Genetic Programming and Evolvable Machines (2023) 24:17
https://doi.org/10.1007/s10710-023-09462-2

Denoising autoencoder genetic programming: strategies to control exploration and exploitation in search

David Wittenberg[1] · Franz Rothlauf[1] · Christian Gagné[2]

Received: 11 November 2022 / Revised: 15 July 2023 / Accepted: 23 September 2023 /
Published online: 8 November 2023

Abstract

Denoising autoencoder genetic programming (DAE-GP) is a novel neural network-based estimation of distribution genetic programming approach that uses denoising autoencoder long short-term memory networks as a probabilistic model to replace the standard mutation and recombination operators of genetic programming. At each generation, the idea is to capture promising properties of the parent population in a probabilistic model and to use corruption to transfer variations of these properties to the offspring. This work studies the influence of corruption and sampling steps on search. Corruption partially mutates candidate solutions that are used as input to the model, whereas the number of sampling steps defines how often we re-use the output during model sampling as input to the model. We study the generalization of the royal tree problem, the Airfoil problem, and the Pagie-1 problem, and find that both corruption strength and the number of sampling steps influence exploration and exploitation in search and affect performance: exploration increases with stronger corruption and lower number of sampling steps. The results indicate that both corruption and sampling steps are key to the success of the DAE-GP: it permits us to balance the exploration and exploitation behavior in search, resulting in an improved search quality. However, also selection is important for exploration and exploitation and should be chosen wisely.

Keywords Genetic programming · Estimation of distribution algorithms · Probabilistic model-building · Denoising autoencoders

✉ David Wittenberg
wittenberg@uni-mainz.de

Franz Rothlauf
rothlauf@uni-mainz.de

Christian Gagné
christian.gagne@gel.ulaval.ca

[1] Johannes Gutenberg University, Mainz, Rhineland Palatinate, Germany

[2] Laval University, Quebec City, QC, Canada

1 Introduction

Estimation of distribution genetic programming (EDA-GP) algorithms are metaheuristics for variable-length combinatorial optimization problems that sample from a learned probabilistic model, replacing the standard mutation and recombination operators of genetic programming (GP). At each generation, the idea is to first learn the properties of promising candidate solutions of the parent population (*model building*) and then to sample from the model to transfer the learned properties to the offspring (*model sampling*) [1].

Due to the complex representation of candidate solutions in GP, the scale of research on EDA-GP is still limited [1]. This is different from regular estimation of distribution algorithms (EDAs), where many promising approaches have already been presented [2] and candidate solutions use a fixed-length representation, known from genetic algorithms (GAs) [3]. In contrast, EDA-GP usually represents candidate solutions as variable-length parse trees, which makes probabilistic model building more difficult. Research on EDA-GP has therefore moved from simple univariate to more complex multivariate approaches [1].

An example of a recent multivariate EDA-GP is the denoising autoencoder genetic programming (DAE-GP) that uses denoising autoencoder long short-term memory networks (DAE-LSTMs) as a probabilistic model [4]. In comparison to other EDA-GP approaches, it has the advantage that the model does not make assumptions about the relationships between problem variables which allows the DAE-GP to flexibly identify and model relevant properties of the parent population. The DAE-GP captures dependencies between problem variables by first encoding candidate solutions (in prefix notation) to the latent space and then reconstructing the candidate solutions from the latent space. During model building, the DAE-GP minimizes the reconstruction error between the encoded and decoded candidate solutions. During model sampling, candidate solutions are propagated through the trained model to transfer the learned properties to the offspring [4].

Denoising prevents the DAE-GP to learn the simple identity function. By partially corrupting input candidate solutions, a more robust model can be created that ignores spurious parts (noise) but focuses on the main elements (signal) of candidate solutions in the parent population [4]. The stronger the corruption, the stronger the generalization of the model [4, 5]. Previous work on estimation of distribution algorithms (EDA), where candidate solutions have a fixed length of size n, showed that we can control exploration and exploitation through modulation of corruption strength [6]. Exploration increases the diversity of a population by introducing new candidate solutions into search; exploitation reduces diversity by focusing a population of candidate solutions on promising areas of the solution space [7]. Adjusting the corruption strength can therefore help to balance exploration and exploitation leading to a more successful search: we either increase diversity to overcome local optima avoiding premature convergence, or we decrease diversity to exploit promising areas of the solution space [6].

Wittenberg [8] studied how different levels of corruption strength affect the exploration and exploitation behavior of the DAE-GP. The author controls the

level of corruption by applying Levenshtein edit on input candidate solutions. Levenshtein edit is similar to the Levenshtein distance [9] and operates on the string representation of a candidate solution (prefix expression). The idea is to apply insertion (add one node), deletion (remove one node), and substitution (replace one node by another node) on candidate solutions. Controlling the number of edits allows us to accurately adjust corruption strength: the more nodes we edit, the stronger the corruption, and the stronger we force the DAE-GP to focus on general properties of the parent population. The generalization of the royal tree (GRT) problem is an easy problem with high locality, where the objective values of neighboring solutions are correlated. Wittenberg found that weak corruption in the GRT problem leads to a strong exploitation of the solution space, which helps the DAE-GP to make search more efficient and to outperform standard GP. In contrast, strong corruption leads to a strong exploration of new solution spaces, which is not useful for solving the GRT problem. However, it can be useful for problem domains where the fitness landscape is more rugged such that we need to be able to escape from local optima. Thus, balancing between exploration and exploitation is achievable by choosing the right corruption strategy, resulting in an improved search quality. Unfortunately, until now results are limited to the GRT problem [8].

This paper builds upon and extends the work of Wittenberg [8], with two key contributions. First, we study the exploration and exploitation behavior of the DAE-GP not only on the GRT problem but also on Airfoil and Pagie-1, which are standard symbolic regression benchmark problems [10–13]. These two last problems are more difficult and with lower locality than GRT (i.e., the objective values of neighboring solutions are not strongly correlated). Second, we do not only analyze the influence of different levels of corruption strength on search, but also consider the number of sampling steps and their impact on exploration and exploitation. The number of sampling steps defines how often we re-inject each individual in the model (output at step s is the input at step $s + 1$). Probst and Rothlauf argue that a large number of sampling steps results in a strong exploitation of the solution space [6].

We compare the performance of the DAE-GP using Levenshtein edit and different levels of corruption strength and number of sampling steps to standard GP, and analyze their impact on search. The results confirm findings from [6, 8], that is corruption strength and the number of sampling steps both strongly influence the exploration and exploitation behavior of the DAE-GP. Indeed, the lower the corruption and the higher the number of sampling steps, the stronger the exploitation of the solution space. This can be helpful for easy, high-locality problems, such as the GRT problem, where a higher degree of exploitation throughout search helps to make search more efficient and to outperform standard GP. In contrast, when facing more rugged fitness landscapes, such as for the Airfoil problem, a stronger level of exploration is needed. However, when facing very difficult problems, such as the Pagie-1 problem, we find that the DAE-GP easily gets stuck in local optima even when choosing strong corruption and a low number of sampling steps. Here, diversity preserving selection methods could help to avoid premature convergence. The results demonstrate that choosing the right corruption and sampling strategy is key

to the success of the DAE-GP: it allows us to control the level of exploration and exploitation in search, helping us to improve search quality. However, also selection strongly influences exploration and exploitation and should be chosen wisely.

In Sect. 2, we present related work on EDA-GP. We describe DAE-LSTMs in Sect. 3, where we focus on the architecture, the corruption strategy, and on model building and sampling. In Sect. 4, we introduce the experiments and discuss the results. We draw conclusions in Sect. 5.

2 Related work

We can categorize research on EDA-GP into two main research streams [1, 14]: probabilistic prototype tree (PPT) models and grammar-based ones. A PPT is a full tree of arity a where we set the depth of the PPT equal to the maximum tree depth d_{max}. Arity a is computed as the maximum arity found in the GP function set. At each node of the PPT, a multinomial probability distribution is evaluated over the set of allowed functions (internal nodes) and terminals (leaf nodes), which is then used to update the distributions according the candidate solutions that are presented to the model. Salustowicz and Schmidhuber [15] introduced PPTs in 1997 in the first probabilistic EDA-GP model called probabilistic incremental program evolution (PIPE) [15]. Based on PIPE, which relies on univariate probability distributions, EDA-GP models have been developed to capture dependencies between nodes in a PPT tree. Examples include the bivariate estimation of distribution programming (EDP) [16] or the multivariate program optimization with linkage estimation (POLE) [17, 18]. Hasegawa and Iba [18] report that POLE needs less fitness evaluations than standard GP to solve the MAX, the deceptive MAX, and the royal tree problem [18].

The second stream of research uses grammars as EDA-GP model [1, 14]. The first grammar-based approach has been proposed by Ratle and Sebag [19] as the stochastic grammar-based genetic programming (SG-GP) in 2001. SG-GP uses stochastic context-free grammar (SCFG) as a probabilistic model. The idea is to first identify a set of production rules for a problem, with weights attached to them, and to then update the weights according to usage counts of the production rules in a parent population [19]. Since SG-GP assumes the production rules to be independent, more sophisticated EDA-GP models capturing more complex grammars have been developed. Consequently, program with annotated grammar estimation (PAGE) is an extension that uses expectation maximization (EM) or variational Bayes (VB) to learn production rules with latent annotations. An example of latent annotation is the position or the depth of a node in a tree [20]. Another extension is grammar-based genetic programming with a Bayesian network (BGBGP) that was introduced by Wong et al. [21] in 2014. BGBGP uses Bayesian networks with stochastic context-sensitive grammars (SCSG) as a model. Compared to SCFG, SCSG additionally incorporate contextual information allowing the Bayesian network to learn dependencies between production rules [21]. To further refine the BGBGP, Wong et al. [22] added (fitness) class labels to the model. The authors argue that this allows the model to differentiate between good and poor candidate solutions helping the model to find better solutions. For the deceptive MAX and the asymmetric royal

tree problem, the model outperforms POLE, PAGE-EM, PAGE-VB, and grammar-based GP in the number of fitness evaluations [22].

One example of an EDA-GP model that neither relies on PPTs nor grammars is the n-gram GP proposed by Poli and McPhee [23], where n-grams are used to model relationships between a group of n consecutive sequences of instructions that can learn dependencies in linear GP. Similarly, Hemberg et al. [24] suggested Operator-free Genetic Programming (OFGP), which learns n-grams of ancestor node chains. An n-gram of ancestors is the sequence of a node and its $n-1$ ancestor nodes in a GP parse tree. However, for the Pagie-1 problem, OFGP could not outperform standard GP [24]. Recently, Liskowski et al. [25] proposed neural program optimization (NPO), where the idea is to first learn neural embeddings of programs in a latent space, and to then use continuous optimization (CMA-ES) to search for promising programs in that latent space. The authors find that several variants of NPO outperform standard GP on program synthesis tasks [25]. Note, however, that the learned embeddings are neither updated nor re-learned during evolution, thus NPO is not considered an EDA-GP.

Wittenberg et al. [4] recently suggested DAE-GP that uses Denoising Autoencoder Long Short-term Memory networks (DAE-LSTMs) as an EDA-GP model. For the GRT problem, the DAE-GP outperforms standard GP. The DAE-GP can better identify promising areas of the solution space compared to standard GP resulting in a more efficient search in the number of fitness evaluations, especially in large search spaces. The authors argue that, compared to previous EDA-GP approaches, the flexible model representation is the key reason for the high performance, allowing the model to identify in parallel, both position as well as context of relevant substructures [4].

The idea of using DAE as probabilistic models in EDA has earlier been presented by Probst [26] who introduced DAE-EDA. DAE-EDA was designed for problems where candidate solutions follow a fixed-length representation [26]. For the NK landscapes, deceptive traps and HIFF problem, Probst and Rothlauf [6] show that the DAE-EDA yields competitive results compared to the Bayesian optimization algorithm (BOA). However, DAE-EDA is better parallelizable, making it the preferred choice especially in large search spaces. Furthermore, the authors show that corruption strength has a strong impact on exploration and exploitation in search. Adjusting the level of corruption can therefore help to either increase exploration which helps to overcome local optima, or to exploit relevant solution spaces making search more efficient [6].

Wittenberg [8] transferred the idea of analyzing the influence of corruption strength to the variable-length problem domain of GP and studied how corruption strength affects the search behavior of the DAE-GP. For the GRT problem, the author found that corruption strength strongly influences search: the stronger we corrupt input candidate solutions, the stronger the generalization of the model and the stronger exploration of new solution spaces. For the given problem (easy problem with high locality), a DAE-GP with weak corruption (Levenshtein edit and 5% corruption strength) performs best, significantly outperforming standard GP. However, results are limited to the GRT problem [8], which we extend with more results and new analysis in the current paper.

3 Denoising autoencoder LSTMs

Autoencoder long short-term memory networks (AE-LSTM) [27] are artificial neural networks that consist of an encoding and a decoding LSTM: the encoding LSTM encodes a candidate solution (a linear sequence in prefix expression) to the latent space; the decoding LSTM decodes the latent space (i.e., the latent vector) back to a candidate solution. DAE-LSTM are a variant of AE-LSTM where corruption is applied to input candidate solutions to prevent the model from learning the simple identity function. DAE-LSTMs are used as a probabilistic model in EDA-GP (DAE-GP) by the repeated application of the following two steps at each generation: train the model for learning relevant properties of our parent population (model building) and propagate candidate solutions through the trained DAE-LSTM to transfer the learned properties to the offspring (model sampling).

In the following sections, we first explain the architecture of AE-LSTMs and the concept of corruption, where we focus on Levenshtein edit as corruption strategy. Then, we describe the training as well as the sampling procedure, where significant differences compared to Wittenberg [8] lie in a new training procedure that increases the quality of our model and the use of several sampling steps for producing the offspring.

3.1 Autoencoder LSTMs

Figure 1 shows the architecture of an AE-LSTM with one input layer, one hidden layer (consisting of LSTM memory cells), and one output layer. It is based on the architecture presented in [4]. x and o represent the input and output candidate solutions of length m and k (sequences in prefix expression), respectively. h is the hidden state at time step (iteration) t, where the total number of time steps (total number of iterations) corresponds to $T = m + k$ ($m, k \in \mathbb{N}$). The encoding LSTM (left) first sequentially encodes a candidate solution x, with x_t, $t \in \{1, 2, \ldots, m\}$ through the

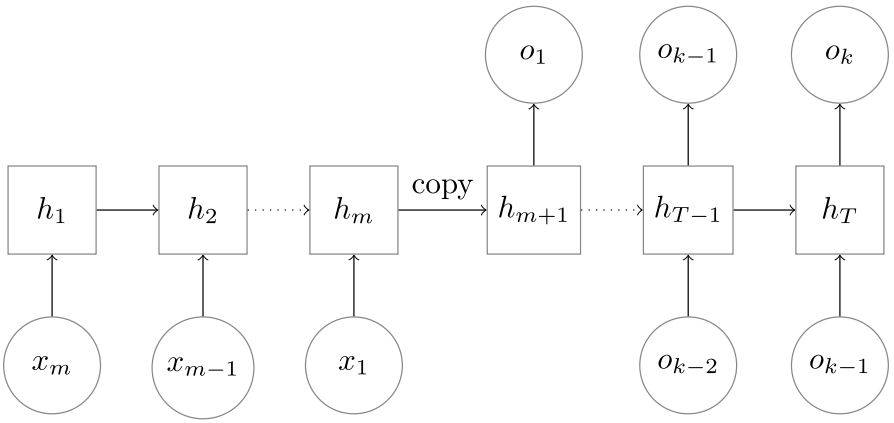


Fig. 1 Autoencoder LSTM with one hidden layer

encoding function $g(x)$, where each x_t represents a function or terminal of a candidate solution in our parent population. At each time step t (except $t = 0$), the LSTM memory cell then receives three inputs: the current input x_t, the previous hidden state h_{t-1} and the previous cell state c_{t-1} (not shown here). The idea of transferring information from one time step to the next is to capture long-term dependencies in training data [28]. After complete processing of the input candidate solution x, we copy h_m and c_m, and transfer it to the decoding LSTM, thus $h_{m+1} = h_m$ and $c_{m+1} = c_m$. The decoding LSTM (right) then uses the decoding function $d(h)$ and decodes h_t back to an output candidate solution o, with each o_t representing either a function or a terminal. The aim of decoding is to reconstruct the input candidate solution x. Using o_t as input in o_{t+1} helps to further reduce the reconstruction error [27]. Similar to [4, 27], we reverse the input candidate solution x to allow the model to learn low range correlations in training data (e.g., x_1 and o_1 are closer to each other when reversing the input), simplifying optimization [27].

3.2 Using Levenshtein edit as corruption strategy

The aim of the AE-LSTM is to reconstruct the input. Given that the hidden layer is sufficiently large, a trivial way to solve this task is to learn the simple identity function, which means that the AE-LSTM simply replicates the candidate solutions given as input. Since we want to learn a more useful representation of the properties of our parent population, we partially corrupt input candidate solutions, transforming the AE-LSTM into a DAE-LSTM. Based on the first DAE presented by Vincent et al. [5] in 2008, the idea is to partially corrupt input candidate solutions making the model robust to noise that is present in our parent population.

At each generation g, we use the corruption function $c(x)$ to corrupt the candidate solutions that were previously selected as promising candidate solutions from population P_g. We can formally describe the process by

$$\tilde{x}^{(i)} = c(x^{(i)}) \quad \forall i \in \{1, \ldots, N\}, \tag{1}$$

where $\tilde{x}^{(i)}$ is the corrupted version of the i-th candidate solution x in the training set X (of size N) [4].

As a corruption function $c(x)$, we use Levenshtein edit that was first introduced in [8]. Levenshtein edit operates on the linear sequence of the input candidate solution x (prefix expression), where each x_t represents a node (function or terminal) of a tree. Levenshtein edit uses insertion (add one node), deletion (remove one node), and substitution (replace one node by another node) to transform x into $\tilde{x}$. We control the corruption strength by a priori defining a corruption percentage p $(0 < p < 1)$. Given a function set F, a terminal set T, and a candidate solution x, with x_j, $j \in \{1, 2, \ldots, m\}$, we corrupt x by iteratively processing each node x_j,

where each x_j has a chance of p to be corrupted: with uniform probability, we either insert a random symbol $s \in F \cup T$ at index j (insertion), we delete x_j (deletion), or we delete x_j *and* insert a random symbol $s \in F \cup T$ at index j (substitution). Note that these edit operations may produce corrupted candidate solutions $\tilde{x}$ that do not obey GP syntax. However, sampling with syntax control (see Sect. 3.4) ensures that output candidate solutions o are syntactically valid.

Using Levenshtein edit as corruption strategy has several advantages: we introduce variance in tree size (number of nodes in a tree). This is desirable since it introduces additional variation into $\tilde{x}$. However, this variation should not lead to a bias towards larger or smaller trees, which was a problem in previous work that used subtree mutation as corruption strategy [4]. Subtree mutation randomly selects a node in tree and replaces the subtree at that node by a new random subtree generated by ramped half-and-half. Depending on the size of the selected subtree to be replaced, we easily corrupt either larger (when closer to the root) or smaller parts (when closer to leaf nodes) of x resulting in a bias in tree size. The situation is different for Levenshtein edit: we randomly choose corruption operators that iteratively either increase (insertion), decrease (deletion), or maintain (substitution) the size of x. Thus, for any p, the expected tree size of $\tilde{x}$ is equal to the tree size of x, which means that we are able to introduce variation without inducing a bias in tree size. Furthermore, we can easily control the corruption strength by adjusting p. The larger p, the stronger the variation, and the stronger the corruption. The results in Sect. 4 will show that this helps to control exploration and exploitation in search.

3.3 Training procedure

At each generation g, we train a DAE-LSTM (from scratch) according to the training procedure shown in Algorithm 1. It is similar to the training procedure presented in [4, 8]. We first initialize the trainable parameters θ of our network (Algorithm 1, line 1). Then, we iteratively adjust the values of the trainable parameters θ using gradient descent. Given the edit percentage p, we first transform the candidate solution $x^{(i)}$ into $\tilde{x}^{(i)}$ (Algorithm 1, line 4). Then, we propagate $\tilde{x}^{(i)}$ through the DAE-LSTM, using the encoding function $g(x)$ (Algorithm 1, line 5) and the decoding function $d(x)$ (Algorithm 1, line 6). We compute the reconstruction error using the multiclass cross entropy loss function by

$$\theta := \min_{\theta} \sum_{i=1}^{N} Err(x^{(i)}, o^{(i)}), \tag{2}$$

where $o^{(i)}$ is the output candidate solution and $x^{(i)}$ the original (not the corrupted) input candidate solution, with N being the size of the training set X. We update the parameters θ into the direction of the negative gradient and control the strength of the update using the learning rate α ($0 < \alpha < 1$) (Algorithm 1, line 7).

Algorithm 1 Pseudocode for training a DAE-LSTM

1: **Initialize** θ
2: **while** not converged **do**
3: **for each** candidate solution $x^{(i)}$ in training set X **do**
4: $\tilde{x}^{(i)} = c(x^{(i)};\ p)$
5: $h = g(\tilde{x}^{(i)};\ \theta)$
6: $o^{(i)} = d(h;\ \theta)$
7: $\theta := \theta - \alpha * \frac{\partial Err(x^{(i)}, o^{(i)})}{\partial \theta}$
8: **end for**
9: **end while**

We stop training as soon as the training error $Err(x^{(i)}, o^{(i)})$, with $x^{(i)}, o^{(i)} \in X$, converges. We measure error convergence by observing the number of epochs that the training error does not improve. As soon as we reach 200 epochs of no improvement, with a minimum change of 0.05 that qualifies for an improvement, we stop training and use for sampling the configuration θ that minimizes the training error. This training procedure follows the recommendations of Probst and Rothlauf [6] but is different to previous work that used early stopping with a hold-out validation set to train the DAE-GP [4, 8]. Here, the idea is to use those parameters for model sampling that minimize the validation error. The problem of using a hold-out validation set is that it would strongly reduce the number of candidate solutions in our training set if we have to assure that the validation set is a representative sample of our parent population [29]. The situation is different when using the new training procedure: here, we use the entire parent population as training set, which allows the DAE-GP to capture more complex dependencies in training data. Still, denoising prevents the model from overfitting.

3.4 Sampling with syntax control

Given the trained DAE-LSTM with parameters θ, we can sample new candidate solutions o forming the offspring population P_{g+1}. Algorithm 2 shows the sampling procedure and is based on [4, 6, 8, 30]. Given θ (Algorithm 2, line 1), we first pick a candidate solution x of our training set X (Algorithm 2, line 2). Then, we corrupt x into $\tilde{x}$ (Algorithm 2, line 3) using Levenshtein edit and the edit percentage v. Note that v can be different to the edit percentage p during training. This is new compared to previous work [4, 6, 8]. We found that exploration can be even stronger when increasing the edit percentage only during sampling, helping the model to overcome local optima. We explain this new property in Sect. 3.5. Then, for s sampling steps, we propagate $\tilde{x}$ through the DAE-LSTM (Algorithm 2, lines 4–7), and add the resulting output candidate solution o to P_{g+1} (Algorithm 2, line 6). The idea of using several sampling steps was presented in previous work [4, 6], but has not yet been explored in the context of the DAE-GP. Probst and Rothlauf argue that choosing several sampling steps results in a stronger exploitation of the solution space [6]. We investigate this property in Sect. 4.

Algorithm 2 Pseudocode for sampling from a DAE-LSTM

1: **Given** the trained DAE-LSTM with θ
2: **Pick** $x \in X$
3: $\tilde{x} = c(x, v)$
4: **for** s sampling steps **do**
5: $\quad h = g(\tilde{x};\ \theta)$
6: $\quad o = d(h;\ \theta)$
7: $\quad \tilde{x} := o$
8: **end for**
9: **Add** o to new population P_{g+1}

We apply syntax control during the last sampling step to ensure that only syntactically valid candidate solutions are added to the new population P_{g+1}. The mechanism proceeds as follows: at each time step t, with $t \in \{m+1, m+2, \ldots, T\}$, when decoding the latent vector h back to an output candidate solution o (Algorithm 2, line 6), the DAE-LSTM implements a probability distribution q over the set of functions and terminals (defined by F and T). Following the procedure of grow initialization [31], we first identify the set of functions and terminals that generate a syntactically valid candidate solution, where we also verify that we do not exceed the recommended maximum allowed tree depth restriction of 17 [31]. Then, we set the classes of invalid functions and terminals in q to zero and normalize the remaining probabilities in q back to one, where we use the updated probability distribution to sample o_t. Note that the DAE-LSTM samples a prefix expression (representing a GP parse tree top down, left to right) where syntax control following grow initialization becomes possible.

Without corrupting the input, syntax control is usually not needed since the complexity of the DAE-LSTM is sufficient to also learn correct syntax. However, the stronger the corruption, the more difficult it becomes for the DAE-LSTM to sample syntactically valid candidate solutions, since corrupted candidate solutions used as input to the model no longer belong to the same parent population as X. In these cases, syntax control is very useful: we prevent the DAE-LSTM from inefficient resampling and allow the model to explore new solution spaces, which can help to overcome local optima and to avoid premature convergence.

3.5 The manifold learning perspective

Compared to previous work [4, 6, 8] that used the same edit percentage for training and sampling, we now split this property into edit percentage for training p and edit percentage for sampling v (see Algorithms 1 and 2). The manifold learning perspective by Vincent et al. [5] helps to understand why we draw this distinction. In the context of the DAE-GP, the manifold is the latent representation of the training set X (a multidimensional surface) that the model learns at each generation g. Vincent et al. argue that training a denoising autoencoder with edit percentage p leads to a model that learns to project corrupted points (the corrupted candidate solutions $\tilde{x}$)

back to the manifold (the original candidate solution x) [5]. When using the same edit percentage during training and sampling (as done in previous work), and given enough epochs and examples (variations of $\tilde{x}$) to train, this must lead (also for different edit percentages) to a DAE-GP that tends the replicate the parent population from population P_g into population P_{g+1}, leading to premature convergence. Instead, when fixing the edit percentage during training (at a low percentage) and varying the edit percentage only during sampling (at a higher percentage, as done in this work), we force the DAE-GP to reconstruct corrupted points in the latent space that are further away from corrupted points that were reconstructed during training. This allows to explore new areas in the solution space, where a stronger corruption results in a stronger exploration of the solution space. Still, the reconstruction through the learned DAE-LSTM leads to a DAE-GP that transfers properties from the parent population to the offspring.

4 Experiments

We study the performance and the search behavior of the DAE-GP not only on the GRT problem, but also on the Airfoil and the Pagie-1 problem. Furthermore, we analyze the influence of corruption strength on search as well as the influence of the number of sampling steps.

4.1 Problems

The GRT problem was first presented in [4] as a test problem. It is based on the royal tree problem introduced by Punch et al. [32] but uses the initialization method ramped half-and-half [31] to generate target candidate solutions x_{opt}. The idea is to define a fitness based on the structure of a candidate solution x by

$$fitness_x = \frac{lev(x, x_{opt})}{max(l_x, l_{x_{opt}})}, \tag{3}$$

where lev is the minimum Levenshtein distance, defined by the minimum number of insertion, deletion, and substitution operations necessary to transform x into x_{opt} [9]. In analogy to [4], we normalize lev by the maximum size l of x and x_{opt}, resulting in $fitness_x \in [0, 1]$: the closer x to x_{opt}, the better the fitness, where $fitness_x = 0$ means that x is identical to x_{opt} [4]. Small changes in the genotype will result in small changes in fitness (high-locality problem) [7].

The Airfoil problem [10] is a more difficult real-world symbolic regression problem. The objective is to find a (black-box) function that best describes a given dataset, where we use the Root Mean Squared Error (RMSE) to measure the fitness of the function. In contrast to the GRT problem, the correlation between the fitness of two neighboring candidate solutions is usually lower (low-locality problem). The Airfoil dataset is often used in GP literature [e.g., 33, 34, 12, 35] and available in the UCI Machine Learning Repository [36]. It consists of 5 features and 1503

observations. We split the dataset into 50% training and 50% test set to study the performance of algorithms on unseen test cases.

The Pagie-1 problem [11] is a synthetic symbolic regression problem, where the objective is to find a function that best approximates synthetic points that were generated using the objective function $\frac{1}{1+z_1{}^{-4}} + \frac{1}{1+z_2{}^{-4}}$. It is known to be a challenging problem [37], where evolutionary search hardly finds the objective function (low-locality problem). Pagie and Hogeweg recommend to generate a set of synthetic evaluation points by using uniform random sampling, with $z_1, z_2 \in [-5, 5]$, and to use an interval of 0.4 resulting in $26 \times 26 = 676$ evaluation points that are part of the training set. Note that it is not standard in the EA community to use a test set [11, 37] or to study the generalizability of the found solutions. Thus, to measure performance, we use the RMSE on the training set.

4.2 Experimental setup

We implemented the experiments in Python using the evolutionary framework DEAP [38] and the neural network framework Keras [39]. We use ramped half-and-half to generate the initial populations and target candidate solutions x_{opt}, where we set the minimum and maximum tree depth to $d_{min} = 2$ and $d_{max} = 6$ (Airfoil and Pagie-1), and $d_{min} = 2$ and $d_{max} = 4$ (GRT), respectively. Note that reducing d_{max} from 6 to 4 strongly reduces the complexity of the problem when searching for target candidate solutions x_{opt} [4]. However, we allow all algorithms to search for solutions up to a maximum tree depth of $d_{max} = 17$ [31].

For the Airfoil and the GRT problem, we use the function set $F = \{+, -, *, \div_{AQ}\}$, where $\div_{AQ}$ represents the analytic quotient (AQ) [40]. The function set is often used to solve real-world symbolic regression tasks [33, 35]. As terminal set, we use $T = \{z_1, z_2, z_3, z_4, z_5\} \cup ERC$, which includes the five problem features of the Airfoil dataset and ephemeral random integer constants ERC. For the Airfoil problem, we use $ERC \in [-5, \ldots, 5]$ whereas for the GRT problem, we set $ERC = 1$. Again, this affects the complexity of the solution space: reducing the number of constants simplifies for both standard GP and the DAE-GP the search of target candidate solutions x_{opt} [4]. For the Pagie-1 problem, we use the Koza function set $F = \{+, -, *, \%, sin, cos, exp, log\}$ [31, 37], where % represents protected division, and the terminal set $T = \{z_1, z_2\} \cup ERC$, where we fix $ERC = 1$.

For GP, we follow the recommendations of Koza [31] and use 90% subtree crossover with internal node bias (90% of the crossover points are functions) and 10% subtree mutation as variation operator. Subtree mutation uses full initialization with $d_{min} = 0$ and $d_{max} = 2$ to generate a new subtree. To evaluate GP on different hyperparameters, we conduct a grid search and vary population size P_s combined with the number of generations g assuring a maximum number of 15,000 fitness evaluations, with $P_s \in \{250, 500, 750, 1000, 1500\}$ and $g \in \{10, 15, 20, 30, 60\}$. Furthermore, we vary the tournament selection size u, with $u \in \{2, 5, 7, 9, 12\}$.

For the DAE-GP, we have to pre-define a set of hyperparameters. We set the number of hidden layers to one and the hidden dimension equal to the maximum size l of the candidate solutions used as input to the model, which allows the model

to flexibly adjust its complexity during evolution. We found that the complexity of the model is sufficient to learn complex relationships in training data while allowing efficient model building and sampling. We train the DAE-GP until the training error converges (as discussed in Sect. 3.3), set the batch size to 50 (10% of X), use a learning rate of $\alpha = 0.001$, and perform adaptive moment estimation (Adam) [41] for gradient descent optimization. We use Levenshtein edit as corruption strategy. To study the impact of corruption and sampling steps on search, we fix the edit percentage $p = 5\%$ during training but vary the edit percentage v and the sampling steps s during sampling: for analysis of both corruption and sampling steps, we study $v \in \{5\%, 20\%, 50\%, 80\%, 95\%\}$ and $s \in \{1, 2, 3, 4, 5\}$.

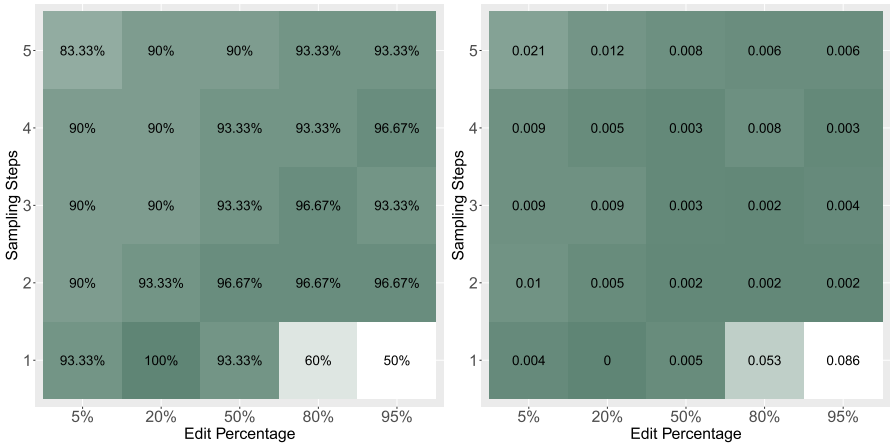
To assure a fair performance comparison between DAE-GP and GP, we first identify the best problem-specific GP hyperparameter setting (by varying population size P_s, the number of generations g, and tournament selection size u) resulting in 25 GP hyperparameter combinations that we test for each problem. Then, we apply the best GP hyperparameter combination of P_s, g, and u to the DAE-GP and vary the edit percentage v and the number of sampling steps s, resulting in again 25 hyperparameter combinations that we evaluate for each problem. We conduct 30 runs per algorithm and hyperparameter setting (30 different x_{opt} for the GRT problem), resulting in a total number of 2250 DAE-GP and 2250 GP runs that we aggregate for performance comparison.

4.3 Performance results

We study the performance of the DAE-GP and GP on the GRT, the Airfoil, and the Pagie-1 problem. We use heat maps for better visualization. For the DAE-GP, we study the influence of the number of sampling steps s and the edit percentage v. According to previous work [6, 8], exploitation should be strongest with weak corruption and a high number of sampling steps (upper left corner of heat map); in contrast, exploration should be strongest with strong corruption and a low number of sampling steps (lower right corner of heat map).

We first focus on the GRT problem. Figure 2 shows the average algorithm success rates (left) that we want to maximize and the average best fitness (right) that we aim to minimize for 25 different DAE-GP configurations after 30 generations, given $P_s = 750$, $g = 20$, and $u = 7$ (best GP configuration). A run is successful as soon as the algorithm finds a candidate solution x during search that is identical to the target candidate solution x_{opt} ($fitness_x = 0$).

We notice that both the average success rates and the average best fitness differ depending on the algorithm configuration considered. For example, when setting the edit percentage to $v = 95\%$ and the number of sampling steps to $s = 1$ (where we expect the strongest exploration), the DAE-GP does only find 50% of the target candidate solutions, with an average best fitness of 0.086. In contrast, when choosing $v = 20\%$ and $s = 1$ (stronger exploitation), the DAE-GP finds all target solutions resulting in a success rate of 100% and an average best fitness of 0 (best DAE-GP configuration).



(a) Average Success Rates (b) Average Best Fitness

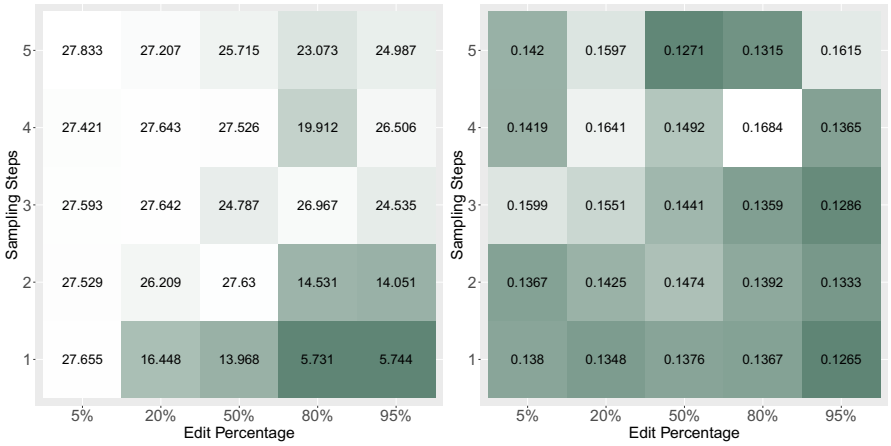
Fig. 2 GRT problem: average success rates (left) and average best fitness (right) of DAE-GP in last generation, given $P_s = 750$, $g = 20$, and $u = 7$. Darker green represents better performance (Color figure online)

We use (pairwise) Mann–Whitney U tests regarding the hypothesis that the best fitness distributions in the last generation are from the same population. We assume a significance level of 0.05. We find that significant differences between the best DAE-GP configuration ($v = 20\%$ and $s = 1$) and other configurations are only observable (p-values < 0.05) when comparing the best configuration to a DAE-GP with success rates lower or equal to 83.33% ($v = 95\%$ and $s = 1$, $v = 80\%$ and $s = 1$, and $v = 5\%$ and $s = 5$). In all other cases, differences are not significant (p-values > 0.05). Thus, both sampling steps and corruption strength influence performance. However, many combinations of sampling steps and corruption strength yield good performance results.

To assess the performance of DAE-GP, we benchmark the results to 25 hyperparameter combinations of standard GP. The best GP configuration ($P_s = 750$, $g = 20$, and $u = 7$) finds 86.67% of the target candidate solutions x_{opt} and yields an average best fitness of 0.017. When comparing the best fitness distribution of the best DAE-GP to the one of standard GP, we find a p-value of 0.042, which indicates that the best DAE-GP significantly outperforms standard GP.

Figure 3 shows a heat map for the Airfoil and Pagie-1 problem, with the median best fitness (RMSE) that we minimize in the last generation on the test set (Airfoil) and training set (Pagie-1). The DAE-GP for the Airfoil problem uses $P_s = 500$, $g = 30$, and $u = 12$, while the DAE-GP for the Pagie-1 problem is set to $P_s = 750$, $g = 20$, and $u = 12$ (each best GP configuration). Similar to Fig. 2, we again vary both the number of sampling steps s and the edit percentage v.

Compared to the GRT problem, we notice stronger differences in performance depending on the chosen sampling step s and edit percentage v used throughout the experiments, especially for the Airfoil problem: while a DAE-GP with $v = 5\%$ and $s = 5$ (where we expect the strongest exploitation) yields a median



(a) Airfoil problem (b) Pagie-1 problem

Fig. 3 Median best RMSE of DAE-GP in last generation. For the Airfoil problem (left), we report results on the test set, given $P_s = 500$, $g = 30$, and $u = 12$. For the Pagie-1 problem (right), we show results on the training set, given $P_s = 750$, $g = 20$, and $u = 12$. Darker green represents better performance (Color figure online)

best fitness of 27.833, we find that a DAE-GP with $v = 80\%$ and $s = 1$ (where we expect strong exploration) finds the best candidate solutions with a median best fitness of 5.731 (best configuration). Pairwise comparisons of the best fitness distributions confirm this observation: for the Airfoil problem, when comparing the best DAE-GP configuration ($v = 80\%$ and $s = 1$) to other DAE-GP settings, we observe that nearly all (pairwise) comparisons are significant (p-values < 0.05), except for the comparison of the best DAE-GP to a DAE-GP with $v = 95\%$ and $s = 1$ (p-value > 0.05). The results indicate that the number of sampling steps and the edit percentage both have a strong influence on the performance of the DAE-GP.

For the Pagie-1 problem, differences in performance are less pronounced: when comparing the best fitness distributions of the best DAE-GP configuration ($v = 95\%$ and $s = 1$) to a DAE-GP with $v = 95\%$ and $s = 2$, $v = 95\%$ and $s = 3$, or $v = 50\%$ and $s = 5$, we find p-values > 0.05. Here, no significant differences in the best fitness distributions exist. Especially the last comparison is interesting, as we would expect exploitation and thus differences in performance to be much stronger here. In contrast, when comparing the best DAE-GP configuration to a DAE-GP with $v = 50\%$ and $s = 1$, $v = 50\%$ and $s = 2$, or $v = 5\%$ and $s = 1$, we find p-values < 0.05, indicating that significant differences exist. However, similar to the Airfoil problem, we find that the DAE-GP with $v = 95\%$ and $s = 1$ (setting with strongest exploration) yields the best results.

Similar to the GRT problem, we benchmark the DAE-GP for each the Airfoil and Pagie-1 problem to 25 hyperparameter combinations of standard GP. The best GP configuration yields a median best RMSE of 7.65 ($P_s = 500$, $g = 30$, and $u = 12$) for the Airfoil problem and a median best RMSE of 0.0055 ($P_s = 750$,

$g = 20$, and $u = 12$) for the Pagie-1 problem. A comparison of the best fitness distributions of the best DAE-GP to the ones of standard GP yields p-values of 0.41 (Airfoil) and 6.6e−10 (Pagie-1): differences between the best DAE-GP and the best GP are not significant on the Airfoil problem, but significant on the Pagie-1 problem.

The results indicate that both the edit percentage and the number of sampling steps seem to have a big influence on solution quality but that several combinations of sampling steps and corruption strength often yield good performance results. We observe the strongest differences in DAE-GP performance on the Airfoil dataset.

For the GRT problem (high-locality problem), we find that strong exploitation leads to best performance results, whereas for both the Airfoil and the Pagie-1 problem (low-locality problems), a strong randomization of the input and choosing a small number of sampling steps seems to improve search, helping the DAE-GP to overcome local optima. However, the results on the Airfoil and the Pagie-1 problem raise the question if the AE as a model is still helpful or if a random corruption (mutation) of candidate solutions (without feeding the solutions to the AE) might be sufficient to obtain good results. We therefore conducted further experiments for both problems, where we use Levenshtein tree edit [13] as random mutation operator (without the AE) at each generation, using the same P_s, g, u, and five edit percentages v as for the original experiments. Levenshtein tree edit is a variation of Levenshtein edit that works on the tree representation of x. The advantage of Levenshtein tree edit is that we do not destroy GP syntax [13]. Thus, it is straightforward to use it as mutation operator. For the Airfoil problem, we find median RMSE values on the test set between 119.8 and 124.8. For Pagie-1, median RMSE values on the training set range from 0.56 to 11.7. When comparing the results to Fig. 3, we notice that all RMSE values on the heat map are significantly smaller to the ones of random corruption, highlighting that propagating candidate solutions through the AE is useful for search.

When comparing the performance of the DAE-GP to standard GP, we notice that the DAE-GP outperforms standard GP on the GRT problem, while standard GP outperforms the DAE-GP on the Pagie-1 problem. We observe no significant differences on the Airfoil problem. The results are in line with recent results by Wittenberg and Rothlauf [13], who compared the performance of the DAE-GP to standard GP, GP-GOMEA [35] and geometric semantic GP [33], given nine real-world symbolic regression datasets (including Airfoil). The authors also do not find a clear winner when comparing the performance of the algorithms to each other. Interestingly, they find that advantages of the DAE-GP lie in the generation of small solutions. Indeed, for the Pagie-1 problem, we find that the median size of the final best solution of the DAE-GP is 9 compared to 40 for standard GP. For the Airfoil problem, we find that the median size of the final best solution of the DAE-GP is 12 compared to 150 for standard GP.

4.4 The influence of corruption and sampling steps on search

The results above demonstrate that both corruption and the number of sampling steps influence the performance of the DAE-GP. This is in line with the results in [6, 8]. To better understand the influence of corruption and sampling steps on search, we study the exploration and exploitation behavior of the algorithms. We expect exploration to be strongest when we use strong corruption and a low number of sampling steps (lower right corner of heat map). Instead, exploitation should be strongest when using weak corruption and many sampling steps (upper left corner of heat map) [6, 8].

According to Rothlauf [7], we need to find an appropriate and problem-specific balance between exploration and exploitation in search. For problems, where small variations on the genotype lead to small variations in fitness (high-locality problems such as the GRT problem), we usually need much less exploration compared to problems, where the fitness landscape is rugged (problems with lower locality such as the Airfoil or the Pagie-1 problem). Thus, depending on the problem at hand, we either need to increase exploitation, making search more efficient, or we need to increase exploration, helping search to keep diversity high and allowing to overcome local optima and to avoid premature convergence [7].

4.4.1 Introducing Levenshtein diversity

Previous work approximated exploration and exploitation by examining the number of new candidate solutions over generations that have never been sampled before [4, 8]. Exploitation is strong if search introduces a low number of new candidate solutions during search. In contrast, the more new candidate solutions we introduce into search, the stronger the exploration. The problem with counting new candidate solutions in search is that we do not take into account how big the differences are between two candidate solutions. For example, changing only one node of a GP tree can result in a candidate solution that has never been sampled before counting towards exploration. Preliminary experiments showed that this can be misleading, especially when properties such as the average solution size (number of nodes in a solution) of a population changes over generations, which we often observe in the problem domain of symbolic regression where size increases over search time.

Therefore, we suggest a new measure that we call Levenshtein diversity to approximate the level of exploration and exploitation in search. The idea is to calculate all pairwise normalized Levenshtein distances between candidate solutions (in prefix expression) in a population and to average the results over the total number of pairwise comparisons (in total $\frac{N*(N-1)}{2}$ pairwise comparisons). The result is a measure scaled between zero and one, where a Levenshtein diversity close to one indicates strong diversity (strong exploration) in a population. In contrast, a Levenshtein diversity close to zero suggests weak diversity (strong exploitation) in a population. The idea of calculating a Levenshtein diversity within a population builds upon work presented by O'Reilly, who introduced averaged pairwise Levenshtein

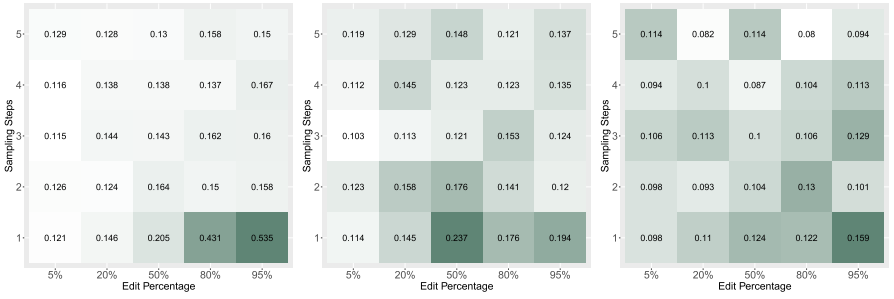
distances to analyze crossover operators [42]. Normalizing those Levenshtein distances is also not new and appears in Kelly et al. [43], who use averaged normalized pairwise Levenshtein distances within a selection operator (knobelty selection) to find new diverse solutions. However, we are not aware of any work that uses averaged pairwise normalized Levenshtein distances to study the exploration and exploitation behavior of a population throughout search.

4.4.2 Analysis of diversity in last generation

Figure 4 shows a heat map of the average normalized Levenshtein diversity in the last generation of the DAE-GP for the GRT problem (left), the Airfoil problem (center), and the Pagie-1 problem (right), given the best GP hyperparameter combination (population size P_s, number of generations g, and tournament size u) that was used for performance comparison in Sect. 4.3. Recap that we expect exploration to be strongest (values closer to 1) in the lower right corner of the heat map; in contrast, we expect exploitation to be strongest in the upper left corner of the heat map (values closer to 0).

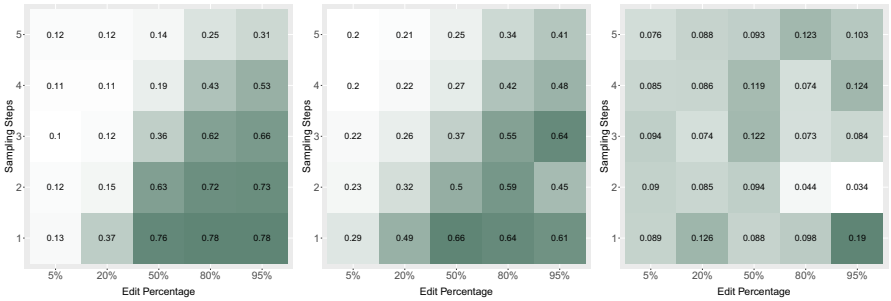
Interestingly, results are not as clear as expected. When comparing diversity values with each other, we notice that differences are often small and difficult to interpret. For example, while for the GRT and the Pagie-1 problem, exploration is indeed highest at $v = 95\%$ and $s = 1$ with Levenshtein diversities of 0.535 and 0.159, respectively, it is lower for the Airfoil problem (0.194) compared to a DAE-GP with $v = 50\%$ and $s = 1$ (0.237). We notice that the low level of diversity (many runs lead to average Leventhtein distances of close to zero) complicates the analysis. It indicates that search has mostly converged, which is probably a result of the strong selection pressure, with tournament selection sizes between 7 and 12. Furthermore, the use of different population sizes, number of generations and tournament selection sizes makes it difficult to compare results between problems.

Therefore, Fig. 5 shows a heat map of the average normalized Levenshtein diversity, where we fix the population size to $P_s = 500$, the number of generations to



(a) GRT problem (b) Airfoil problem (c) Pagie-1 problem

Fig. 4 Average Normalized Levenshtein diversity of DAE-GP in last generation, given $P_s = 750$, $g = 20$, and $u = 7$ for the GRT problem (left), $P_s = 500$, $g = 30$, $u = 12$ for the Airfoil problem (center), and $P_s = 750$, $g = 20$, and $u = 12$ for the Pagie-1 problem (right). Darker green represents higher diversity (Color figure online)



(a) GRT problem (b) Airfoil problem (c) Pagie-1 problem

Fig. 5 Average Normalized Levenshtein diversity of DAE-GP in last generation, given $P_s = 500$, $g = 30$, and $u = 2$ for the GRT (left), the Airfoil (center), and the Pagie-1 (right) problem. Darker green represents higher diversity (Color figure online)

$g = 30$, and the tournament selection size to $u = 2$ for all three problems. The results are interesting, since the hyperparameter configuration of the DAE-GP is exactly the same for all problems. However, remind that the problems use different function and terminal sets (see Sect. 4.2), making a direct comparison of diversities between problems still difficult.

When comparing the results to Fig. 4, we notice strong differences in diversity, highlighting that varying the hyperparameters strongly affects exploration and exploitation in search. Here, we expect especially tournament selection size to have a strong influence on diversity. Indeed, this can best be observed when comparing results between Figs. 4b and 5b (Airfoil problem). Here, we only adjust u (from 12 to 2), which strongly increases diversity. For example, for a DAE-GP with $v = 95\%$ and $s = 1$ on the Airfoil problem, reducing selection pressure leads to an increase in diversity in the final population from 0.194 to 0.61.

The diversity values in the heat map in Fig. 5 are now much more as expected, especially for the GRT problem: we can well observe that increasing the edit percentage v and lowering the number of sampling steps s increases the diversity of the solutions. Here, a DAE-GP with $v = 95\%$ and $s = 1$ yields a diversity of 0.78, while choosing $v = 5\%$ and $s = 5$ results in a diversity of 0.12. For the Airfoil problem, results are similar: while setting the DAE-GP to $v = 95\%$ and $s = 1$ yields a diversity of 0.61, choosing $v = 5\%$ and $s = 5$ results in a much lower diversity of 0.2. However, there are exceptions. For example, a DAE-GP with $v = 95\%$ and $s = 2$ produces less diverse solutions (0.45) than a DAE-GP with $v = 95\%$ and $s = 3$ (0.64). Probably, the DAE-GP with $v = 95\%$ and $s = 2$ has already started to converge to a local optimum. For the Pagie-1 problem, differences in diversity are more difficult to interpret: the DAE-GP with $v = 95\%$ and $s = 1$ again yields the most diverse solutions (0.19). However, the level of diversity is still very low, which is surprising as we decreased the tournament selection size u from 12 (Fig. 4c) to 2 (Fig. 5c). Also, it is surprising that a DAE-GP with $v = 95\%$ and $s = 2$ yields the lowest diversity value in the heatmap (0.034). The results indicate that the DAE-GP converges very early for the Pagie-1 problem, even when given a low tournament selection

Fig. 6 Average normalized Levenshtein diversity over generations, given $P_s = 500$, $g = 30$, and $u = 2$, ▶ for the GRT (left), Airfoil (center), and Pagie-1 (right) problems, with fixed number of sampling steps s (**a–f**) and fixed edit percentage v (**g–l**)

size, which makes a comparison of diversity values still difficult. It also shows that adjusting the edit percentage and the number of sampling steps does not seem to affect the exploration and exploitation behavior as much as we can observe it for the GRT or the Airfoil problem.

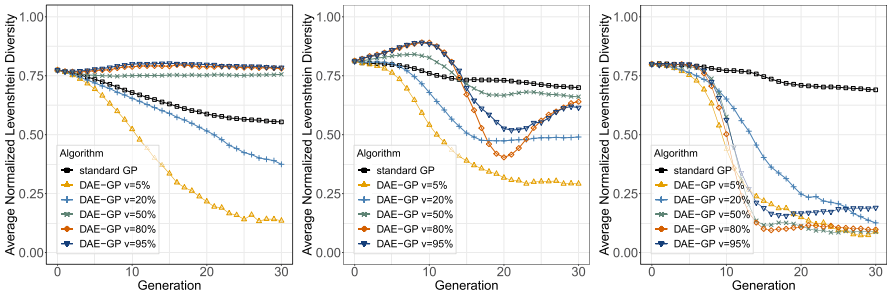
For completeness, we report the problem-specific configuration of the DAE-GP with fixed hyperparameters ($P_s = 500$, $g = 30$, $u = 2$) that performs best: for the GRT problem, a DAE-GP with $v = 20\%$ and $s = 2$ yields the best results, with a success rate of 86.67% and a median best fitness of 0.019. For the Airfoil problem, a DAE-GP with $v = 95\%$ and $s = 4$ is most successful with a median best RMSE of 8.77 on the test set. For the Pagie-1 problem, the DAE-GP with $v = 20\%$ and $s = 3$ performs best, with a median best RMSE of 0.1754 on the training set.

4.4.3 Analysis of diversity over generations

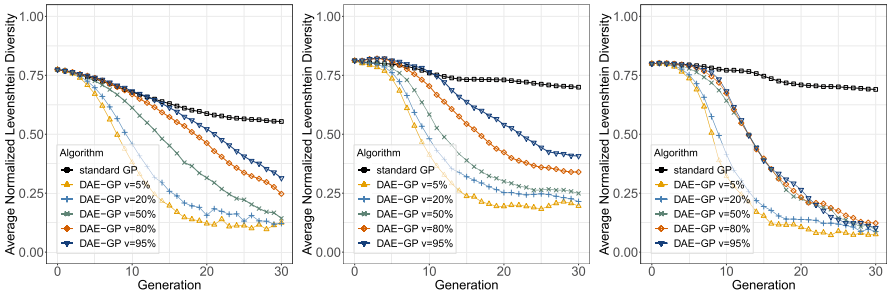
Previous results focused on diversity only in the last generation. To gain insight into the development of diversity throughout search, Fig. 6 plots the average normalized Levenshtein diversity over generations. We show results for the GRT (left), Airfoil (center), and Pagie-1 (right) problems, given $P_s = 500$, $g = 30$, and $u = 2$, for a selection of DAE-GP configurations. We fix either the number of sampling steps s (a–f) or the edit percentage v (g–l) for easier visualization and compare the results to standard GP that uses the same P_s, g, and u as DAE-GP. The results give a fine-grained idea of how exploration and exploitation develops throughout search.

We find that diversity is usually highest in the initial population (at around 0.8) and then decreases (with a few exceptions) over generations. We expect this behavior since selection focuses search on specific areas of the solution space. However, the decrease is very different depending on the considered problem and the chosen edit percentage v and sampling step s.

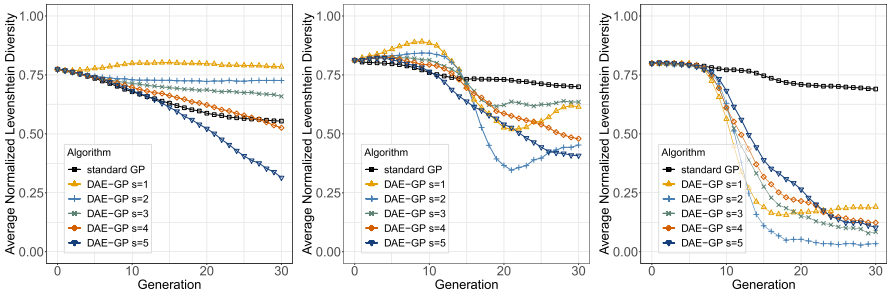
We first focus on the GRT problem. When fixing the number of sampling steps and focusing on $s = 1$ (Fig. 6a), we find that the decrease (and thus the exploitation) of the DAE-GP is strongest when using an edit percentage of $v = 5\%$. In contrast, when we set the edit percentage to $v = 95\%$, exploration is much stronger. The results confirm the observations made in Wittenberg [8]: the stronger the corruption, the stronger the exploration. Compared to standard GP, the level of exploration is higher when using a corruption strength of 50%, 80%, or 95%. However, there seems to be an upper boundary in exploration, since differences between $v = 80\%$ and 95% are only hardly distinguishable. When increasing the sampling steps from $s = 1$ to $s = 5$ (Fig. 6d), the general level of exploration decreases (which is what we expect) such that the diversity of even the DAE-GP with strongest $v = 95\%$ falls below the level of exploration of standard GP. We fix the edit percentage and focus on $v = 95\%$ (Fig. 6g). The results confirm that the number of sampling steps s also has a strong influence on exploration and exploitation. A DAE-GP with $s = 1$ shows



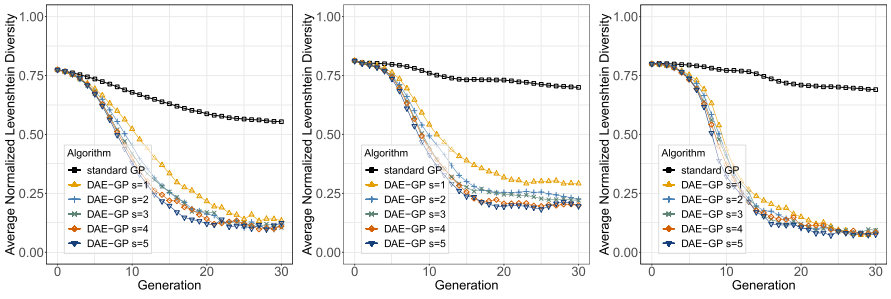
(a) GRT $s = 1$ (b) Airfoil $s = 1$ (c) Pagie-1 $s = 1$



(d) GRT $s = 5$ (e) Airfoil $s = 5$ (f) Pagie-1 $s = 5$



(g) GRT $v = 95\%$ (h) Airfoil $v = 95\%$ (i) Pagie-1 $v = 95\%$



(j) GRT $v = 5\%$ (k) Airfoil $v = 5\%$ (l) Pagie-1 $v = 5\%$

the strongest exploration, whereas increasing the number of sampling steps leads to a stronger exploitation of the solution space. This in line with previous work by Probst and Rothlauf [6]: the more sampling steps we use, the stronger the exploitation. When lowering the edit percentage from $v = 95\%$ to $v = 5\%$ (Fig. 6j), we notice a general (and expected) decrease of the level of exploration. The results indicate that we seem to be approaching a lower bound on exploration where differences in diversity become smaller when adjusting the number of sampling steps.

Let us now consider the Airfoil problem, where we first fix the number of sampling steps to $s = 1$ (Fig. 6b). Similar to the GRT problem, we find that setting the edit percentage to $v = 5\%$ leads to the strongest decrease in diversity and that exploration generally increases when elevating corruption strength. However, for $v \geq 50\%$, we find that the diversity shifts strongly over generations. Diversity first increases to a level larger than the initial population, then drops (partially below the diversity of the DAE-GP with $v = 20\%$), and finally increases again. Interestingly, at $g = 30$, the DAE-GP configurations that use $v = 50\%$ and $v = 80\%$ are slightly more diverse than the DAE-GP configuration using $v = 95\%$ (confirming the observations made in Fig. 5b). Probably the rugged fitness landscape of the Airfoil problem is the reason for strong changes in diversity, where some configurations get stuck in local optima. When focusing on standard GP and comparing the diversity to the GRT problem, we notice that diversity remains at a higher level for the Airfoil problem, which can probably be explained by the strong bloat behavior of standard GP for regression problems [13] (recap that we found a median best solution size of GP of 150). For the DAE-GP, when increasing the number of sampling steps to $s = 5$ (Fig. 6e), the level of exploration of the DAE-GP decreases. Results are similar to the ones of the GRT problem (Fig. 6d). When fixing the edit percentage to $v = 95\%$ (Fig. 6h), results are similar to the ones in Fig. 6b. We again observe a strong change in diversity, where the DAE-GP with $s = 1$ generates the most diverse solutions only in the very first generations. In the last generation, the configuration using $s = 3$ yields the highest diversity. When reducing corruption strength to $v = 5\%$ (Fig. 6k), we find a general decrease in exploration, where differences in diversity between DAE-GP configurations become smaller. Again, we seem to be approaching a lower bound on exploration of the DAE-GP.

For the Pagie-1 problem, when the number of sampling steps is fixed to $s = 1$ (Fig. 6c), we notice that the development of diversity differs strongly from observations made for the GRT and the Airfoil problem (Fig. 6a, b): the diversity of all DAE-GP configurations strongly converges towards a level below 0.25, indicating that the DAE-GP quickly gets stuck in a local optima. Only for $v = 95\%$ (where we expect the strongest exploration), we observe that diversity slightly increases in later generations. The results suggest that the DAE-GP has severe problems to solve the Pagie-1 problem due to premature convergence, where also strong randomization techniques do not help to overcome local optima. We believe that this is due to the very rugged fitness landscape of the Pagie-1 problem. Observations are similar when increasing the number of sampling steps to $s = 5$ (Fig. 6f), or when fixing the edit percentage and varying v (Fig. 6f, l). Interestingly, when comparing the level of diversity to the one of GP, we notice that diversity is much higher. Again, this is probably due to the strong bloat behavior of standard GP that helps to keep diversity

high. For the Pagie-1 problem, we postulate that this helps standard GP to yield superior performance results compared to the DAE-GP (see Sect. 4.3).

The analysis of diversity over generations confirms previous observations. For the GRT and the Airfoil problem, we find that both corruption strength and the number of sampling steps have a strong influence on exploration and exploitation in search: the weaker the corruption and the more sampling steps we use, the stronger the exploitation. This can help to improve search performance for high-locality problems such as the the GRT problem (where the DAE-GP significantly outperforms standard GP). In contrast, the stronger the corruption and the fewer the sampling steps, the stronger the exploration. This can help to solve problems where we face more rugged fitness landscapes such as the Airfoil problem.

However, for very difficult problems such as the Pagie-1 problem, where we expect the fitness landscape to be very rugged, the DAE-GP easily gets stuck in local optima, even when applying strong randomization techniques. Here, we observe strong losses in diversity, also when setting selection pressure to a low level $u = 2$. We think that diversity preserving selection methods such as Epsilon-Lexicase selection [44] could help to further improve search quality of the DAE-GP.

In general, we do not expect the DAE-GP to yield superior performance results compared to standard GP when facing very rugged fitness landscapes. Here, the only chance to overcome local optima is to apply strong randomization techniques and diversity preserving mechanisms. Instead, we should rather focus on problems with higher locality. For these problems, we believe that performance advantages from probabilistic model building and sampling will occur. However, recent work [13] has also shown that other properties such as the solution size is a promising property of the DAE-GP that can help to find small solutions and improve interpretability for real-world regression tasks.

5 Conclusions

The DAE-GP is an EDA-GP model based on artificial neural networks that flexibly identifies and models hidden relationships in training data. It corrupts input candidate solutions to make the model robust to noise that is present in the parent population. For the GRT problem, Airfoil and Pagie-1 problems, this paper studied the influence of corruption and sampling steps on search.

We find that both the level of corruption and the number of sampling steps strongly influence exploration and exploitation in search and affect performance. The stronger we corrupt input candidate solutions and the less sampling steps we use, the stronger the exploration. High levels of exploration is especially useful for low-locality problems such as the Airfoil problem where we want to escape from local optima. In contrast, for high-locality problems, such as the GRT problem, stronger exploitation is needed. The results show that the corruption and sampling strategy is key to the success of the DAE-GP: it permits us to control the exploration and exploitation behavior in search leading to an improved search quality. For the Pagie-1 problem, which is a very difficult problem with low-locality, we find that the DAE-GP easily gets stuck in local optima, also when choosing strong corruption and

a low number of sampling steps. For these problems, diversity preserving mechanisms (such as diversity preserving selection methods) are needed to avoid premature convergence.

In future work, we want to study if we can dynamically control both corruption strength and the number of sampling steps throughout search. To prevent the DAE-GP from premature convergence, we want to study the performance of the DAE-GP using diversity preserving methods, such as Epsilon-Lexicase selection [44] or novelty initialization [45]. In addition, preliminary results suggest that pre-training and re-using the model during evolution strongly reduces computation time [46]. We plan to explore this model strategy further.

Acknowledgements We thank our group in Mainz and Quebec City for insightful discussions and previous work on this topic. Furthermore, we thank the reviewers for thoughtful comments that helped improve the manuscript. Parts of this research were conducted using the supercomputer Mogon and/or advisory services offered by Johannes Gutenberg University Mainz (http://hpc.uni-mainz.de), which is a member of the AHRP (Alliance for High Performance Computing in Rhineland Palatinate, http://www.ahrp.info) and the Gauss Alliance e.V. The authors gratefully acknowledge the computing time granted.

Author contributions David Wittenberg wrote the manuscript. All authors reviewed the manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. This work was supported by a fellowship of the German Academic Exchange Service (DAAD) and by a funding from the Interdisciplinary Public Policy (IPP) Mainz.

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

References

1. K. Kim, Y. Shan, X.H. Nguyen, R.I. McKay, Probabilistic model building in genetic programming: a critical review. Genet. Program Evolvable Mach. **15**(2), 115–167 (2014). https://doi.org/10.1007/s10710-013-9205-x
2. M. Pelikan, M.W. Hauschild, F.G. Lobo, Introduction to estimation of distribution algorithms. Missouri Estimation of Distribution Algorithms Laboratory (MEDAL), Report Nr. 2012003 (2012)
3. J.H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence* (University of Michigan Press, Ann Arbor, 1975)
4. D. Wittenberg, F. Rothlauf, D. Schweim, DAE-GP: Denoising autoencoder LSTM networks as probabilistic models in estimation of distribution genetic programming, in *Proceedings of the 2020*

Genetic and Evolutionary Computation Conference. GECCO'20 (ACM, New York, 2020), pp. 1037–1045. https://doi.org/10.1145/3377930.3390180
5. P. Vincent, H. Larochelle, Y. Bengio, P.A. Manzagol, Extracting and composing robust features with denoising autoencoders, in *Proceedings of the 25th International Conference on Machine Learning (ICML'08)* (ACM, Helsinki, 2008), pp. 1096–1103. https://doi.org/10.1145/1390156.1390294
6. M. Probst, F. Rothlauf, Harmless overfitting: Using denoising autoencoders in estimation of distribution algorithms. J. Mach. Learn. Res. **21**(78), 1–31 (2020)
7. F. Rothlauf, *Design of Modern Heuristics: Principles and Application*, 1st edn. (Springer, Berlin, 2011). https://doi.org/10.1007/978-3-540-72962-4
8. D. Wittenberg, Using denoising autoencoder genetic programming to control exploration and exploitation in search, in *Genetic Programming: 25th European Conference, EuroGP 2022, Held as Part of EvoStar 2022, Madrid, Spain, April 20–22, 2022, Proceedings* (Springer, Berlin, 2022), pp. 102–117. https://doi.org/10.1007/978-3-031-02056-8_7
9. J.B. Kruskal, An overview of sequence comparison: time warps, string edits, and macromolecules. Soc. Ind. Appl. Math. (SIAM) Rev. **25**(2), 201–237 (1983). https://doi.org/10.1137/1025045
10. T.F. Brooks, D.S. Pope, M.A. Marcolini, *Airfoil Self-noise and Prediction*, vol. 1218. National Aeronautics and Space Administration, Office of Management, Scientific and Technical Information Division (1989)
11. L. Pagie, P. Hogeweg, Evolutionary consequences of coevolving targets. Evol. Comput. **5**(4), 401–418 (1997)
12. D. Wittenberg, F. Rothlauf, Denoising autoencoder genetic programming for real-world symbolic regression, in *Proceedings of the Genetic and Evolutionary Computation Conference Companion. GECCO'22* (Association for Computing Machinery, New York, 2022), pp. 612–614. https://doi.org/10.1145/3520304.3528921
13. D. Wittenberg, F. Rothlauf, Small solutions for real-world symbolic regression using denoising autoencoder genetic programming, in *Genetic Programming: 26th European Conference, EuroGP 2023, Held as Part of EvoStar 2023, Brno, Czech Republic, April 12–14, 2023, Proceedings* (Springer, Berlin, 2023), pp. 101–116. https://doi.org/10.1007/978-3-031-29573-7_7
14. Y. Shan, R. McKay, D. Essam, H. Abbass, A survey of probabilistic model building genetic programming, in *Scalable Optimization Via Probabilistic Modeling*. ed. by M. Pelikan, K. Sastry, E. CantúPaz (Springer, Berlin, 2006), pp.121–160
15. R. Salustowicz, J. Schmidhuber, Probabilistic incremental program evolution. Evol. Comput. **5**(2), 123–141 (1997). https://doi.org/10.1162/evco.1997.5.2.123
16. K. Yanai, H. Iba, Estimation of distribution programming based on Bayesian network, in *IEEE Congress on Evolutionary Computation (CEC'03)* (IEEE, Canberra, 2003), pp. 1618–1625. https://doi.org/10.1109/CEC.2003.1299866
17. Y. Hasegawa, H. Iba, Estimation of Bayesian network for program generation, in *Proceedings of the Third Asian-Pacific Workshop on Genetic Programming Hanoi, Vietnam* (2006), pp. 35–46
18. Y. Hasegawa, H. Iba, A Bayesian network approach to program generation. IEEE Trans. Evol. Comput. **12**(6), 750–764 (2008). https://doi.org/10.1109/tevc.2008.915999
19. A. Ratle, M. Sebag, Avoiding the bloat with probabilistic grammar-based genetic programming, in *5th International Conference on Artificial Evolution (EA'01)* (Springer, Le Creusot, 2001), pp. 255–266
20. Y. Hasegawa, H. Iba, Estimation of distribution algorithm based on probabilistic grammar with latent annotations, in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'17)* (IEEE, Singapore, 2007), pp. 1043–1050. https://doi.org/10.1109/CEC.2007.4424585
21. P.-K. Wong, L.-Y. Lo, M.-L. Wong, K.-S. Leung, Grammar-based genetic programming with Bayesian network, in *IEEE Congress on Evolutionary Computation (CEC'14)* (IEEE, Beijing, 2014), pp. 739–746
22. P.-K. Wong, L.-Y. Lo, M.-L. Wong, K.-S. Leung, grammar-based genetic programming with dependence learning and Bayesian network classifier, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'14)* (ACM, Vancouver, 2014), pp. 959–966. https://doi.org/10.1145/2576768.2598256
23. R. Poli, N.F. McPhee, A Linear estimation-of-distribution GP system, in *Proceedings of the 11th European Conference on Genetic Programming (EuroGP'08)* (Springer, Neapel, 2008), pp. 206–217
24. E. Hemberg, K. Veeramachaneni, J. McDermott, C. Berzan, U.-M. O'Reilly, An investigation of local patterns for estimation of distribution genetic programming, in *Proceedings of the genetic*

and evolutionary computation conference (GECCO '12) (ACM, Philadelphia, 2012), pp. 767–774. https://doi.org/10.1145/2330163.2330270
25. P. Liskowski, K. Krawiec, N.E. Toklu, J. Swan, Program synthesis as latent continuous optimization: evolutionary search in neural embeddings, in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference. GECCO'20* (Association for Computing Machinery, New York, 2020), pp. 359–367. https://doi.org/10.1145/3377930.3390213
26. M. Probst, Denoising autoencoders for fast combinatorial black box optimization, in *Proceedings of the Companion Publication of the Annual Conference on Genetic and Evolutionary Computation* (ACM, Madrid, 2015), pp. 1459–1460
27. N. Srivastava, E. Mansimov, R. Salakhutdinov, Unsupervised learning of video representations using LSTMs, in *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)* (ACM, Lille, 2015), pp. 843–852. https://doi.org/10.5555/3045118.3045209
28. S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
29. D. Schweim, D. Wittenberg, F. Rothlauf, On sampling error in genetic programming. Natural Comput. (2021). https://doi.org/10.1007/s11047-020-09828-w
30. Y. Bengio, L. Yao, G. Alain, P. Vincent, Generalized denoising auto-encoders as generative models, in *Advances on Neural Information Processing Systems (NIPS'13)*, vol. 26, pp. 899–907 (2013)
31. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT press, Cambridge, 1992)
32. B. Punch, D. Zongker, E. Goodman, The royal tree problem, a benchmark for single and multi-population genetic programming, in *Advances in Genetic Programming II*. ed. by P.J. Angeline, K.E. Kinnear Jr. (MIT Press, Cambridge, 1996), pp.299–316
33. J.F.B.S. Martins, L.O.V.B. Oliveira, L.F. Miranda, F. Casadei, G.L. Pappa, Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming, in *Proceedings of the Genetic and Evolutionary Computation Conference. GECCO'18* (Association for Computing Machinery, New York, 2018), pp. 1151–1158. https://doi.org/10.1145/3205455.3205593
34. V.V. de Melo, D.V. Vargas, W. Banzhaf, Batch tournament selection for genetic programming: The quality of lexicase, the speed of tournament, in *Proceedings of the Genetic and Evolutionary Computation Conference. GECCO'19* (Association for Computing Machinery, New York, 2019), pp. 994–1002. https://doi.org/10.1145/3321707.3321793
35. M. Virgolin, T. Alderliesten, C. Witteveen, P.A.N. Bosman, Improving model-based genetic programming for symbolic regression of small expressions. Evol. Comput. **29**(2), 211–237 (2021). https://doi.org/10.1162/evco_a_00278
36. D. Dua, C. Graff, *UCI Machine Learning Repository* (2017). http://archive.ics.uci.edu/ml
37. J. McDermott, D.R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, U.-M. O'Reilly, Genetic programming needs better benchmarks, in *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation. GECCO'12* (Association for Computing Machinery, New York, 2012), pp. 791–798. https://doi.org/10.1145/2330163.2330273
38. F.A. Fortin, F.M. De Rainville, M.A. Gardner, M. Parizeau, C. Gagńe, DEAP: Evolutionary algorithms made easy. J. Mach. Learn. Res. **13**(1), 2171–2175 (2012)
39. F. Chollet, keras. GitHub (2015). https://github.com/fchollet/keras
40. J. Ni, R.H. Drieberg, P.I. Rockett, The use of an analytic quotient operator in genetic programming. IEEE Trans. Evol. Comput. **17**(1), 146–152 (2013). https://doi.org/10.1109/TEVC.2012.2195319
41. D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in *International Conference on Learning Representations, San Diego, CA, USA* (2015)
42. U.-M. O'Reilly, Using a distance metric on genetic programs to understand genetic operators, in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 5 (1997), pp. 4092–4097. https://doi.org/10.1109/ICSMC.1997.637337
43. J. Kelly, E. Hemberg, U.-M. O'Reilly, Improving genetic programming with novel exploration–exploitation control, in *Genetic Programming*. ed. by L. Sekanina, T. Hu, N. Lourenço, H. Richter, P. García-Sánchez (Springer, Cham, 2019), pp.64–80
44. W. La Cava, L. Spector, K. Danai, Epsilon-lexicase selection for regression, in *Proceedings of the Genetic and Evolutionary Computation Conference 2016. GECCO'16* (Association for Computing Machinery, New York, 2016), pp. 741–748. https://doi.org/10.1145/2908812.2908898
45. C. Olmscheid, D. Wittenberg, D. Sobania, F. Rothlauf, Improving estimation of distribution genetic programming with novelty initialization, in *Proceedings of the Genetic and Evolutionary*

Computation Conference Companion. GECCO'21 (Association for Computing Machinery, New York, 2021), pp. 261–262. https://doi.org/10.1145/3449726.3459410

46. J. Reiter, D. Schweim, D. Wittenberg, Pretraining reduces runtime in denoising autoencoder genetic programming by an order of magnitude, in *Proceedings of the Genetic and Evolutionary Computation Conference Companion. GECCO'23* (Association for Computing Machinery, New York, 2023). https://doi.org/10.1145/3583133.3596332