UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Victor de Souza Caetano

Embedded representation of genetic programming trees

Belo Horizonte
2023

Victor de Souza Caetano

Embedded representation of genetic programming trees

Final Version

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Gisele Lobo Pappa

Belo Horizonte
2023

2023, Víctor de Souza Caetano.
Todos os direitos reservados

Caetano, Víctor de Souza.

C128e Embedded representation of genetic programming trees [recurso eletrônico] / Víctor de Souza Caetano. – 2023.

1 recurso online (84 f. il., color.) : pdf.

Orientadora: Gisele Lobo Pappa

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f. 65-68.

1. Computação – Teses. 2. Aprendizado do computador – Teses. 3. Programação genética (Computação) – Teses. 4. Redes neurais (Computação) – Teses. I. Pappa, Gisele Lobo. II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. III. Título.

CDU 519.6*82(043)

Ficha catalográfica elaborada pela bibliotecária Irenquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais – ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Embedded representation of genetic programming trees

VÍCTOR DE SOUZA CAETANO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Profa. Gisele Lobo Pappa - Orientadora
Departamento de Ciência da Computação - UFMG

Prof. Luiz Chaimowicz
Departamento de Ciência da Computação - UFMG

Prof. Renato Tinós
Departamento de Computação e Matemática - USP

Belo Horizonte, 12 de setembro de 2023.

Resumo

Aprendizado de representações é uma área que investiga formas de representações de dados que facilitem para os algoritmos de aprendizado de máquina extraírem informações úteis deles. O aprendizado profundo atualmente possui os métodos mais eficazes para essa tarefa e pode aprender representações vetoriais reais - também conhecidas como *embeddings* - capazes de representar diferentes propriedades dos dados e suas relações.

Nessa direção, este trabalho introduz uma nova maneira de visualizar indivíduos de programação genética (GP) em formato de árvore para regressão simbólica. A Programação Genética é um ramo da computação evolucionária onde os indivíduos são gerados para representar estruturas gerais, também conhecidas como programas. Quando essas estruturas são usadas em regressão simbólica, os indivíduos geralmente são representados por árvores, onde cada árvore é mapeada para uma função matemática. Como em qualquer tarefa de regressão, o objetivo da regressão simbólica é encontrar a função f que mapeia o conjunto de variáveis que descrevem um exemplo de dados para um valor real y, $y \in \mathbf{R}$.

Assim como em outros algoritmos evolucionários, os GPs exploram o espaço de busca de funções f usando os princípios da evolução e sobrevivência do mais apto. Primeiro, cada solução é avaliada para medir sua capacidade de resolver o problema, gerando o valor da função de aptidão (*fitness*). A aptidão é então usada em um processo probabilístico de seleção de indivíduos. Em seguida, os indivíduos selecionados passam por operações de cruzamento e mutação. Nos GPs baseados em árvores, é sabido que as operações genéticas acabam sendo restritas pela sintaxe e na maioria das vezes não há garantia de que o resultado das operações de cruzamento seja significativo ou apenas equivalente a uma mutação. Há muitas tentativas de levar a semântica em consideração nesse tipo de algoritmo, mas as abordagens propostas até agora não alteram as próprias representações individuais para tentar aproveitar a robustez fornecida pelos embeddings.

Motivados por isso, dado um conjunto de operadores predefinidos usados para gerar as soluções para o problema (funções) representadas por árvores e um número suficientemente grande de árvores amostradas do espaço, treinamos um *transformer* para aprender uma função de codificação/decodificação.

Ao transformar uma representação de árvore em uma representação vetorial numérica, somos capazes de medir distâncias entre árvores de maneira muito mais eficiente e, mais importante, gerar o potencial para que essas representações capturem semântica.

Mostramos que a distância considerando o embedding apresenta resultados muito

semelhantes aos de uma distância de edição de árvore, o que reflete sua similaridade sintática. Embora o modelo ainda não possa capturar a semântica, mostramos seu potencial usando o modelo de representação de árvore gerado em tarefas simples: medindo distâncias entre árvores em um cenário de *fitness sharing*, onde a diversidade é a propriedade desejável e pode ser medida considerando distâncias entre árvores, e gerando visualizações das árvores dentro de uma população evoluída.

Palavras-chave: programação genética, representações embutidas, transformers, semântica, aprendizado de representação, redes neurais.

Abstract

Representation learning is an area responsible for learning data representations that makes it easier for machine learning algorithms to extract useful information from them. Deep learning currently has the most effective methods for this task and can learn distributed representations – also known as embeddings – able to represent different properties of the data and their relationship. In this direction, this work introduces a new way to look at tree-like genetic programming (GP) individuals for symbolic regression. Genetic Programming is a branch of evolutionary computation where individuals are generated to represent general structures also known as programs. When used for symbolic regression, individuals are usually represented by trees, where each tree maps to a mathematical function. As in any regression task, the objective of symbolic regression is to find the function f that maps the set of variables describing a data example into a real-value y, where $y \in \mathbf{R}$. As with other evolutionary algorithms, GPs explore the search space of functions f using the principles of evolution and survival of the fittest. First, each solution is evaluated to measure its ability to solve the problem, generating the value of the fitness function. The fitness is then used in a probabilistic individual selection process. Next, individuals selected undergo crossover and mutation operations. In tree-based GPs, it is well-known that genetic operations end up being restricted by syntax and most of the time there is no guarantee the result of crossover operations is meaningful or is just equivalent to a mutation. There have been many attempts to take semantics into consideration in this type of algorithm, but the approaches proposed so far do not change the individual's representations themselves to try to take advantage of the robustness provided by embeddings. Motivated by that, given a set of predefined operators used to generate the solutions to the problem (functions) represented by trees and a sufficiently large number of trees sampled from the space, we train a transformer to learn an encoding/decoding function. By transforming a tree representation into a distributed representation, we are able to measure distances between trees in a much more efficient way and, more importantly, generate the potential for these representations to capture semantics. We show the distance accounting for embedding presents results very similar to those of a tree-edition distance, which reflects their syntactic similarity. Although the model cannot capture semantics yet, we show its potential by using the generated tree-representation model in simple tasks: measuring distances between trees in a fitness-sharing scenario, where diversity is the desirable property and can be measured considering distances between trees or and generating visualizations of the trees within an evolved population.

Keywords: genetic programming, embedded representations, transformers, semantics, representation learning, neural networks.

List of Figures

2.1 Transformer architecture proposed by [36]. 21
2.2 Self-attention mechanism proposed by [36]. 22

3.1 Proposed methodology considering a maximum tree depth of 3. 26
3.2 Tree linearization process. 28
3.3 Transformer training process. 29
3.4 Correlation between the embedding distance and the edit distance 33
3.5 Heatmap of the correlation of the embedding representation with edit distance. 35
3.6 Heatmap of the correlation of the embedding representation with the fitness distance. 36
3.7 Heatmap of the correlation of the embedding distances and the edit distance. . 37
3.8 Distances considering the embeddings (size = 64). 38

4.1 Violinplots of the individual's pairwise embedding distances for the dataset RedWine. 42
4.2 Violinplots of the individual's pairwise embedding distances for the dataset Pagie-1. 43
4.3 Violinplots of the individual's pairwise embedding distances for the dataset Concrete. 43
4.4 Coordinate-based plot of 6 runs for Pagie-1 dataset. 56
4.5 Neighborhood-based plot of 6 runs for Pagie-1 dataset. 56
4.6 Plots of 2 runs for Pagie-1 dataset, with parameters random seed 20230610, tournament sizes 2 (blue) and 7 (orange), and mutation probability 10%. On the right the coordinate-based plot, and on the left the neighborhood-based one. 57
4.7 Coordinate-based plot of 6 runs for DM4 dataset. 58
4.8 Neighborhood-based plot of 6 runs for DM4 dataset. 58
4.9 Plots of 2 runs for DM4 dataset, with parameters random seed 20230611, tournament size 2, and mutation probabilities 10% (blue) and 40% (orange). On the right is the coordinate-based plot, and on the left is the neighborhood-based one. 59
4.10 Plots of 2 runs for DM4 dataset with the trajectories highlighted, with parameters random seed 20230611, tournament size 2, and mutation probabilities 10% (blue) and 40% (orange). On the right is the coordinate-based plot, and on the left is the neighborhood-based one. 60

4.11 Coordinate-based plot of 6 runs for DM7 dataset. 61
4.12 Neighborhood-based plot of 6 runs for DM7 dataset. 61
4.13 Plots of 2 runs for DM4 dataset with parameters random seed 20230610, tournament size 2, and mutation probabilities 10% (blue) and 40% (orange). On the right is the coordinate-based plot, and on the left is the neighborhood-based one. 62

List of Tables

3.1 Datasets used to train the model. 30
3.2 Subset of parameters tuned. 33
3.3 Hand-designed trees for testing embedding distances. 37

4.1 Datasets for experiments with diversity. 41
4.2 Parameters tested for the model selection experiment. 44
4.3 Parameters of the best-performing models and their results of median MSE (fitness) when applied to a new group of datasets. 45
4.4 Parameters of the best-performing models and their results of Median Entropy Transformed Semantics diversity when applied to a new group of datasets. . . 46
4.5 Parameters of the best-performing models and their results of Median Pseudo-Isomorph diversity when applied to a new group of datasets. 46
4.6 Median MSE over 50 runs of GP in different versions: with no fitness sharing, using the fitness and edit distances for fitness sharing, and using the proposed method based on embeddings (generated for dataset DM7) in Table 4.1. . . . 47
4.7 Median Entropy Transformed Semantics over 50 runs of GP in different versions: with no fitness sharing, using the fitness and edit distances for fitness sharing, and using the proposed method based on embeddings (generated for dataset DM7) in Table 4.1. 48
4.8 Median Pseudo-Isomorph over 50 runs of GP in different versions: with no fitness sharing, using the fitness and edit distances for fitness sharing, and using the proposed method based on embeddings (generated for dataset DM7) in Table 4.1. 48
4.9 Parameters tested for evaluating the impact of tree size. 49
4.10 Median MSE over 50 runs of GP in different versions: Using the proposed method based on embeddings (generated for dataset DM7) with tree maximum size parameter set as 4, 6, 8, and with no fitness sharing. 50
4.11 Median Entropy Transformed Semantics over 50 runs of GP in different versions: Using the proposed method based on embeddings (generated for dataset DM7) with tree maximum size parameter set as 4, 6, 8, and GP with fitness sharing using the fitness difference as distance. 50

4.12 Median Pseudo-Isomorph over 50 runs of GP in different versions: Using the proposed method based on embeddings (generated for dataset DM7) with tree maximum size parameter set as 4, 6, 8, and GP with fitness sharing using the edit distance. 51
4.13 Symbolic Regression parameters used in the visualization experiments. 55
4.14 Color and size scheme for nodes and edges. 55
4.15 Fitness and trajectory size per run for the dataset Pagie-1. 56
4.16 Fitness and trajectory size per run for dataset DM4. 58
4.17 Fitness and trajectory size per run for dataset DM7. 61

A.1 Parameters of the models trained using dataset DM4 and their results of Median MSE when applied to a new group of datasets. 69
A.2 Parameters of the models trained using dataset DL1 and their results of Median MSE when applied to a new group of datasets. 70
A.3 Parameters of the models trained using dataset DM7 and their results of Median MSE when applied to a new group of datasets. 70
A.4 Parameters of the models trained using dataset DM4 and their results of Median MSE when applied to a new group of datasets. 71
A.5 Parameters of the models trained using dataset DL1 and their results of Median MSE when applied to a new group of datasets. 71
A.6 Parameters of the models trained using dataset DM7 and their results of Median MSE when applied to a new group of datasets. 72
A.7 Parameters of the models trained using dataset DM4 and their results of Median MSE when applied to a new group of datasets. 72
A.8 Parameters of the models trained using dataset DL1 and their results of Median MSE when applied to a new group of datasets. 73
A.9 Parameters of the models trained using dataset DM7 and their results of Median MSE when applied to a new group of datasets. 73
A.10 Parameters of the models trained using dataset DM4 and their results of Median MSE when applied to a new group of datasets. 74
A.11 Parameters of the models trained using dataset DL1 and their results of Median MSE when applied to a new group of datasets 74
A.12 Parameters of the models trained using dataset DM7 and their results of Median MSE when applied to a new group of datasets 75
A.13 Parameters of the models trained using datasets DM4 and DL1 and their results of Median Entropy Transformed Semantics when applied to a new group of datasets. 75

A.14 Parameters of the models trained using datasets DM7 and DM4 and their results of Median Entropy Transformed Semantics when applied to a new group of datasets 76
A.15 Parameters of the models trained using datasets DL1 and DM7 and their results of Median Entropy Transformed Semantics when applied to a new group of datasets. 76
A.16 Parameters of the models trained using datasets DM4 and DL1 and their results of Median Entropy Transformed Semantics when applied to a new group of datasets. 77
A.17 Parameters of the models trained using datasets DM7 and DM4 and their results of Median Entropy Transformed Semantics when applied to a new group of datasets 77
A.18 Parameters of the models trained using datasets DL1 and DM7 and their results of Median Entropy Transformed Semantics when applied to a new group of datasets. 78
A.19 Parameters of the models trained using dataset DM4 and their results of Median Pseudo-Isomorph when applied to a new group of datasets 78
A.20 Parameters of the models trained using dataset DM4 and their results of Median Pseudo-Isomorph when applied to a new group of datasets. 79
A.21 Parameters of the models trained using dataset DM7 and their results of Median Pseudo-Isomorph when applied to a new group of datasets. 79
A.22 Parameters of the models trained using dataset DM4 and their results of Median Pseudo-Isomorph when applied to a new group of datasets. 80
A.23 Parameters of the models trained using dataset DL1 and their results of Median Pseudo-Isomorph when applied to a new group of datasets. 80
A.24 Parameters of the models trained using dataset DM7 and their results of Median Pseudo-Isomorph when applied to a new group of datasets. 81
A.25 Parameters of the models trained using dataset DM4 and their results of Median Pseudo-Isomorph when applied to a new group of datasets. 81
A.26 Parameters of the models trained using dataset DL1 and their results of Median Pseudo-Isomorph when applied to a new group of datasets. 82
A.27 Parameters of the models trained using dataset DM7 and their results of Median Pseudo-Isomorph when applied to a new group of datasets 82
A.28 Parameters of the models trained using dataset DM4 and their results of Median Pseudo-Isomorph when applied to a new group of datasets 83
A.29 Parameters of the models trained using dataset DL1 and their results of Median Pseudo-Isomorph when applied to a new group of datasets 83
A.30 Parameters of the models trained using dataset DM7 and their results of Median Pseudo-Isomorph when applied to a new group of datasets 84

Contents

1 Introduction **15**

2 Background and Literature Review **18**

2.1 Distributed Representations 18

2.1.1 Transformer network 19

2.1.2 Transformers for representing tree structures 22

2.2 GP for Symbolic Regression 23

2.3 Embedded representations in evolutionary algorithms 24

3 Embedded representations for Genetic Programming trees **25**

3.1 From trees to distributed representations 25

3.1.1 Generating GP trees 26

3.1.2 Tree linearization and Tokenization 27

3.1.3 Representation Learning Model 28

3.2 Model training 30

3.2.1 Generating training and test data 31

3.2.2 Model evaluation 31

3.2.3 Parameter tuning 32

3.3 Experimental Analysis 34

3.3.1 Model effectiveness and generalization 34

3.3.2 Representation evaluation 35

3.3.3 A note on execution time 37

4 Case studies on Distributed Representation of GP Trees **39**

4.1 Individuals' diversity and fitness sharing 39

4.1.1 Experimental Setup 41

4.1.2 Model generation and selection 42

4.2 Impact of input trees to the models 49

4.3 Embedded tree representations for visualization 51

4.3.1 Coordinate-based trajectory visualization 53

4.3.2 Neighborhood-based trajectory visualization 53

4.3.3 Experimental Results 54

4.3.3.1 Pagie-1 plots 55

4.3.3.2 DM4 plots 57

4.3.3.3 DM7 plots 60

5 Conclusions and Future Work **63**

Bibliography **65**

Appendix A Tables **69**

A.1 Evaluation using Median MSE 69

A.2 Evaluation using Median Entropy Transformed Semantics 75

A.3 Evaluation using Pseudo-Isomorph Diversity 78

Chapter 1

Introduction

Representation learning is currently one of the most studied areas in machine learning [24]. In short, the basic idea behind this area is to learn a data representation that makes easier for machine learning algorithms to extract useful information from it. Although representation learning can be performed using a different range of methods, those based on deep learning are currently considered the most powerful [19].

There is a set of characteristics deep neural networks can add to representations that can make them more expressive [3]. In general, they can learn distributed representations able to represent different properties of the data and their relationships. For instance, the use of embeddings generated by neural networks have completely changed the way text is represented in a wide range of applications [19]. Since the initial proposal of word2vec [25], more powerful language representation methods have been proposed and become state-of-the-art for text representation [24].

For this reason, the use of embeddings has extrapolated the area of language models, and many different areas have been exploring this idea [39]. There is a wide range of "*tovec" methods, where * can stand for author, graph, tree, etc, and they have been shown to be robust to represent a diverse set of concepts. One of the main advantages of using embedded representations is that they are able to capture the semantics of concepts/objects being represented. For language models, we know the representation goes beyond lexical, as shown by the results of vector operations, capable of adding or subtracting concepts [25].

In evolutionary algorithms (EAs), individuals – which represent candidate solutions to a problem of interest – are generated based on a "representation model", which can vary from a vector of numbers to more complex structures, including trees and graphs [12]. The idea behind these algorithms is to search for a solution to a problem following the principles of evolution, which are driven by the ideas of survival of the fittest and random mutations, which introduce variability to the solutions being explored. Solutions are represented by individuals, and evaluated to measure their ability to solve the problem (fitness). The fitness value of the individuals is then used to probabilistic select a subset to undergo crossover and mutation operations, which are responsible for exploring the space of solutions. While crossover changes genetic material from two individuals, mutations

replace parts of a single individual.

When EA individuals are represented by more complex structures, different types of analysis and genetic operations are limited to the representation used. In this thesis, we are interested in the case of symbolic regression, where individuals are usually represented by trees. As in any regression task, the objective of symbolic regression is to find a function f that maps the set of variables (or attributes) X describing a data example into a real-value y, where $y \in \mathbf{R}$. This problem has been successfully solved by a variant of evolutionary computation called genetic programming (GP) [21]. For tree-based GPs, it is well-known that genetic operations end up being restricted by syntax and most of the time there is no guarantee the result of crossover operations is meaningful or is just equivalent to a mutation. There have been many attempts to take semantics into consideration in this type of algorithm [27], but the approaches proposed so far do not change the individual's representations themselves to try to take advantage of the robustness provided by embeddings.

In natural language processing (NLP), embeddings are used to generate text representations, which can then be used as input to other models more specific to the task of interest. Inspired by that, this thesis introduces a new way to look at tree-like genetic programming (GP) individuals. Given a set of predefined operators chosen to define the types of functions f that can be generated and a sufficiently large number of solutions sampled from the space, we train a neural network, more specifically, a transformer [36], to learn an encoding/decoding function from the tree. By transforming a tree representation into a vector, we are able to measure distances between trees in a much more efficient way. More importantly, the main question is whether these representations can also capture the semantics of the functions represented by the trees. We show the distance accounting for embedding presents results very similar to those of a tree-edition, which reflects their syntactic similarity. We go beyond and use the fitness, measured using an error metric, to evaluate the semantic distance between trees.

Although the model seems not to capture semantics yet, we show its potential by using the generated tree-representation model in two case studies. In the first, we use the generated representations to measure distances between trees to measure and account for diversity in these algorithms. This is interesting as measuring distances between trees can be a very computationally expensive problem. In the second, we use the generated representations to generate visualizations of the search space, which is a difficult problem given the dimensionality of the solutions. We use these visualizations to evaluate the explored search space and the trajectory of the solutions found by different search algorithms. We conclude by showing that the proposed model opens up a diverse set of opportunities to incorporate, in the near future, both syntactic and semantics into GP-based representation models.

The remainder of this thesis is organized as follows. Chapter 2 provides a compre-

hensive exploration of the background and literature relevant to the work. In Chapter 3, the experimental methodology employed for this study is explained in detail, including the methodology, modelling details, evaluation, as well as an initial experiment setup and analysis. Chapter 4 presents two case studies on the use of the methodology described in the previous chapter, including the experiment setup, analysis, and discussion of the results. Finally, Chapter 5 provides a summary of the thesis as well as the conclusion drawn from the experiments, and possibilities for future research.

Chapter 2

Background and Literature Review

This chapter introduces the concepts of representation learning, which serves as inspiration for this work. Representation learning is a sub-field of machine learning that focuses on learning useful and meaningful representations of data [15]. These representations are typically used as input to other machine learning algorithms or models, and the goal is to learn a representation that captures the essential aspects of the data and that is useful for a particular task.

There are many different approaches to representation learning, including unsupervised learning methods such as autoencoders, generative models, and dimensionality reduction techniques, as well as supervised learning methods [15]. Representation learning has been successful in a wide range of applications, including image and video analysis, natural language processing, and speech recognition [4].

In this work we are particularly interested in unsupervised representation learning based on artificial neural networks. In order to contextualize the methods that first served as inspiration to this work, this chapter reviews the first methods proposed in the area of natural language processing [4]. This methods are interesting because the distributed representation of words they generate are able to capture problem semantics by coding each word and its context – an important component for assigning meaning to it – in a real vector-space embedding [4]. Next, we give an overview of distributed representations and introduce transformer models, which are the base of the model proposed in this work. Following, we describe other works that used embedded representations in the context of evolutionary algorithms.

2.1 Distributed Representations

Among the initial distributed representations for text based on neural networks are the word2vec [25], which trains a multi-layer perception and uses as embeddings the weights of the neurons in the hidden layer. The most interesting aspect of word vectors

representations is they are expected to be consistent with vector algebra, in the sense that some operations in the vector domain (such as sum or difference) should keep some degree of consistency with similar semantic manipulations. For example, [25] shows word vectors models in where the operation $vector("king")$ - $vector("man")$ + $vector("woman")$ produces a vector that is close to $vector("queen")$[1]. Even more abstract concepts could be derived, not only terms from a vocabulary, like the similarity between the results of operations $vector("cars")$ - $vector("car")$ and $vector("apples")$ - $vector("apple")$, which capture the concept of plurality.

Texts usually follow a meaningful order, which makes tasks in Natural Language Processing (NLP) tasks to represent the inputs as words in sequence. Recurrent neural networks, such as Long Short-Term Memory (LSTM) [17], excel at this kind of problem because they have an internal structure that enables information to be passed forward when processing different words. The main issue with this architecture is that, for long sequences, the computational power required to run these networks increases drastically. This problems was recently solved by transformers [36].

2.1.1 Transformer network

As previously mentioned, transformers [36] are neural networks that have been developed to solve the issue with long sequences of recurrent neural networks have. They usually follow and encoder-decoder architecture – although they can be trained to solve end-to-end tasks, where the encoder learns a set of features (embeddings) from the data, and the decoder uses this set of features to generate the output of the network.

The main difference from transformers to other recurrent networks is its self-attention mechanism, which enables the encoder to extract information from the whole sequence, and allows the decoder to assign more importance to specific elements of the input depending on which token is being processed at the time. In NLP, for example, self-attention consists in multiplying the value that represents a word w by a factor that represents whether other words affect the meaning of w. For example, in the sentence "The house is red", when passing through the word "'red", "house" should also have a high activation in the network as it is directly related to "house". By combining multiple "heads" of self-attention it is possible to map multiple information into a single word, which would often happen in a sentence. This is what we call multi-head attention, which will be described in more details later.

Figure 2.1 shows the transformer architecture. The network architecture starts

[1]Here $vector(w)$ means the vector representation of word w.

with an input embeddings layer, which transforms tokens of the vocabulary (e.g., words) into numeric vectors. Next, there is the positional encoding, used to describe the location of the tokens in the sequence. The positional encoding is generated by applying multiple sinusoidal functions to the index of the tokens and getting the results for each of the sinusoidal functions into a matrix. Therefore, there will be different values for each index.

After getting the embeddings with positional encoding, there is one block of operations, namely the encoder: it consists of one multi-head attention, which captures the importance of the other tokens of the sequence related to the current token. In this layer there are multiple attention heads working in parallel, and their results need to be combined. That is what the next layer does: it adds the results and normalizes them using batch normalization. Note that this layer also has a skip connection, which is a connection to the original data. Skip connections are often used in deep learning because they make the loss landscape smoother [37] for some problems.

Following, the encoder block has a feed forward layer followed by the same addition and normalization layer with skip connections previously mentioned. The output of the encoder is an intermediate representation of the input in a latent space, also known as embedding, and which is the core of the work proposed in this thesis.

This representation then passes through the decoder block, which has a few differences from the encoder. The decoder block starts using the output values and the same positional encoding as the input, but after that it uses a masked Multi-Head attention layer. The mask guarantees that the model is only able to use output tokens that appeared before the token that is being currently processed, avoiding that the model has access to current and future output values. On top of that there is an attention mechanism, which generates a representation of the previous values of the output. The representation of the input and the representation of the previous outputs are combined and used in the next part of the decoding block, which is equal to the encoding block. After the decoding block, there is a fully connected layer followed by a softmax layer to get the probabilities of the tokens. Note that, in the figure, "Nx" indicates that we have N encoder blocks and N decoder blocks. They are stacked on top of each other, so a network with 3 stacks would have a 3-stage encoder, then an intermediate representation, followed by a 3-stage decoder.

As already mentioned, the major breakthrough of transformers was the self-attention mechanism, illustrated in Figure 2.2. Before talking about it in more details, we make a brief introduction of the attention mechanism. The attention mechanism is usually found in sequence modeling and often used in encoder-decoder networks, where the decoder uses information about the input sequence in order to perform the decoding more efficiently. This is done by learning a weight for each element in the input sequence, which indicates how much attention the network should give to that element when processing the input. These weights are learned as part of the training process and can be used to weight the

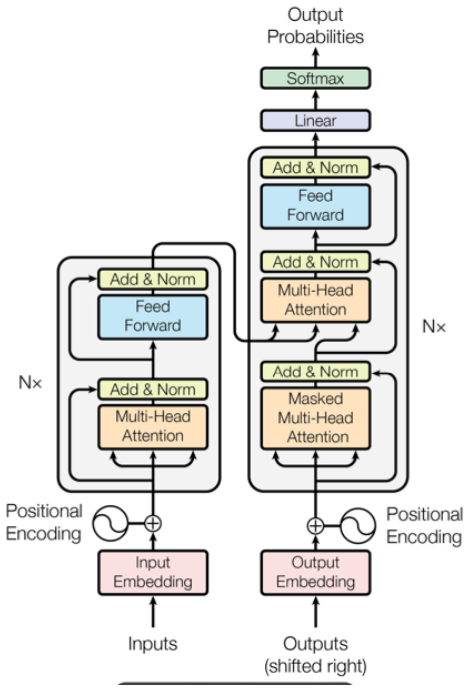


Figure 2.1: Transformer architecture proposed by [36].

contribution of each element to the final prediction made by the network.

In the transformer architecture, the self-attention mechanism is used to allow the model to consider the relationships between all the elements in the input sequence when processing it within the encoder or the decoder. The self-attention mechanism works by first computing a weight for each element in the input sequence, which indicates how much attention the model should give to that element when processing the input. These weights are computed using a dot-product attention function, which takes as input a query vector (represented by Q in the figure), a key vector (represented by K in the figure), and a value vector (represented by V in the figure) for each element in the input sequence. The query and key vectors are used to compute a similarity score between each element and the other elements in the input sequence, as shown in the left side of Figure 2.2. The result forms a mask-like output that should represent the importance of the tokens, which then are multiplied by their values in order to get the output of this layer. In transformers, multiple layers of self-attention (Scaled Dot-Product Attention) are then stacked, allowing the model to capture complex relationships by taking into consideration multiple aspects in the input or output sequence into consideration.

Note that the transformer works by passing through each element of the input in order to get an output. Hence, when creating new representations via embedding, it will produce a set of elements for each element of the input, i.e., in the case of NLP, for each word.

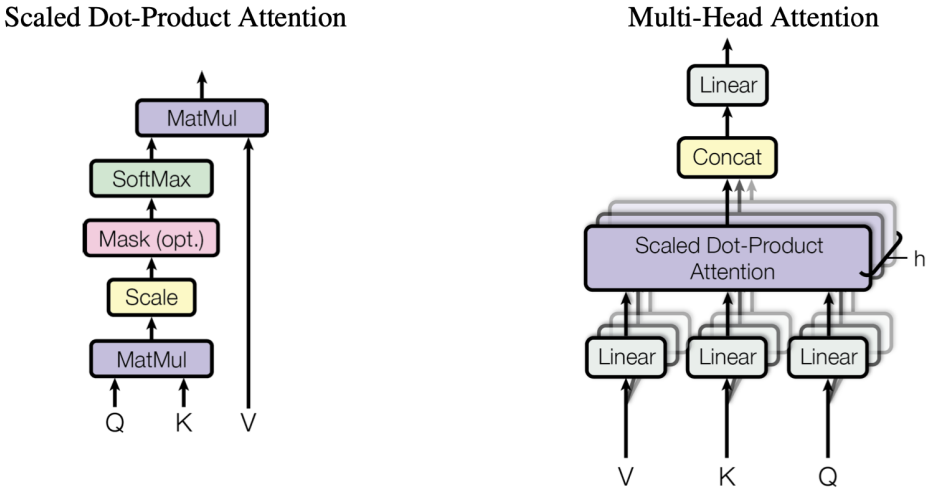


Figure 2.2: Self-attention mechanism proposed by [36].

2.1.2 Transformers for representing tree structures

So far, we have talked about transformers working with sequences of data, such as text. However, there are types of problems that can benefit from transformers but do not follow a natural sequential structure. This is the case of tree-like structures. Tree-like structures are specially useful for data representation in tasks that involve hierarchical or recursive data, such as compiler parsing or grammar parsing of texts. Because of that, transformer and other recurrent networks have been used and adapted to do this kind of representation [20].

The authors in [16] performed a theoretical analysis to confirm that transformers can learn tree structures, and have also proposed Tree Transformers, a variant that makes changes in the self-attention mechanism in order to encourage the attention heads to follow tree structures. They have tested the results in parsing problems. The authors in [22], in turn, have proposed a variational autoencoder to represent trees from symbolic regression and molecular analysis.

The work of [10] developed a tree-to-tree neural network with its own attention mechanism that boosted performance of program translation tasks. The authors in [41] have proposed a variational autoencoder for directed acyclic graphs (DAGs) using graph neural networks to encode the graphs in a latent space that allowed searching for DAGs with better performance. Finally, the authors in [39] have proposed a framework that encodes trees into a latent space and evaluated it on formula reconstruction and formula retrieval tasks.

Our work goes into this direction, but starts simple: we transform our trees in sequences, as done by [10], to create a neural network. Then, similarly to [39], we use the

latent space of the network as a representation for the tree.

2.2 GP for Symbolic Regression

As already mentioned, the objective of symbolic regression is to find a function f that maps the set of variables (or attributes) X describing a data example into a real-value y. In GP, these functions are represented by trees, built from a set of terminals (problem variables or constants) and operators (mathematical functions) predefined by the system user.

Having these set of candidate solutions of the GP, each representing a function, the search process starts following the principles of evolution. Individuals are evaluated according to their ability to solve the problem, generating their fitness. In symbolic regression methods, the fitness is generally related to the error of the function regarding the real output y. The fitness value of the individuals is then used to probabilistic selected a subset to undergo crossover and mutation operations, which are responsible for exploring the space of solutions. While crossover changes genetic material from two individuals, mutations replace parts of a single individual.

A problem that can appear in GP algorithms - for symbolic regression or other types of problems – is the population premature convergence. Premature convergence is most of the time related to a lack of population diversity, which can have many different causes, including a high selective pressure. There are a set of methods that can be used to increase population diversity, being one of the most well-known called fitness sharing [12]. Fitness sharing is a strategy that actively changes the fitness of individuals that are densely populated areas of the search space. In this approach, the original fitness f_i of an individual i is replaced by a shared fitness f'_i, given by:

$$f'_i = \frac{f_i}{m_i} \tag{2.1}$$

where m_i is a function that measures the number of individuals that are close to the individual i, which is called the sharing function, where

$$m_i = \sum_{j=1}^{N} sh(d_{ij}) \tag{2.2}$$

In Equation 2.2, N is the population size and d_{ij} is the distance between individuals i and j. The most widely used sharing function is defined in Equation 2.3:

$$sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\theta_{share}}\right)^{\alpha}, & \text{if } d_{ij} < \theta_{share} \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

In Equation 2.3, θ_{share} is the threshold of the dissimilarity, also called distance cut-off, and α is a constant that regulates the shape of the sharing function and is commonly set to 1.

Note that Eqs. 2.2 and 2.3 depend on a distance d_{ij}. This distance can be based on the individual phenotype – i.e., the distance between two fitness functions – or the individual genotype – the distances among the trees that represent the function. Usually, fitness sharing uses the phenotype as the distance, mainly because the computational complexity to calculate genotype distances is too high. The proposed embedded representation makes the time complexity as high as the size of the embeddings, making it more feasible to measure distances using the solutions genotype.

2.3 Embedded representations in evolutionary algorithms

Some works have already been successful in using different types of representations to improve performance in evolutionary algorithms (EA). One possibility is to use embeddings created by neural networks to make a representation of the space that can be better suited for the optimization problem.

In this direction, the authors in [23] used a sequence-to-sequence recurrent neural network to create a continuous latent space that encoded programs. The programs were represented as strings that were used to train a problem-agnostic autoencoder. This has enabled the use of continuous optimization methods for solving discrete optimization problems.

The work of [5] presents a methodology that uses an optimizer to generate solutions to a problem that satisfies one criterion, then those solutions are passed through a variational autoencoder (VAE). The VAE learns how to represent a set of solutions that satisfy the criterion, and therefore creates a biased representation. Next, a genetic algorithm (GA) is run on the latent representations of the VAE to find solutions that satisfy both the criterion from the first step and the objective function. Note that the evaluation of the objective function is made by using the decoder of the VAE on the latent representation to find a solution that is in the original problem space, and then applying the fitness function of the GA.

Chapter 3

Embedded representations for Genetic Programming trees

This chapter presents a new way of representing genetic programming (GP) trees so that operations such as distance between solutions can be performed in a more straightforward manner. We focus on a specific task that is commonly solved with tree-based GPs: symbolic regression [2]. The objective of symbolic regression is to find a function f that maps the set of variables (or attributes) X describing a data example into a real-value y, where $y \in \mathbf{R}$. Symbolic regression tasks have the advantage of not assuming the shape of the function f (e.g., linear or polynomial) *a priori*, and using the GP algorithm to find both the shape and coefficients of the function.

We propose to encode GP trees into vectors using an embedded representation created by a neural network. We hope the embedded representations can capture both the syntax and some semantics involved in the trees, as detailed in Section 3.1. In Section 3.2, we show how we trained the network to generate the embeddings, while Section 3.3 reports the obtained results.

3.1 From trees to distributed representations

Aiming to encode GP trees into vectors using an embedded representation created by a neural network, the following steps were followed: (i) generate a set of trees; (ii) represent the trees in a way it can be fed to neural networks; (iii) choose the neural network architecture and hyperparameters; (iv) train the network with generated input data from the GP; (v) extract the embedded representation from the hidden-layers of the trained network; (vi) propose a representation to the trees based on the embedded vectors.

These steps are illustrated in Figure 3.1. Given a set of trees, they will be linearized and mapped to a set of tokens. This new representation of the trees will be given as input

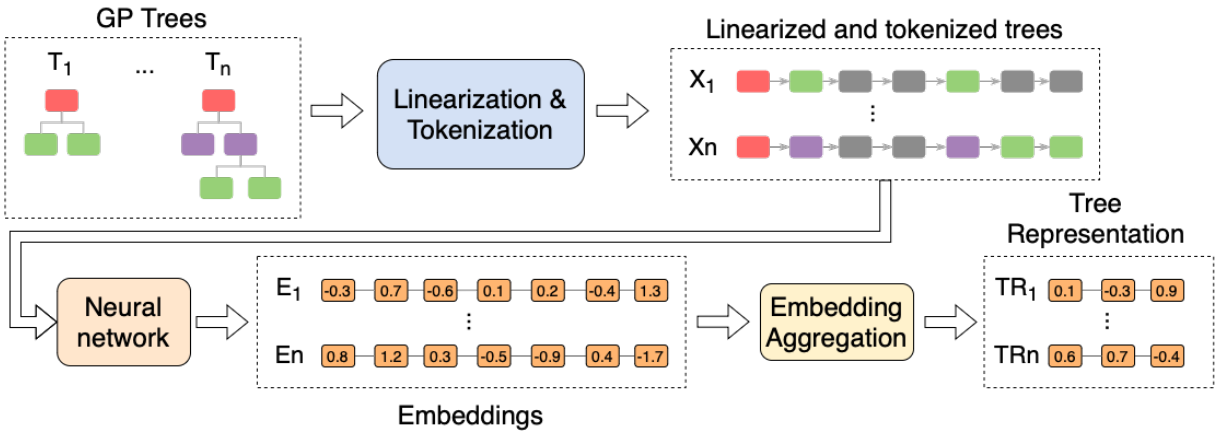


Figure 3.1: Proposed methodology considering a maximum tree depth of 3.

to an artificial neural network (ANN). The artificial neural network (ANN) will be trained to learn a new representation for this tree – using an encoder-decoder architecture. In this type of architecture, the encoder part of the network receives the input and transforms it into a more compact representation, expressed by the weights of the neural network in the encoding layer. This is the principle of representation learning: to use vectors of weights as intermediate representations to objects, words, images, etc.

The network will then output an embedded representation for each node of the tree, and these embedded representations will be combined by an aggregation process, generating the tree representation. Each phase of this process is detailed in the next sections.

Although it might sound a too complicated process that will add time to the already computationally expensive process of a GP, it is important to emphasize that, in theory, with enough data, the networks need to be trained only once, given a set of operators and terminals we want to deal with, and this model can then be simply reused over and over. As a seminal work, here we train a small and relatively simple network. But having enough data from different runs of GPs into different problems, one may be able to create a generic representation.

3.1.1 Generating GP trees

GP individuals are usually generated based on two sets defined when modeling the problem: a set of operators and a set of terminals. The set of operators should include mathematical or logical operators that are needed to solve the problem. The terminals are constants and the input variables of the problem, previously defined by X. The trees also obey a maximum depth, so they will not grow excessively.

In our case, we are dealing with symbolic regression and hence only mathematical operators are defined [2]. We use the four basic mathematical operators together with sine and square root, but these can be changed according to the nature of the problem being considered. The terminal set includes constant values uniformly picked from [-1, 1] and the number of variables of the problem being tackled.

Given a problem of interest and the sets of operators and terminals, we use the ramped half-and-half [2] method to generate the trees. This method works as follows. Given a maximum tree depth d_{max} and the number of tree to be generated (i.e., population size), it generates a uniform number of trees with depths varying from 2 to d_{max}. For each tree depth, half of the trees will be complete (i.e., the method first selects only symbols from the set of operators to generate the tree, and only adds terminals when the maximum tree depth is about to be reached) and the other half will be generated by randomly selecting symbols from both the set of operators and terminals, having a a variety of possible shapes.

3.1.2 Tree linearization and Tokenization

As we use ANNs to generate the embedded representations for GP trees, there are a few limitations on the shape of the input training data for ANNs that need to be addressed: it is necessary to transform the trees into a linear, non-hierarchical structure with a fixed size. For that, we first selected a maximum height for the tree. Any tree with a height above the maximum has its leaf nodes after the maximum height discarded. This is also the standard procedure for learning representations from text [8]). However, we also tested an alternative solution to this problem when dealing with specific problems, as discussed in Section 5.2. In the case of trees smaller than the maximum height, we create a special token for padding (<pad_token>) and complete the missing nodes of the tree.

Having these definitions, the first step was to prepare the tree to be linearized, as illustrated in Figure 3.2. We first normalize the trees by excluding nodes that extrapolate the maximum height (in the figure, the maximum height of the trees is 4) and filling incomplete nodes with the padding symbol <pad>. We then use a depth-first search (DFS), as proposed in [34], to linearize the trees.

Recall that, for GP trees, we generate a random constant, which is a continuous value, to be part of the tree. This representation is not straightforward, as embeddings are not the best yet in representing numbers. To simplify that, we categorize the constants using uniform bins in the -1 to 1 interval and use them together with operators and

terminals as inputs to the network.

At the end of this process, for each tree, we generate an input example. The vocabulary of these examples, i.e., the nodes that might appear in the training set, include the GP operators, the categorized constants, the terminals and the symbol <pad_token>. Each of these symbols is inserted into a hash table, and mapped into a numeric value, which we call token. The inputs for the network will be this tokenized tree.

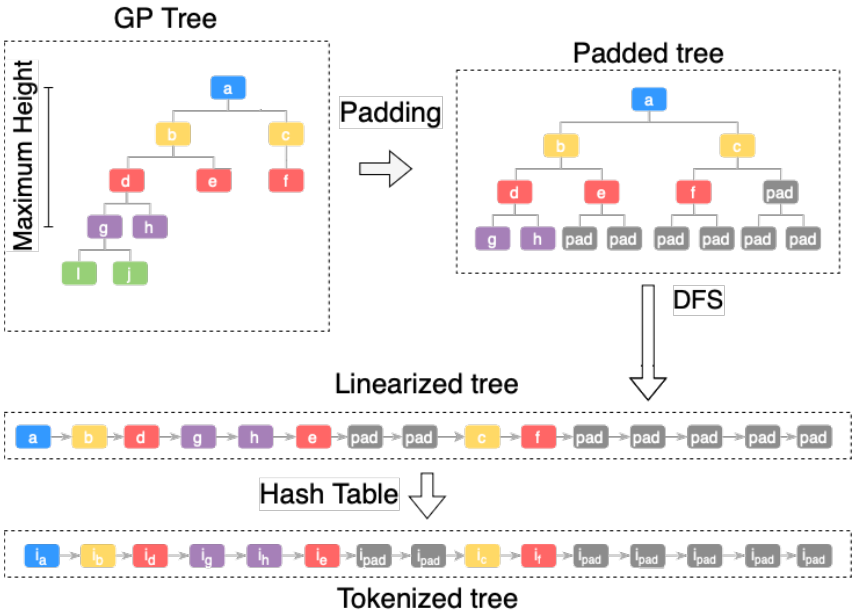


Figure 3.2: Tree linearization process.

3.1.3 Representation Learning Model

Having a training set of tokenized trees, the next step is to train the ANN to generate the desired representation. Any ANN could be used for that, and we performed tests with a transformer [36], as this is considered state-of-the-art in many representation learning tasks.

A transformer is a neural network architecture initially proposed to deal with sequence-to-sequence problems in Natural Language Processing (NLP). It can be trained in different ways, and here it follows an attention-based encoder-decoder type of architecture. A high-level picture of how the transformer is applied to our problem is illustrated in Figure 3.3. We will treat the transformer here as a black box, focusing only on what is needed to understand the process followed to generate the tree embeddings. For more details on the transformer architecture, the user is referred to Section 2.1.1.

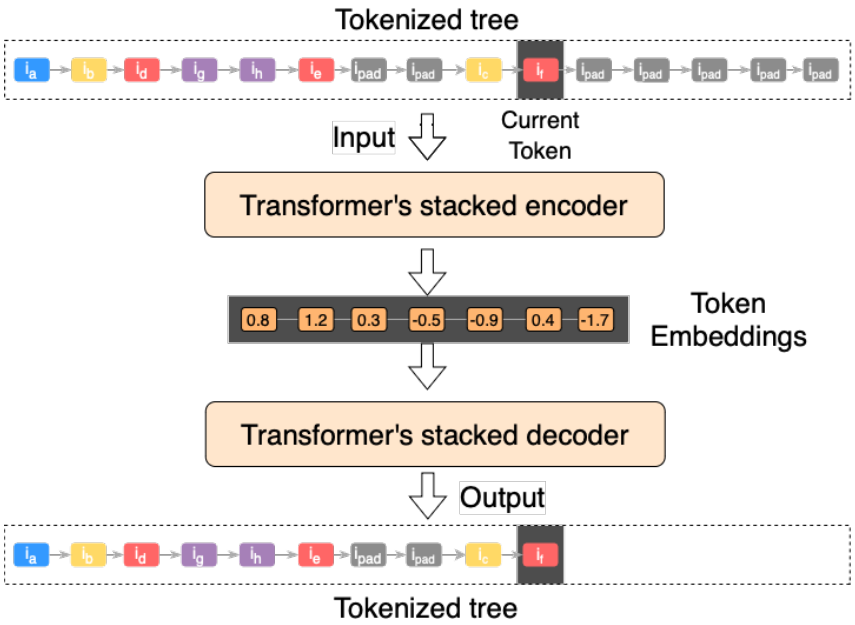


Figure 3.3: Transformer training process.

On a high level, the transformer stacked encoder (it can stack as many layers as needed, and here we use a single one, due to the simplicity of the problem) maps an input sequence (a tokenized tree) into an abstract continuous representation (embedding) that holds all the learned information of that input. At this point, it is important to mention that the network processes one token (one tree node, highlighted in the figure) at a time. However, while processing that token it still knows the surrounding tokens and learns to give higher weights to those more relevant to the current token. The self-attention mechanism is the one capable of weighting the significance of each part of the input data differently. This is what makes transformers powerful networks: their ability to learn context.

Once transformers deal with one token at a time, the embeddings generated are related to a single token and not the whole tree. Following, the decoder takes this continuous representation (embedding) and, step by step, generates a single output. This process is auto-regressive, as the first output of the decoder becomes part of its next input.

After the network is trained, given a tree we want to represent, we extract the embeddings of the tokens that compose the tree and need to combine them to generate the tree embedding. Some of the simplest ways of aggregating these embeddings is by summing, averaging or concatenating them. We have experimented with average and concatenation of the vectors.

Note that, as in any learning problem, the network will first be trained, and then can be later used to generate representations for any GP tree that shares the vocabulary of the trees used for training.

Table 3.1: Datasets used to train the model.

Datasets	Alias	Instances	Features	Dimensionality
4544_GeographicOriginalMusic	**DL1**	1059	117	High
505_tecator	DL2	240	124	High
588_fri_c4_1000_100	DL3	1000	100	High
590_fri_c0_1000_50	DM1	1000	50	Medium
633_fri_c0_500_25	DM2	500	25	Medium
592_fri_c4_1000_25	DM3	1000	25	Medium
197_cpu_act	**DM4**	8192	21	Medium
503_wind	DM5	6574	14	Medium
622_fri_c2_1000_50	DM6	1000	50	Medium
658_fri_c3_250_25	**DM7**	250	25	Medium
597_fri_c2_500_5	DS1	500	5	Low
635_fri_c0_250_10	DS2	250	10	Low
192_vineyard	DS3	52	2	Low
656_fri_c1_100_5	DS4	100	5	Low
228_elusage	**DS5**	55	2	Low
594_fri_c2_100_5	DS6	100	5	Low
537_houses	DS7	20640	8	Low
529_pollen	**DS8**	3848	4	Low
1096_FacultySal.	DS9	50	4	Low
595_fri_c0_1000_10	DSA	1000	10	Low

3.2 Model training

The neural network is the core component of the proposed methodology. The data used to train the network and the parameters used have a big impact on the embeddings generated. This section describes how we generate training data, train the representation model and evaluate it. To perform these three steps, we selected a set of 20 real-world regression datasets from Penn Machine Learning Benchmarks (PMLB) repository [30] with numeric features only, presented in Table 3.1. We grouped the datasets into three categories based on their number of features: low (up to 10 features), medium (from 11 to 99 features) and high (from 100 onwards) dimensional features. This was done because the number of features impacts model training: as the number of features increases the size of the vocabulary also increases. These data were then used to generate training and test data.

3.2.1 Generating training and test data

The input data for the ANN is a set of trees generated using a genetic programming algorithm for symbolic regression, described in Section 3.1.1. As already mentioned, the GP trees were generated using six arithmetic operators $(+, -, \times, \div, sin, sqrt)$ and the terminal set comprised the variables of the problem and constant values uniformly picked from [-1, 1]. As previously mentioned, constants were discretized. In NLP we call vocabulary the set of words that a model will be able to understand. For our problems, the vocabulary V will contain the operators, the variables and the discretized values of constants. On top of that, the vocabulary includes the special token for padding (<pad_token>).

We generated 100,000 trees using the ramped-half-and-half method with a maximum tree depth of 7. However, we later decided to add to this set also trees evolved by a GP algorithm. The idea was to simulate how the embeddings would be used by a GP. Hence, we run a GP algorithm with 900 individuals for 55 generations with tournament selection of size 7. Crossover and mutation probabilities were set to 0.8 and 0.05, respectively. The maximum height of the tree was kept as 7. As we already had random trees, the trees from the first 5 generations were disregarded, and the 45,000 trees from the last 50 generations were merged with the set of 100,000 randomly generated trees.

These 145,000 trees were randomly sampled to generate datasets with different sizes for network training, split into training and validation sets in a 70/30 proportion. The training data was used to train the network and the validation data to evaluate the network reconstruction error to select the model with the best performance to represent trees. In order to generate test data, the GP was run with a new seed to gather a new set of trees , used in an external evaluation. A total of 30 trees was generated for test,using both random and evolved trees. The idea is to have a small set of trees that could be inspected, out of which some parts of them would be more correlated with each other than the rest. This set is made to make an initial evaluation of the models, which will evaluate the correlations between embeddings distances and other metrics such as differences in fitness, and edit distance.

3.2.2 Model evaluation

With the training and test data in hand, we need to run the network and perform parameter tuning. However, measuring the quality of the representations generated with-

out a proxy task is not simple. Given the lack of objective criteria, we assessed the quality of the embeddings generated using two metrics. First, for each pair of trees, we calculated their syntactic distance considering their original representations using the edit distance [40] described in Eq. 3.1.

$$D(T_1, T_2) = \begin{cases} 0 & if \quad T_1 = T_2 \\ min \begin{cases} D(T_1 - l_1, T_2 - l_2) + cost(l_1, l_2) \\ D(T_1 - l_1, T_2) \\ D(T_1, T_2 - l_2)) \end{cases} & otherwise \end{cases} \tag{3.1}$$

Next, we calculated the distance from the embedded representations using both the cosine and the Euclidean distance. Finally, we calculated the Pearson correlation (ρ) between the distances in the original representation and the distributed representation, generating our first syntactic metric.

Next, we calculate the absolute fitness difference between pairs of individuals considering both representations (which we call fitness distance). The fitness of a tree (individual) is given by its mean squared error in the validation set. Again, we measure the Pearson correlation of the fitness distances between the two representations. Since we do not make any explicit semantic modelling, we expected that the models obtain a higher correlation with the edit distance and a lower correlation with the fitness distance.

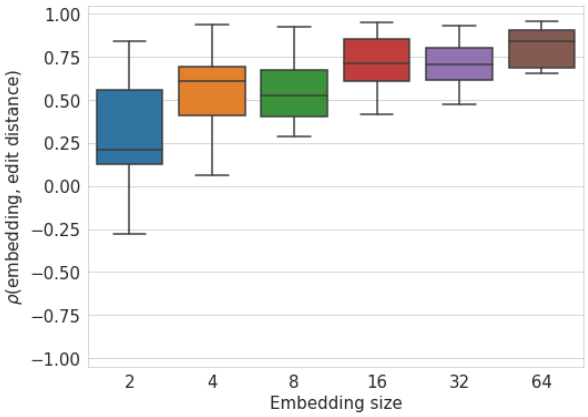
3.2.3 Parameter tuning

The transformer model trained had a single stack – a single encoder and a single decoded – due to the small vocabulary size of the problem when compared to regular the ones found in NLP. We used KL divergence loss to train the network, and dropout between each layer, with 30% probability. The model was trained for a maximum of 50 epochs using an early stop strategy – if the validation loss did not go down for 5 epochs, the training stopped and the weights of the best epoch were restored.

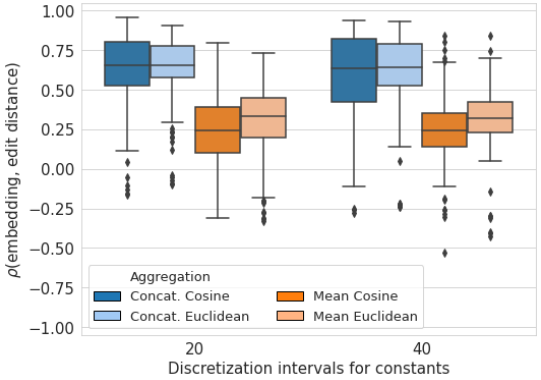
We started by performing some simple parameter tuning. To reduce computational cost, we selected 5 datasets, marked in bold in Table 3.1. For each of the chosen datasets we trained a model varying the parameters described in Table 3.2 for 50 epochs. In Table 3.2, embedding size represents the size of the distributed representation for each term of the vocabulary, embedding aggregation the method used to generate the final representation and training size, the number of samples used for training. The parameter "discretization intervals for constants" represents the number of intervals used to uniformly discretize the values of the constants (which vary from -1 to 1).

Table 3.2: Subset of parameters tuned.

Parameter	Values
Embedding size	2, 4, 8, 16, 32, 64
Embedding aggregation	Concatenation, Mean
Training size	5k, 10k, 25k, 50k, 100k
Discretization intervals for constants	20, 40



(a) Model embedding size



(b) Constant interval number and aggregation type

Figure 3.4: Correlation between the embedding distance and the edit distance

Figure 3.4a shows that increasing the embedding size also tends to increase the correlation with the edit distance, which is over 0.9 for most datasets with size 64. Each bar represents the box plot considering the 5 datasets used for tuning. This is expected, as larger embeddings can store more information about the data used from training.

Figure 3.4b shows that aggregating embeddings by the concatenation of representations presented a higher overall correlation with the edit distance than averaging them. Also, note that the number of intervals used for discretizing the constants had low impact on the results. Regarding the fitness correlation, most of the parameters do not make a significant difference in the results, so the choice of parameters was made considering solely the correlation with the edit distance.

When choosing a single model, one can use a smaller number of discretization intervals (20), concatenate the outputs of the network and choose the size according to the application at hand. The smaller the model, the more compact the representation but the smaller the correlation with the edit distance. Note that in the case where a correlation with fitness is more relevant, none of the parameter tests seemed to have a relevant influence on that.

3.3 Experimental Analysis

This section presents an overview of the experimental analysis in which multiple transformer models were trained. As we had already found the parameters that obtained the most promising results, here we used the model trained using those parameters to evaluate the representation generated from our method and compare it to other metrics that are currently used in the GP field.

3.3.1 Model effectiveness and generalization

After understanding the impact of the transformer parameters on generating embeddings for GP trees, we trained models to generate tree representations for the 20 selected datasets considering the parameters chosen. Next, we performed a cross-evaluation of the models to understand generalisation. Each model was used to generate embeddings for individuals from the other 19 datasets that had a smaller number of input variables than the trained model. Again, instead of evaluating the loss, which may not necessarily correlate to a better representation from the GP standpoint, we used the same evaluation metrics described in the previous section: Pearson correlation for the edit distance and the fitness distance.

The heatmap in Figure 3.5 shows these correlations for the 20 datasets with the edit distance and Figure 3.6 presents the correlation with the fitness distance. Blanks represent datasets where the model could not be tested, as the number of variables of the input dataset was smaller than the number of variables of the test dataset. Note that the correlations vary from 0.38 to 0.99. This is good for understanding how the embeddings compare to a traditional syntactic distance but does not help gain insights on semantics or generalisation.

Most of the datasets presented a better average correlation with the edit distance, and in most cases higher than 0.9. This is expected since the parameters were selected based on them. Regarding the fitness correlation, it only presented high values for a few cases, where they reached up to 0.8. Thus, for the general case, it is reasonable to assume that the embeddings can capture syntactic relationships. They are generally not able to represent semantic relationships and recall they were not trained for that.

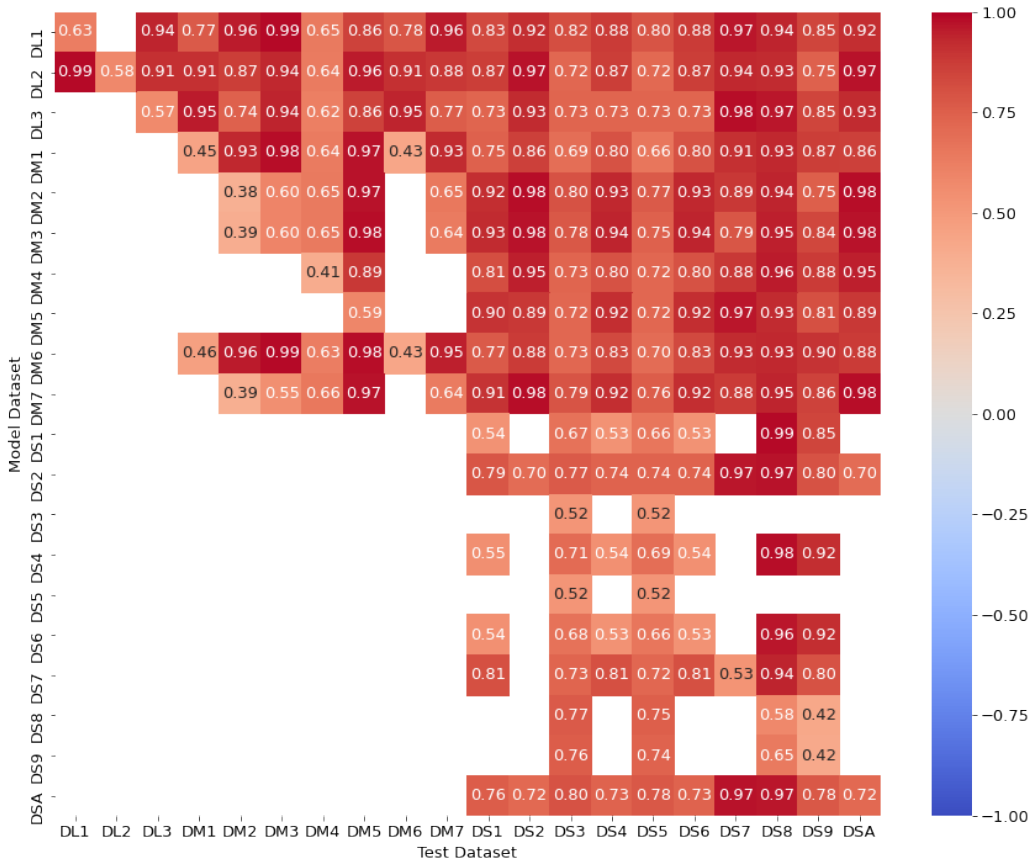


Figure 3.5: Heatmap of the correlation of the embedding representation with edit distance.

3.3.2 Representation evaluation

However, to actually see whether the syntax of the trees is present in the embeddings, we need to manually evaluate a set of trees. With that in mind, we designed a set of 17 simple trees, shown in Table 3.3, to evaluate the embeddings themselves. The alias of the trees show their number, e.g., T1, and can have two suffixes. Suffix "R" indicates the expression is the same as the one without the suffix, but the left and right nodes were reversed – see T1 and T1R. The idea here is to show how the commutative property holds since the trees are semantically the same but have a slightly different syntax. The suffix "EQ" refers to trees with the same semantics as the original numbered one but with a higher number of nodes and different syntax.

Each of these trees was given as input to the pre-trained model of dataset DM7 (the choice of DM7 was arbitrary but based on the fact that it is a mid-size dataset

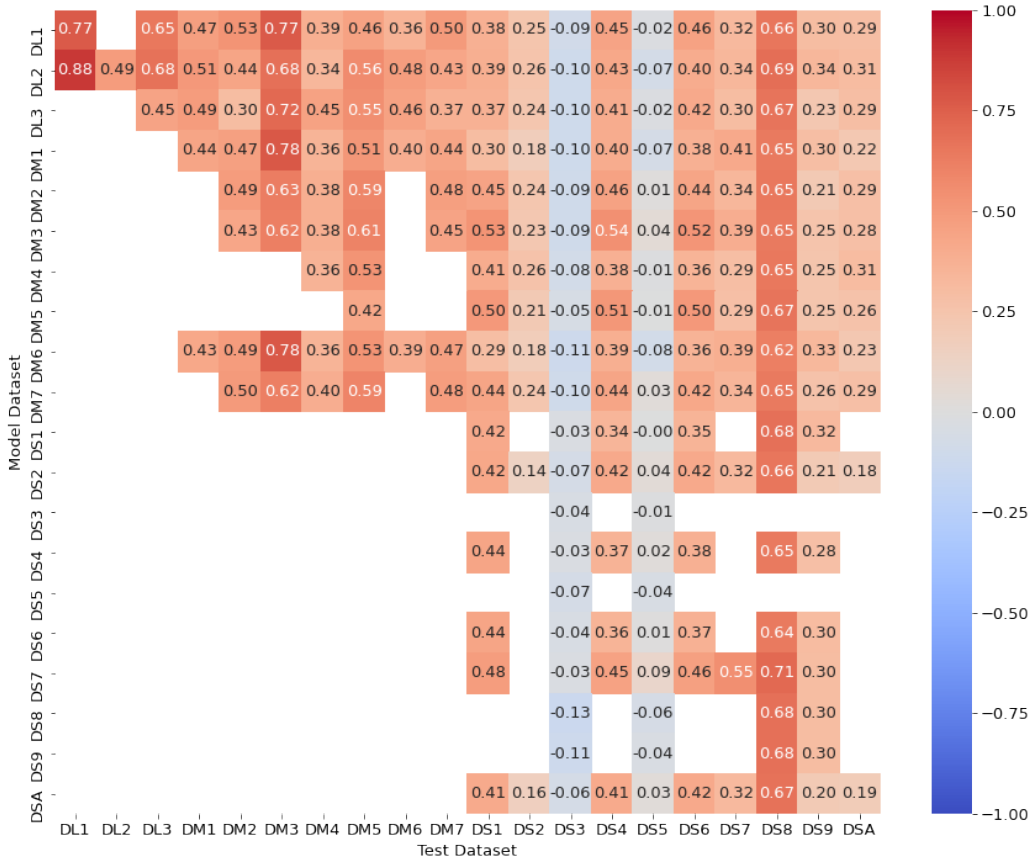


Figure 3.6: Heatmap of the correlation of the embedding representation with the fitness distance.

and generalization from a larger to smaller dataset could be evaluated) and had their embedding generated. We then calculated distances between these pairs of trees using the embedding of size 64 and the cosine distance, shown in Figure 3.8. In the heatmap, trees that should be similar appear close to each other in the matrix. Hence, we expect that the diagonals and their neighbour cells are much darker than the other parts of the matrix. Analysing the values of distance, we observe that the trees and their reverse values are always the closest among the possible trees. However, when we look at the trees with the suffix "EQ", we can see that the syntactic is better captured than the semantics. For example, the most similar tree to T3EQ – div(X0, add(1, 1)) and T4 –add(div(X0, 0.5), 1). Syntactically, these trees are indeed more similar than T3EQ is to T3. In general, trees with the suffix "EQ" are closer to each other and then closer to their respective semantic equivalent trees. This shows us that the current model gives more weight to syntax because that is the information we gave to the model. The next step is

Table 3.3: Hand-designed trees for testing embedding distances.

Alias	Tree expression	Alias	Tree expression
T1	add(X0, 1)	T4	add(div(X0, 0.5), 1)
T1R	add(1, X0)	T4R	add(1, div(0.5, X0))
T1EQ	add(X0, add(0.5, 0.5))	T5	div(X0, div(0.2, X0))
T2	div(X0, 0.5)	T5R	div(div(X0, 0.2), X0)
T2R	div(0.5, X0)	T6	add(mul(X0, div(X0, 0.5)), 1)
T2EQ	mul(X0, add(1, 1))	T6R	add(1, mul(div(0.5, X0), X0))
T3	mul(X0, 0.5)	T7	sub(mul(X0, 0.3), sub(X0, 0.8))
T3R	mul(0.5, X0)	T7R	sub(sub(0.8, X0), mul(0.3, X0))
T3EQ	div(X0, add(1, 1))		

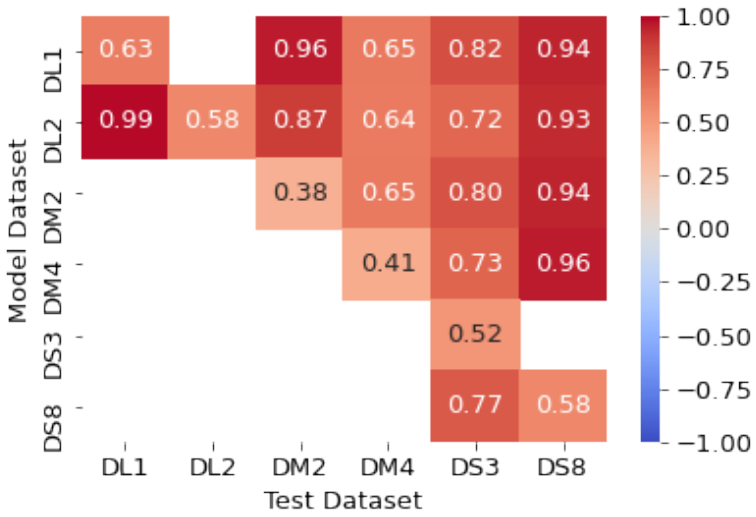


Figure 3.7: Heatmap of the correlation of the embedding distances and the edit distance.

to provide explicit semantic information to the model. Regardless of how much semantics we have involved, the model can be used as a replacement for tree edit distance, being much cheaper, as discussed later.

3.3.3 A note on execution time

The experiments presented so far were run on an Intel Core i5-9300H with 16GB of memory. Considering DNN models are usually trained on GPUs, it is important to notice that the times reported here can be at least 10 times lower if the models are trained on GPUs. As explained, the methodology works in two phases: we first train the model and then use it to generate representations for new individuals. Regarding the training time

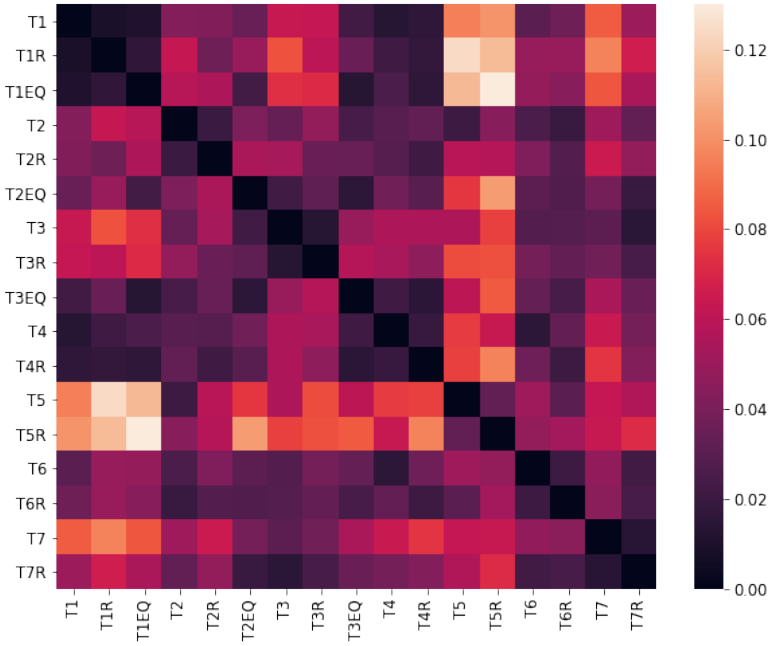


Figure 3.8: Distances considering the embeddings (size = 64).

(always with 100k trees), it varies significantly from one dataset to the other. For DL1, for example, which has 117 input features, 50 epochs took 10h53min to train. For DM7, which has 25 features, the training time was 3h36min. DS8, which has only 4 features, had an early stop (which happens when the validation loss does not change for 5 epochs) at the 37th iteration, which took 10h14min. We can say that learning from datasets with smaller sets of features seems more difficult than learning on larger datasets.

Chapter 4

Case studies on Distributed Representation of GP Trees

Chapter 3 provided evidence that embedded representations have properties that are common to the edit distance, a metric commonly used to calculate distances between trees. In this chapter, we use the embedded representations in more realistic scenarios, integrating them into a GP-based symbolic regression algorithm.

We analyze two different case studies. In the first, we look at how the distributed representation impacts the way we measure the diversity of the population. For that, we use the commonly used technique for improving diversity in GP called fitness sharing, explained in detail in Section 2.2. In this case, the distance between trees used to measure diversity is computed from the embeddings instead of using the edit distance of the trees, the fitness or other metrics.

In our second case study, we use the embeddings generated from one model to propose different types of visualizations of GP candidate solutions and how they evolve. The first shows the search space explored by the GP and the second illustrates the trajectory followed by the algorithm to find the best solution to the problem. Before that, we briefly describe how symbolic regression algorithms with GP works.

4.1 Individuals' diversity and fitness sharing

In evolutionary algorithms in general, including GP, diversity refers to the presence of a range of different solutions within a population of individuals. Diversity allows evolutionary algorithms to explore a larger space of potential solutions and helps avoid local optima [2]. Diversity might also be useful to identify (premature) convergence, and can also be a measure evaluated when adjusting parameters

As already mentioned, diversity can be measured according to two characteristics of GP individuals: the genotype (represented by the tree structure) or the phenotype

(usually indirectly measured by the fitness). According to [9], the simplest way to measure genotype diversity is to make a node-to-node comparison between programs. This can be measured by the edit distance. The work [31] used edit distance to analyze differences between the best individuals and the whole population. A modified version of the edit distance that also considers the depth of the node was used by [13]. The work [7] used the top proportion of the GP tree as a genetic marker and introduced a measure called genetic marker diversity, which is defined as the density of genetic markers. A metric based on isomorphism, in turn, was described in [6], which defined three-tuple of counts of terminals, non-terminals and the depth of the tree and compared the number of unique triplets, since finding out if the trees are isomorphic would be too computationally expensive.

On the other hand, phenotype diversity measures the difference in the behaviour of the programs, which for GP usually translates into fitness. The simplest way to represent phenotype diversity would be to use the number of unique values of fitness in the population. [32] defined a metric based on the entropy and free energy of the fitness in a population, where the entropy was found to be positively correlated with the diversity. [18] assessed diversity as the number of distinct behaviours in the population, where a distinct behaviour is considered to be when the distance of semantics between individuals is above a threshold. [28] presented phenotype diversity as the distribution of phenotypic characteristics in the population. [9] used PCA to reduce the dimension of the semantics and then defined partitions in the data to calculate a diversity measure, which is the entropy of the partitions. In this work, we refer to Genotype and Phenotype diversity as Syntactic and Semantic diversity, respectively.

In this section, we use distances between tree embeddings to generate metrics of diversity. Diversity was measured using two metrics that evaluate different aspects of the trees. For the syntactic diversity measurement we used the Pseudo-isomorph described in [6]. As already mentioned, this metric defines, for each tree, a three-tuple of terminals, non-terminals and depth and then counts the number of unique tuples. For the semantic diversity measure we used the Transformed Semantics Entropy, described in [9]. This metric uses a principal component analysis (PCA) to reduce the dimension of the target semantics vector (i.e., the vector containing the values of y for each input example) into a two-dimensional space. The 2D space is then split into 8 quadrants and the entropy of each partition is calculated. Next, we detail our experimental setup.

Table 4.1: Datasets for experiments with diversity.

Dataset	Instances	Features	Dimensionality
197_cpu_act (DM4)	8192	21	Medium
658_fri_c3_250_25 (DM7)	250	25	Medium
228_elusage (DS5)	55	2	Low
529_pollen (DS8)	3848	4	Low
BHouse (BHS)	506	13	Medium
Concrete (CCT)	1030	8	Low
RedWine (RWN)	1599	11	Medium
WhiteWine (WWN)	4898	12	Medium
Diabetes (DBT)	442	10	Low
Vladislaveva-4 (VLAD4)	6024	5	Low
Pagie-1 (PG1)	1352	2	Low

4.1.1 Experimental Setup

The experiments presented in this section are divided into two parts. First, we conduct one experiment to help us select the best model to generate the embeddings we are going to use to calculate distances. Datasets used in this part can be found in Table 4.1: In the upper part of the table, there are 4 out of the 5 datasets that lead to models with the best results from the previous chapter - we removed *DL1* due to its number of features, which would make it harder for us to test it on models trained in medium-sized datasets. This is true because, if we use a dataset that has a way larger number of features than a model was trained on, it would introduce too many tokens that are unknown to the trained model. The motivation to remove DL1 was to avoid this situation.

In the lower part of the table are the datasets used in the second part of our experiments: there are new datasets, borrowed from [9] and commonly used to test GPs and diversity. For them, we run four different versions of GP: (i) a GP without any diversity mechanism, (ii) a GP with fitness sharing, using fitness difference (or fitness distance) to measure diversity, (iii) a GP with fitness sharing, using tree edit distances to measure diversity, and (iv) a GP with fitness sharing but using the representations generated by our transformer to calculate distances between individuals. As explained in Chapter 2, fitness sharing is a common mechanism used to ensure population diversity.

In this second part of the experiments, the GP parameters were kept the same as the ones reported in Chapter 3. The values of θ_{share} were customized to each dataset and were defined based on an *ad-hoc* analysis of pairwise distances when using different metrics. In this case, we ran a GP without any fitness sharing. After that, we calculated pairwise distances for each of the distance metrics - fitness distance, edit distance, and the distance based on the embeddings - and analyzed them using the violin plots shown

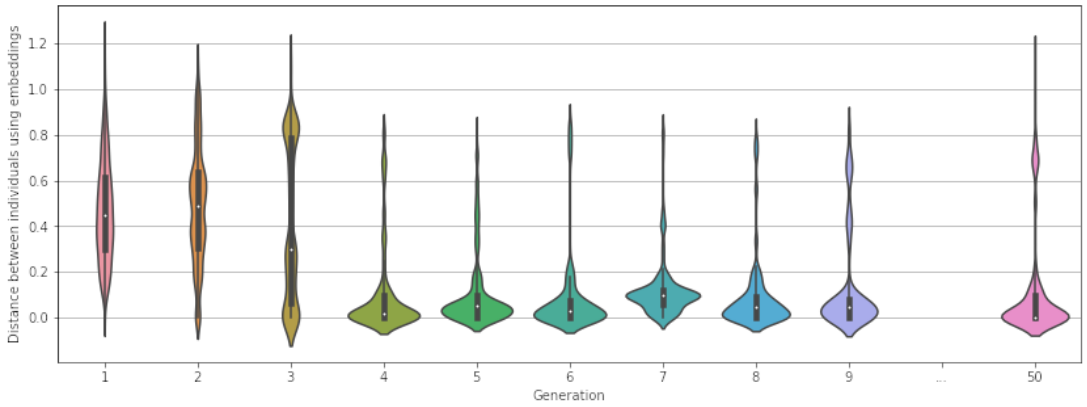


Figure 4.1: Violinplots of the individual's pairwise embedding distances for the dataset RedWine.

in Figures 4.1, 4.2, and 4.3. The figures show the violin plots for the datasets RedWine, Pagie-1, and Concrete, respectively, where each violin shows the results of pairwise distances of the embedded representations in one generation. In every case, there were signs of convergence before the 10th generation, *i.e.* the variance of the pairwise distances dropped drastically. According to the format of the distribution, we chose the value of θ_{share}. When the distribution was not bimodal, we used the Interquartile Range (IQR). For bimodal distributions, we used half the value of IQR. For example, in Figures 4.1 and 4.3, we chose generations 4 and 3, respectively, as the generation where convergence started. Both distributions are unimodal, and hence we use the IQR of the distances between the individuals in the respective generations as the value of θ_{share}. In Figure 4.2, in contrast, there seem to be signs of convergence in generation 3, and it does not seem to be unimodal, so we used half the value of the IQR as the value of θ_{share}. Although we just showed the graphs for finding the values of θ for the embeddings, we followed the same procedure for fitness difference and edit distance. They presented similar behaviour with a much higher variability in the first generations. In this case, it was necessary to ignore the first 1-3 generations to visualize distributions properly.

4.1.2 Model generation and selection

Having the GP parameters set, we need to define which version of the transformer will be used to generate the embeddings. As in the previous chapter we only evaluated how different network parameters impact edit distance and fitness correlations, here we perform additional tests considering the same parameters tested before and presented in Table 4.2. We trained the transformers again using each combination of embedding

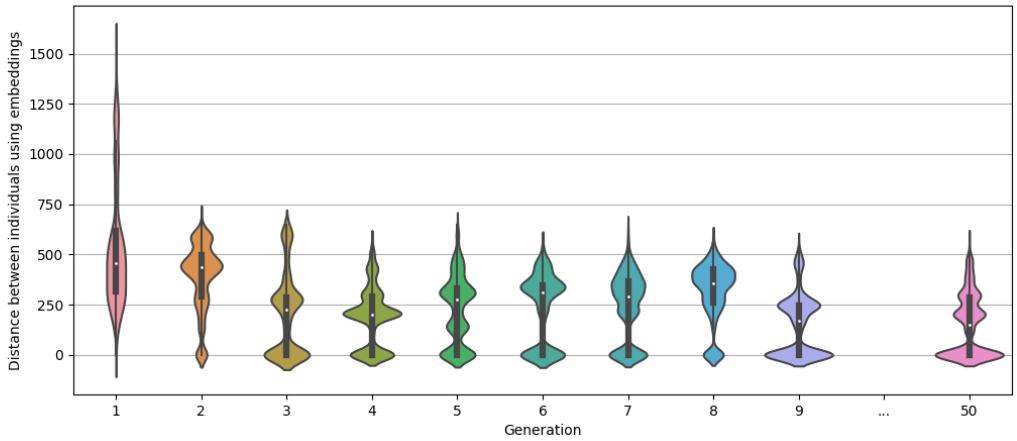


Figure 4.2: Violinplots of the individual's pairwise embedding distances for the dataset Pagie-1.

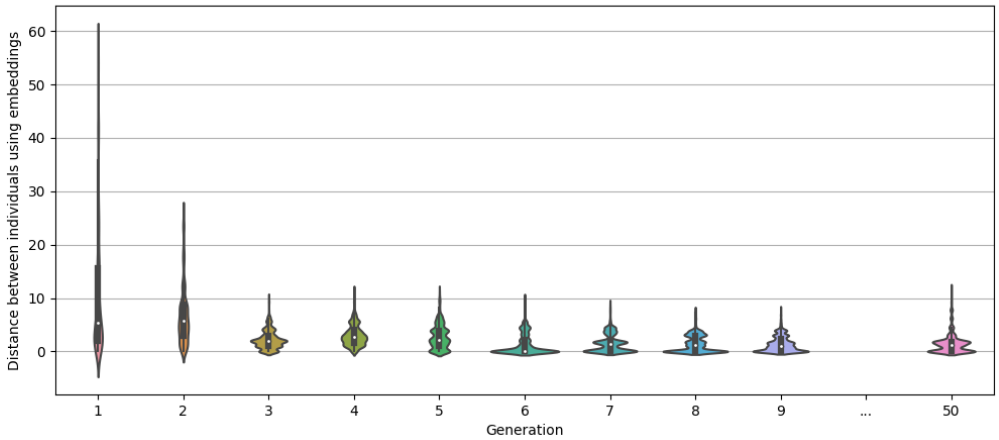


Figure 4.3: Violinplots of the individual's pairwise embedding distances for the dataset Concrete.

size and training dataset. Then we evaluated each of the models using the different combinations of distances and embedding aggregation strategies. This generated a total of 9 models (3 embeddings sizes in 3 datasets), and 36 different parameters in total (9 models evaluated with 4 different combinations of distance and embedding aggregation), which yielded 1800 symbolic regression runs for each dataset in Table 4.1 - 50 GP repetitions[1] for each.

Considering the diversity metrics we chose, we always evaluate the model that generates an individual regarding the median mean squared error (fitness), the median pseudo-isomorph (syntactic diversity), and the median transformed semantic entropy (semantic diversity) across all runs. For fitness, the median fitness considers the best individual returned in the last generation. For syntactic and semantic diversity, we consider

[1]For fitness sharing using edit distance we used 15 repetitions due to time constraints.

Table 4.2: Parameters tested for the model selection experiment.

Parameter	Values
Embedding Size	8, 16, 64
Training dataset	DL1, DM4, DM7
Distance	Cosine, Euclidean
Embedding aggregation	Concatenating, Average

the average pseudo-isomorph/transformed semantic entropy across all the generations.

As we ran many models and was difficult to evaluate all of them, for reporting the results the models were split into groups based on the distance method used to compare embedding (Euclidean distances vs cosine distance) and embedding aggregation type (concatenation or average) , leading to 4 groups. Within each group we picked the model that had the smallest fitness overall. After that, between groups winners, we ran a Friedman test, to check if there was a model that had a significantly better result than the others regarding fitness and diversity. In case the test was significantly different, we used the model with the smallest median. In case the test was not significantly different we could choose any of the models, but we chose the one with the smallest median to be consistent with the previous choice. Please refer to Appendix A for the results of all the models that were tested for each of those metrics.

Tables 4.3, 4.4, and 4.5 present the results for the fitness, the syntactic diversity, and semantic diversity, respectively, when running a GP where the diversity of the individuals is generated using embeddings produced by transformers with the settings that had best fitness out of each group. The upper part of the table shows the parameters of the models selected within each group, while the bottom part shows the use of these models in a set of new datasets. Results in bold show the metrics with the best results of each row. The model selection was based only on the results from Table 4.3, which contains information about the fitness of the models. For Symbolic Regression problems, it does not make sense to consider diversity metrics alone. They must be complementary to the fitness, which is the main task.

When running a Friedman test on the data presented in Tables 4.3, 4.4, and 4.5 they did not present statistically significant differences between each other, and the p-values were 0.263, 0.504, and 0.409, respectively. Thus, to represent the embeddings we chose the model that was trained on the dataset DM7 with embedding size 8, mean as embedding aggregation, and cosine distance as the distance metric between embeddings. This combination presented the best absolute results overall when compared to the other combinations.

When comparing the results in the tables 4.3, 4.4, and 4.5 there were no clear patterns that related median MSE and median diversity. For example, when comparing Table 4.3 to the other two, when a model had the best fitness value, in 5 datasets it had

Table 4.3: Parameters of the best-performing models and their results of median MSE (fitness) when applied to a new group of datasets.

	Parameter of the best embedding models			
Training dataset	DM4	DM7	DM7	DM7
Emb. Size	16	8	8	16
Emb. type	concat	mean	concat	mean
Distance	cosine	cosine	Euclidian	Euclidian
Test dataset	**Median MSE (IQR)**			
Boston House	49.5 (31)	**45 (24)**	61.3 (35)	51.2 (29)
Concrete	259 (77)	244 (100)	274 (75)	**231 (92)**
DM4	**628 (500)**	838 (560)	1050 (920)	802 (400)
DM7	1.09 (0.045)	1.07 (0.063)	**0.962 (0.085)**	1.098 (0.013)
DS5	441 (550)	**415 (470)**	451 (480)	424 (500)
DS8	9.05 (1.4)	**8.25 (1.7)**	9.46 (0.55)	9.67 (0.064)
Red Wine	0.808 (0.5)	**0.763 (0.39)**	0.807 (0.46)	0.772 (0.4)
White Wine	0.949 (0.38)	**0.878 (0.24)**	0.957 (0.43)	1.01 (0.31)
Diabetes	16900 (9300)	17600 (5300)	**16500 (12000)**	16800 (5500)
Pagie-1	0.18 (0.044)	0.156 (0.047)	**0.147 (0.022)**	0.191 (0.062)
Vladslaveva-4	0.04 (0.01)	0.0384 (0.0039)	0.0379 (0.0021)	**0.0379 (0.0007)**

less diversity, in 4 datasets it had better diversity and in two other datasets there were no difference in diversity. Making this same evaluation for the syntactic diversity, when the model had the best fitness, in datasets the syntactic diversity was lower, for one dataset it was higher and in the other 6 cases they were similar. However, notice that both diversity results seem to be correlated: When the semantic diversity was the highest, in 8 out of 11 datasets the syntactic diversity was also the highest.

The way that we calculated these metrics can also have an impact on these results. We use the median of the average diversity across all the generations. If we restrict the average to certain generations instead of using all of them, or if we change the average for other metrics, such as maximum diversity, the difference between minimum and maximum may lead to different results regarding correlations between semantic and syntactic diversities.

After selecting the model, we tested it against the baseline models for the same metrics again. In this test, we had 4 different scenarios: for baselines, we used (i) a plain symbolic regression, a symbolic regression with fitness sharing using (ii) fitness distance, and (iii) edit distance. Which were compared to the results using fitness sharing using the distance of the embeddings generated by the selected model. Next, we performed a Friedman test with a Nemenyi *post-hoc* test - for each of the three metrics - to evaluate which pairs of models were statistically different, as suggested in [11]. The results are presented in Tables 4.6, 4.7, 4.8 comparing fitness, semantic diversity and syntactic diversity, respectively.

When comparing with the baselines every Friedman test had statistically significant

Table 4.4: Parameters of the best-performing models and their results of Median Entropy Transformed Semantics diversity when applied to a new group of datasets.

Parameter of the best embedding models				
Training dataset	DM4	DM7	DM7	DM7
Emb. Size	16	8	8	16
Emb. type	concat	mean	concat	mean
Distance	cosine	cosine	Euclidian	Euclidian
Test dataset	**Median Entropy Transformed Semantics (IQR)**			
Boston House	1.56 (0.36)	1.42 (0.37)	**1.62 (0.27)**	1.59 (0.29)
Concrete	1.3 (0.31)	1.27 (0.36)	**1.35 (0.31)**	1.29 (0.31)
DM4	1.11 (0.38)	1.15 (0.34)	**1.25 (0.34)**	1.24 (0.26)
DM7	**1.74 (0.34)**	1.17 (0.25)	1.51 (0.22)	1.48 (0.39)
DS5	1.5 (0.47)	1.46 (0.48)	**1.56 (0.38)**	1.54 (0.41)
DS8	1.06 (0.26)	0.824 (0.52)	**1.42 (0.39)**	1.17 (0.31)
Red Wine	**1.7 (0.24)**	1.71 (0.28)	1.43 (0.11)	1.6 (0.25)
White Wine	**1.63 (0.2)**	1.62 (0.19)	1.32 (0.19)	1.55 (0.27)
Diabetes	**1.09 (0.42)**	1.08 (0.42)	1.08 (0.52)	1.05 (0.38)
Pagie-1	**2.06 (0.26)**	2.05 (0.31)	1.31 (0.19)	1.59 (0.25)
Vladslaveva-4	2.07 (0.13)	**2.36 (0.095)**	1 (0.56)	1.46 (0.26)

Table 4.5: Parameters of the best-performing models and their results of Median Pseudo-Isomorph diversity when applied to a new group of datasets.

Parameter of the best embedding models				
Training dataset	DM4	DM7	DM7	DM7
Emb. Size	16	8	8	16
Emb. type	concat	mean	concat	mean
Distance	cosine	cosine	Euclidian	Euclidian
Test dataset	**Median Pseudo-Isomorph (IQR)**			
Boston House	29.22 (3.804)	28.75 (2.436)	**29.96 (2.779)**	29.17 (3.789)
Concrete	29.02 (2.721)	28.48 (2.613)	**30.08 (2.471)**	29.27 (1.309)
DM4	28.26 (3.441)	27.3 (2.77)	**28.98 (3.696)**	28.37 (2.623)
DM7	**29.96 (2.485)**	29.69 (2.618)	21.38 (4.544)	28.25 (4.564)
DS5	29.17 (2.373)	28.63 (2.627)	**30.67 (2.73)**	29.26 (1.873)
DS8	28.21 (2.52)	28.85 (2.966)	30.6 (3.564)	**31.01 (2.02)**
Red Wine	**29.68 (2.466)**	28.58 (2.01)	25.47 (4.039)	29.42 (2.99)
White Wine	**29.5 (2.48)**	28.2 (3.819)	25.21 (3.525)	29.29 (2.779)
Diabetes	26.41 (2.956)	26.58 (2.828)	**27.75 (2.441)**	26.92 (3.088)
Pagie-1	30.7 (2.319)	**30.68 (1.985)**	18.67 (3.24)	24.71 (3.578)
Vladslaveva-4	20.14 (2.779)	**23.65 (3.52)**	9.725 (7.735)	14.25 (7.725)

Table 4.6: Median MSE over 50 runs of GP in different versions: with no fitness sharing, using the fitness and edit distances for fitness sharing, and using the proposed method based on embeddings (generated for dataset DM7) in Table 4.1.

Dataset	GP	GP with Fitness Sharing		
		Fitness	Edit-distance	Embedding
Boston House	**40.8 (26)**	60.7 (38)	62.1 (25)	45 (24)
Concrete	244 (340)	282 (80)	268 (180)	**244 (100)**
DM4	839 (1700)	1350 (1600)	1310 (1200)	**838 (560)**
DM7	0.946 (0.081)	1.09 (0.032)	**0.945 (0.05)**	1.07 (0.063)
DS5	**410 (470)**	457 (470)	768 (490)	415 (470)
DS8	**8.02 (1.9)**	9.71 (0.025)	8.74 (1.1)	8.25 (1.7)
Red Wine	0.761 (0.47)	**0.685 (0.29)**	0.878 (0.42)	0.763 (0.39)
White Wine	**0.854 (0.49)**	0.969 (0.32)	0.881 (0.5)	0.878 (0.24)
Diabetes	**16300 (5600)**	17100 (13000)	25400 (2700)	17600 (5300)
Pagie-1	**0.145 (0.025)**	0.162 (0.071)	0.17 (0.043)	0.156 (0.047)
Vladslaveva-4	**0.0379 (0.0021)**	0.0503 (0.021)	0.0482 (0.018)	0.0384 (0.0039)

results. The p-values were 3.48×10^{-4}, 8.11×10^{-5}, and 3.70×10^{-5} for fitness, semantic diversity, and syntactic diversity, respectively. The results of the Nemenyi *post-hoc* test for fitness presented a statistically significant difference between the plain symbolic regression and the symbolic regression with fitness sharing using edit distance and fitness. By inspecting Table 4.6, we realize that the plain GP has an advantage over the other baseline models when we measure error. Although it is not possible to claim that embedding methods are statistically significantly better than baselines, it is safe to assume that it can yield results that are similar to them, and in some cases, it may outperform them, which is not true for the general case, considering a plain GP algorithm. A somewhat similar situation happens for both diversity metrics. But in those two cases, the plain GP algorithm is expected to perform poorly, because it has no mechanism to maintain diversity. Therefore, for those cases, the Nemenyi *post-hoc* test shows a significant difference between the plain GP algorithm and every other, which can also be inferred by looking at Tables 4.7 and 4.8. Other than that, no significant differences were found.

Note that the correlation that was presented in Tables 4.4 and 4.5 was not found here. This was expected since each method uses a different mechanism to maintain diversity, which could give more emphasis on one type of diversity than the other. This can indicate that embeddings are probably able to capture relationships that fitness or edit distance alone are not able to, which is reasonable. However, since the experiment was not designed to understand this difference, we leave the study of the properties of the embeddings to future work.

Table 4.7: Median Entropy Transformed Semantics over 50 runs of GP in different versions: with no fitness sharing, using the fitness and edit distances for fitness sharing, and using the proposed method based on embeddings (generated for dataset DM7) in Table 4.1.

Dataset	GP	GP with Fitness Sharing		
		Fitness	Edit-distance	Embedding
Boston House	1.01 (0.28)	**1.55 (0.32)**	1.07 (0.3)	1.42 (0.37)
Concrete	0.989 (0.23)	**1.32 (0.22)**	1.11 (0.26)	1.27 (0.36)
DM4	0.875 (0.23)	**1.35 (0.24)**	0.994 (0.28)	1.15 (0.34)
DM7	0.332 (0.17)	**1.67 (0.41)**	0.462 (0.15)	1.17 (0.25)
DS5	1.08 (0.27)	1.2 (0.32)	1.21 (0.17)	**1.46 (0.48)**
DS8	0.617 (0.41)	**1.21 (0.37)**	0.742 (0.42)	0.824 (0.52)
Red Wine	1.07 (0.22)	1.32 (0.2)	**1.7 (0.26)**	1.71 (0.28)
White Wine	1.07 (0.13)	**1.62 (0.19)**	1.43 (0.31)	**1.62 (0.19)**
Diabetes	0.633 (0.48)	1 (0.37)	0.327 (0.084)	**1.08 (0.42)**
Pagie-1	1.14 (0.23)	1.4 (0.18)	2.05 (0.47)	**2.05 (0.31)**
Vladslaveva-4	0.996 (0.53)	2.29 (0.14)	2.31 (0.15)	**2.36 (0.095)**

Table 4.8: Median Pseudo-Isomorph over 50 runs of GP in different versions: with no fitness sharing, using the fitness and edit distances for fitness sharing, and using the proposed method based on embeddings (generated for dataset DM7) in Table 4.1.

Dataset	GP	GP with Fitness Sharing		
		Fitness	Edit-distance	Embedding
Boston House	16.84 (5.549)	28.81 (2.74)	**31.91 (1.917)**	28.75 (2.436)
Concrete	16.61 (4.064)	27.65 (3.956)	**31.87 (2.157)**	28.48 (2.613)
DM4	16.36 (4.343)	23.79 (5.319)	**30.16 (4.201)**	27.3 (2.77)
DM7	14.03 (3.064)	**30.01 (1.995)**	25.15 (2.25)	29.69 (2.618)
DS5	18.09 (2.534)	29.24 (3.941)	**32.32 (2.132)**	28.63 (2.627)
DS8	15.41 (2.936)	**31.84 (2.304)**	31.55 (4.034)	28.85 (2.966)
Red Wine	17.09 (2.696)	22.51 (4.392)	28.1 (1.765)	**28.58 (2.01)**
White Wine	16.64 (3.015)	**30.09 (3.118)**	28.41 (4.5)	28.2 (3.819)
Diabetes	16.94 (1.931)	28.97 (2.652)	**32.89 (2.235)**	26.58 (2.828)
Pagie-1	17.45 (3.172)	20.87 (4.789)	**32.61 (1.691)**	30.68 (1.985)
Vladslaveva-4	10.95 (7.559)	36.93 (8.794)	**37.76 (6.843)**	23.65 (3.52)

Table 4.9: Parameters tested for evaluating the impact of tree size.

Parameter	Transformer model
Training Dataset	DM7
Embedding Size	08
Embedding Aggregation	Mean
Distance	Cosine
Tree Size Cutoff	4, 6, **8**

4.2 Impact of input trees to the models

After choosing one model to represent individuals in the fitness sharing scenario using embeddings, we wanted to run additional tests to verify whether changing the size of the trees that serve as input to the model has an impact on the performance and compare them to the current size used.

Table 4.9 presents the parameters fixed in our first experiments in the upper part of the table, while new parameters appear in the lower part. In this experiment, we also trained new models for each of the different tree sizes, since it changes the dimensions of the vectors that we are using - an essential parameter for the neural network. Please note that the models from the previous sections used a Tree Size Cutoff of 8 - highlighted in bold in the parameter table - so we ran tests for tree sizes (depths) 4 and 6.

The experiment follows the same methodology used in the experiment from the previous section, using the same datasets and set of random seeds. The only difference is that, having some previous knowledge of the baselines performance for each metric, we do not have to compare the model results to every baseline model. The results are presented in Tables 4.10, 4.11, and 4.12, and the results were again evaluated by performing a Friedman test followed by a Nemenyi *post-hoc* test. We found that evaluating by the fitness, the Friedman test had a p-value of 1.34×10^{-5}, which means that there is a statistically significant difference between the groups. The Nemenyi test did not show a statistically significant difference between the groups with tree sizes 8 and the plain-GP algorithm. However, it did show a difference between the 2 algorithms above and the ones with tree sizes 4 and 6, which can be confirmed by looking at the results in Table 4.10.

When considering semantic diversity as the metric for success, the Friedman test has a p-value of 8.81×10^{-3}, which presents a statistical difference, but none of the groups tested had a difference when compared with the baseline, which is fitness sharing using the fitness. When using syntactic diversity as the evaluation metric, the Friedman test presented a p-value of 0.126, which is above the threshold for reaching statistical significance. Therefore, in this case, none of the metrics are statistically different. However,

Table 4.10: Median MSE over 50 runs of GP in different versions: Using the proposed method based on embeddings (generated for dataset DM7) with tree maximum size parameter set as 4, 6, 8, and with no fitness sharing.

Dataset	GP with Fitness Sharing			GP
	Embedding			
	Tree size 4	Tree size 6	Tree size 8	
Boston House	57.8 (24)	58.7 (24)	45 (24)	**40.8 (26)**
Concrete	647 (720)	287 (310)	**244 (100)**	244 (340)
DM4	**832 (570)**	1200 (1200)	838 (560)	839 (1700)
DM7	1.09 (0.012)	1.04 (0.012)	1.07 (0.063)	**0.946 (0.081)**
DS5	1120 (210)	1330 (2300)	415 (470)	**410 (470)**
DS8	9.65 (0.033)	9.66 (0.03)	8.25 (1.7)	**8.02 (1.9)**
Red Wine	0.822 (0.55)	0.806 (0.8)	**0.763 (0.39)**	**0.761 (0.47)**
White Wine	1.2 (0.79)	1.03 (0.65)	**0.878 (0.24)**	**0.854 (0.49)**
Diabetes	22500 (4700)	17100 (13000)	17600 (5300)	**16300 (5600)**
Pagie-1	0.179 (0.054)	0.196 (0.046)	0.156 (0.047)	**0.145 (0.025)**
Vladslaveva-4	0.0385 (0.0047)	0.0509 (0.021)	0.0384 (0.0039)	**0.0379 (0.0021)**

Table 4.11: Median Entropy Transformed Semantics over 50 runs of GP in different versions: Using the proposed method based on embeddings (generated for dataset DM7) with tree maximum size parameter set as 4, 6, 8, and GP with fitness sharing using the fitness difference as distance.

Dataset	**GP with Fitness Sharing**			
	Embeddings			**Fitness**
	Tree Size 4	**Tree Size 6**	**Tree Size 8**	
Boston House	1.46 (0.3)	1.39 (0.4)	1.42 (0.37)	**1.55 (0.32)**
Concrete	0.635 (0.59)	0.974 (0.37)	1.27 (0.36)	**1.32 (0.22)**
DM4	1.03 (0.41)	1.12 (0.38)	1.15 (0.34)	**1.35 (0.24)**
DM7	0.38 (0.087)	0.35 (0.08)	1.17 (0.25)	**1.67 (0.41)**
DS5	0.882 (0.39)	0.42 (0.26)	**1.46 (0.48)**	1.2 (0.32)
DS8	0.57 (0.1)	0.588 (0.099)	0.824 (0.52)	**1.21 (0.37)**
Red Wine	1.47 (0.32)	1.4 (0.23)	**1.71 (0.28)**	1.32 (0.2)
White Wine	1.43 (0.29)	1.52 (0.23)	**1.62 (0.19)**	**1.62 (0.19)**
Diabetes	0.534 (0.26)	1.03 (0.41)	**1.08 (0.42)**	1.0 (0.37)
Pagie-1	1.82 (0.53)	1.71 (0.49)	**2.05 (0.31)**	1.4 (0.18)
Vladslaveva-4	2.12 (0.11)	2.29 (0.14)	**2.36 (0.095)**	2.29 (0.14)

in this case, the baseline metric is the edit distance, and by using fitness sharing with embeddings it is possible to achieve similar results in a lower computational time, mainly if it is not necessary to train the models.

It is important to mention that we also performed preliminary tests to represent trees that are deeper than the threshold cutoff of the trees. In the current approach, tree nodes that exceed the transformer input size are dropped. In this test, instead of dropping the trees, we tried cutting them at the maximum possible size and treating each of the subtrees that would be dropped as a different tree. Every tree would pass through

Table 4.12: Median Pseudo-Isomorph over 50 runs of GP in different versions: Using the proposed method based on embeddings (generated for dataset DM7) with tree maximum size parameter set as 4, 6, 8, and GP with fitness sharing using the edit distance.

Dataset	GP with Fitness Sharing			
	Embeddings			Edit distance
	Tree Size 4	Tree Size 6	Tree Size 8	
Boston House	28.7 (3.5)	27.6 (4.4)	28.8 (2.4)	**31.9 (1.9)**
Concrete	23.3 (3.9)	25.6 (2.6)	28.5 (2.6)	**31.9 (2.2)**
DM4	27.7 (3.9)	28.7 (3.1)	27.3 (2.8)	**30.2 (4.2)**
DM7	15.6 (1.5)	22.5 (3)	**29.7 (2.6)**	25.1 (2.2)
DS5	23.7 (2.8)	19.4 (1.4)	28.6 (2.6)	**32.3 (2.1)**
DS8	18 (2.4)	17.2 (1.4)	28.9(3.0)	**31.5 (4)**
Red Wine	25.9 (3.9)	21.9 (3.2)	**28.6 (2.0)**	28.1 (1.8)
White Wine	24.1 (4.2)	24.9 (3.6)	**28.2 (3.8)**	28.4 (4.5)
Diabetes	28.6 (3.1)	29.5 (2.5)	26.6 (2.8)	**32.9 (2.2)**
Pagie-1	29.6 (3.3)	28 (2.5)	30.7 (2.0)	**32.6 (1.7)**
Vladslaveva-4	13.7 (2)	37.3 (8.7)	23.7 (3.5)	**37.8 (6.8)**

the transformer model to generate embeddings. Then the embeddings for each large tree would be averaged to generate a single vector, just like in the previous approach. This kind of approach is often used in the NLP field to represent different sentences or paragraphs from a text. By doing preliminary tests with a smaller set of random seeds, we found out that this approach was taking too long to run, and the results were consistently worse than the approach used in the previous tests. We hypothesise that the root of the trees and the nodes close to it carry the most important syntactic information. Thus, when we average the embeddings, the importance of these nodes is decreased when averaged with many other sub-trees. Therefore, we leave exploring other forms of representing large trees for future work.

4.3 Embedded tree representations for visualization

This section presents our second case study, which is based on using distributed representations to create two different types of visualizations to explore the trajectory of a solution:

1. Using a dimensionality reduction technique to reduce the dimension of the embeddings to 2, and then plot the reduced dimensions directly;

2. Creating a visualization based on a neighborhood graph generated from the distances of the embeddings.

We were first inspired by the work of Search Trajectory Networks (STNs) [29]. STNs are graphs created to connect solutions - in the order they were found - during the optimization process, generating a search trajectory. For the particular case of symbolic regression, we can think of the solutions as the trees that represent the functions, and the search trajectory would be the path of trees until the best solution of the run is found. We will refer to trajectory as the smallest search trajectory. Overall, STNs are useful to make visualizations possible and this is valuable for understanding how the algorithm performed during the evolutionary process.

For these visualizations to be done properly, we need to calculate the trajectory followed by the algorithm to find these solutions. Given one or more GP runs, we get the best solution returned by the algorithm in each run and then select all of their ancestors (parents, then parents of the parents, and so on) recursively to create a new dataset. To track the ancestors, we use IDs to identify parents and children. This new dataset contains the best solutions found by different GP runs with different parameters and all the ancestors trees generated in order to reach the best solution.

After that, to find the trajectory we created a graph based on the parents and children of the trees in the dataset. Each individual is a node in the graph T, and children nodes are connected to their parent nodes. Note that, in T, nodes with in-degree 0 (here named root nodes) represent the initial population, and nodes with out-degree 0 (here named leaf nodes) represent the solutions of the last generation.

Note that, along a GP run, there is a chance that multiple trees represent the same function, even though they may have different parents. When the evolution process starts to converge, this situation becomes common. Here we remove duplicated trees by using the embeddings to compare pairs of trees quickly[2]. We grouped trees that have the same embeddings, and then we selected the first tree found (the one with the lowest ID) and kept track of parents and children by simply adding every parent and son to the same graph node. We also removed self-loops from the graph – i.e., the case when a tree is its own parent/child so that we could remove repeated trees from the dataset without losing information.

Having graph T, we can then calculate the shortest path from each root node to the leaf node of the same run. Out of these, the smallest shortest path is called the trajectory. This will give us the shortest number of steps for leaving a random initial solution and finding the best solutions returned for different parameters.

Next, we present two different ways of visualizing this trajectory graph: using a coordinate-based system or a neighborhood-based trajectory.

[2]Given that we already had a trained model.

4.3.1 Coordinate-based trajectory visualization

After the dataset was generated, we used Uniform Manifold Approximation and Projections (UMAP) [33] to reduce the dimensionality of the embeddings from 8 to 2. We used the reduced space to plot the trees. To find optimal parameters for UMAP we ran a greedy search over the two parameters that have the highest impact on the results: number of neighbors and minimum distance. We then compared the distances of the embeddings in the original space with the cosine distances of the reduced set of dimensions and selected the parameters that had distances that were most similar to the distances in the original space.

4.3.2 Neighborhood-based trajectory visualization

Inspired by the node-edge diagram proposed by the authors in [29], we used a different method to recreate the search trajectory in trees. After gathering the dataset with embeddings calculated, we create a *neighborhood-based graph*, which will be represented by a Gabriel Graph [14]. A Gabriel Graph is a geometric structure used in computational geometry and graph theory. It involves a set of nodes (S) and edges (E), where two distinct nodes p and q in S are connected by an edge $e \in E$ if the hyper-sphere with pq as its diameter contains no other nodes from S. This geometric criterion establishes a direct "neighborhood relationship" between nodes. Notably, the Gabriel Graph is a subgraph of the Delaunay triangulation, revealing its close connection to another important geometric construct. This relationship simplifies complex networks and can aid tasks like clustering and nearest neighbor search. Gabriel graphs have been applied in distinct domains such as pattern recognition [38, 1], and wireless sensor networks [26].

Since we already have a graph T, we add a new set of edges to this graph based on the definition of the Gabriel Graph using every dimension of the embeddings. Therefore, each node of the graph T can now have two types of edges:

1. An edge that connects parent nodes to its sons (relationship edge);
2. An edge that connects any two nodes based on the proximity according to the Gabriel Graph definition (gg edge).

We used this graph to create a force-based diagram but, instead of showing all the relationship edges, we show just the ones that belong to the trajectory, and hence we call

them trajectory edges. Force-based diagrams are a popular way of visualizing graphs. In our case, it maintains some of the syntactic properties of the trees captured by the embeddings.

4.3.3 Experimental Results

To analyze the quality of the visualizations produced, we focused on a subset of 3 datasets used in our experiments for diversity. Ideally, we should be able to represent any kind of problem, so we picked datasets with different characteristics: the first, Pagie-1, is easy to solve using symbolic regression and has few (2) features; the second, DM4, has a higher number of features, and the last, DM7, is a problem hard to solve with the current operations and parameters used by the GP. To define hard or easy problems we used the fitness (MSE) achieved: problems that presented low values of error were considered easy, and the hard ones were the ones with high values of error.

The parameters used for the symbolic regression to compare different GP runs are presented in Table 4.13. We focused on varying two parameters that have an impact on the time of convergence and diversity of GP algorithms: tournament selection size k and mutation rate. Recall that a tournament selection works as follows: it randomly selects a set of k individuals from the population that will compete among themselves. The individual with the best fitness among the k is considered the winner. The smaller the value of k, the lower the selective pressure, which is responsible for determining the speed of convergence. Regarding mutation, the higher the mutation rate, the bigger the chance of inserting new genetic material into the population. Higher mutation rates usually increase diversity and tend to delay convergence. Notice that not all combinations of tournament size and mutation probability were used to avoid too many trajectories in the plots. We did not test tournament size 7 with 40% mutation probability.

Also note that we use two different random seeds, running the tests for the same set of parameters twice. Although the graphs could show a single run, we found it interesting to represent the stochastic nature of the process.

We loaded the graphs into a Neo4j database[3] and used the library Bloom[4] to generate the plots. In every chart, there are different types of nodes and edges. They were colored differently using the color scheme detailed in Table 4.14, where node size refers to the size of the dots plotted. Note that for plots that compare just two combinations of parameters, the color scheme for regular nodes does not apply. In those cases, the

[3]https://neo4j.com
[4]https://neo4j.com/product/bloom/

Table 4.13: Symbolic Regression parameters used in the visualization experiments.

Fixed parameters	**Value**	
Datasets	Pagie-1, DM4, DM7	
Random seed	20230610, 20230611	
Number of generations	100	
Population size	100	
Crossover probability	90%	
Tournament size	2, 7	2
Mutation probability	10%	10%, 40%

parameters are described in the caption of the figure. Since it is not possible to plot the graphs in this document in high resolution, they are also available online to make a more careful inspection possible.

Table 4.14: Color and size scheme for nodes and edges.

Type	**Random Seed**	**Tournament Size**	**Mutation probability**	**Color**	**Node Size**
Root node		any		brown	2
Leaf node		any		green	4
Regular node	20230610	2	10%	yellow	1
	20230611	2	10%	cyan	
	20230610	7	10%	pink	
	20230611	7	10%	red	
	20230610	2	40%	orange	
	20230611	2	40%	beige	
GG edge		any		light gray	N/A
Trajectory edge		any		black	N/A

4.3.3.1 Pagie-1 plots

This section shows and discusses the plots for dataset Pagie-1. First, we present an overall plot that represents the individuals present in the trajectory in a total of 6 different runs. Each overall plot is shown side-to-side by the same plot but with the trajectories highlighted. Figure 4.4 is calculated using the coordinate-based layout, and Figure 4.5 using the Gabriel Graph. The smallest trajectory was given by random seed 20230611, tournament size 2, and mutation probability 40% with size 17 in generation 38. On the other hand, the run that had the best fitness presented the same random seed, but tournament size 7 and mutation probability 10%. The complete results are shown in

Table 4.15: Fitness and trajectory size per run for the dataset Pagie-1.

Random Seed	Tournament Size	Mutation Probability	Fitness	Trajectory Size	1st Generation Found
20230610	2	0.1	0.148	31	75
20230610	2	0.4	0.140	30	53
20230610	7	0.1	0.139	32	78
20230611	2	0.1	0.132	40	79
20230611	2	0.4	0.146	17	38
20230611	7	0.1	0.130	17	52

Table 4.15, where 1st generation found is the generation where the returned solution was first found by the GP.

Figure 4.4: Coordinate-based plot of 6 runs for Pagie-1 dataset.

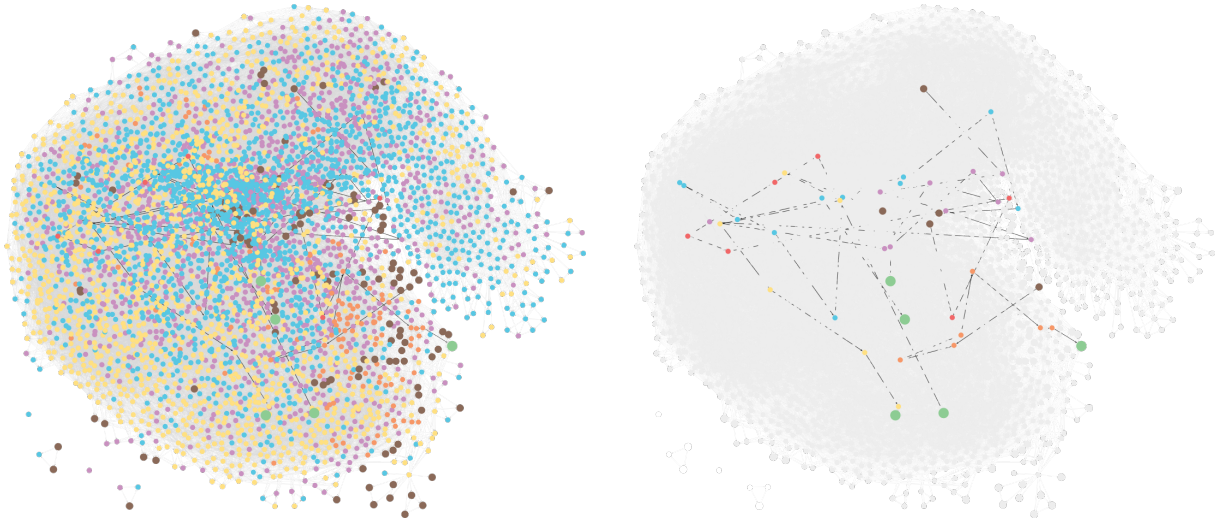


Figure 4.5: Neighborhood-based plot of 6 runs for Pagie-1 dataset.

It is possible to check that the coordinate-based layout tends to be more sparse, which is expected since it has to respect a set of coordinates, and is not able to accommodate the nodes on arbitrary positions. Also in both charts it is possible to see that the yellow dots (tournament 2, mutation rate 10) tend to be in an outer region, and a little

bit more separated than the others. Orange dots (tournament 2, mutation rate 40) are more concentrated in specific region of the space in both plots. It is also possible to see that the blue dots (tournament 2, mutation rate 10) tend to be spread across almost all the regions, and it tend to mix more with other runs, which can be seen in both graphs.

Next, we present a comparison plot between two runs with different parameters. We will not go over all of them to limit the number of figures. So we will cover all the parameters at least once across all datasets, and upload the other files to the image repository. Figure 4.6 presents the two different types of plots in the same figure, the coordinate-based on the left and the neighborhood-based on the right. This figure presents a comparison between the tournament size parameter, being it 2 or 7. A tournament 7 would increase selective pressure, making the algorithm more prone to early convergence and a lower diversity. It is important to note, though, that only a subset of the trees generated for the GP is shown in these graphs. Because of that, the graph is not suitable to compare the explored search space. We can see that both have a similar content in the way that in the middle are of the chart there is a higher mix of points from both parameter configurations, while closer to the edges one parameter tend to dominate those regions concentration.

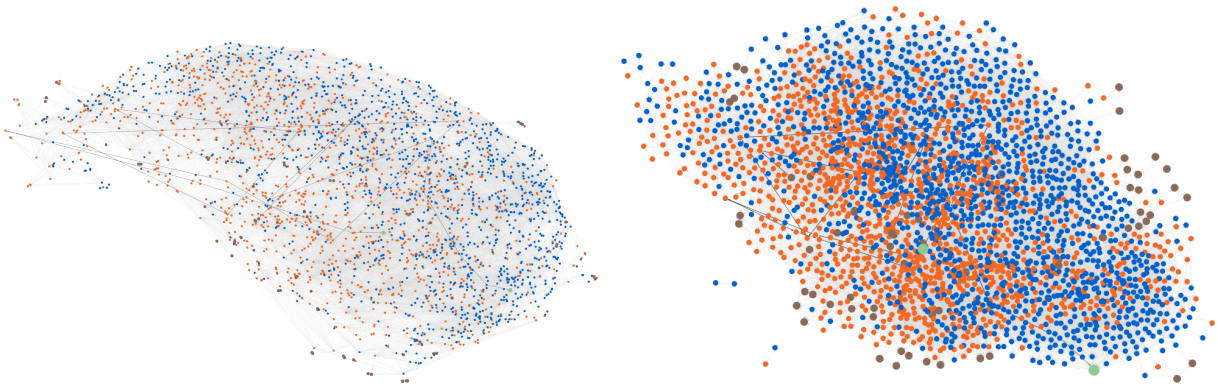


Figure 4.6: Plots of 2 runs for Pagie-1 dataset, with parameters random seed 20230610, tournament sizes 2 (blue) and 7 (orange), and mutation probability 10%. On the right the coordinate-based plot, and on the left the neighborhood-based one.

4.3.3.2 DM4 plots

This section discusses the plots for dataset DM4, which represents the Cpu-Act dataset. We present the plots the same way as before. Figures 4.7 and 4.8 present an overall plot that has 6 runs of the GP. The plots on the right side are the same, but with the trajectories highlighted.

Table 4.16: Fitness and trajectory size per run for dataset DM4.

Random Seed	Tournament Size	Mutation Probability	Fitness	Trajectory Size	1st Generation Found
20230610	2	0.1	717	38	98
20230610	2	0.4	982	36	75
20230610	7	0.1	202	28	93
20230611	2	0.1	274	30	69
20230611	2	0.4	318	42	83
20230611	7	0.1	1264	34	98

Figure 4.7: Coordinate-based plot of 6 runs for DM4 dataset.

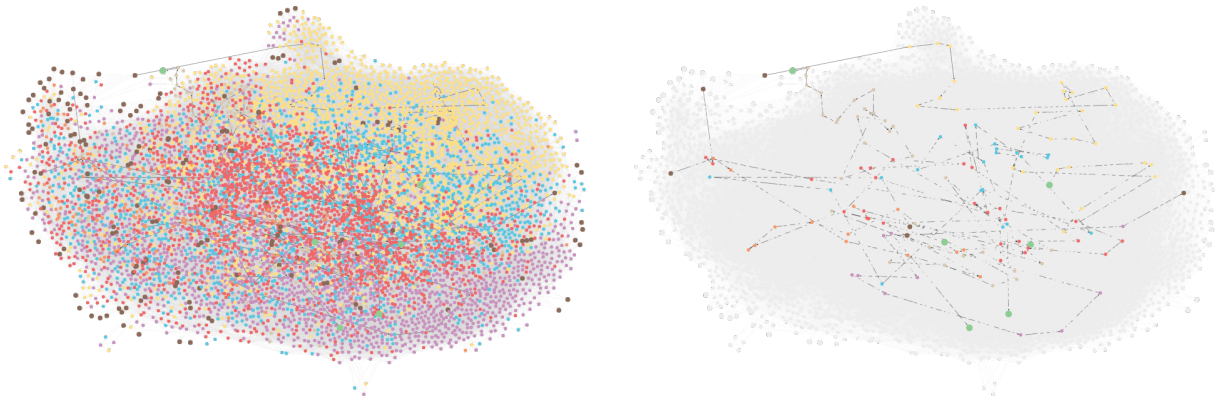


Figure 4.8: Neighborhood-based plot of 6 runs for DM4 dataset.

Both overall plots for the dataset DM4 have four areas that have way more points from one specific color. In Figure 4.7 it is possible to see that the concentration of yellow dots (tournament size 2 and mutation rate 0.1) is higher in the bottom right portion of the plot, while the same happens for the blue in the bottom middle section (second run of tournament size 2 and mutation rate 0.1), which seems to show they explore different parts of the search space). Pink dots (tournament size 7 and mutation rate 0.1) are more present along all the left portion of the image, in a somehow narrower region of the graph. This run presented the best solution and smallest trajectory of all runs, as observed in Table 4.16 – which shows the size of the trajectories and the fitness values of the solutions found in each GP run.

Red dots (second run of tournament size 7 and mutation rate 0.1) are more concentrated in the very middle of the graph and, although they use the same parameters as the pink dots, they present the overall worst results – due to a different random seed. Table 4.16 shows the size of the trajectories and the fitness values of the solutions found in each GP run.

Something similar happens in Figure 4.8: Yellows are more concentrated on the top right part of the plot, while blue is more concentrated towards the right middle part. Pink has more dots along the bottom of the picture, and reds are more distributed in the middle of the image. It is important to note that the relative position of the colors seems to be about the same, showing both graphs are capturing the same notion of distances.

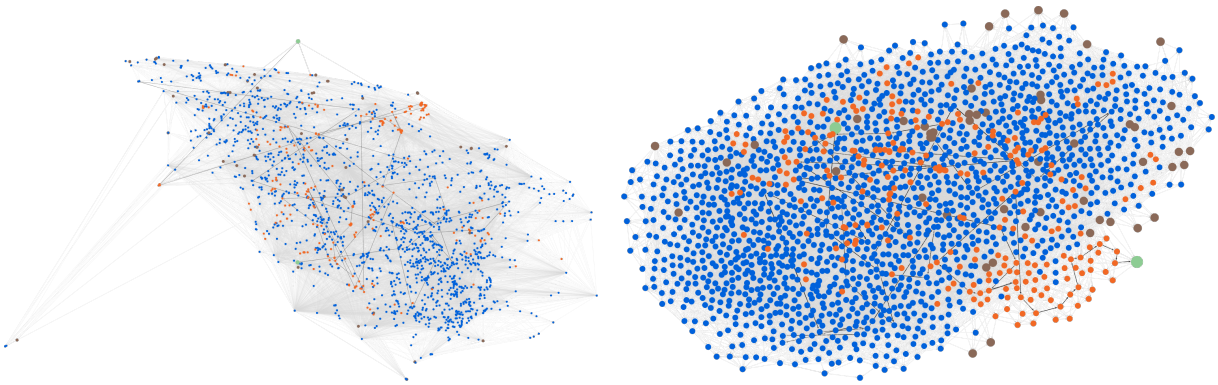


Figure 4.9: Plots of 2 runs for DM4 dataset, with parameters random seed 20230611, tournament size 2, and mutation probabilities 10% (blue) and 40% (orange). On the right is the coordinate-based plot, and on the left is the neighborhood-based one.

The selected comparison plot for DM4 dataset is presented in Figure 4.9. This time we selected a different set of parameters. In this case, in the neighborhood-based visualization, it is possible to see a path of trajectory in the lower part of the chart, where orange dots (tournament size 2 and mutation rate of 0.4) have a higher concentration. Therefore, we also added Figure 4.10, which is the same as Figure 4.9 but with the trajectories highlighted.

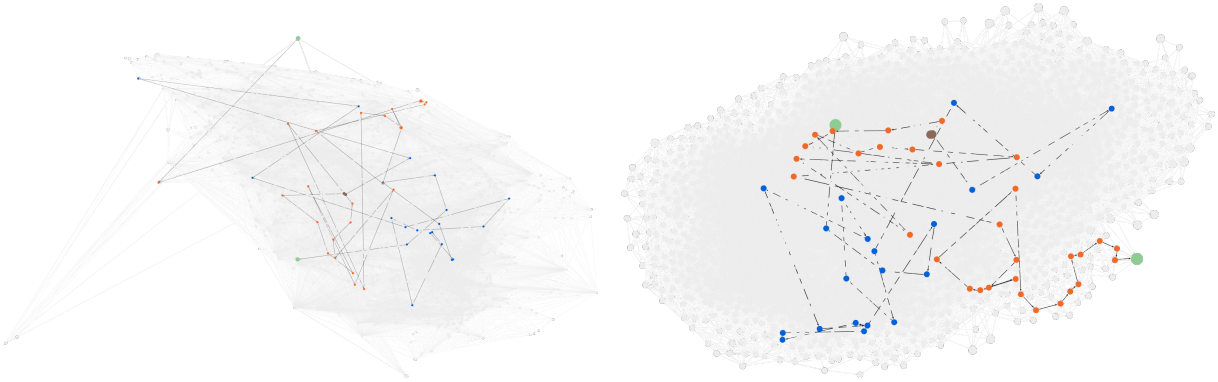


Figure 4.10: Plots of 2 runs for DM4 dataset with the trajectories highlighted, with parameters random seed 20230611, tournament size 2, and mutation probabilities 10% (blue) and 40% (orange). On the right is the coordinate-based plot, and on the left is the neighborhood-based one.

Comparing the two trajectories from the two graphs it is clear neighborhood-based plot tends to have a more concise representation of the trajectory. This is expected because of the process we did in the neighborhood-based plots[5].

4.3.3.3 DM7 plots

In this section, we show and discuss the charts generated using the dataset DM7 - fri_c3_250_25. The plots are presented in the same way as in previous sections. Table 4.17 presents complete results of the final fitness of the runs, as well as the size of the trajectory and the first generation where the solution was found. The smallest trajectory of 32 was found using random seed 20230610, tournament size 2, and mutation probability 0.4 - found in orange at figures 4.11 and 4.12. This run also was the one where the solution was found in the earliest generation. The one with the best fitness was the run with random seed 20230610, tournament size 7, and mutation probability 0.1 - presented in pink in figures 4.11 and 4.12. This solution was found at the last generation and the size of the trajectory is 39. Figures 4.11 and 4.12 present an overall plot that has 6 runs of the GP and the plots on the right side are the same, but with the trajectories highlighted. Next, we present the comparison plot in Figure 4.13, which contains a comparison of the mutation probability. In the comparison chart, note that towards the middle, there is a noticeably higher concentration of points in the neighborhood-based plot. This also seems to happen on the coordinate one but in a more concise space. The force-based layout tends to spread closer points and compress when they are more distant.

[5]We artificially removed the trajectory points from the graph after it converged and added them again, so they would show in the outmost layer of the graph

Table 4.17: Fitness and trajectory size per run for dataset DM7.

Random Seed	Tournament Size	Mutation Probability	Fitness	Trajectory Size	1st Generation Found
20230610	2	0.1	1.000	43	100
20230610	2	0.4	0.888	32	71
20230610	7	0.1	0.671	39	100
20230611	2	0.1	1.003	46	100
20230611	2	0.4	0.987	48	95
20230611	7	0.1	0.940	36	100

Figure 4.11: Coordinate-based plot of 6 runs for DM7 dataset.

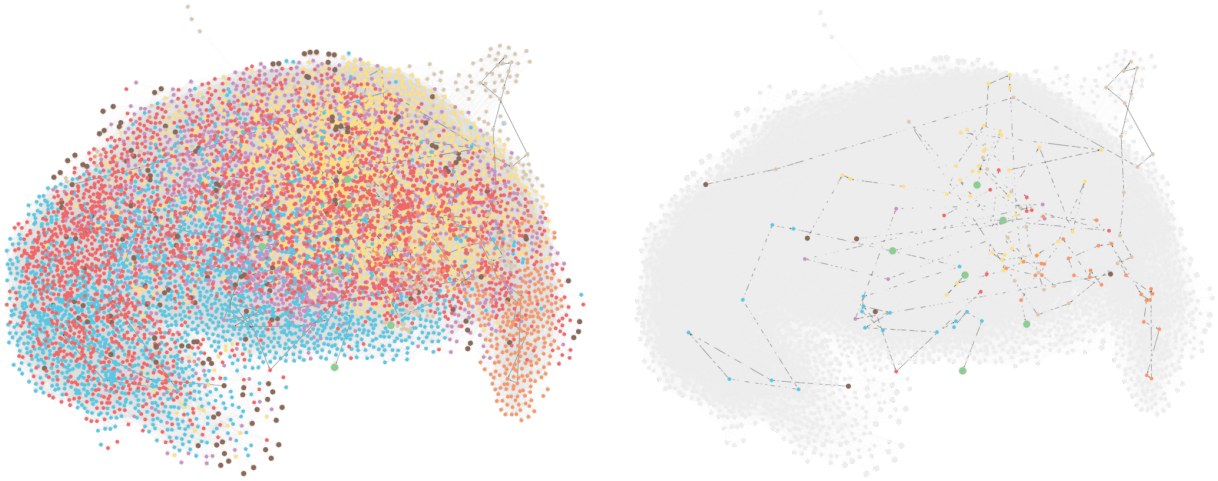


Figure 4.12: Neighborhood-based plot of 6 runs for DM7 dataset.

Looking at the charts from dataset DM7 there is not much new from the previous datasets. Probably, the behavior of the plots based on coordinates and the plots based on the Gabriel Graph present similar results. The coordinate-based plots are more strict in the sense that the coordinates are not able to be moved, so they tend to take a larger space. The neighborhood-based plot presents a higher concentration of points, therefore, they tend to accommodate well a large set of points. Overall, they also seem to represent the points in a meaningful way. The trajectories presented very different patterns across both types of plots. This is expected, but we may be able to improve the force-based plots by simply removing the trajectory edges from them, and plotting them afterward.

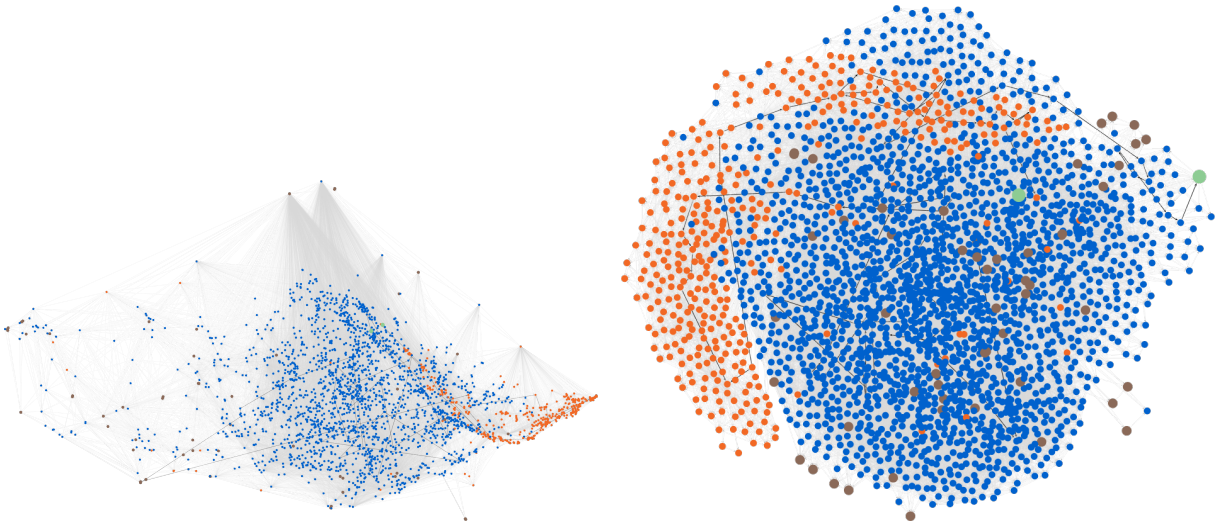


Figure 4.13: Plots of 2 runs for DM4 dataset with parameters random seed 20230610, tournament size 2, and mutation probabilities 10% (blue) and 40% (orange). On the right is the coordinate-based plot, and on the left is the neighborhood-based one.

We were not able to do that with the current tools we used.

Chapter 5

Conclusions and Future Work

In this work, we presented a novel form of representation for genetic programming trees. We assessed this representation indirectly comparing it to other metrics and evaluated the results using fitness sharing. The indirect result measurements showed that the syntactic information captured by the representations increases as the representation size increases, and the relation of this representation with semantic information is problem-dependent. We tested the representation in two different scenarios: to measure diversity in a fitness-sharing scenario and to provide a new representation of how GPs explore the search space.

In the first scenario, we show the models seem to generalize from one dataset to the other, which is a really interesting feature. In the second scenario, we showed robust representations of the space being explored by the GP, which can be further developed to better understand what happens in tree-based GP algorithms.

There are many aspects of the current work that can be improved in the future. Although in terms of diversity, the model seems to be good at generalizing to new datasets, a good way to make this model even more generalizable would be to use genetic programming trees from various datasets during training instead of relying solely on one problem.

Still thinking about the current solution, the way we categorized constants is probably an interesting feature that can be modified with support from the NLP literature. Additionally, further testing the representation of trees that exceed the defined threshold could yield valuable insights.

We have also noticed a strong correlation between two diversity metrics associated with distinct strategies, particularly evident when employing fitness sharing with embeddings. This correlation was not present in the other scenarios tested and could be worth investigating to have a better understanding of the embeddings.

Embeddings of GP trees enable a distinct range of operations to be conducted on vectors rather than trees. Thus, embeddings can be used to perform evolutionary operations, such as mutation and crossover, which are already used in other domains. Extending this idea, the embeddings might potentially serve as constraints for these operations. For instance, mutations or crossovers could be restricted if the tree distance fails to exceed a designated threshold.

Expanding the visualization technique that allows us to see how the search space is being explored, we could use embeddings as ways to represent trees in tasks of fitness landscape analysis, where the neighborhood of a solution is an important concept. Likewise, while the GP algorithm is running, embeddings can be used to check whether the algorithm is getting stuck in certain areas of the space or if it's adequately exploring it. It is also possible to try mapping constraints of optimization algorithms into the embedding space to restrict search in certain regions.

Finally, it is important to note that the representation we constructed in this study exclusively considers the syntax of the trees. It is worth mentioning that syntax representations already exist, and are commonly used in GP. In future research, there is potential to integrate this network with other models, or even construct a novel multimodal network to also account for semantics. This hybrid model could take both the trees and the values of independent and dependent variables to execute tasks that involve semantics or a combination of semantic and syntactic aspects. In this case, we would rely on what definition of semantics we consider. In many works, the semantics of a GP tree for symbolic regression are given by the differences of the expected versus predicted outputs individually [35]. Having defined what semantics is in this context, syntactic and semantic concepts could be learned simultaneously.

Bibliography

[1] Wagner J Alvarenga, Felipe V Campos, Alexsander CAA Costa, Turibio T Salis, Eduardo Magalhães, Luiz CB Torres, and Antonio P Braga. Time domain graph-based anomaly detection approach applied to a real industrial problem. *Computers in Industry*, 142:103714, 2022.

[2] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[3] Yoshua Bengio. Deep learning of representations: Looking forward. In *International conference on statistical language and speech processing*, pages 1–37. Springer, 2013.

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[5] Peter J Bentley, Soo Ling Lim, Adam Gaier, and Linh Tran. Evolving through the looking glass: Learning improved search spaces with variational autoencoders. In *International Conference on Parallel Problem Solving from Nature*, pages 371–384. Springer, 2022.

[6] Edmund K. Burke, Steven Gustafson, and Graham Kendall. Diversity in Genetic Programming: An Analysis of Measures and Correlation with Fitness. *IEEE Trans. Evol. Comput.*, 8(1):47–62, 2004.

[7] Armand R Burks and William F Punch. An analysis of the genetic marker diversity algorithm for genetic programming. *Genetic Programming and Evolvable Machines*, 18(2):213–245, 2017.

[8] Junyi Chai and Anming Li. Deep learning in natural language processing: A state-of-the-art survey. In *2019 International Conference on Machine Learning and Cybernetics (ICMLC)*, pages 1–6. IEEE, 2019.

[9] Qi Chen, Bing Xue, and Mengjie Zhang. Preserving Population Diversity Based on Transformed Semantics in Genetic Programming for Symbolic Regression. *IEEE Trans. Evol. Comput.*, 25(3):433–447, 2021.

[10] Sander de Bruin, Vadim Liventsev, and Milan Petković. Autoencoders as tools for program synthesis. *arXiv preprint arXiv:2108.07129*, 2021.

[11] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, 7:1–30, 2006.

[12] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[13] Anikó Ekárt and Sándor Zoltán Németh. A metric for genetic programs and fitness sharing. In *European Conference on Genetic Programming*, pages 259–270. Springer, 2000.

[14] K Ruben Gabriel and Robert R Sokal. A new statistical approach to geographic variation analysis. *Systematic zoology*, 18(3):259–278, 1969.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[16] Qi He, João Sedoc, and Jordan Rodu. Trees in transformers: a theoretical analysis of the transformer's ability to represent trees. *arXiv preprint arXiv:2112.11913*, 2021.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[18] David Jackson. Phenotypic diversity in initial genetic programming populations. In *European Conference on Genetic Programming*, pages 98–109. Springer, 2010.

[19] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[20] SEOHYUN KIM, JINMAN ZHAO, YUCHI TIAN, and SATISH CHANDRA. Code prediction by feeding trees to transformers. *arXiv*, 2020.

[21] John R Koza. *Genetic programming II: automatic discovery of reusable programs*. MIT press, 1994.

[22] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *International conference on machine learning*, pages 1945–1954. PMLR, 2017.

[23] Pawe Liskowski, Krzysztof Krawiec, Nihat Engin Toklu, and Jerry Swan. Program synthesis as latent continuous optimization: Evolutionary search in neural embeddings. *GECCO 2020 - Proc. 2020 Genet. Evol. Comput. Conf.*, pages 359–367, 2020.

[24] Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation learning for natural language processing*. Springer Nature, 2020.

[25] Tomas Mikolov, Kai Chen, G.s Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 01 2013.

[26] Tomas Mikolov, Greg Corrado, Kai Chen, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *ICLR*, pages 1–12, 2013.

[27] Alberto Moraglio, Krzysztof Krawiec, and Colin G Johnson. Geometric semantic genetic programming. In *International Conference on Parallel Problem Solving from Nature*, pages 21–31. Springer, 2012.

[28] Su Nguyen, Mengjei Zhang, Damminda Alahakoon, and Kay Chen Tan. Visualizing the evolution of computer programs for genetic programming [research frontier]. *IEEE Computational Intelligence Magazine*, 13(4):77–94, 2018.

[29] Gabriela Ochoa, Katherine M Malan, and Christian Blum. Search trajectory networks: A tool for analysing and visualising the behaviour of metaheuristics. *Applied Soft Computing*, 109:107492, 2021.

[30] Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, Dec 2017.

[31] 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*, volume 5, pages 4092–4097. IEEE, 1997.

[32] Justinian P Rosca. Entropy-driven adaptive representation. In *Proceedings of the workshop on genetic programming: From theory to real-world applications*, volume 9, pages 23–32. Citeseer, 1995.

[33] Tim Sainburg, Leland McInnes, and Timothy Q Gentner. Parametric umap embeddings for representation and semisupervised learning. *Neural Computation*, 33(11):2881–2907, 2021.

[34] Vighnesh Leonardo Shiv and Chris Quirk. Novel positional encodings to enable tree-based transformers. In *NeurIPS*, pages 12058–12068, 2019.

[35] Leonardo Vanneschi, Mauro Castelli, and Sara Silva. A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, 15, 06 2014.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[37] Lifu Wang, Bo Shen, Ning Zhao, and Zhiyuan Zhang. Is the skip connection provable to reform the neural network loss landscape? *arXiv preprint arXiv:2006.05939*, 2020.

[38] Zhe Wang, Qida Dong, Wei Guo, Dongdong Li, Jing Zhang, and Wenli Du. Geometric imbalanced deep learning with feature scaling and boundary sample mining. *Pattern Recognition*, 126:108564, 2022.

[39] Zichao Wang, Mengxue Zhang, Richard G Baraniuk, and Andrew S Lan. Scientific formula retrieval via tree embeddings. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 1493–1503. IEEE, 2021.

[40] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262, 1989.

[41] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. *Advances in Neural Information Processing Systems*, 32, 2019.

Appendix A

Tables

A.1 Evaluation using Median MSE

Table A.1: Parameters of the models trained using dataset DM4 and their results of Median MSE when applied to a new group of datasets.

Dataset	**DM4**	**DM4**	**DM4**
Emb. Size	**8**	**16**	**64**
Emb. type	**mean**	**mean**	**mean**
Distance	**cos**	**cos**	**cos**
Dataset		**Median MSE**	
BHouse	42.9 (24)	48.7 (25)	46.7 (32)
Concrete	248 (1.8e+02)	252 (81)	245 (1.9e+02)
DM4	932 (6.3e+02)	646 (4.8e+02)	957 (6.8e+02)
DM7	1.04 (0.13)	1.09 (0.017)	1.09 (0.0078)
DS5	418 (5.1e+02)	415 (4.8e+02)	415 (5.2e+02)
DS8	7.98 (1.8)	9.6 (0.41)	9.69 (0.066)
R.Wine	0.701 (0.55)	0.758 (0.4)	0.76 (0.41)
W.Wine	0.927 (0.24)	0.949 (0.31)	0.992 (0.47)
diabetes	1.76e+04 (5.3e+03)	1.76e+04 (5.8e+03)	1.74e+04 (5.6e+03)
pagie-1	0.155 (0.046)	0.162 (0.047)	0.204 (0.062)
vlad.4	0.038 (0.0018)	0.0392 (0.0049)	0.0379 (0.0016)

Table A.2: Parameters of the models trained using dataset DL1 and their results of Median MSE when applied to a new group of datasets.

Dataset	**DL1**	**DL1**	**DL1**
Emb. Size	**8**	**16**	**64**
Emb. type	**mean**	**mean**	**mean**
Distance	**cos**	**cos**	**cos**
Dataset		**Fitness**	
BHouse	47.8 (26)	46.6 (29)	47.9 (26)
Concrete	248 (83)	248 (90)	241 (74)
DM4	831 (7.4e+02)	751 (6.3e+02)	899 (6.5e+02)
DM7	1.08 (0.084)	1.09 (0.038)	1.09 (0.0078)
DS5	424 (5.7e+02)	410 (4.8e+02)	415 (5e+02)
DS8	8.4 (1.8)	8.57 (1.8)	9.66 (0.14)
R.Wine	0.729 (0.47)	0.761 (0.35)	0.65 (0.25)
W.Wine	0.906 (0.3)	0.935 (0.28)	0.87 (0.24)
diabetes	1.76e+04 (5.3e+03)	1.74e+04 (5.7e+03)	1.76e+04 (5.3e+03)
pagie-1	0.164 (0.039)	0.185 (0.059)	0.179 (0.061)
vlad.4	0.0503 (0.021)	0.0503 (0.021)	0.0397 (0.01)

Table A.3: Parameters of the models trained using dataset DM7 and their results of Median MSE when applied to a new group of datasets.

Dataset	**DM7**	**DM7**	**DM7**
Emb. Size	**8**	**16**	**64**
Emb. type	**mean**	**mean**	**mean**
Distance	**cos**	**cos**	**cos**
Dataset		**Fitness**	
BHouse	45 (24)	46.3 (26)	49.8 (24)
Concrete	244 (1e+02)	247 (78)	238 (77)
DM4	838 (5.6e+02)	643 (6.5e+02)	836 (6.7e+02)
DM7	1.07 (0.063)	1.09 (0.047)	1.09 (0.0078)
DS5	415 (4.7e+02)	409 (4.8e+02)	415 (5.6e+02)
DS8	8.25 (1.7)	8.85 (1.5)	9.66 (0.11)
R.Wine	0.763 (0.39)	0.769 (0.39)	0.656 (0.35)
W.Wine	0.878 (0.24)	0.999 (0.33)	0.934 (0.24)
diabetes	1.76e+04 (5.3e+03)	1.76e+04 (5.3e+03)	1.76e+04 (5.4e+03)
pagie-1	0.156 (0.047)	0.159 (0.045)	0.175 (0.052)
vlad.4	0.0384 (0.0039)	0.0401 (0.0079)	0.0382 (0.0032)

Table A.4: Parameters of the models trained using dataset DM4 and their results of Median MSE when applied to a new group of datasets.

Dataset	**DM4**	**DM4**	**DM4**
Emb. Size	**8**	**16**	**64**
Emb. type	**mean**	**mean**	**mean**
Distance	**euc**	**euc**	**euc**
Dataset		**Fitness**	
BHouse	51.2 (29)	52.2 (29)	61.7 (34)
Concrete	231 (92)	248 (2.2e+02)	246 (2.3e+02)
DM4	802 (4e+02)	924 (9.2e+02)	1.02e+03 (7.2e+02)
DM7		1.09 (0.013)	1.1 (0.011)
DS5	424 (5e+02)	396 (6.3e+02)	412 (4.9e+02)
DS8	9.67 (0.064)	9.69 (0.046)	9.68 (0.053)
R.Wine	0.772 (0.4)	0.939 (0.46)	0.75 (0.23)
W.Wine	1.01 (0.31)	0.906 (0.29)	0.906 (0.33)
diabetes	1.68e+04 (5.5e+03)	1.78e+04 (4.4e+03)	1.62e+04 (7.6e+03)
pagie-1	0.191 (0.062)	0.197 (0.068)	0.217 (0.069)
vlad.4	0.0379 (0.00067)	0.0379 (0.003)	0.0379 (0.0019)

Table A.5: Parameters of the models trained using dataset DL1 and their results of Median MSE when applied to a new group of datasets.

Dataset	**DL1**	**DL1**	**DL1**
Emb. Size	**8**	**16**	**64**
Emb. type	**mean**	**mean**	**mean**
Distance	**euc**	**euc**	**euc**
Dataset		**Fitness**	
BHouse	55.9 (27)	52.8 (26)	48.3 (24)
Concrete	241 (60)	248 (80)	239 (52)
DM4	821 (7.6e+02)	902 (8e+02)	879 (4.8e+02)
DM7	1.1 (0.015)	1.1 (0.011)	1.09 (0.0078)
DS5	400 (6.2e+02)	404 (5.4e+02)	405 (4.1e+02)
DS8	9.69 (0.047)	9.7 (0.031)	9.7 (0.028)
R.Wine	0.773 (0.35)	1.03 (0.63)	9.55 (7)
W.Wine	0.943 (0.31)	0.971 (0.3)	0.935 (0.37)
diabetes	1.77e+04 (4.4e+03)	1.76e+04 (5.1e+03)	1.71e+04 (5.6e+03)
pagie-1	0.182 (0.07)	0.171 (0.066)	0.228 (0.071)
vlad.4	0.0379 (0.0026)	0.0503 (0.021)	0.0379 (0.0025)

Table A.6: Parameters of the models trained using dataset DM7 and their results of Median MSE when applied to a new group of datasets.

Dataset	**DM7**	**DM7**	**DM7**
Emb. Size	**8**	**16**	**64**
Emb. type	**mean**	**mean**	**mean**
Distance	**euc**	**euc**	**euc**
Dataset		**Fitness**	
BHouse	43.7 (29)	47.3 (27)	57.2 (29)
Concrete	247 (1e+02)	247 (1e+02)	267 (1.1e+02)
DM4	853 (7.8e+02)	990 (9.3e+02)	986 (7e+02)
DM7		1.1 (0.013)	1.09 (0.0078)
DS5	424 (4.9e+02)	377 (3.3e+02)	390 (4.3e+02)
DS8	9.66 (0.066)	9.7 (0.044)	9.7 (0.044)
R.Wine	0.752 (0.36)	0.741 (0.43)	9.55 (7)
W.Wine	0.988 (0.41)	0.943 (0.24)	0.915 (0.26)
diabetes	1.68e+04 (7.4e+03)	1.78e+04 (5.1e+03)	1.61e+04 (5.9e+03)
pagie-1	0.179 (0.059)	0.182 (0.069)	0.205 (0.066)
vlad.4	0.0379 (0.002)	0.0379 (0.003)	0.0379 (0.002)

Table A.7: Parameters of the models trained using dataset DM4 and their results of Median MSE when applied to a new group of datasets.

Dataset	**DM4**	**DM4**	**DM4**
Emb. Size	**8**	**16**	**64**
Emb. type	**concat**	**concat**	**concat**
Distance	**cos**	**cos**	**cos**
Dataset		**Fitness**	
BHouse	47.1 (27)	47 (29)	43.7 (25)
Concrete	256 (66)	258 (72)	251 (73)
DM4	678 (6.2e+02)	706 (6.2e+02)	802 (6.2e+02)
DM7	1.08 (0.083)	1.09 (0.021)	1.09 (0.017)
DS5	441 (5.4e+02)	446 (4.9e+02)	459 (5.6e+02)
DS8	8.49 (1.7)	9.39 (0.93)	9.5 (0.8)
R.Wine	0.779 (0.36)	0.74 (0.53)	0.732 (0.35)
W.Wine	0.988 (0.35)	0.938 (0.34)	0.998 (0.43)
diabetes	1.69e+04 (8.1e+03)	1.69e+04 (6e+03)	1.72e+04 (1e+04)
pagie-1	0.188 (0.053)	0.164 (0.031)	0.176 (0.057)
vlad.4	0.038 (0.0027)	0.0394 (0.0066)	0.0405 (0.0083)

Table A.8: Parameters of the models trained using dataset DL1 and their results of Median MSE when applied to a new group of datasets.

Dataset	**DL1**	**DL1**	**DL1**
Emb. Size	**8**	**16**	**64**
Emb. type	**concat**	**concat**	**concat**
Distance	**cos**	**cos**	**cos**
Dataset		**Fitness**	
BHouse	49.8 (28)	45.2 (30)	49.5 (31)
Concrete	231 (72)	243 (53)	259 (77)
DM4	732 (7.1e+02)	902 (7.4e+02)	628 (5e+02)
DM7	1.09 (0.025)	1.09 (0.072)	1.09 (0.045)
DS5	440 (5.4e+02)	446 (4.9e+02)	441 (5.5e+02)
DS8	8.8 (1.1)	8.99 (1.3)	9.05 (1.4)
R.Wine	0.709 (0.52)	0.712 (0.3)	0.808 (0.5)
W.Wine	1.01 (0.36)	1.03 (0.4)	0.949 (0.38)
diabetes	1.7e+04 (8.2e+03)	1.7e+04 (5.5e+03)	1.69e+04 (9.3e+03)
pagie-1	0.176 (0.052)	0.193 (0.066)	0.18 (0.044)
vlad.4	0.0503 (0.021)	0.0503 (0.021)	0.04 (0.01)

Table A.9: Parameters of the models trained using dataset DM7 and their results of Median MSE when applied to a new group of datasets.

Dataset	**DM7**	**DM7**	**DM7**
Emb. Size	**8**	**16**	**64**
Emb. type	**concat**	**concat**	**concat**
Distance	**cos**	**cos**	**cos**
Dataset		**Fitness**	
BHouse	44.3 (23)	44.6 (30)	55.2 (29)
Concrete	248 (57)	248 (86)	248 (94)
DM4	816 (7.6e+02)	694 (5.2e+02)	743 (5.8e+02)
DM7	1.09 (0.099)	1.09 (0.083)	1.09 (0.018)
DS5	444 (4.3e+02)	439 (5.4e+02)	411 (4.5e+02)
DS8	9.07 (1)	9.11 (1.5)	9.57 (0.55)
R.Wine	0.775 (0.39)	0.752 (0.47)	0.796 (0.36)
W.Wine	0.927 (0.35)	1.03 (0.34)	1.01 (0.33)
diabetes	1.69e+04 (9e+03)	1.69e+04 (7.1e+03)	1.69e+04 (8.1e+03)
pagie-1	0.175 (0.047)	0.177 (0.053)	0.179 (0.061)
vlad.4	0.0394 (0.0039)	0.0392 (0.01)	0.0408 (0.0095)

Table A.10: Parameters of the models trained using dataset DM4 and their results of Median MSE when applied to a new group of datasets.

Dataset	**DM4**	**DM4**	**DM4**
Emb. Size	**8**	**16**	**64**
Emb. type	**concat**	**concat**	**concat**
Distance	**euc**	**euc**	**euc**
Dataset		**Fitness**	
BHouse	62.6 (32)	63.4 (28)	60 (31)
Concrete	263 (76)	260 (61)	274 (1.3e+02)
DM4	891 (9.3e+02)	1.03e+03 (9.5e+02)	1.18e+03 (8.7e+02)
DM7	0.957 (0.067)	0.95 (0.058)	0.962 (0.065)
DS5	484 (6.2e+02)	420 (1.7e+02)	547 (5.9e+02)
DS8	9.69 (0.045)	9.68 (0.15)	9.55 (0.62)
R.Wine	0.727 (0.58)	0.703 (0.23)	0.772 (0.55)
W.Wine	0.881 (0.23)	0.865 (0.22)	0.942 (0.36)
diabetes	1.74e+04 (1.1e+04)	1.68e+04 (1.2e+04)	1.77e+04 (8.6e+03)
pagie-1	0.148 (0.027)	0.148 (0.022)	0.145 (0.014)
vlad.4	0.0379 (0.0013)	0.0379 (0.0021)	0.0379 (0.0021)

Table A.11: Parameters of the models trained using dataset DL1 and their results of Median MSE when applied to a new group of datasets

Dataset	**DL1**	**DL1**	**DL1**
Emb. Size	**8**	**16**	**64**
Emb. type	**concat**	**concat**	**concat**
Distance	**euc**	**euc**	**euc**
Dataset		**Fitness**	
BHouse	57.4 (27)	62.4 (34)	45.1 (28)
Concrete	250 (78)	260 (67)	270 (82)
DM4	1.04e+03 (6.9e+02)	1.01e+03 (7.3e+02)	1.03e+03 (8.1e+02)
DM7	0.945 (0.076)	0.958 (0.067)	0.948 (0.076)
DS5	421 (4.8e+02)	440 (4.8e+02)	464 (5.5e+02)
DS8	9.68 (0.035)	9.64 (0.19)	9.23 (0.99)
R.Wine	0.664 (0.38)	0.819 (0.34)	0.75 (0.5)
W.Wine	0.888 (0.33)	0.84 (0.2)	0.856 (0.29)
diabetes	1.75e+04 (1.1e+04)	1.74e+04 (8.4e+03)	1.73e+04 (1.1e+04)
pagie-1	0.149 (0.029)	0.147 (0.018)	0.149 (0.021)
vlad.4	0.0379 (0.0019)	0.0503 (0.021)	0.0379 (0.0021)

Table A.12: Parameters of the models trained using dataset DM7 and their results of Median MSE when applied to a new group of datasets

Dataset	**DM7**	**DM7**	**DM7**
Emb. Size	**8**	**16**	**64**
Emb. type	**concat**	**concat**	**concat**
Distance	**euc**	**euc**	**euc**
Dataset		**Fitness**	
BHouse	56.2 (32)	52.1 (36)	61.3 (35)
Concrete	276 (3.5e+02)	261 (82)	274 (75)
DM4	973 (6.8e+02)	715 (7.3e+02)	1.05e+03 (9.2e+02)
DM7	1.03 (0.16)	0.96 (0.065)	0.962 (0.085)
DS5	502 (5.7e+02)	455 (4.3e+02)	451 (4.8e+02)
DS8	9.69 (0.049)	9.68 (0.078)	9.46 (0.55)
R.Wine	0.674 (0.45)	0.754 (0.58)	0.807 (0.46)
W.Wine	0.889 (0.39)	0.796 (0.17)	0.957 (0.43)
diabetes	1.7e+04 (1e+04)	1.75e+04 (1.1e+04)	1.65e+04 (1.2e+04)
pagie-1	0.144 (0.027)	0.145 (0.02)	0.147 (0.022)
vlad.4	0.0379 (0.0019)	0.0379 (0.0021)	0.0379 (0.0021)

A.2 Evaluation using Median Entropy Transformed Semantics

Table A.13: Parameters of the models trained using datasets DM4 and DL1 and their results of Median Entropy Transformed Semantics when applied to a new group of datasets.

Dataset	**DM4**	**DM4**	**DM4**	**DL1**	**DL1**	**DL1**
Emb. Size	**8**	**16**	**64**	**8**	**16**	**64**
Emb. type	**mean**	**mean**	**mean**	**mean**	**mean**	**mean**
Distance	**cos**	**cos**	**cos**	**cos**	**cos**	**cos**
Dataset			**Entropy Transformed Semantics**			
BHouse	1.44 (0.41)	1.54 (0.36)	1.62 (0.49)	1.52 (0.38)	1.5 (0.35)	1.6 (0.32)
Concrete	1.26 (0.34)	1.3 (0.37)	1.39 (0.24)	1.33 (0.34)	1.3 (0.37)	1.29 (0.36)
DM4	1.11 (0.35)	1.12 (0.35)	1.22 (0.38)	1.25 (0.42)	1.13 (0.41)	1.15 (0.22)
DM7	1.01 (0.17)	1.77 (0.25)	1.59 (0.48)	1.64 (0.22)	1.74 (0.23)	1.59 (0.48)
DS5	1.43 (0.49)	1.42 (0.45)	1.54 (0.44)	1.49 (0.44)	1.43 (0.45)	1.46 (0.45)
DS8	0.76 (0.48)	1.03 (0.29)	1.12 (0.51)	0.69 (0.56)	0.803 (0.45)	1.12 (0.24)
R.Wine	1.77 (0.24)	1.71 (0.32)	1.67 (0.22)	1.7 (0.39)	1.69 (0.27)	1.71 (0.26)
W.Wine	1.64 (0.26)	1.63 (0.22)	1.55 (0.21)	1.6 (0.22)	1.63 (0.27)	1.63 (0.21)
diabetes	1.08 (0.42)	1.08 (0.44)	1.08 (0.45)	1.08 (0.42)	1.08 (0.45)	1.08 (0.42)
pagie-1	2.01 (0.29)	2.07 (0.3)	1.71 (0.39)	2.04 (0.27)	1.96 (0.33)	1.92 (0.27)
vlad.4	2.39 (0.11)	2.08 (0.11)	1.68 (0.19)	2.29 (0.14)	2.29 (0.14)	2 (0.15)

Table A.14: Parameters of the models trained using datasets DM7 and DM4 and their results of Median Entropy Transformed Semantics when applied to a new group of datasets

Dataset	**DM7**	**DM7**	**DM7**	**DM4**	**DM4**	**DM4**
Emb. Size	**8**	**16**	**64**	**8**	**16**	**64**
Emb. type	**mean**	**mean**	**mean**	**mean**	**mean**	**mean**
Distance	**cos**	**cos**	**cos**	**euc**	**euc**	**euc**
Dataset	**Entropy Transformed Semantics**					
BHouse	1.42 (0.37)	1.54 (0.28)	1.56 (0.25)	1.57 (0.34)	1.61 (0.28)	1.67 (0.24)
Concrete	1.27 (0.36)	1.33 (0.37)	1.34 (0.31)	1.38 (0.24)	1.36 (0.34)	1.35 (0.32)
DM4	1.15 (0.34)	1.19 (0.34)	1.22 (0.42)	1.23 (0.32)	1.17 (0.44)	1.22 (0.33)
DM7	1.17 (0.25)	1.6 (0.3)	1.59 (0.48)		1.57 (0.28)	1.52 (0.29)
DS5	1.46 (0.48)	1.45 (0.46)	1.47 (0.47)	1.49 (0.41)	1.49 (0.41)	1.55 (0.52)
DS8	0.824 (0.52)	0.766 (0.37)	1.14 (0.25)	1.18 (0.26)	1.09 (0.48)	1.19 (0.43)
R.Wine	1.71 (0.28)	1.69 (0.3)	1.67 (0.29)	1.68 (0.24)	1.58 (0.28)	1.65 (0.19)
W.Wine	1.62 (0.19)	1.64 (0.31)	1.61 (0.18)	1.59 (0.2)	1.55 (0.17)	1.53 (0.23)
diabetes	1.08 (0.42)	1.08 (0.42)	1.08 (0.47)	1.05 (0.34)	1.08 (0.36)	1.06 (0.3)
pagie-1	2.05 (0.31)	2.09 (0.32)	1.98 (0.44)	1.71 (0.29)	1.52 (0.24)	1.54 (0.22)
vlad.4	2.36 (0.095)	2.08 (0.11)	1.98 (0.12)	1.88 (0.14)	1.44 (0.27)	0.984 (0.45)

Table A.15: Parameters of the models trained using datasets DL1 and DM7 and their results of Median Entropy Transformed Semantics when applied to a new group of datasets.

Dataset	**DL1**	**DL1**	**DL1**	**DM7**	**DM7**	**DM7**
Emb. Size	**8**	**16**	**64**	**8**	**16**	**64**
Emb. type	**mean**	**mean**	**mean**	**mean**	**mean**	**mean**
Distance	**euc**	**euc**	**euc**	**euc**	**euc**	**euc**
Dataset	**Entropy Transformed Semantics**					
BHouse	1.58 (0.41)	1.56 (0.43)	1.6 (0.27)	1.6 (0.26)	1.59 (0.29)	1.7 (0.25)
Concrete	1.38 (0.25)	1.36 (0.29)	1.38 (0.21)	1.33 (0.23)	1.29 (0.31)	1.37 (0.23)
DM4	1.23 (0.25)	1.22 (0.36)	1.19 (0.31)	1.18 (0.38)	1.24 (0.26)	1.3 (0.37)
DM7	1.46 (0.36)	1.56 (0.31)	1.59 (0.48)		1.48 (0.39)	1.59 (0.48)
DS5	1.53 (0.44)	1.56 (0.39)	1.55 (0.5)	1.52 (0.41)	1.54 (0.41)	1.56 (0.38)
DS8	0.997 (0.45)	1.19 (0.42)	1.16 (0.26)	1.13 (0.41)	1.17 (0.31)	1.2 (0.42)
R.Wine	1.66 (0.27)	1.58 (0.25)	0.684 (0.4)	1.65 (0.24)	1.6 (0.25)	0.684 (0.4)
W.Wine	1.5 (0.25)	1.52 (0.27)	1.56 (0.26)	1.58 (0.22)	1.55 (0.27)	1.58 (0.29)
diabetes	1.07 (0.33)	1.07 (0.39)	1.04 (0.36)	1.05 (0.37)	1.05 (0.38)	1.04 (0.35)
pagie-1	1.6 (0.28)	1.49 (0.22)	1.54 (0.2)	1.81 (0.3)	1.59 (0.25)	1.75 (0.37)
vlad.4	1.61 (0.17)	2.29 (0.14)	1.36 (0.27)	1.76 (0.13)	1.46 (0.26)	1.3 (0.27)

Table A.16: Parameters of the models trained using datasets DM4 and DL1 and their results of Median Entropy Transformed Semantics when applied to a new group of datasets.

Dataset	**DM4**	**DM4**	**DM4**	**DL1**	**DL1**	**DL1**
Emb. Size	**8**	**16**	**64**	**8**	**16**	**64**
Emb. type	**concat**	**concat**	**concat**	**concat**	**concat**	**concat**
Distance	**cos**	**cos**	**cos**	**cos**	**cos**	**cos**
Dataset	**Entropy Transformed Semantics**					
BHouse	1.52 (0.37)	1.56 (0.36)	1.57 (0.39)	1.54 (0.37)	1.51 (0.27)	1.46 (0.34)
Concrete	1.32 (0.39)	1.3 (0.31)	1.3 (0.33)	1.33 (0.34)	1.3 (0.27)	1.3 (0.31)
DM4	1.12 (0.36)	1.11 (0.38)	1.18 (0.33)	1.13 (0.46)	1.17 (0.36)	1.1 (0.33)
DM7	1.47 (0.39)	1.74 (0.34)	1.81 (0.14)	1.75 (0.37)	1.78 (0.21)	1.7 (0.19)
DS5	1.44 (0.47)	1.5 (0.47)	1.5 (0.45)	1.45 (0.51)	1.5 (0.49)	1.44 (0.47)
DS8	0.836 (0.64)	1.06 (0.26)	1.07 (0.27)	0.862 (0.42)	0.896 (0.33)	0.884 (0.41)
R.Wine	1.73 (0.21)	1.7 (0.24)	1.69 (0.24)	1.67 (0.29)	1.71 (0.23)	1.75 (0.22)
W.Wine	1.61 (0.23)	1.63 (0.2)	1.62 (0.27)	1.61 (0.28)	1.62 (0.23)	1.59 (0.22)
diabetes	1.09 (0.39)	1.09 (0.42)	1.04 (0.41)	1.09 (0.38)	1.09 (0.42)	1.08 (0.42)
pagie-1	2 (0.25)	2.06 (0.26)	1.86 (0.37)	2.08 (0.4)	1.92 (0.39)	1.97 (0.25)
vlad.4	2.4 (0.091)	2.07 (0.13)	2.01 (0.14)	2.29 (0.14)	2.29 (0.14)	2.05 (0.14)

Table A.17: Parameters of the models trained using datasets DM7 and DM4 and their results of Median Entropy Transformed Semantics when applied to a new group of datasets

Dataset	**DM7**	**DM7**	**DM7**	**DM4**	**DM4**	**DM4**
Emb. Size	**8**	**16**	**64**	**8**	**16**	**64**
Emb. type	**concat**	**concat**	**concat**	**concat**	**concat**	**concat**
Distance	**cos**	**cos**	**cos**	**euc**	**euc**	**euc**
Dataset	**Entropy Transformed Semantics**					
BHouse	1.54 (0.35)	1.52 (0.31)	1.54 (0.42)	1.6 (0.27)	1.56 (0.25)	1.49 (0.27)
Concrete	1.31 (0.4)	1.31 (0.33)	1.31 (0.34)	1.39 (0.25)	1.32 (0.34)	1.28 (0.33)
DM4	1.18 (0.34)	1.21 (0.38)	1.17 (0.39)	1.24 (0.33)	1.22 (0.45)	1.2 (0.38)
DM7	1.36 (0.32)	1.6 (0.29)	1.76 (0.24)	1 (0.65)	0.698 (0.57)	0.575 (0.42)
DS5	1.42 (0.47)	1.44 (0.46)	1.45 (0.39)	1.65 (0.43)	1.62 (0.35)	1.53 (0.36)
DS8	0.946 (0.25)	0.921 (0.3)	1.12 (0.18)	1.35 (0.3)	1.47 (0.31)	1.51 (0.21)
R.Wine	1.69 (0.28)	1.66 (0.18)	1.73 (0.15)	1.35 (0.21)	1.37 (0.23)	1.29 (0.19)
W.Wine	1.61 (0.18)	1.66 (0.25)	1.62 (0.26)	1.38 (0.16)	1.33 (0.18)	1.26 (0.18)
diabetes	1.05 (0.46)	1.08 (0.41)	1.07 (0.42)	1.1 (0.3)	1.05 (0.45)	0.944 (0.43)
pagie-1	2.02 (0.36)	1.96 (0.41)	1.87 (0.36)	1.27 (0.18)	1.21 (0.14)	1.17 (0.18)
vlad.4	2.21 (0.12)	2.05 (0.11)	2.01 (0.11)	0.986 (0.51)	0.996 (0.53)	0.996 (0.53)

Table A.18: Parameters of the models trained using datasets DL1 and DM7 and their results of Median Entropy Transformed Semantics when applied to a new group of datasets.

Dataset	**DL1**	**DL1**	**DL1**	**DM7**	**DM7**	**DM7**
Emb. Size	**8**	**16**	**64**	**8**	**16**	**64**
Emb. type	**concat**	**concat**	**concat**	**concat**	**concat**	**concat**
Distance	**euc**	**euc**	**euc**	**euc**	**euc**	**euc**
Dataset	**Entropy Transformed Semantics**					
BHouse	1.66 (0.28)	1.6 (0.25)	1.59 (0.28)	1.62 (0.27)	1.62 (0.29)	1.54 (0.26)
Concrete	1.43 (0.24)	1.35 (0.26)	1.24 (0.29)	1.35 (0.31)	1.34 (0.23)	1.29 (0.32)
DM4	1.28 (0.33)	1.29 (0.43)	1.26 (0.37)	1.25 (0.34)	1.28 (0.33)	1.16 (0.47)
DM7	0.815 (0.57)	0.714 (0.58)	0.636 (0.44)	1.51 (0.22)	0.79 (0.59)	0.594 (0.5)
DS5	1.48 (0.3)	1.56 (0.33)	1.53 (0.27)	1.56 (0.38)	1.55 (0.42)	1.5 (0.31)
DS8	1.41 (0.26)	1.5 (0.15)	1.49 (0.22)	1.42 (0.39)	1.45 (0.25)	1.48 (0.23)
R.Wine	1.34 (0.19)	1.29 (0.18)	1.29 (0.21)	1.43 (0.11)	1.38 (0.26)	1.31 (0.21)
W.Wine	1.35 (0.12)	1.35 (0.21)	1.3 (0.18)	1.32 (0.19)	1.35 (0.15)	1.25 (0.17)
diabetes	1.05 (0.4)	1.05 (0.38)	1.03 (0.32)	1.08 (0.52)	1.06 (0.36)	1.13 (0.32)
pagie-1	1.31 (0.2)	1.19 (0.17)	1.12 (0.16)	1.31 (0.19)	1.18 (0.2)	1.14 (0.18)
vlad.4	0.912 (0.52)	2.29 (0.14)	0.996 (0.53)	1 (0.56)	0.996 (0.53)	0.996 (0.53)

A.3 Evaluation using Pseudo-Isomorph Diversity

Table A.19: Parameters of the models trained using dataset DM4 and their results of Median Pseudo-Isomorph when applied to a new group of datasets

Dataset	**DM4**	**DM4**	**DM4**
Emb. Size	**8**	**16**	**64**
Emb. type	**mean**	**mean**	**mean**
Distance	**cos**	**cos**	**cos**
Dataset	**Pseudo-Isomorph**		
BHouse	28.52 (2.623)	28.6 (2.882)	29.16 (3.074)
Concrete	28.53 (2.417)	29.35 (1.98)	28.78 (2.922)
DM4	27.53 (3.196)	27.92 (3.549)	28.52 (4.304)
DM7	28.2 (2.961)	29.93 (2.529)	37.13 (7.618)
DS5	29.05 (2.152)	28.95 (2.623)	28.7 (2.716)
DS8	28.34 (3.387)	28.54 (2.863)	31.24 (2.618)
R.Wine	28.5 (2.686)	29.28 (3.539)	29.69 (3.564)
W.Wine	28.79 (3.73)	29.07 (2.971)	29.85 (2.51)
diabetes	26.58 (2.828)	26.53 (2.814)	26.68 (2.49)
pagie-1	30.88 (2.529)	30.79 (1.99)	29.85 (3.181)
vlad.4	23.6 (3.593)	20.96 (2.402)	16.62 (6.696)

Table A.20: Parameters of the models trained using dataset DM4 and their results of Median Pseudo-Isomorph when applied to a new group of datasets.

Dataset		**DM4**	
Emb. Size	**8**	**16**	**64**
Emb. type		**mean**	
Distance		**cos**	
Dataset		**Pseudo-Isomorph**	
BHouse	28.6 (2.755)	29.01 (3.206)	28.8 (2.446)
Concrete	28.8 (2.701)	28.79 (2.27)	28.94 (2.475)
DM4	28.01 (3.461)	28.13 (2.917)	28.47 (3.912)
DM7	29.79 (2.76)	29.77 (3.005)	37.13 (7.618)
DS5	28.97 (2.652)	28.81 (2.623)	28.65 (2.819)
DS8	29.19 (3.137)	28.54 (2.696)	29.79 (3.51)
R.Wine	28.68 (2.853)	29.64 (3.358)	28.68 (2.245)
W.Wine	29.63 (2.985)	29.6 (1.745)	29.38 (2.99)
diabetes	26.53 (2.828)	26.68 (2.632)	26.53 (2.961)
pagie-1	30.99 (2.784)	30.6 (2.294)	30.08 (2.985)
vlad.4	36.93 (8.794)	36.93 (8.794)	20.44 (3.353)

Table A.21: Parameters of the models trained using dataset DM7 and their results of Median Pseudo-Isomorph when applied to a new group of datasets.

Dataset		**DM7**	
Emb. Size	**8**	**16**	**64**
Emb. type		**mean**	
Distance		**cos**	
Dataset		**Pseudo-Isomorph**	
BHouse	28.75 (2.436)	28.66 (3.02)	28.86 (2.755)
Concrete	28.48 (2.613)	29.74 (3.147)	28.88 (2.328)
DM4	27.3 (2.77)	27.51 (3.637)	28.47 (3.897)
DM7	29.69 (2.618)	31 (2.176)	37.13 (7.618)
DS5	28.63 (2.627)	28.65 (2.73)	28.74 (2.623)
DS8	28.85 (2.966)	29.02 (3.539)	29.84 (3.304)
R.Wine	28.58 (2.01)	29.24 (3.755)	28.68 (2.422)
W.Wine	28.2 (3.819)	29.25 (2.902)	30.2 (2.613)
diabetes	26.58 (2.828)	26.53 (2.706)	26.53 (2.275)
pagie-1	30.68 (1.985)	30.84 (2.603)	30.85 (1.966)
vlad.4	23.65 (3.52)	20.08 (2.868)	19.34 (3.363)

Table A.22: Parameters of the models trained using dataset DM4 and their results of Median Pseudo-Isomorph when applied to a new group of datasets.

Dataset		**DM4**	
Emb. Size	**8**	**16**	**64**
Emb. type		**mean**	
Distance		**Euclidean**	
Dataset		**Pseudo-Isomorph**	
BHouse	28.77 (2.201)	29.45 (3.103)	29.31 (2.574)
Concrete	28.58 (2.676)	28.83 (2.397)	29.29 (2.946)
DM4	28.11 (4.098)	27.74 (3.225)	28.3 (4.167)
DM7		29.21 (6.054)	27.32 (3.196)
DS5	29.18 (2.451)	29.57 (1.471)	29.89 (3.123)
DS8	29.59 (2.029)	31.36 (2.686)	30.95 (2.456)
R.Wine	29.54 (3.328)	29.27 (3.25)	29.43 (2.843)
W.Wine	29.79 (2.784)	29.26 (2.24)	28.27 (3.652)
diabetes	26.56 (2.716)	27 (3.034)	27.03 (3.475)
pagie-1	27.94 (2.578)	24.48 (3.471)	24.15 (3.088)
vlad.4	18.9 (4.475)	15.08 (6.667)	10.03 (6.48)

Table A.23: Parameters of the models trained using dataset DL1 and their results of Median Pseudo-Isomorph when applied to a new group of datasets.

Dataset		**DL1**	
Emb. Size	**8**	**16**	**64**
Emb. type		**mean**	
Distance		**Euclidean**	
Dataset		**Pseudo-Isomorph**	
BHouse	29.58 (2.279)	29.13 (3.348)	28.9 (4.02)
Concrete	28.82 (2.598)	29.24 (2.716)	28.91 (2.397)
DM4	28.36 (3.289)	27.64 (4.333)	27.44 (3.324)
DM7	28.36 (3.505)	28.11 (4.054)	37.13 (7.618)
DS5	29.64 (1.961)	29.38 (2.368)	30.06 (3.314)
DS8	31.27 (2.789)	31.63 (2.588)	31.27 (3.132)
R.Wine	29.31 (3.225)	28.56 (4.025)	36.61 (7.647)
W.Wine	28.98 (3.319)	28.89 (2.711)	29.08 (3.696)
diabetes	26.67 (2.966)	26.4 (2.627)	27.04 (3.765)
pagie-1	26.77 (2.377)	23.26 (3.863)	23.37 (3.373)
vlad.4	15.81 (7.123)	36.93 (8.794)	12.41 (6.833)

Table A.24: Parameters of the models trained using dataset DM7 and their results of Median Pseudo-Isomorph when applied to a new group of datasets.

Dataset		**DM7**	
Emb. Size	**8**	**16**	**64**
Emb. type		**mean**	
Distance		**Euclidean**	
Dataset		**Pseudo-Isomorph**	
BHouse	29.32 (2.549)	29.17 (3.789)	29.61 (3.255)
Concrete	28.87 (3.485)	29.27 (1.309)	29.35 (3.27)
DM4	28.9 (3.966)	28.37 (2.623)	28.39 (4.441)
DM7		28.25 (4.564)	37.13 (7.618)
DS5	29.41 (2.51)	29.26 (1.873)	29.09 (2.691)
DS8	29.6 (2.265)	31.01 (2.02)	30.83 (3.196)
R.Wine	29.8 (3.529)	29.42 (2.99)	36.61 (7.647)
W.Wine	29.87 (3.451)	29.29 (2.779)	29.36 (2.809)
diabetes	26.46 (2.799)	26.92 (3.088)	27.01 (3.534)
pagie-1	30.04 (2.897)	24.71 (3.578)	28.63 (3.995)
vlad.4	17.86 (5.201)	14.25 (7.725)	13.02 (6.956)

Table A.25: Parameters of the models trained using dataset DM4 and their results of Median Pseudo-Isomorph when applied to a new group of datasets.

Dataset		**DM4**	
Emb. Size	**8**	**16**	**64**
Emb. type		**mean**	
Distance		**Cosine**	
Dataset		**Pseudo-Isomorph**	
BHouse	28.9 (2.922)	29.22 (3.804)	28.92 (3.02)
Concrete	28.92 (2.368)	29.02 (2.721)	28.84 (2.093)
DM4	28.08 (3.667)	28.26 (3.441)	28.02 (3.039)
DM7	29.59 (2.402)	29.96 (2.485)	30.15 (2.417)
DS5	29.13 (1.931)	29.17 (2.373)	29.22 (2.328)
DS8	28.8 (3.319)	28.21 (2.52)	27.8 (3.608)
R.Wine	29.53 (3.304)	29.68 (2.466)	29.43 (3.054)
W.Wine	29.77 (2.853)	29.5 (2.48)	29.96 (2.216)
diabetes	26.53 (2.642)	26.41 (2.956)	26.69 (2.809)
pagie-1	31.09 (2.176)	30.7 (2.319)	30.53 (3.446)
vlad.4	23.4 (2.598)	20.14 (2.779)	20.81 (3.809)

Table A.26: Parameters of the models trained using dataset DL1 and their results of Median Pseudo-Isomorph when applied to a new group of datasets.

Dataset		**DL1**	
Emb. Size	**8**	**16**	**64**
Emb. type		**mean**	
Distance		**Cosine**	
Dataset		**Pseudo-Isomorph**	
BHouse	29.48 (2.706)	29.25 (3.23)	29.18 (2.779)
Concrete	28.91 (2.162)	29.15 (2.946)	29.02 (2.716)
DM4	28.04 (2.73)	27.88 (3.593)	27.94 (2.873)
DM7	29.7 (2.123)	30.28 (1.672)	29.78 (2.539)
DS5	29.04 (1.809)	29.19 (2.373)	29.13 (1.931)
DS8	28.65 (4.069)	28.84 (3.794)	28.6 (3.926)
R.Wine	29.4 (2.711)	29.01 (3.603)	29.05 (2.887)
W.Wine	29.82 (2.245)	29.25 (2.475)	29.57 (2.471)
diabetes	26.72 (2.819)	26.67 (2.745)	26.72 (2.74)
pagie-1	31.08 (2.221)	30.37 (2.902)	30.31 (2.201)
vlad.4	36.93 (8.794)	36.93 (8.794)	20.78 (3.426)

Table A.27: Parameters of the models trained using dataset DM7 and their results of Median Pseudo-Isomorph when applied to a new group of datasets

Dataset		**DM7**	
Emb. Size	**8**	**16**	**64**
Emb. type		**mean**	
Distance		**Cosine**	
Dataset		**Pseudo-Isomorph**	
BHouse	28.6 (3.255)	28.98 (1.657)	29.32 (2.99)
Concrete	28.91 (2.539)	28.79 (2.118)	28.88 (2.216)
DM4	27.83 (3.475)	28.81 (3.662)	28.74 (3.588)
DM7	29.76 (2.755)	30.09 (1.99)	29.85 (3.446)
DS5	29.13 (2.211)	29.04 (1.931)	29.26 (2.078)
DS8	27.87 (3.716)	28.26 (3.779)	29.22 (3.049)
R.Wine	28.91 (3.917)	29.19 (3.034)	30.12 (4.147)
W.Wine	29.6 (2.392)	29.51 (3.113)	29.37 (3.426)
diabetes	26.72 (2.701)	26.49 (2.73)	26.48 (2.716)
pagie-1	31.03 (2.176)	31.17 (2.098)	30.28 (3.083)
vlad.4	23.39 (3.152)	21.09 (3.49)	20.52 (3.118)

Table A.28: Parameters of the models trained using dataset DM4 and their results of Median Pseudo-Isomorph when applied to a new group of datasets

Dataset		**DM4**	
Emb. Size	**8**	**16**	**64**
Emb. type		**mean**	
Distance		**Euclidean**	
Dataset		**Pseudo-Isomorph**	
BHouse	29.45 (4.093)	29.92 (3.618)	28.36 (3.647)
Concrete	30.45 (2.946)	30.35 (3.319)	30.35 (2.912)
DM4	28.91 (3.647)	28.27 (4.25)	28.83 (5.093)
DM7	17.18 (4.946)	15.26 (3.77)	15.06 (2.809)
DS5	31.1 (2.328)	30.47 (2.431)	31.35 (2.412)
DS8	29.7 (3.828)	28.7 (2.848)	27.21 (3.417)
R.Wine	22.02 (4.559)	21.92 (4.833)	19.64 (3.691)
W.Wine	22.8 (2.426)	22.08 (3.368)	18.52 (4.034)
diabetes	28.13 (2.593)	27.73 (2.324)	28.56 (2.892)
pagie-1	18.29 (3.593)	17.56 (2.716)	17.89 (3.088)
vlad.4	9.529 (7.186)	10.95 (7.559)	10.95 (7.559)

Table A.29: Parameters of the models trained using dataset DL1 and their results of Median Pseudo-Isomorph when applied to a new group of datasets

Dataset		**DL1**	
Emb. Size	**8**	**16**	**64**
Emb. type		**mean**	
Distance		**Euclidean**	
Dataset		**Pseudo-Isomorph**	
BHouse	30.44 (3.039)	29.73 (3.652)	29.38 (3.814)
Concrete	29.6 (1.858)	30.46 (2.123)	30.85 (2.672)
DM4	28.56 (3.525)	28.02 (3.338)	28.71 (3.735)
DM7	16.02 (3.882)	15.16 (3.966)	14.74 (4.083)
DS5	30.33 (2.324)	30.57 (3.24)	30.81 (2.186)
DS8	30.72 (2.843)	27.99 (3.328)	25.82 (3.534)
R.Wine	22.55 (5.691)	20.12 (3.627)	19.15 (3.578)
W.Wine	25.58 (4.99)	22.95 (3.549)	20.68 (3.387)
diabetes	27.87 (2.696)	27.64 (2.99)	27.85 (2)
pagie-1	19.38 (3.505)	17.05 (2.417)	17.01 (3.348)
vlad.4	9.843 (7.363)	36.93 (8.794)	10.95 (7.559)

Table A.30: Parameters of the models trained using dataset DM7 and their results of Median Pseudo-Isomorph when applied to a new group of datasets

Dataset	**DM7**		
Emb. Size	**8**	**16**	**64**
Emb. type	**mean**		
Distance	**Euclidean**		
Dataset	**Pseudo-Isomorph**		
BHouse	29.96 (2.779)	29.78 (3.01)	28.39 (3.392)
Concrete	30.08 (2.471)	30.28 (2.564)	30.39 (3.436)
DM4	28.98 (3.696)	28.16 (3.598)	28.81 (3.745)
DM7	21.38 (4.544)	15.91 (4.074)	15.21 (3.284)
DS5	30.67 (2.73)	30.72 (2.098)	30.86 (2.936)
DS8	30.6 (3.564)	29.22 (3.299)	26.81 (4.225)
R.Wine	25.47 (4.039)	23.03 (3.848)	19.38 (4.201)
W.Wine	25.21 (3.525)	22.86 (3.451)	18.54 (3.181)
diabetes	27.75 (2.441)	27.55 (2.206)	28.09 (2.603)
pagie-1	18.67 (3.24)	17.83 (3.338)	16.91 (3.348)
vlad.4	9.725 (7.735)	10.95 (7.559)	10.95 (7.559)