

Figure 3: MountainCar-v0 results. Please mind the different scales. The Base variant was able to solve the environment in all cases (median=13457, gold dashed line). The results of the unfinished runs are not comparable. In their case, the attached number of finished runs is meaningful.

Moreover, both should be solvable in less than 20,000 steps. They present a baseline for a real-world scenario, where typically only a low number of steps is possible due to the high resource cost.

Cartpole-v1 is a standard benchmark, where the goal is to balance a pole placed on a cart. The environment has four observable variables (x position, x velocity, pole angle, angular velocity) and two discrete actions (drive left or right). The target is to keep the pole upright in a slight angle range for an average of 195 steps over 100 consecutive episodes. If the pole fails to balance (i.e., the angle reaches a threshold) or the cart drives out of a certain x-range, an episode is stopped. For each step, the agent receives a reward of +1.

In **MountainCar-v0**, a car is located between two mountains and has to drive to the top of one of them. As the direct acceleration is not high enough, it has to build up momentum by alternatively driving up and down the mountains. The environment has two observable variables (x-position, velocity) and three actions (accelerate left/right, do nothing). The target is to drive to a goal position on the proper mountain in less than average 110 time-steps. Each step is rewarded by -1, and the environment is stopped if either the step limit (200) or the goal is reached. The environment requires considerable exploration to find a solution to reach the goal point. If the exploration is unsuccessful, it remains with a -200 reward in each episode and gains no valuable experience. This flat reward landscape renders the environment as difficult to solve in a small number of steps. For both environments, the starting state of the pole or car is randomly set in a small range, leading to different initial scenarios for each episode. Therefore, each setup was repeated at least 20 times with random initial seeds. The run was stopped if a found policy reached the required average target, which was evaluated in an extra function to save unnecessary computation time. The fitness or experiences of these stopping criteria evaluations were not utilized in any other form (e.g., for the algorithm itself).

The first benchmarks include setup variants of BNET, where selected candidates were generated in each iteration. The elitist was always kept and repeated (for the robust evaluation). The setups are: *Base* (all proposed modules are active), *BDist* (behavior distance), *Cross* (cross-entropy), *Surr* (SMBO) and *Mut* (champion mutation). For each variant, the population consists of the champion and a single candidate per active module (e.g., BDist has two candidates per iteration), and the maximum number of episodes was fixed to 1000, except for the *Mut* variant, which served as an additional internal baseline and was run until the environments were solved. In MountainCar-v0, we always kept the mutation candidate in the population. Further, we added additional random exploration (30%) to its policy. Random exploration was also added to the environment evaluations of the initial candidates. All remaining policies are always evaluated deterministically. Each run starts with five initial, random candidates, while the elitist experience set contains a maximum of ten archived episode results.

4.2 OpenAI Benchmark Results

(Cartpole-v0) Fig. 2 illustrates our results from the CartPole-V0 environment. For the plot, each algorithm was repeated 20 times, except for the *Base* and *BDist* variants, which were repeated 50 times (in order to make more minor differences visible). The surrogate-only variant of BNET only succeeded in 16 out of 20 runs to find a solution in 1000 iterations. Overall, the *BDist* variant only generating the behavior-distance optimized candidate (and keeping the robust champion), and the *Base* version using all proposed search methods were the most successful. As expected, the use of only direct mutation performed slower than all other variants. The overall result was surprising for us, as we expected that one variant beside *Base* would be the best performing, as it includes a relatively large sampling overhead by the larger population. However, the algorithm seems to leverage from a diverse set of candidates, and each search variant seems to contribute to the overall algorithm's performance. To test this assumption, we tracked the candidate type with the best fitness (mean over 100 iterations) in each iteration of the BNET *Base* setup. Fig. 4 shows the results. The underlying data is from the 50 repeats of the BNET *Base* runs, with 432 iterations.

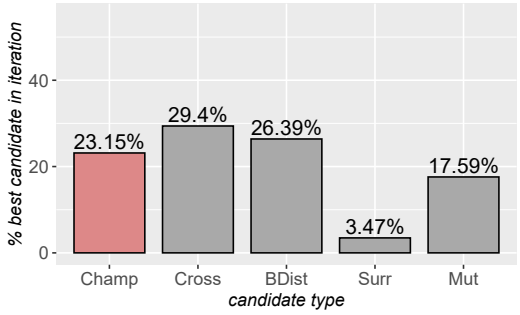

Figure 4: Frequency proportion of each best performing candidate in each iteration of Base BNET for CartPole-v0.

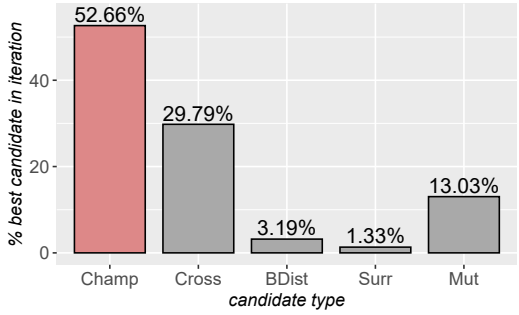

Figure 5: Frequency proportion of each best performing candidate in each iteration of Base BNET for MountainCar-v0.

The plot shows two insights: first, in 77% of the iterations, one of the generated candidates was superior to the stored champion, this implies a reasonable learning rate; second, all candidates of BNET contribute to the performance, where only the surrogate selected candidate performs significantly worse. The surrogate candidate's inferiority could be due to poor parametrization or due to the low number of evaluations, which might not be sufficient to create a proper surrogate model for the complex search space. Interestingly, the direct mutation also generated the best candidates and thus added significantly to the overall performance.

(MountainCar-v0) Due to the challenging nature of this problem, we were not able to find working setups for our baseline algorithms A2C and PPO. All tested parametrization showed no learning effect and got stuck at a reward level of -200, even considering significantly large timestep budgets. We thus tested an additional algorithm, *Deep Q Networks* (DQN) [18] in available setups (double-DQN, dueling-DQN, prioritized experience replay) and were able to solve the environment using prioritized experience replay [24] and high exploration constants. However, DQN still took a vast number of steps to solve the environment and performs even inferior to the BNET mutation variant. Fig. 5 displays the MountainCar-v0 results. As visible, the BNET Base variant is dominating this benchmark and remains the only variant that solves the environment in the 100k step limit. Still, the other algorithms' result is quite interesting, as they tend to either solve the environment in a small number of steps or seem to get completely stuck. As Fig. 5 illustrates, the percentage of successful candidates in the Base variant per iteration is 47%, much less compared to CartPole-v0, with a clear lead of cross-entropy and mutation. The BDist optimization clearly falls behind for this environment, visible in both Fig 5 and 5. We assume this issue is related to the problematic value estimation due to the flat reward landscape of MountainCar-v0. Again, the surrogate search does not perform as desired, which is also caused by the very flat fitness landscape at the beginning, which does not include much valuable information.

4.3 Real-World Robot Maze Setup

Our robot maze problem was explicitly designed to represent a costly real-world demonstrator to test RL algorithms on different setups. We chose a classic maze problem to track and observe an agent's progress and performance efficiently.

The maze consists of a lego brick base plate with 250 x 250 mm, 4x2 black brick walls, and 4x2 white tiles floor, covered by an acrylic glass cover, where a camera is mounted on top. The camera is used to track the position of the red marble in the maze. The system is mounted on a universal robot UR10e 6-axis robotic arm, allowing to move the maze in all directions. The setup is displayed in Fig. 6. The target is to move the marble to the upper left position from the starting point. A central challenge in designing real-world problems is learning without manual user interaction (i.e., resetting a robot position). Our demonstrator allows the automatic resetting by flipping the complete maze and navigating the marble on the glass window to the start. This reset allows episode-to-episode learning and further remote control of the environment without any presence in the lab. The demonstrator is adaptable, i.e., the maze can be redesigned, and the action and observation space can be adapted to discrete or continuous values.

For this work, we restricted the action space to the discrete four cardinal directions and defined designated robot movements, which tilt the maze by an angle of 24,6° and then move back to its base position. Each action takes about 10 seconds and lets the marble roll in a straight line. For the illustrated maze setup, only 23 correct actions are required to reach the target area. The observation space

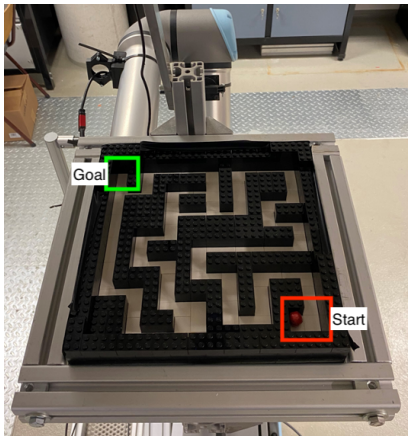

Figure 6: Robot maze test environment mounted on a universal robots UR-10 collaborative robot.

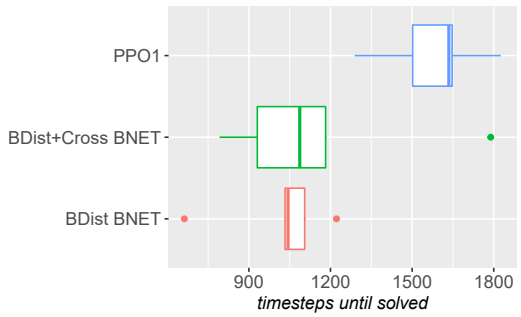

Figure 7: Maze results. Steps until the correct way is learned; The target is consequently reached afterwards. All tested algorithms learn fast. Means (top-down) = 1580, 1156, 1013.

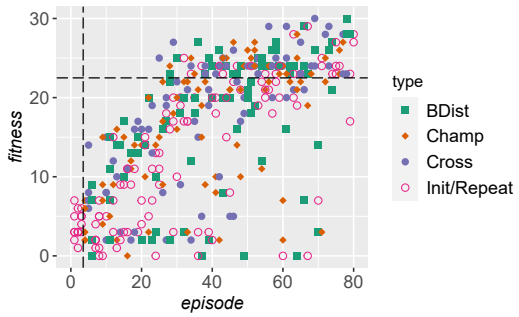

Figure 8: Learning progress of all runs of BDist+Cross BNET on robotmaze. Considered solved at fitness>=23 (horizontal line). BNET shows fast and stable learning process.

is set to a discrete matrix of 15x15, equal to the maze size, where the marble's current position is set to one, else zero. The reward function forces exploration of the maze by rewarding the agent with 0.1 if he drives the marble to a prior unseen position. If run against a wall, it is penalized by -0.75 and by -0.25 if moved to an already discovered position. Reaching the goal is worth +10. The setup is reset if the goal, a number of 75 steps, or a cumulative reward of -12.5 is reached. The robot maze environment should be fast to learn, as, for most positions, only a single action is correct. However, it requires a perfect mixture of exploration and exploitation to find and learn the correct actions sequentially, as the probability of getting stuck is high, and only a small number of steps is possible in each episode. Moreover, the environment fitness is noisy, as sometimes the marble rolls to a difficult position (i.e., edges of a brick), or the camera position detections are incorrect.

Due to the natural time constraints (each run took 8-24h) and several trials to set up the environment, we could not run an extensive experimental design for the benchmark. Thus, we restricted the final results to the most promising ones for this setup: the behavior distance and cross-entropy versions. Each variant was run five times. Regarding the algorithm setup, we set the fitness value in BNET to the number of correctly performed steps. The behavior and cross-entropy distances were weighted by the direct reward instead of a critic and advantage estimation. The purpose-built reward function already delivers correct action value information, and no additional value critic is required. As a baseline, we utilize PPO that showed very stable results in preliminary runs. We set gamma=0.5 due to the apparent correlation between correct actions and rewards and 50 steps per actorbatch for frequent learning.

Fig. 7 illustrates the results. All pre-selected algorithms were quite fast in solving the problem, with the *BDist* and *BDist+Cross* versions performing best. In Fig. 8 we visualize the learning progress of the *BDist+Cross* runs. The algorithm advances quite fast and even learns to take more reward-giving actions (>23), assumingly by taking extra detours in the maze. This behavior is similar for PPO and caused by the definition of the rewarding of previous unvisited positions. However, the underlying NE-EA with the behavior-based losses demonstrates to be capable of fitting the CGP-ANN policies excellently to the collected experience, even given the sizeable neuroevolutionary search space, and without the use of gradients.

5 CONCLUSION AND OUTLOOK

In this work, we investigated methods to combine neuroevolutionary search with standard value-based RL methods. The target was to leverage from gathered experiences and to create a sample-efficient RL algorithm applicable to real-world problems. We evolve CGP-ANNs as our agents' policies and search for the best network topologies and optimal weights simultaneously. We defined a hybrid, population-based algorithm, called BNET, which utilizes different methods to generate new candidates: direct NE-based mutation, behavior optimization using gathered experience, and finally, behavior-surrogate-based learning. We discovered that the behavior-based search significantly supports the performance. Combining all methods in a population with shared experience and fitness pools leads to excellent sample efficiency and extraordinary explorative abilities. Moreover, even elementary direct neuroevolutionary mutation steps can contribute significantly to the overall algorithm's performance. As our real-world experiment demonstrates, the defined behavior-loss functions seem well suited to optimize complex networks with changing topologies if the actions' value is estimated correctly. Furthermore, the robust selection with an adaptive re-evaluation of candidates significantly improves our learning progress' stability, as shown in the experiments with a real-world setup. They prove the ability to learn fast and adapt the CGP-ANN policies by solely relying on a gradient-free evolutionary algorithm for optimization. In future work, we want to tackle several open issues:

Framework: We presented an implementation prototype of BNET. The current version is rather slow due to a single-thread implementation in R. The underlying ideas need to be transferred to a faster and computationally more efficient implementation.

Analysis / Tuning: The algorithm structure and parameters need to be profoundly analyzed, understood, and optimized.

Offline initialization: Real-world problems can benefit considerably from experience from prior runs or human demonstrations. We want to implement and test offline-initialization methods.

Environments: We focused on discrete action spaces. Extending benchmarks to continuous action spaces would be interesting.

Modified Real-World Experiment: Our real-world experiment can be adapted to create more challenging RL problems.

REFERENCES

[1] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[3] François Chollet et al. 2015. Keras. https://keras.io.

[4] Agoston E Eiben, James E Smith, et al. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.

[5] Alexander Forrester, Andras Sobester, and Andy Keane. 2008. *Engineering Design via Surrogate Modelling*. Wiley.

[6] Jörg KH Franke, Gregor Köhler, Noor Awad, and Frank Hutter. 2019. Neural Architecture Evolution in Deep Reinforcement Learning for Continuous Control. *arXiv preprint arXiv:1910.12824* (2019).

[7] Joerg M Gablonsky and Carl T Kelley. 2001. A locally-biased form of the DIRECT algorithm. *Journal of Global Optimization* 21, 1 (2001), 27–37.

[8] Adam Gaier, Alexander Asteroth, and Jean-Baptiste Mouret. 2018. Data-efficient neuroevolution with kernel-based surrogate models. In *Proceedings of the genetic and evolutionary computation conference*. 85–92.

[9] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2018. Stable Baselines. https://github.com/hill-a/stable-baselines.

[10] Dario Izzo, Francesco Biscani, and Alessio Mereta. 2017. Differentiable genetic programming. In *European conference on genetic programming*. Springer, 35–51.

[11] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. 2017. Population based training of neural networks. *arXiv preprint arXiv:1711.09846* (2017).

[12] Donald R. Jones, Matthias Schonlau, and William J. Welch. 1998. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization* 13, 4 (1998), 455–492.

[13] Whiyoung Jung, Giseung Park, and Youngchul Sung. 2020. Population-guided parallel policy search for reinforcement learning. *arXiv preprint arXiv:2001.02907* (2020).

[14] Shauharda Khadka, Somdeb Majumdar, Tarek Nassar, Zach Dwiel, Evren Tumer, Santiago Miret, Yinyin Liu, and Kagan Tumer. 2019. Collaborative evolutionary reinforcement learning. In *International Conference on Machine Learning*. PMLR, 3341–3350.

[15] Shauharda Khadka and Kagan Tumer. 2018. Evolution-guided policy gradient in reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 1196–1208.

[16] Julian F Miller and Peter Thomson. 2000. Cartesian genetic programming. In *European Conference on Genetic Programming*. Springer, 121–132.

[17] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[19] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. 1978. *Towards Global Optimization 2*. North-Holland, Chapter The application of Bayesian methods for seeking the extremum, 117–129.

[20] Alberto Moraglio and Ahmed Kattan. 2011. Geometric Generalisation of Surrogate Model Based Optimisation to Combinatorial Spaces. In *Proceedings of the 11th European Conference on Evolutionary Computation in Combinatorial Optimization* (Torino, Italy) *(EvoCOP'11)*. Springer, 142–154.

[21] Jack Parker-Holder, Aldo Pacchiano, Krzysztof Choromanski, and Stephen Roberts. 2020. Effective diversity in population-based reinforcement learning. *arXiv preprint arXiv:2002.00632* (2020).

[22] Frederik Rehbach, Martin Zaefferer, Boris Naujoks, and Thomas Bartz-Beielstein. 2020. Expected improvement versus predicted value in surrogate-based optimization. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. ACM, 868–876.

[23] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).

[24] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).

[25] Simon Schmitt, Matteo Hessel, and Karen Simonyan. 2020. Off-policy actor-critic with shared experience replay. In *International Conference on Machine Learning*. PMLR, 8545–8554.

[26] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[27] Jörg Stork, Martin Zaefferer, and Thomas Bartz-Beielstein. 2019. Improving NeuroEvolution Efficiency by Surrogate Model-Based Optimization with Phenotypic Distance Kernels. In *Applications of Evolutionary Computation*, Paul Kaufmann and Pedro A. Castillo (Eds.). Springer International Publishing, Cham, 504–519.

[28] Jörg Stork, Martin Zaefferer, Thomas Bartz-Beielstein, and AE Eiben. 2019. Surrogate models for enhancing the efficiency of neuroevolution in reinforcement learning. In *Proceedings of the genetic and evolutionary computation conference*. 934–942.

[29] Jörg Stork, Martin Zaefferer, Thomas Bartz-Beielstein, and AE Eiben. 2020. Understanding the Behavior of Reinforcement Learning Agents. In *International Conference on Bioinspired Methods and Their Applications*. Springer, 148–160.

[30] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[31] Andrew James Turner and Julian Francis Miller. 2015. Introducing a cross platform open source cartesian genetic programming library. *Genetic Programming and Evolvable Machines* 16, 1 (2015), 83–91.

[32] Wim CM Van Beers and Jack PC Kleijnen. 2003. Kriging for interpolation in random simulation. *Journal of the Operational Research Society* 54, 3 (2003), 255–262.

[33] Simon Wessing and Mike Preuss. 2017. The true destination of EGO is multi-local optimization. In *2017 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*. IEEE, 1–6.

[34] Martin Zaefferer. 2017. Combinatorial Efficient Global Optimization in R - CEGO v2.2.0. online: https://cran.r-project.org/package=CEGO. accessed: 2018-01-10.

[35] Martin Zaefferer, Jörg Stork, Martina Friese, Andreas Fischbach, Boris Naujoks, and Thomas Bartz-Beielstein. 2014. Efficient Global Optimization for Combinatorial Problems. In *Proc. GECCO'14* (Vancouver, BC, Canada). ACM, 871–878.