

MDSAA

Master Degree Program in
Data Science and Advanced Analytics

MULTITASK SYMBOLIC REGRESSION IN GENETIC PROGRAMMING

Adriana Monteiro

Master Thesis

presented as partial requirement for obtaining a Master's Degree in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

MULTITASK SYMBOLIC REGRESSION
IN GENETIC PROGRAMMING

by

Adriana Monteiro

Master Thesis presented as a partial requirement for obtaining the Master's degree in Data Science and Advanced Analytics, with a specialization in Data Science.

Supervised by

Leonardo Vanneschi, PhD, NOVA Information Management School

July, 2024

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Lisbon, 02-12-2024

Multitask Symbolic Regression in Genetic Programming

Copyright © Adriana Monteiro, NOVA Information Management School, NOVA University Lisbon.

The NOVA Information Management School and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

This document was created with the (pdf/Xe/Lua) $\LaTeX$ processor and the [NOVAthesis](#) template (v7.1.18) (Lourenço, 2021).

”

*“And here, poor fool!
With all my lore, I stand
no wiser than before.”*

— **Goethe**, *Faust*

ABSTRACT

Multitask Learning (MTL) is the process in which multiple problems are solved simultaneously, usually with shared representations or parameters. This approach is known to improve generalisation, performance and training time. MTL in Genetic Programming (GP) has received limited attention, due to the assumption that GP tailors solutions to a specific task and thus, that it is challenging to share these amongst different problems. This work introduces a novel method that evolves a tree that is shared between two tasks (also called problems), in which its terminals are pairs of the original features of the datasets. The fitness function of the common evolution is the average performance on each task, as an attempt to balance the improvement of both. The impact of different metrics for the common fitness function was studied, in specific the difference between Root Mean Squared Error (RMSE) and Relative Squared Error (RSE). The results revealed that the RSE is a more adequate measure than the other, as it achieves a better balance between the minimisation of error across both tasks. The effectiveness of the Common Tree (CT) in reducing overfitting and program size was verified for some cases, but it is not evident under what conditions this effect is predictable. In addition, across all datasets, the common evolution produces trees that for the same level of fitness, are smaller in size, indicating that it is preventing bloat. Other conclusions relate to the nature of the relationship between tasks. It was verified that the domain of problems is not sufficient to justify if a joint evolution will be successful. Additionally, some problems were verified to be better pairs than others and that their effectiveness is not symmetric. The CT is not expected to capture the complexity of each problem, leading to poorer results when compared with Standard Genetic Programming (StdGP). This was verified for all cases with the exception of the Istanbul dataset. To tackle this issue, the CT is transformed into a new feature that is later inserted into the dataset of each problem. A final evolutionary process is conducted to leverage the information from both evolutions. The final results yield no difference when compared with StdGP.

Keywords: Genetic Programming, Symbolic Regression, Multitask Learning

CONTENTS

List of Figures	ix
List of Tables	xi
List of Algorithms	xiii
Acronyms	xv
1 Introduction	1
2 Theoretical Background	5
2.1 Machine Learning	5
2.1.1 Machine Learning Categories	6
2.1.2 Model Training and Monitoring	6
2.1.3 Model Evaluation	7
2.2 Transfer Learning	7
2.2.1 Definitions	8
2.2.2 Categorization of Transfer Learning	9
2.2.3 Multitask Learning	11
2.3 Algorithms and Optimisation	11
2.4 Evolutionary Computation	12
2.4.1 Genetic Programming	12
3 Literature Review	19
3.1 Transfer Learning in Genetic Programming	19
3.2 Evolutionary Multitask Learning	21
3.2.1 Multitask Learning in Genetic Algorithms	21
3.2.2 Multitask Learning in Genetic Programming	22
4 Methodology	25
4.1 Overview of the Proposed Approach	25

4.2	Common Tree	25
4.2.1	Representation	25
4.2.2	Fitness Evaluation	27
4.2.3	Evolution	28
4.3	Information Transfer to Specific Problems	28
4.4	Framework	29
5	Experimental Study	31
5.1	Case Studies	31
5.1.1	QSAR Aquatic Toxicity (AT)	32
5.1.2	QSAR Fish Toxicity (FT)	33
5.1.3	Unified Parkinson's Disease Rating Scale (UPDRS)	34
5.1.4	Istanbul Stock Exchange	35
5.1.5	Human Oral Bioavailability	36
5.1.6	Human Oral Lethal Dose (LD50)	36
5.1.7	Plasma Protein Binding Levels (PPB)	36
5.2	Experiments	36
5.2.1	Parameter Settings	37
6	Results and Discussion	39
6.1	Common Evolution	39
6.1.1	Fitness Function	39
6.1.2	Performance	41
6.1.3	Overfitting Reduction	43
6.1.4	Program Size	46
6.2	Specific Evolution	52
7	Conclusions and Future Work	57
	Bibliography	59

LIST OF FIGURES

2.1	Example of a confusion matrix for a binary classification problem with classes 0 and 1. The darkest blue depicts predictions which are correct (TP and TN), and the incorrect predictions are shown with the lightest blue (FP and FN). The formulas for calculating the error metrics of accuracy, recall, precision and f1-score are also shown on the right side of the image. . . .	8
2.2	Transfer Learning Categories (Zhuang et al., 2021).	10
2.3	General process of an evolutionary algorithm, inspired by Vanneschi and Silva (2023)	13
2.4	Example of a symbolic regression genetic programming tree built with the sets $T = \{x_1, x_3, 4\}$ and $F = \{+, -\}$. The tree has depth $d = 2$ and corresponds to the mathematical function $g(x_1, x_3) = (x_1 + 4) - x_3$. . .	14
2.5	Example of a standard crossover. The grey boxes contain the crossover fragments.	16
2.6	Example of a standard mutation. The grey boxes contain the mutation fragments.	16
4.1	Dataset transformation for evaluating the common tree. Each pair of terminals of the original datasets, (x_i, y_j) , is mapped to a new terminal c_k . To facilitate the evaluation, the vector feature columns of the datasets are systematically replicated to form pairs of features.	26
4.2	An example of a common tree representation. (a) represents the tree with the common features c_k , which correspond to pairs of features of two tasks. (b) is feature correspondence of (a) into the features x_i , and (c) into the features y_j	27
6.1	Median best RMSE of CTGP50_RMSE (green) and CTGP50_RSE (orange) of the pair composed by the Istanbul and LD50 datasets. The RMSE shown in each plot is of the dataset in bold.	40

6.2	Median RMSE of Train (thicker curve) and Validation (thinner curve) across generations, comparing the best combinations of CTGP (orange) and StdGP (blue) for the Istanbul and UPDRS datasets across all population sizes. .	44
6.3	Boxplot of the best individual size at generation 100 over 30 runs of the AT, Istanbul and UPDRS datasets. The non-shaded part refers to CTGP and the shaded part to StdGP. The labels indicate the paired dataset if applicable, and population size.	47
6.4	Boxplot of the best individual size at generation 100 over 30 runs of the LD50, Bioavailability and PPB datasets. The non-shaded part refers to CTGP and the shaded part to StdGP. The labels indicate the paired dataset if applicable, and population size.	48
6.5	Boxplot of the best individual size at generation 100 over 30 runs of the FT dataset. The non-shaded part refers to CTGP and the shaded part to StdGP. The labels indicate the paired dataset if applicable, and population size.	49
6.6	Comparison of validation fitness against program size of the best individuals between CTGP (black dots) and StdGP (red triangles) for the AT, FT, I and UPDRS datasets. Columns are of the same population (P) size and rows of the same dataset.	50
6.7	Comparison of validation fitness against program size of the best individuals between CTGP (black dots) and StdGP (red triangles) for the Bioavailability, LD50 and PPB datasets. Columns are of the same population (P) size and rows of the same dataset.	51
6.8	Train (thinner line) and Test (thicker line) RMSE of CTGP (orange) and SpTGP (yellow) against StdGP (blue) across generations. The shaded area refers to the p-value (right axis) of the comparison of the test scores between methods. The dashed line references the significance level of 0.05. . . .	55
6.9	Train (thinner line) and Test (thicker line) RMSE of CTGP (orange) and SpTGP (yellow) of the Istanbul dataset.	56

LIST OF TABLES

2.1	Notations	9
5.1	Number of attributes (including target) and instances of the datasets used.	31
5.2	General statistics of the Aquatic Toxicity dataset attributes rounded to two decimal places.	32
5.3	General statistics of the Fish Toxicity dataset attributes rounded to two decimal places.	33
5.4	General statistics of the UPDRS dataset attributes.	34
5.5	General statistics of the Istanbul dataset attributes rounded to the third decimal place.	35
5.6	Parameter settings. The parameters enclosed in { } represent those that differed between experiments.	37
5.7	List of experiments conducted.	38
6.1	Median Train and Validation errors (Train : Validation) at the initial (G=0) and final (G=100) generations for datasets where the error increased when using RMSE in the common fitness function. Results are shown for Task1 of the pair Task1/Task2 of the experiment CTGP50_RMSE.	40
6.2	Median Train and Validation errors (Train : Validation) of the best CTGP experiments. For each dataset on the left, the corresponding error values and dataset pair are shown. The bold values indicate the best-performing experiment among the three population sizes.	41
6.3	Median Train and Validation errors (Train : Validation) of each StdGP experiment at generation 100.	42
6.4	Comparison of overfitting reduction between CTGP and StdGP across datasets and population sizes. The left column indicates in which dataset the overfitting was reduced. The terms used in the table are defined in the legend below.	45

6.5	Median train and validation error (Train : Validation) of the best SpTGP experiments for the AT, Bioavailability, FT and Istanbul datasets. For each dataset, the corresponding dataset pair is also shown above the error values. The bold values indicate the best performing experiment. The last row shows the performance of StdGP at generation 200.	53
6.6	Median train and validation error (Train : Validation) of the best SpTGP experiments for the LD50, PPB and UPDRS datasets. For each dataset, the corresponding dataset pair is also shown above the error values. The bold values indicate the best performing experiment. The last row shows the performance of StdGP at generation 200.	54
6.7	Train and test errors (Train : Test) of the proposed method and StdGP. CT(%) indicates the percentage of best performing individuals that contained the common tree as a terminal. The CT percentage for the Istanbul dataset is omitted, as the final result is the CT.	54

LIST OF ALGORITHMS

1	Pseudo-code of the common tree evolution.	29
2	RHH Initialisation with Custom Individual Creation Function	30
3	Pseudo-code for the individual generation.	30

ACRONYMS

R^2	Coefficient of determination <i>p.</i> 7
AI	Artificial Intelligence <i>pp.</i> 1, 5
AT	QSAR Aquatic Toxicity <i>pp.</i> 31, 41, 43, 44, 46, 52, 57, 58
Bioavailability	Human Oral Bioavailability <i>pp.</i> 31, 36, 44, 46
CT	Common Tree <i>pp.</i> v, xii, 2, 3, 25–28, 37, 39, 52, 54, 57, 58
CTGP	Common Tree Genetic Programming <i>pp.</i> 2, 3, 25, 31, 36–39, 41–44, 46, 52, 57, 58
DEAP	Distributed Evolutionary Algorithms in Python <i>p.</i> 29
DJSS	Dynamic Job Shop Scheduling <i>p.</i> 2
EC	Evolutionary Computation <i>pp.</i> 1, 2, 5, 12, 19, 21
FA	Feature Alignment <i>pp.</i> 20, 21, 23, 26
FT	QSAR Fish Toxicity <i>pp.</i> 31, 41–44, 46, 52, 57, 58
GA	Genetic Algorithm <i>pp.</i> 12, 13, 15, 22
GP	Genetic Programming <i>pp.</i> v, 1–3, 5, 12–15, 19–23, 25, 26, 36, 37, 46, 52, 57
Istanbul	Istanbul Stock Exchange <i>pp.</i> 31, 39, 41–43, 46, 52, 58
KNN	K-Nearest Neighbours <i>p.</i> 21
LD50	Median Oral Lethal Dose <i>pp.</i> 31, 36, 39, 41–43, 46, 58
MAE	Mean Absolute Error <i>p.</i> 7

MAPE	Mean Absolute Percentage Error <i>p.</i> 7
MFEA	Multifactorial Evolutionary Algorithm <i>pp.</i> 2, 21, 22
MFO	Multifactorial Optimisation <i>pp.</i> 2, 21, 22
ML	Machine Learning <i>pp.</i> 1, 5–7, 9, 11, 31
MTL	Multitask Learning <i>pp.</i> v, 1–3, 5, 11, 19, 21–23, 26, 31, 41, 57
PPB	Plasma Protein Binding Levels <i>pp.</i> 31, 36, 44, 46, 58
RHH	Ramped Half-and-Half <i>pp.</i> 14, 28, 37
RMSE	Root Mean Squared Error <i>pp.</i> v, 7, 27, 38–40
RSE	Relative Squared Error <i>pp.</i> v, 7, 27, 38–40
SpTGP	Specific Tree Genetic Programming <i>pp.</i> 2, 3, 25, 31, 36–38, 52, 57
SR	Symbolic Regression <i>pp.</i> 2, 13, 19, 20, 22, 23, 25, 28
StdGP	Standard Genetic Programming <i>pp.</i> v, 19, 20, 22, 28, 31, 36–38, 42–44, 46, 52, 57, 58
TL	Transfer Learning <i>pp.</i> 1–3, 5, 7–11, 19–21, 23, 42, 57
UPDRS	Unified Parkinson’s Disease Rating Scale <i>pp.</i> 31, 42–44, 46, 52, 57

INTRODUCTION

Genetic Programming (GP) is an Evolutionary Computation (EC) algorithm that automatically searches for the computer program that best solves a particular task (Koza, 1992). The advantage of GP in comparison to other Machine Learning (ML) techniques is its ability to perform implicit feature selection and derive mathematical expressions without predefining their structure. It has been successfully applied in several domains, such as classification (Espejo et al., 2010), regression (Muñoz et al., 2019) and data mining (Ravisankar et al., 2011).

GP iteratively improves solutions by evolving them in a process inspired by the Darwinian principles of natural selection and survival of the fittest (Darwin, 1859). Each potential solution is traditionally represented as a tree and assigned a fitness value, which quantifies how close the solution's output is to its target value. The evolutionary process begins with a population of programs that are iteratively modified at each generation until a certain stopping criteria is met. During each generation, a subset of the population is selected based on their fitness, and new solutions are generated through crossover and mutation operators.

Since the beginning of Artificial Intelligence (AI) researchers have attempted to replicate human behaviour using computational algorithms. Such is the case of Transfer Learning (TL), which aims to use the knowledge acquired from one or more source tasks and transfer it to a target task (Pan & Yang, 2010). When the objective shifts to optimising all tasks simultaneously, it is referred as Multitask Learning (MTL). MTL attempts to learn shared representations or parameters between problems in order to improve performance, training time and generalisation (Caruana, 1997; Y. Zhang & Yang, 2018). Furthermore, taking advantage of information from multiple problems also alleviates the need for large amounts of data that traditional ML models require to perform effectively (Niu et al., 2020).

This thesis explores the application of MTL in the context of GP. Although the concept of transferring information between problems in GP is not new, current research in this area is both recent and limited. Even though earlier related fields, such as population seeding (Langdon & Nordin, 2000; Perry, 1994) and layered learning

(Jackson & Gibbons, 2007) had been explored before, the first work on TL in GP was proposed by Huong Dinh et al. (2015). Despite its novelty, his work focused on simple Symbolic Regression (SR) tasks, lacking applicability to real-world scenarios. Later research also shared the same limitations (Haslam et al., 2016; Huong Dinh et al., 2015; O'Neill et al., 2017). Furthermore, other studies address situations where the target and source tasks are of the same domain, with solutions that may not extend besides this condition (Iqbal, Xue, & Zhang, 2016; Iqbal, Zhang, & Xue, 2016; Iqbal et al., 2017, 2019). An exception is Muñoz et al. (2020), who studied the effects of transferring evolved features between problems with different domains using M3GP (Muñoz et al., 2019). However, this strategy was suboptimal for minimising the negative impact of directly transferring evolve features, as it required feature reduction and experimentation with all possible combinations of features. Other studies involving heterogeneous domains proposed techniques to improve the performance of target task with limited data by imputing it with features from a source task (Al-Helali et al., 2020a, 2020b, 2024).

On the other hand, research on MTL within EC only gained popularity upon the introduction of Multifactorial Optimisation (MFO) and Multifactorial Evolutionary Algorithm (MFEA) (Gupta et al., 2016). Subsequent works focused on improving the MFEA algorithm, either by expanding its applicability to multiobjective optimisation (Gupta et al., 2017) or by attempting to reduce negative repercussions inherent to knowledge transfer (Bali et al., 2017, 2020; Tang et al., 2021; Zheng et al., 2020; Zhou et al., 2021).

Specifically within GP, the first work on MTL was published by Jaśkowski et al. (2007) on applications for visual learning. However, the majority of the studies focus around optimising Dynamic Job Shop Scheduling (DJSS) heuristics (F. Zhang, Mei, Nguyen, & Zhang, 2023; F. Zhang et al., 2021, 2022). The MFEA process was adapted to GP by Wen and Ting (2016) as an ensemble learning algorithm for single task learning, and only later for multitasking, by Zhong et al. (2020), but this approach was only tested in time series and simple SR tasks. Additionally, another study applied GP MTL to image feature learning, by transforming it into a multiobjective problem (Bi et al., 2023) and through explicit share of a GP tree between tasks (Bi et al., 2022). Both techniques improved the generalisation of the model.

Building on this foundation, this thesis extends the research between MTL and GP by introducing a new system that begins with evolving two tasks that share a representation. This shared representation, a Common Tree (CT) between problems, acts as a form of regularisation because it depends on the simultaneous improvement of the two problems. In this first phase, denominated Common Tree Genetic Programming (CTGP), the algorithm automatically selects the pairs of features between two tasks which reduce the most the error in each problem. However, the CT is not expected to be enough to capture the complexity of each individual task. To address this limitation, the proposed method progresses to a separate evolution for each task, denominated Specific Tree Genetic Programming (SpTGP). To leverage information of the common evolution,

the CT is transferred to each problem as a new feature which is used in the final specific evolution, denominated SpTGP. Transferring the CT as a feature allows for it to not be destroyed during the second evolutionary process. This method is expected to exploit both common and specific features of different datasets, improve generalisation and reduce training time, since during the common evolution the number of fitness evaluations is reduced by half.

The document is structured as follows. In Chapter 2, the Theoretical Background, all concepts relevant to this work are defined, providing the reader with the necessary knowledge to understand the subsequent chapters. Chapter 3, the Literature Review, discusses the previous studies that associate the fields of TL and MTL with GP. A detailed description of the CTGP and SpTGP algorithms, as well as their implementation is described in Chapter 4, the Methodology. The experimental study is reported in Chapter 5, depicting the case studies and parameter settings used in the experiments. In Chapter 6, the results and discussion are provided. Chapter 7 concludes this work by providing an overview of the thesis, including its results, limitations and possible directions for future research.

THEORETICAL BACKGROUND

This chapter introduces the theoretical concepts, algorithms and techniques relevant to this work. Section 2.1 provides an overview of ML, followed by the concepts of TL and MTL in Section 2.2. Section 2.3 briefly describes algorithms and optimisation. Finally, Section 2.4 focuses on EC, with a particular emphasis on GP.

2.1 Machine Learning

AI is a field that aims to simulate intelligent systems in computers. When this is achieved by fitting mathematical models to data, it is referred as ML. Although ML is typically associated with statistics and computer science, it can also be considered a multi-disciplinary field, due to its wide range of applications across various research domains, including fraud detection, face and shape recognition, medical diagnosis and product recommendation (Alzubi et al., 2018).

According to Mitchell (1997), ML is defined as learning from experience with respect to a specific task and performance measure, where the performance improves over time with accumulated experience. The primary objective of these models is to capture patterns that can be later used to make predictions on unseen data.

In a general ML scenario, a prediction model is built based on a set of features that describe the problem to be solved. These features are observed across a number of subjects and organised in a structured manner, a dataset (Mohri et al., 2012). A dataset is generally formatted in the following way:

$$\begin{array}{cccccc|c} x_{11} & x_{12} & \dots & x_{1j} & \dots & x_{1n} & y_1 \\ x_{i1} & x_{i2} & \dots & x_{ij} & \dots & x_{in} & y_i \\ x_{m1} & x_{m2} & \dots & x_{mj} & \dots & x_{mn} & y_m \end{array}$$

In this representation, x_{ij} is the j -th feature or attribute of the i -th data point. In this example, there are a total of n features and m data points. y_i denotes the target or label associated with the i -th data point, indicating the value the model is expected to predict.

The nature of this output vector depends on the problem type. In classification tasks, the output represents a category assigned to the input, whereas in regression tasks, the labels consist of real-values. Additionally, the target vector can either be known or not, depending on the availability of the labels.

2.1.1 Machine Learning Categories

ML can be divided in a wide range of categories, but its main areas of study include supervised learning, unsupervised learning and reinforcement learning.

Supervised Learning: The goal of the model is to map input data to a known target variable. The relationship between the input and output values is learned during training, resulting in a model that can generate predictions on unseen data (Murphy, 2012).

Unsupervised Learning: Given the inputs, the objective is to find hidden patterns in data, not requiring a labelled target variable (Murphy, 2012). The most popular tasks of this type are clustering and dimensionality reduction algorithms (Mohri et al., 2012).

Reinforcement Learning: In this scenario, an agent makes decisions by interacting with the environment. Through trial and error, and guided by rewards and penalties, the agent learns which actions are optimal to reach the desired solution. Robotics, gaming and recommendation systems are some examples of applications of reinforcement learning (Sutton & Barto, 2018).

2.1.2 Model Training and Monitoring

For a model to learn, it is necessary to define a measure of misfit between the model predictions and the actual observations. This measure, commonly referred to as a loss function, is the key guide during training of the model, which adjusts its trainable parameters to minimise this function (Bishop, 2006).

While ML learns from a set of data, the goal of modelling is not to replicate this set, but to make accurate predictions on new, unseen data. In other words, it is necessary for the model to generalise to data it has not encountered during training. To monitor this during training, the input data is usually divided into three sets:

Training Set: A collection of observations used to train the model.

Validation Set: A collection of observations used to optimise the hyperparameters of the model.

Test Set: A collection of observations that is independent from the training and tuning process. Its purpose is to assess the performance and generalisation of the learned algorithm in unseen data.

By carefully evaluating these sets, it is possible to ensure that the model matches the complexity of the underlying data patterns. This is important because, if the model is too simple, it will underfit the data, failing to capture important trends. On the other

hand, if the model fits patterns that are unique to just a subset of the data, it will overfit, leading to poor performance on unseen data (Mohri et al., 2012).

2.1.3 Model Evaluation

Given that loss function values might not be intuitive, to evaluate and interpret the performance of ML mode, an comprehensible metric needs to exist. This metric has many names such as performance, error, evaluation or forecasting measure.

Although this depends on several factors and can be domain-dependent, the most commonly used metrics in regression are the Mean Absolute Error (MAE), the Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), R and Coefficient of determination (R^2) (Naser & Alavi, 2023).

In the following chapters, RMSE and Relative Squared Error (RSE) are chosen to measure goodness of fit. RMSE is good for optimisation because it is differentiable in all real valued numbers, while RSE is not scale-dependent, since it is ratio of the total squared error with the total squared error of a simple predictor, the average of the target values. The mathematical expression of the RMSE is presented in Equation 2.1 and of the RSE is presented in Equation 2.2.

$$RMSE = \sqrt{\sum_{i=1}^N \frac{(y_i - \hat{y}_i)^2}{N}} \quad (2.1)$$

$$RSE = \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2} \quad (2.2)$$

In the above equations, y_i represents the target values, $\hat{y}_i$ denotes the predicted values, $\bar{y}$ is the mean of the target values and N refers the total of data points in the dataset.

For classification tasks, the output of the model is a target class. Thus, to evaluate the model, the distribution of predictions across each class is calculated. From these distributions, metrics are derived, and usually include true positives (TP), false positives (FP), false negatives (FN) and true negatives (TN), which can be represented through a confusion matrix (Lever, 2016). An example of a confusion matrix for a binary classification problem with classes 0 and 1, as well as some popular metrics are shown in Figure 2.1. Amongst all performance metrics, the accuracy is the most utilised for binary and multi-classification problems (Hossin & Sulaiman, 2015).

2.2 Transfer Learning

Typically, ML problems are solved in isolation, with each model being trained independently for a specific task. However, this is does not apply to TL, which attempts

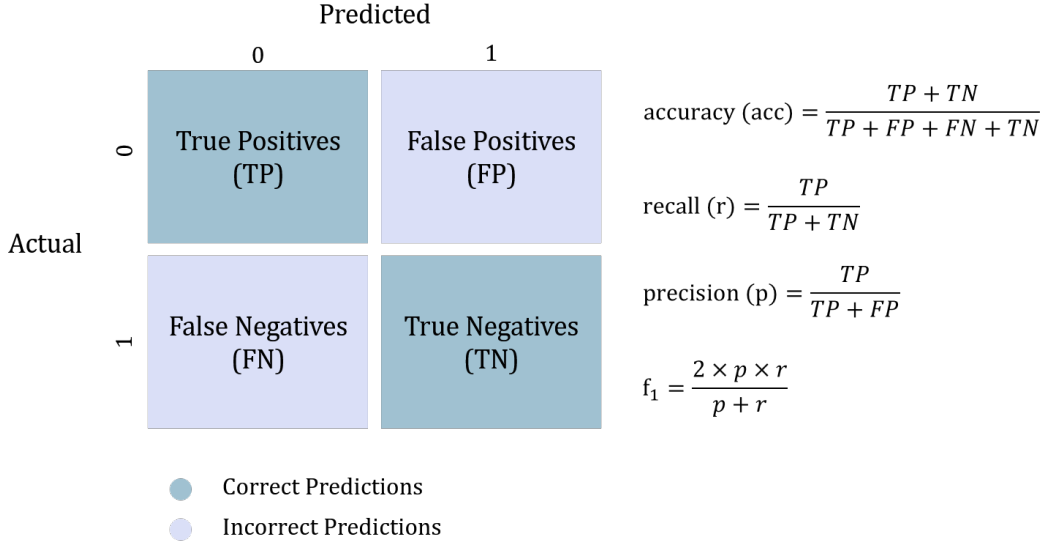


Figure 2.1: Example of a confusion matrix for a binary classification problem with classes 0 and 1. The darkest blue depicts predictions which are correct (TP and TN), and the incorrect predictions are shown with the lightest blue (FP and FN). The formulas for calculating the error metrics of accuracy, recall, precision and f1-score are also shown on the right side of the image.

to improve the performance of a target task by transferring knowledge from one or more source tasks (Torrey & Shavlik, 2010). This transfer occurs sequentially: when the source task finishes training, some of its learned parameters or representations are transferred to the target task.

The transfer might not always be beneficial, leading to poorer results compared to solving the problems independently. This phenomenon is called negative transfer (Wang et al., 2019), and it is inspired by the same concept in the field of psychology (Perkins, Salomon, et al., 1992). Negative transfer can happen due to many reasons, such as the proximity between the source and target domains, and the learners capacity of identifying the patterns that benefit from the knowledge across tasks (Zhuang et al., 2021).

2.2.1 Definitions

To formally define TL, it is necessary to first define the concepts of domain and task, as these are fundamental. The notation used is presented in Table 2.1. In addition, the superscripts ^S and ^T denote that the symbol is referring, respectively, to a source and target domain.

Definition 1 (Domain) A domain $\mathcal{D}$ is composed of a feature space $\mathcal{X}$ and a marginal distribution $P(X)$. In other words, $\mathcal{D} = \{\mathcal{X}, P(X)\}$. The symbol X denotes an instance set, which is defined as $X = \{\mathbf{x} | \mathbf{x}_i \in \mathcal{X}, i = 1, \dots, n\}$.

Definition 2 (Task) A task $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a decision function f , that is, $\mathcal{T} = \{\mathcal{Y}, f\}$. The decision function f is implicit and expected to be learned from the sample data. In the case that the ML model outputs the predicted conditional distribution of instances, $f(\mathbf{x}_j) = \{P(y_k|\mathbf{x}_j)|y_k \in \mathcal{Y}, k = 1, \dots, |\mathcal{Y}|\}$

Given the above definitions, it is possible now to define TL.

Definition 3 (Transfer Learning) Given some observation(s) corresponding to $m^S \in \mathbb{N}^+$ source domain(s) and task(s) (i.e., $\{(\mathcal{D}^{S_i}, \mathcal{T}^{S_i})|i = 1, \dots, m^S\}$), and some observation(s) corresponding to $m^T \in \mathbb{N}^+$ target domain(s) and task(s) (i.e. $\{(\mathcal{D}^{T_j}, \mathcal{T}^{T_j})|j = 1, \dots, m^T\}$), transfer learning utilizes the knowledge implied in the source domain(s) to improve the performance of the learned decision functions f^{T_j} ($j = 1, \dots, m^T$) on the target domain(s).

The definitions can be found in Zhuang et al. (2021). The definition of TL builds on that of Pan and Yang (2010) and expands it to include multi-source TL, a scenario in which $m^S > 1$.

Table 2.1: Notations

Symbol	Definition
n	Number of instances
m	Number of domains
$\mathcal{D}$	Domain
$\mathcal{T}$	Task
$\mathcal{X}$	Feature space
$\mathbf{x}$	Feature vector
$\mathcal{Y}$	Label space
y	Label
X	Instance set
Y	Label set corresponding to X
S	Source domain
T	Target domain
f	Decision function

2.2.2 Categorization of Transfer Learning

TL can be categorised from different perspectives based on the criteria used. It can be grouped by problem type, which can be further divided into label-based or space-based categories, or by solution type. A schematic representation of these categories is shown in Figure 2.2.

Pan and Yang (2010) describe inductive TL as when there is label information in the target task, and transductive TL as when the label information is only available in the source task. When both tasks do not have labels, it is considered unsupervised TL.

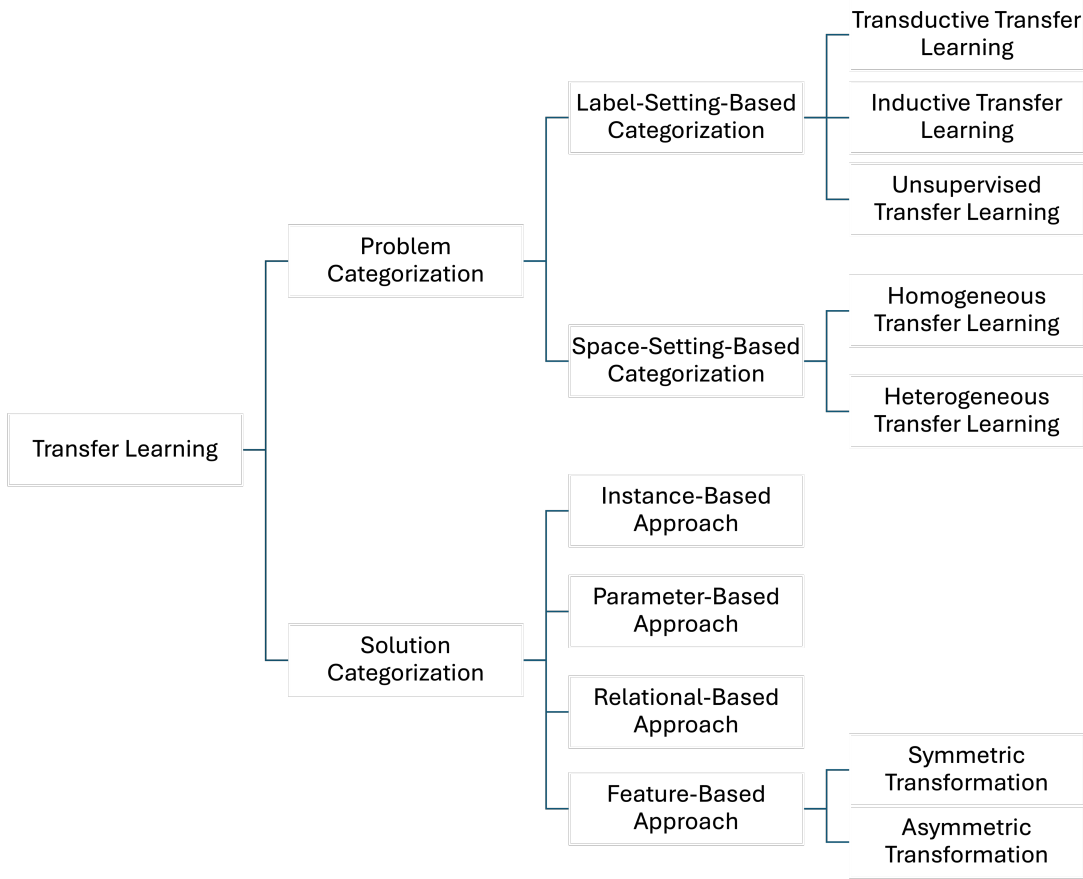


Figure 2.2: Transfer Learning Categories (Zhuang et al., 2021).

Another criterion for categorising TL depends on the relationship between the feature and label spaces of the source and target tasks. If $\mathcal{X}^S \neq \mathcal{X}^T$ or $\mathcal{Y}^S \neq \mathcal{Y}^T$, then it is heterogeneous TL. On the contrary, if $\mathcal{X}^S = \mathcal{X}^T$ and $\mathcal{Y}^S = \mathcal{Y}^T$, it is considered homogeneous TL.

Furthermore, it is also possible to group techniques according to the method used for knowledge transfer, leading to the categories of instance-based, feature-based, parameter-based, and relational-based approaches. Instance-based methods attempt to correct the marginal distribution difference between the source and target domains by changing the weights of the source task(s) accordingly. In feature-based TL, algorithms are applied to approximate as closely as possible the feature spaces of the tasks. This transformation is considered asymmetric if applied to the source domain in order to match the target one. Whereas, if it is applied to both feature spaces to create a common latent feature space, it is considered symmetric. When what is being shared between the source and target are parameters of the models, it is referred to as parameter-based TL. Relational-based approaches are used with domains where the instances are related, attempting to transfer the mechanism behind relationships in the data.

2.2.3 Multitask Learning

In TL the relationship between the source and target problems is predominantly unilateral. This means that only the target task benefits from the transfer, as it is the one that is affected by the transferred knowledge from the source problem. In contrast, in MTL, multiple tasks receive information from each other. One of the main advantages of MTL is that by sharing knowledge across tasks, it can help the model generalize better, especially in scenarios where individual tasks suffer from limited data (Caruana, 1997).

According to Definition 3 of TL, MTL can be seen as a specific case of TL, where the tasks are both source and target tasks. Additionally, MTL is related to other fields such as multi-label learning (M.-L. Zhang & Zhou, 2014) and multi-output regression (Borchani et al., 2015). In these cases, data instances are associated with more than one label, categorical or continuous. Similarly, in continual learning, models learn incrementally with different tasks. It shares similarities with MTL because both methods work with more than one task (Parisi et al., 2019).

Y. Zhang and Yang (2022) classify MTL based on the underlying ML approach, including supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning, multi-view learning, active learning and graphical models. Moreover, MTL can be categorized by how tasks share knowledge. These categories include feature-based methods, which involve symmetric transformations or selection of features, and parameter-based, which are further divided into low-rank, task clustering, task relation and task decomposition.

In low-rank methods, similar tasks are assumed to share a low-rank matrix of parameters. Task clustering relies on the assumption that tasks with similar parameters will cluster together. Task relation exploits task similarities and covariances to share parameters. Task decomposition assumes the weight matrix can be decomposed into two matrices: one that generally determines the relatedness between the problems through approaches such as feature selection and low-rank, and another that uses sparsity to make the model robust to noise and outliers.

2.3 Algorithms and Optimisation

Optimisation is the field that focuses on selecting the best solutions, known as optima, from a set possible options using algorithms. It is usually formulated as an optimization problem, a mathematical model designed to understand complex phenomena (Gill et al., 1981).

An instance of an optimization problem is defined as:

$$(S, f)$$

where S is the search space, which contains all possible solutions and f is a fitness function $f : X \rightarrow \mathbb{R}$, that quantifies the quality of the solutions. An optimisation

problem may either be a maximisation problem, in which the optima are solutions that maximise f , or a minimisation problem, where optima are solutions that minimise f . In both cases the objective is to find the global optimum of the function f .

2.4 Evolutionary Computation

Evolutionary algorithms are a subset of algorithms that fall under the scope of EC. These algorithms are inspired by the Darwinian principles of natural selection and evolution, first articulated by Darwin (1859). To solve an optimisation problem, evolutionary algorithms use analogies of biological processes such as reproduction, adaptation, inheritance, variation and competition. These natural phenomena are transformed into computational methods, allowing for an iterative process to improve solutions over time (Back & Schwefel, 1996; Vanneschi & Silva, 2023). One of the most widely known evolutionary algorithms is the Genetic Algorithm (GA), introduced by Holland (1975).

In a GA process, a group of potential solutions (a population of individuals) is altered at each iteration, referred as a generation in evolutionary terms. These solutions are represented as a fixed-length string of characters (the *genotype*).

During the evolution, individuals are evaluated and selected based on their fitness (the *phenotype*). The selection is probabilistic and, typically, the better the solution, the more likely it is to be selected. Once selected, the individuals undergo variation via genetic operators to generate new solutions. The most widely used genetic operators are crossover (combination of two individuals) and mutation (randomly altering parts of an individual).

After the first generation, the original population is replaced by the newly created one. This iterative process continues until a predefined stopping condition is met, such as a maximum number of generations or a pre-defined fitness level. The final solution is the fittest individual at this step. The general process of an evolutionary algorithm is represented in Figure 2.3.

2.4.1 Genetic Programming

Given the limitations concerning the representation of GAs, first recognised by De Jong (1988), Koza (1992) proposed GP. GP evolves a population of individuals alike GAs, guided by fitness. However, instead of solutions having a fixed length, each individual is a computer program. While selection algorithms of GAs can be applied to GP individuals, as they act on the *phenotype*, the same does not apply to genetic operators of GAs due to the difference in *genotype*.

The GP process can be represented in the following steps:

1. Initialise a population of individuals.
2. Until a predefined criteria is met, usually the number of generations, execute the following steps:

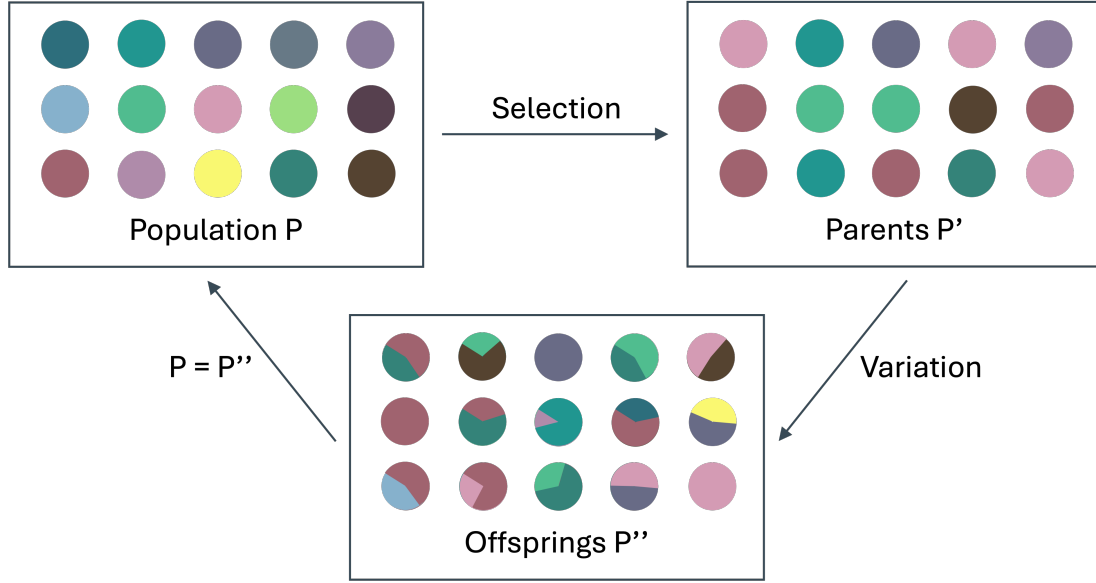


Figure 2.3: General process of an evolutionary algorithm, inspired by Vanneschi and Silva (2023)

- 2.1. Evaluate the individuals and assign them a fitness.
- 2.2. Generate a new population with the same size as the initial by applying one of the following genetic operators:
 - 2.2.1. Select a set of individuals and genetically recombine them (crossover).
 - 2.2.2. Select an individual and randomly substitute parts of it with newly generated ones (mutation).
 - 2.2.3. Select an individual and replicate it (replication).

GP, in specific SR, differs from typical regression, because it finds both the desired expression and its coefficients (Espejo et al., 2010).

2.4.1.1 Individuals

GP individuals are hierarchical computer programs, typically represented by trees constituted by a set of nodes. These nodes can be either taken from a set of primitive functions $F = \{f_1, f_2, \dots, f_n\}$ or a set of terminals $T = \{t_1, t_2, \dots, t_n\}$. The function set F takes a fixed number of arguments dependent on the domain of the problem. Examples of the former include arithmetic operators ($\{+, -, \times, \div, \dots\}$), boolean functions ($\{AND, OR, NOT\}$), mathematical functions ($\{\sin, \cos, \log, \dots\}$), and conditional or iterative operators ($\{If-Then-Else, While-Do\}$). The terminals are usually variables and constants, defined with basis on the problem.

By having a tree-based representation, GP allows for more flexibility than GAs. To limit the possible infinite search space, trees can be controlled by their depth d , which is defined as the longest possible path from the root of the tree to one of its leaves. An example of a SR GP tree is presented in Figure 2.4.

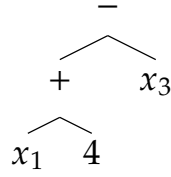


Figure 2.4: Example of a symbolic regression genetic programming tree built with the sets $T = \{x_1, x_3, 4\}$ and $F = \{+, -\}$. The tree has depth $d = 2$ and corresponds to the mathematical function $g(x_1, x_3) = (x_1 + 4) - x_3$

As mentioned before, each individual in a GP run is attributed a fitness, which describes how successful it is in solving the problem.

2.4.1.2 Initialisation

The first step in the GP evolution is the creation of the trees of the first population. The most common initialisation techniques, all proposed by Koza (1992), are the Grow, Full and Ramped Half-and-Half (RHH) methods.

Grow: When the Grow method is used, trees do not necessarily reach the maximum depth initially established and are likely to have an irregular shape (Vanneschi & Silva, 2023). The steps of this initialisation algorithm are represented below. d_{max} is the maximum depth desired and all selections are performed with uniform probability.

1. Select a random primitive from F ;
2. If n is the arity of the primitive selected, choose n random symbols from $F \cup T$ to be its subsequent nodes;
3. Repeat step 2. recursively for any branch which ends with a function node, until $d_{max} - 1$;
4. At depth d , select only from T .

Full: With the Full method, trees always reach the maximum depth defined. The steps are the same as in Grow with the exception of step 2., where symbols are only selected from F .

RHH: RHH utilises both Grow and Full in its composition. It increases diversity in population compared with using solely the previous methods, which might create trees that are too similar between each other (Koza, 1992). This is relevant because having a diverse population allows for a better exploration of the search space. The algorithm, for each depth between the minimum and maximum depths, generates half of the trees with the Grow method and the other half with the Full method.

2.4.1.3 Selection

Selection is the method with which solutions are chosen to be inserted in the offspring population. The majority of selection algorithms are probabilistic, dependent on fitness

and ensure that no individual, even if it is the worst, has zero chances of being selected. This is important because selecting always the best individuals might increase loss of diversity and the evolution may stagnate on a local optima.

The most used selection algorithms are fitness proportionate selection (or "roulette wheel"), ranking selection and tournament selection. In this work, tournament selection was chosen as the selection algorithm.

Tournament selection, opposed to the other algorithms mentioned before, does not require the evaluation of the entire population when selecting an individual. The exception to this is when elitism is used (elitism is further explained in Section 2.4.1.7). Initially, a pool of k individual (tournament) is chosen randomly with uniform distribution from the population. The fittest solution of the group is the selected individual. The first part of the method is random and independent of fitness, while the second part is completely deterministic and solely based on the fitness of the individuals in the tournament. The only parameter of this method is, k , the tournament size. The higher k is, higher the selection pressure and the probability of selecting better individuals increases. On the other hand, if k is low, the convergence rate is slower and it will take longer to find the best solution (Miller, Goldberg, et al., 1995).

2.4.1.4 Genetic Operators

During evolution, solutions are modified via genetic operators, such as crossover, mutation and replication. Crossover and mutation are further explained in sections 2.4.1.5 and 2.4.1.6, respectively. In addition to these operators, individuals can also be replicated without change. At each generation, only one operator is chosen probabilistically, due to the increased destructive nature of genetic operators in GP. This differs from GAs, where genetic operators are applied in sequence. The individuals produced by these operators are typically called offspring.

2.4.1.5 Crossover

Crossover is the operator that attempts to represent the exchange of genetic characteristics between two individuals, similar to the process of reproduction.

Standard crossover (Koza, 1992) begins by independently selecting one random point in each parent tree, T_1 and T_2 . The crossover fragment of a parent is defined as the subtree rooted at the node underneath the selected crossover point. The first offspring is produced by deleting the crossover fragment of T_1 and inserting the crossover fragment of T_2 at the crossover point of T_1 . The second offspring is produced in a symmetric manner. A graphical representation of this process is shown in Figure 2.5.

2.4.1.6 Mutation

Unlike crossover, in mutation new genetic material is created and used to modify an already existent individual. In standard mutation (Koza, 1992), a random node of

the individual is selected with uniform distribution and the subtree attached to this mutation point is substituted with a newly generated one. A graphical representation of this process is shown in Figure 2.6.

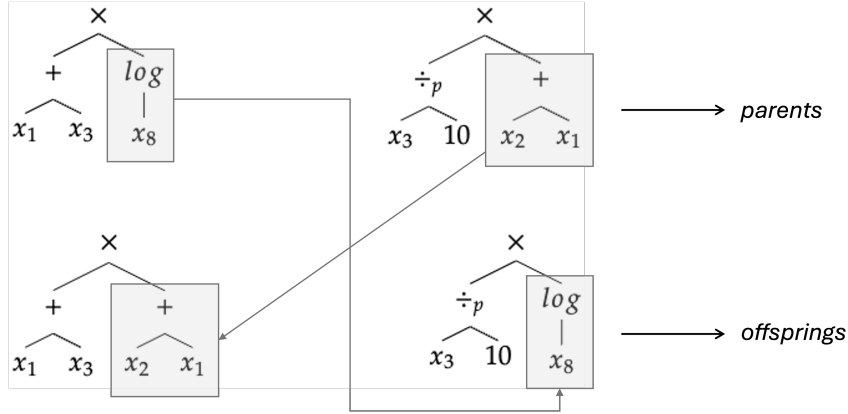


Figure 2.5: Example of a standard crossover. The grey boxes contain the crossover fragments.

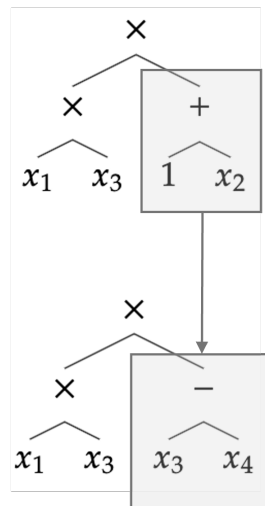


Figure 2.6: Example of a standard mutation. The grey boxes contain the mutation fragments.

2.4.1.7 Elitism

When solutions pass through crossover or mutation, the offspring are, in most cases, different from the parents. Therefore, it is not ensured that the best individual of the next generation is better than the one of the current. To avoid decreasing fitness when moving to the next generation, a portion of the best individuals, the elites, are copied to the next generation unchanged. This is called applying elitism to the evolution.

2.4.1.8 Bloat

Bloat is the phenomenon where individuals show an excessive growth in size without a corresponding improvement in fitness, which harms the efficiency of the algorithm. The most popular technique to counter bloat was introduced by (Koza, [1992](#)), which involves limiting the breeding to individuals with depth less than 17. Additionally, methods that control parsimony pressure of an individual are widely used, since they factor in tree size in the probability of being selected (Luke & Panait, [2006](#)). An example of this method is the double tournament, where k tournaments are conducted according to a specific criteria, and the k winners of these tournaments compete in another tournament with a different criteria. To minimise the effects of bloat, parsimony pressure is defined as one of the criteria of the double tournament (Luke & Panait, [2002](#); Silva & Costa, [2009](#)).

LITERATURE REVIEW

This chapter provides a comprehensive review of the relevant studies within the scope of this thesis, focusing on the topics of TL and MTL in the context of EC. Section 3.1 discusses TL, specifically highlighting works that used GP. In section 3.2, a broader overview of Evolutionary MTL is provided.

3.1 Transfer Learning in Genetic Programming

TL aims to improve the performance of a target task by using knowledge from one or more source tasks (Pan & Yang, 2010). Although TL differs from MTL in certain aspects, the primary focus of this work, both methods share similarities, per example, learning common representations or parameters.

In GP, TL is not inherently intuitive because GP generates solutions tailored for a specific problem (Y. Zhang & Rockett, 2009), which raises concerns about its transferability across tasks. Nonetheless, the idea of extracting knowledge from evolved GP programs is not entirely new. While population seeding (Langdon & Nordin, 2000; Perry, 1994) and layered learning (Jackson & Gibbons, 2007) were not originally designed for transferring knowledge between distinct problems, they laid the foundation for TL in GP.

The first work that explicitly mentions TL in GP was proposed by Huong Dinh et al. (2015). The authors explored three approaches for initialising the target problem's population with information from the source task: with a percentage of its best individuals, with randomly selected subtrees from the population of the last generation, or with elite individuals from each generation. This system outperformed StdGP in two sets of SR problems. In the same work, the authors suggested that the performance of the algorithm could increase if the number of transferred solutions were dependent on the correlation between problems. This hypothesis was tested by Haslam et al. (2016), however, tuning this parameter did not reach significantly different results. The authors concluded that the GP process was already automatically selecting the best transferred solutions and that this adaptation was not necessary. In addition, the system was tested

in a wider range of problems.

O'Neill et al. (2017) developed a method to transfer common trees from the best individuals of two related problems into the function set of a target problem, under the assumption that these trees represent useful building blocks. The method was tested on SR and boolean problems and obtained better results than the previous studies. Despite these strategies outperforming StdGP, it is important to note that they were evaluated in simple synthetic tasks and may not be directly applicable in real-world datasets.

In cases where real-world data is used, the research is focused on specific domains. For example, several studies applied GP-based TL in texture image classification (Iqbal, Xue, & Zhang, 2016; Iqbal, Zhang, & Xue, 2016; Iqbal et al., 2017, 2019). This is explained by the fact that computer vision is a recurrent domain in TL, since most images share general low-level features that can be transferred between different problems. Beyond computer vision, additional studies applied TL to hyper-heuristic optimisation (Mei et al., 2021) and document classification (Fu et al., 2017). In the latter case, the model trained on the source task was directly applied on the target task.

Furthermore, instance-based TL has been explored in GP by applying different weighting strategies to instances of the source dataset in an attempt to correct sample selection bias. In Chen et al. (2019b), the instances were weighted based on the performance of the target task after the weighting, whereas in Chen et al. (2019a), the authors used differential evolution as the weighing strategy. The latter study was improved later (Chen et al., 2022). However, similar to the earlier studies mentioned, when applied to real-world data, the datasets often originate from the same domain. The strategy used to create source and target datasets is to split the original dataset into two with respect to one feature. Then, the features values are permuted randomly with values from different normal distributions, generating datasets of the same domain with different distributions.

One of the biggest challenges of directly transferring GP models across real-worlds datasets is the Feature Alignment (FA) problem, highlighted in Muñoz et al. (2020). It refers to the requirement that the number and order of features must be independent in both problems. As an example, the feature labelled x_1 of a dataset does not necessarily correspond to the x_1 of another. In SR trees, terminals are typically variables of a dataset, so reusing the same tree in another dataset is not trivial, since unlike other typical TL scenarios, there is no spatial (images) or temporal (time series) feature space assumptions. Nevertheless, Muñoz et al. (2020) studied the effects of directly transferring the evolved features of one problem to another using M3GP (Muñoz et al., 2019). The transfer was conducted without any further evolution, regardless of each task's domain. To counteract the FA problem described before, the researchers performed feature reduction and the space was truncated to six features. Next, all combinations of feature mapping between the two tasks were tested to ensure the best performance. The findings of the study reveal that TL outperforms the baseline

methods chosen (mahalanobis distance classifier and multiple linear regression for classification and regression, respectively) in most cases, but does not reach M3GP’s performance. Moreover, based on the success of the transfer, the authors also concluded that some problems are better suited to be either source or target tasks and that the result of transferring knowledge between two problems is not necessarily symmetric.

In another study addressing both homogeneous and heterogeneous TL, Al-Helali et al. (2020b) utilised Multitree GP to transform features of a source domain to impute a smaller target dataset with a weighted K-Nearest Neighbours (KNN). By letting the GP evolution guide the transformation of the source dataset, the FA is implicitly handled without directly mapping feature spaces. This method was improved in Al-Helali et al. (2020a) by also considering a Kolmogorov-Smirnov measure to account for the distribution difference of the source and target datasets. Furthermore, it was extended for datasets with a large number of features in Al-Helali et al. (2024). In Al-Helali et al. (2021) the initial methodology was refined by weighting the transformed features and instances, as well as implementing a probabilistic crossover and mutation operator. Imputation algorithms are particularly relevant because data scarcity is one of the major motivations for using TL.

3.2 Evolutionary Multitask Learning

3.2.1 Multitask Learning in Genetic Algorithms

MTL is a knowledge-sharing mechanism that exploits similarities and differences between multiple tasks by solving them simultaneously (Caruana, 1997).

Even though MTL in EC was preliminary studied for visual learning (Jaśkowski et al., 2007, 2008, 2014), was not until the introduction of MFO (Gupta et al., 2016). Alongside this new paradigm, the researchers proposed MFEA, a unified search space algorithm where knowledge is shared implicitly between individuals through assortative mating and vertical cultural transmission. According to their findings, this method accelerated convergence speed compared with single-task methods.

The first improvement of MFEA was expanding it to solve multiobjective problems (Gupta et al., 2017). However, most research focuses on reducing the negative impact of cross-domain learning. For instance, Bali et al. (2017) integrated linear domain adaptation and Bali et al. (2020) enabled online parameter transfer into MFEA, which improved the original method. The latter was also adapted for multiobjective learning (Bali et al., 2021). Additionally, Zheng et al. (2020) developed a self-regulated method to perform knowledge-sharing based on task relatedness. Tang et al. (2021) proposed an evolutionary mechanism to align subspaces of different tasks. Their method first generated low-dimensional subspaces for each task with PCA. Then, an alignment matrix was computed by minimising the discrepancies between these subspaces. Furthermore, Zhou et al. (2021) introduced an algorithm that guides the evolution process

with an adaptive crossover operator. All these approaches surpassed MFEA.

While MFEA primarily relies on implicit knowledge-sharing, some researchers have proposed mechanisms for explicit knowledge transfer. For instance, Feng et al. (2019) presented an optimisation GA with explicit knowledge-sharing. This algorithm used independent representations for each task, and individuals of one problem were inserted in the population of the other every ten generations. The transfer occurred after the individuals passed through a denoising autoencoder, trained to find the mappings between the problems. This approach surpassed MFEA and was later extended to combinatorial problems (Feng et al., 2021).

Evolutionary MTL has also been applied to constrained multiobjective optimisation problems (Qiao et al., 2022, 2023) and feature selection (Lin et al., 2023). The latter study involved evolving a binary vectors that encoded which features to select. The study included both common and specific features for each problem.

3.2.2 Multitask Learning in Genetic Programming

The first attempt to adapt the MFEA to Genetic Programming (GP) was proposed by Wen and Ting (2016). The authors decomposed a complex task into a series of simpler sub-tasks, and then applied the evolutionary strategy of MFEA. The final model was an ensemble of GP-based decision tree classifiers. However, this approach is not considered a multitask learning system, because it does not solve different problems.

Later, MFO was adapted to SR problems. Instead of using a tree representation, which is characteristic of StdGP, Zhong et al. (2020) opted for a gene expression representation with automatically defined functions (Zhong et al., 2016), modified to encode information from multiple tasks. Nevertheless, the practical utility of this method outside of time series optimisation or synthetic SR problems was not tested.

In the field of image feature learning, Bi et al. (2022) applied MTL to image classification problems by introducing a shared a strongly typed GP tree (Montana, 1995) between two tasks. The full representation of an individual was composed by this common tree and an additional task-specific tree for each problem. The two trees evolve simultaneously, however, their fitness evaluation differed. The fitness of common tree depended on the mean accuracy of both tasks and the size of the tree to restrict bloat. The fitness of the task-specific tree depended on the accuracy of the combined representation of both trees. Their research findings indicate that the inclusion of a common tree allows for the model to have a similar or better generalisation ability. Additionally, the authors studied the transferability of the evolved common tree to different datasets besides the used for training. The analysis revealed that the common tree performed better than the task-specific tree when transferred to new tasks. This further demonstrates that the shared tree contributed to the generalisation of the model, making it more adaptable to new tasks.

Another approach to multitask image feature learning was presented by Bi et

al. (2023). It proposes a method where all tasks share the same search space and reformulates the multitask problem as a multiobjective one. This method generated solutions that generalised more and have reduced risk of overfitting.

Similarly to some GP-based TL methods, both of these models are limited to problems that share the same spatial feature space and cannot be directly used in other contexts, such as the case of tasks that suffer from the FA problem.

The field in which MTL in a GP system has been most extensively researched is for modelling heuristics for combinatorial problems, specially for Dynamic Job Shop Scheduling (Park et al., 2018; F. Zhang, Mei, Nguyen, Tan, & Zhang, 2023; F. Zhang, Mei, Nguyen, & Zhang, 2023; F. Zhang et al., 2020, 2021, 2022). Despite this progress, the conditions of these methods can't be directly applied to regression problems.

Overall, the majority of research in evolutionary MTL is focused on optimisation and simple synthetic problems or is limited to domains where some assumption is made regarding the search space, such as image, text or time series feature learning. In SR, little to no work has been done. Thus, the objective of this thesis is to develop and study a GP SR multitask method with a representation that accommodates both common and task-specific numerical features.

METHODOLOGY

This chapter outlines the methodology conducted in this research, with the objective of explaining how a MTL SR GP system was designed and studied. The aim of this approach is to evaluate the advantages and disadvantages of having a system which utilises information from two problems opposed to solving each as an independent task. The chapter is organised as follows: Section 4.1 provides a general overview of the proposed method; Section 4.2 details CTGP, including the representation, evaluation and evolution of the CT; Section 4.3 explains how the CT is integrated into each problem; and, finally, the implementation framework details are described in Section 4.4.

4.1 Overview of the Proposed Approach

The approach to knowledge sharing between two SR tasks is to divide the evolution process into two stages. Firstly, it requires evolving the tree structure that best represents the two tasks. To accommodate this method, a CT representation was developed and its fitness depends on the performance of both problems. The goal at this stage is to find the best tree that can generalise across both problems. This section of the evolution will be referred as CTGP.

In the subsequent stage, the information encoded in the CT is transferred to each problem as a new feature and another evolution process begins to refine features that are specific to each task that the CT is incapable of capturing. This section of the evolution will be referred as SpTGP.

The idea behind this methodology is that the solution benefits from the generalisation obtained from the CT and the specificity of an individual evolution process.

4.2 Common Tree

4.2.1 Representation

The approach to creating an unified representation for two distinct tasks is to express them a single SR tree, the CT. In practice, this means the CT needs to correspond

to an expression that is able to output a result regardless of the two problems it is solving. This has been presented as the FA problem, previously mentioned in Section 3. In summary, the FA problem regards the fact that it is not guaranteed that the order or number of the features in one dataset will correspond to the order or number of features of another.

The strategy used to solve this problem was to implement a terminal set that instead of having individual features, it includes pairs of features of each problem. For example, if one of the tasks has three features, its terminal set would contain, along with potentially added constants, $\{x_1, x_2, x_3\}$. Similarly, if the other task has two features, its terminal set would consist of $\{y_1, y_2\}$. The terminal set of the MTL system between these tasks, would contain six pairs of features: $\{c_1, c_2, c_3, c_4, c_5, c_6\} = \{(x_1, y_1), (x_2, y_1), (x_3, y_1), (x_1, y_2), (x_2, y_2), (x_3, y_2)\}$. In practice, this was achieved by replicating the column vectors so that the columns with the same index form a pair. A representation of this is presented in Figure 4.1. Additionally, a visual representation of a possible individual with this feature set is provided in Figure 4.2. As is it shown, the CT, constituted by the features c_k , can be represented into trees that with features that are specific to each task (x_i and y_j).

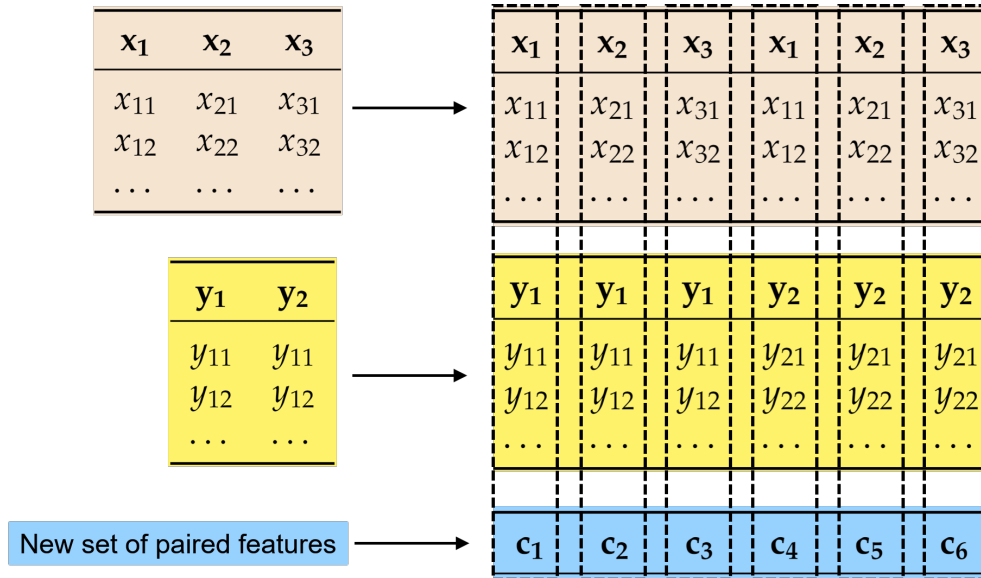


Figure 4.1: Dataset transformation for evaluating the common tree. Each pair of terminals of the original datasets, (x_i, y_j) , is mapped to a new terminal c_k . To facilitate the evaluation, the vector feature columns of the datasets are systematically replicated to form pairs of features.

Including all pairs of features effectively addresses the FA problem. GP has the advantage of automatic feature selection and can choose the relevant features, that guided by fitness, improve the performance of the model (Icke & Bongard, 2013). Thus, this design choice eliminates issues associated with the obligation of studying the problem's domains prior to evolution and pre-define feature pairs, as the decision of which pairs are relevant is solely done based on their impact on the overall fitness of

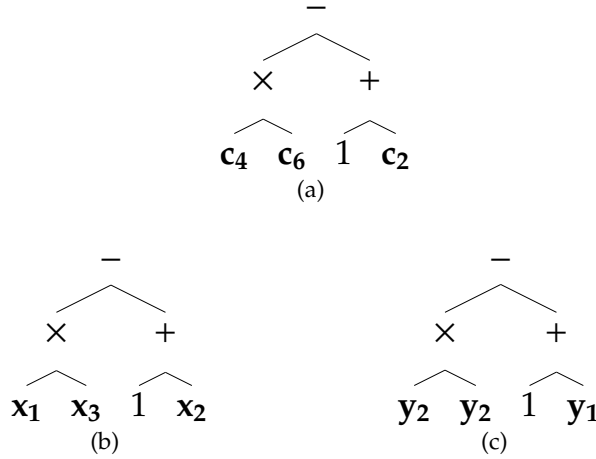


Figure 4.2: An example of a common tree representation. (a) represents the tree with the common features c_k , which correspond to pairs of features of two tasks. (b) is feature correspondence of (a) into the features x_i , and (c) into the features y_j .

the CT. Furthermore, it avoids the additional work of experimenting all combinations possible in different evolutionary processes.

Another advantage of using a unified set of features for both problems is the decrease in fitness evaluations. Rather than having separate populations, where each individual of each population would require their own fitness evaluation, this approach cuts the number of evaluations in half as only a single common individual between problems is evaluated. Fitness evaluations is a common measure for computational effort (Vanneschi & Silva, 2023), so this method also increases the efficiency of the algorithm.

Even though the number of features increases significantly with this choice, dimensionality reductions methods such as feature reduction or selection were not considered and are out of the scope of this work, but should be considered in future work for combinations of problems with high dimensionality.

4.2.2 Fitness Evaluation

After defining the representation of the CT, it is important to explain how its fitness is evaluated. Given that each terminal in a CT is either a constant or a pair of variables, the CT when evaluated in each dataset produces a different fitness value. The fitness of the CT is then calculated by averaging the fitness values of each dataset, which were modified like Figure 4.1 exemplifies. In the Results and Discussion chapter, RMSE and RSE will be compared as possible fitness functions for this part of the evolution. While the RMSE is a very common metric in single-task regression problems, it might be prone disproportionately favour one of the tasks over the other. In contrast, the RSE is expected to be more robust.

A more refined weighting strategy for the two problems is not within the scope of this work, but should be considered in future studies.

4.2.3 Evolution

The evolution of the CT is the same as in StdGP. The process starts with initialising the common population (P_0^c) by generating *pop_size* individuals. After calculating the fitness of each of them (explained in Section 4.2.2), a new empty population (P_{g+1}^c) is created to be filled with the first generation individuals. Next, elitism is employed by inserting the elite individual(s) in the new population. The new population is filled with the offspring of individuals that are selected to go through either crossover, mutation or replication until it has *pop_size* individuals. This choice is dependent on the probabilities p_c and p_m . Then, the individuals are once again evaluated. The steps from creating a new population until updating the fitness of each individual represent a single generation. After repeating the process for G generations, the best CT, *ct*, is returned. The pseudo-code of the evolution is presented in Algorithm 1.

4.3 Information Transfer to Specific Problems

After optimising the CT, the subsequent step involves transferring its information to the individual tasks. The outputted CT is as a SR tree that can be decoded into two alternative trees, each expressible as a column of predictions for each task. Integrating this prediction as a feature of each problem allows for assessing its potential contribution to enhancing each of the tasks's performance and generalisation. In addition, if the CT were to be inserted directly, without creating a new feature based on its predictions, crossover and mutation operators could alter it, potential losing part of its information in the process.

To explore this, an algorithm was developed to ensure that all individuals within the initialised population have at least once the new feature as a terminal. This ensures a consistent initial presence of the CT across the population in the first generation, facilitating the inclusion of this feature as a relevant step in the evolution.

To test the second approach, a RHH initialisation method with a modified individual creation function was developed. In addition to the standard RHH parameters, it includes *forced_terminal*. The parameter *forced_terminal* corresponds to the terminal created with the CT. If *forced_terminal* is not None, then it is necessary to force the inclusion of the CT terminal in all individuals of the population to be created. In case the condition is verified the algorithm verifies if the generated individuals contain terminal *forced_terminal*. If this condition is not met, a random terminal is selected and substituted by the CT terminal. The pseudo-code of the RHH algorithm is presented in Algorithm 2 and of the new individual creation function in Algorithm 3.

Algorithm 1: Pseudo-code of the common tree evolution.

```

1 def Evolution(pop_size, max_depth):
2    $g \leftarrow 0$ 
3   Initialise a population  $P_0^c$  of textitpop_size individuals
4   Calculate the fitness of each individual in  $P_0^c$ 
5   while  $g \leq \textit{number of generations}$  :
6     Create empty offspring population  $P_{g+1}^c$ 
7     if number of elites  $\geq 1$  :
8       Insert elites in  $P_{g+1}^c$ 
9     while  $\text{len}((P_{g+1}^c)) \leq N$  :
10       $r \leftarrow \text{random number} \in [0, 1]$ 
11      if  $r \leq p_c$  :
12        Select individuals  $Ind_1, Ind_2$  from  $P_g^c$ 
13         $O_1, O_2 \leftarrow \text{XO}(Ind_1, Ind_2)$ 
14        Insert  $O_1, O_2$  in  $P_{g+1}^c$ 
15      else if  $r \leq p_c + p_m$  :
16        Select individual  $Ind_1$  from  $P_g^c$   $O_1 \leftarrow \text{MUT}(Ind_1)$  Insert  $O_1$  in
            $P_{g+1}^c$ 
17      else :
18        Select 1 individual from  $P_g^c$  and insert it in  $P_{g+1}^c$ 
19       $P_{g+1}^c \leftarrow P_{g+1}^c[:N]$ 
20      Calculate the fitness of each individual in  $P_{g+1}^c$ 
21       $g \leftarrow g + 1$ 
22  return ct

```

4.4 Framework

The methodology steps delineated before were implemented having as a base the Distributed Evolutionary Algorithms in Python (DEAP) package (Fortin et al., 2012). DEAP is a free and downloadable python-based framework for evolutionary learning. In DEAP, it is possible to define a flexible primitive set, which will be used to create and assign trees as individuals, constituted of primitives (functions) and terminals (variables and constants).

Algorithm 2: RHH Initialisation with Custom Individual Creation Function

```
1 Function RHH(pop_size, max_depth):
2   n_trees_per_depth  $\leftarrow$  int(pop_size/(max_depth/2)) pop  $\leftarrow$  empty population
   for depth d from 1 to max_depth :
3     for _from 1 to n_trees_per_depth :
4       Create individual with the Grow method (grow  $\leftarrow$  True) and depth
        $\leftarrow d$  (Algorithm 3)
5       Append individual to pop
6       Create individual with the Full method (grow  $\leftarrow$  False) and depth  $\leftarrow$ 
       d (Algorithm 3)
7       Append individual to pop
8   while len(pop) < pop_size :
9     Create individual with grow  $\leftarrow$  True and depth  $\leftarrow$  max_depth (Algorithm
       3)
10    Append individual to pop
11  return pop
```

Algorithm 3: Pseudo-code for the individual generation.

```
1 Function generate_individual(grow, max_depth, current_depth,
   forced_terminal):
2   if grow is True :
3     Generate individual indv with maximum depth current_depth using the
       Grow method
4   else:
5     Generate individual indv with maximum depth current_depth using the
       Full method
6   count  $\leftarrow$  0 ;
7   while indv not in population and count < 50 :
8     if grow is True :
9       Generate individual indv with maximum depth current_depth using
       the Grow method
10    else:
11      Generate individual indv with maximum depth current_depth using
       the Full method
12    count  $\leftarrow$  count + 1 ;
13  if forced_terminal not None :
14    Replace random terminal of indv with the common tree terminal
15  return indv
```

EXPERIMENTAL STUDY

This chapter delineates the experimental study following the methodology detailed in the previous chapter, ensuring the reproducibility of results. It includes the regression case studies used and the parameter settings of both StdGP, CTGP and SpTGP experiments.

5.1 Case Studies

The datasets used in the experiments, as well as the number of instances and attributes of each are presented in Table 5.1. The QSAR Aquatic Toxicity (AT) and QSAR Fish Toxicity (FT) datasets were chosen because of their similarity in domain, which is something often assumed, but not necessary in MTL. The Human Oral Bioavailability (Bioavailability), Median Oral Lethal Dose (LD50) and Plasma Protein Binding Levels (PPB) were chosen due to their tendency to overfit in StdGP. In addition, the Unified Parkinson’s Disease Rating Scale (UPDRS) and Istanbul Stock Exchange (Istanbul) datasets were included in the experiments because they are widely used benchmarks in the ML community.

Table 5.1: Number of attributes (including target) and instances of the datasets used.

Dataset Name	Instances	Attributes
QSAR Aquatic Toxicity	546	9
QSAR Fish Toxicity	908	7
Unified Parkinson’s Disease Rating Scale	5875	20
Istanbul Stock Exchange	536	8
Human Oral Bioavailability	359	242
Median Oral Lethal Dose	234	627
Plasma Protein Binding Levels	131	627

The AT, FT, UPDRS and Istanbul datasets are available in the UCI Machine Learning

Repository (Dua & Graff, 2017). The next sections describe each dataset in greater detail. If the number of variables is not very high, an explanation of each feature is provided when available, as well as their data type, mean, standard deviation (Std.), minimum (Min.) and maximum values (Max.).

5.1.1 QSAR Aquatic Toxicity (AT)

This dataset is the result of a study of a development of a QSAR model from a dataset consisting of 546 organic molecules, with the objective of predicting acute aquatic toxicity toward *Daphnia magna* (Cassotti et al., 2014). The response variable, LC_{50} , represents the concentration that causes death of 50% of the *Daphnia magna* population over a duration of 48 hours. The attributes of the dataset and its statistics are presented below and in Table 5.2.

Table 5.2: General statistics of the Aquatic Toxicity dataset attributes rounded to two decimal places.

	Type	Mean	Std.	Min.	Max.
TPSA	float64	48.47	46.76	0.00	347.32
SAcc	float64	58.87	68.17	0.00	571.95
H-050	int64	0.94	1.62	0.00	18.00
MLOGP	float64	2.31	1.74	-6.45	9.15
RDCHI	float64	2.49	0.81	1.00	6.44
GATS1p	float64	1.05	0.40	0.28	2.50
nN	int64	1.00	1.40	0.00	11.00
C-040	int64	0.35	0.81	0.00	11.00
LC50	float64	4.66	1.67	0.12	10.05

The description of each feature is the following:

- **MLOGP**: Octanol-water partition coefficient calculated from the Moriguchi model.
- **RDCHI**: Topological index that encodes information about molecular size and branching, but does not account for heteroatoms.
- **SAacc**: Van der Waals surface area (VSA) of atoms that are acceptors of hydrogen bonds.
- **TPSA(tot)**: Topological polar surface area calculated by means of a contribution method that takes into account N, O, P and S.
- **H-050**: Number of hydrogen atoms bonded to heteroatoms.

- **nN**: Number of nitrogen atoms present in the molecule.
- **C-040**: Number of carbon atoms of the type $R-C(=X)-X$ / $R-C\#X$ / $X=C=X$, where X represents any electronegative atom (O, N, S, P, Se, halogens).
- **GATS1p**: Information on molecular polarizability.
- **LC50**: Concentration that causes death in 50% of test *D. magna* ($-\log_{10}(\text{mol/L})$)

5.1.2 QSAR Fish Toxicity (FT)

Similarly to the previous dataset, this dataset is the result of a study which objective was to predict acute toxicity (M. Cassotti & Consonni, 2015). However, in this study, the number of instances was different and the response variable is the LC_{50} of 96 hours of the population of fathead minnow (*Pimephales promelas*). Consequently, the attributes of the dataset are not the same as in the AT dataset, with exception of MLOGP. The variables of the dataset are shown below and in Table 5.3.

Table 5.3: General statistics of the Fish Toxicity dataset attributes rounded to two decimal places.

	Type	Mean.	Std.	Min.	Max.
CIC0	float64	2.90	0.76	0.67	5.93
SM1Dz	float64	0.63	0.43	0.00	2.17
GATS1i	float64	1.29	0.39	0.40	2.92
NdssCH	int64	0.23	0.61	0.00	4.00
NdssC	int64	0.49	0.86	0.00	6.00
MLOGP	float64	2.11	1.43	-2.88	6.52
LC50	float64	4.06	1.46	0.05	9.61

The description of each variable is the following:

- **CIC0**: Set of indices of neighbourhood symmetry, derived from a partitioning of the vertices of the hydrogen-filled molecular graph into equivalency classes.
- **SM1_Dz (Z)**: Set of descriptors calculated from 2D matrices derived from the molecular graph.
- **GATS1i**: 2D Geary autocorrelation descriptor.
- **NdsCH**: Atom-type E-state counts ($=CH-$).
- **NdssC**: Atom-type E-state counts ($=CH<$).
- **LC50**: Concentration that causes death in 50% of test *P. promelas* ($-\log_{10}(\text{mol/L})$).

5.1.3 Unified Parkinson's Disease Rating Scale (UPDRS)

This dataset is composed of several biomedical voice measurements taken from 42 people with early-stage Parkinson's disease. During a six months long trial, the patients were monitored with a telemonitoring device that remotely tracked their symptom progression (Tsanas et al., 2010). The description of the attributes and their statistics is shown below and in Table 5.4.

Table 5.4: General statistics of the UPDRS dataset attributes.

	Type	Mean	Std.	Min.	Max.
Age	int64	64.80	8.82	36.00	85.00
Sex	int64	0.32	0.47	0	1
Jitter (%)	float64	0.01	0.01	0.00	0.10
Jitter(Abs)	float64	0.00	0.00	0.00	0.00
Jitter (RAP)	float64	0.00	0.00	0.00	0.06
Jitter (PPQ5)	float64	0.00	0.00	0.00	0.07
Jitter (DDP)	float64	0.01	0.01	0.00	0.17
Shimmer	float64	0.03	0.03	0.00	0.27
Shimmer (dB)	float64	0.31	0.23	0.03	2.11
Shimmer: APQ3	float64	0.02	0.01	0.00	0.16
Shimmer: APQ5	float64	0.02	0.02	0.00	0.17
Shimmer: APQ11	float64	0.03	0.02	0.00	0.28
Shimmer: DDA	float64	0.05	0.04	0.00	0.49
NHR	float64	0.03	0.06	0.00	0.75
HNR	float64	21.68	4.29	1.66	37.88
RPDE	float64	0.54	0.10	0.15	0.97
DFA	float64	0.65	0.07	0.51	0.87
PPE	float64	0.22	0.09	0.02	0.73
Total UPDRS	float64	29.02	10.70	7.00	54.99

The description of each feature is the following:

- **Age:** Subject age.
- **Sex:** Subject gender '0' - male, '1' - female.
- **Jitter (%)**, **Jitter (Abs)**, **Jitter: RAP**, **Jitter: PPQ5**, **Jitter: DDP:** Measures of variation in fundamental frequency.
- **Shimmer**, **Shimmer(dB)**, **Shimmer:APQ3**, **Shimmer:APQ5**, **Shimmer:APQ11**, **Shimmer:DDA:** Measures of variation in amplitude.

- **NHR, HNR:** Measures of ratio of noise to tonal components in the voice.
- **RPDE:** Nonlinear dynamical complexity measure.
- **DFA:** Signal fractal scaling exponent.
- **PPE:** Nonlinear measure of fundamental frequency variation.
- **Total_UPDRS:** Clinician’s total UPDRS score, linearly interpolated.

5.1.4 Istanbul Stock Exchange

This dataset was created because the authors wished to study the influence of international markets on the Istanbul Stock Exchange and predict how the National 100 index would vary (Akbulgic et al., 2014). The description of each variable are as follows and their statistics are presented in Table 5.5.

Table 5.5: General statistics of the Istanbul dataset attributes rounded to the third decimal place.

	Type	Mean	Std.	Min.	Max.
SP	float64	0.001	0.014	-0.054	0.068
DAX	float64	0.001	0.015	-0.052	0.059
FTSE	float64	0.001	0.013	-0.055	0.050
NIKKEI	float64	0.000	0.015	-0.050	0.061
BOVESPA	float64	0.001	0.016	-0.054	0.064
EU	float64	0.000	0.013	-0.049	0.067
EM	float64	0.001	0.011	-0.039	0.048
ISE (TL-Based)	float64	0.002	0.016	-0.062	0.069

The description of each variable is the following:

- **SP:** Stock market return index of Standard and Poor’s 500.
- **DAX:** Stock market return index of Germany.
- **FTSE:** Stock market return index of UK.
- **NIKKEI:** Stock market return index of Japan.
- **BOVESPA:** Stock market return index of Brazil.
- **EU:** MSCI European index.
- **EM:** SCI emerging markets index.
- **ISE (TL-Based):** Istanbul stock exchange national 100 index.

5.1.5 Human Oral Bioavailability

Human oral bioavailability ($F\%$) is a parameter that represents the fraction of an orally administered drug that successfully reaches systemic circulation. Accurate prediction of ($F\%$) is pivotal in pharmacokinetics, as errors in this parameter have negatively affected compound development (Kennedy, 1997; Kola & Landis, 2004). It includes 241 molecular descriptors for 359 drug compounds, with the target variable being the oral bioavailability. This dataset has been used in previous GP studies (Archetti et al., 2006, 2007a; Silva & Vanneschi, 2012). Statistics and descriptions are not detailed given the extensive number of features.

5.1.6 Human Oral Lethal Dose (LD50)

The LD50 is another dataset in the field of pharmacokinetics that has been used in GP studies (Archetti et al., 2007a, 2007b). LD50 is a critical parameter for assessing the toxicity of a substance, representing the single-dose amount required to cause the death of 50% of the test organisms. In this set of data, there is information on 234 drugs, with the target variable being their LD50, tested on rats. The description of features and their statistics are not presented for the same reason as the previous dataset.

5.1.7 Plasma Protein Binding Levels (PPB)

This dataset contains molecular descriptors of 131 different drugs, with the target variable being the Plasma Protein Binding level. It refers to the percentage of the drug initial dose that reaches blood circulation and binds the proteins of plasma. This metric is important to predict because only the unbound fraction of a drug can pass through cell membranes and interact with its intended targets. GP has been used to predict this percentage in previous studies (Archetti et al., 2007a, 2007b). Similarly to the previous datasets, statistics are not shown due to the high number of features.

5.2 Experiments

The objective of this work is to analyse the performance of inserting a previously evolved common feature between two different problems in a GP environment. With this aim, several experiments were conducted with regression tasks. To achieve this, StdGP was performed for each problem, while CTGP and SpTGP were run for all possible pairs of problems, with the exception of the LD50, PPB and Bioavailability datasets, which were not run with each other given their high dimensionality.

Due to the randomness of the initialisation and genetic operators in the GP algorithm, a single run is not deterministic of the general performance. Therefore, to obtain significant results, 30 independent runs of the algorithm were performed for each experiment, with randomised train, validation and test splits of 70%, 15% and 15%,

respectively. A validation set was used due to this study having experiments that differ parameter-wise.

5.2.1 Parameter Settings

The GP parameters used in the experiments are outlined in Table 5.6. These were determined based on previous literature.

Table 5.6: Parameter settings. The parameters enclosed in { } represent those that differed between experiments.

Parameters	Value
Population size	{50,100, 200}
Number of generations	200 (Total)
Maximum initial depth	7
Overall Maximum depth	17
Function Set	$+, -, \times, \div_p$
Terminal Set	Features (StdGP), Pairs of features (CTGP)
Selection method	Tournament of size 2
Number of elites	1
Crossover method	Standard tree crossover
Crossover rate	0.8
Mutation method	Standard tree mutation
Mutation rate	0.2
Fitness Function	{RMSE, RSE} (CTGP), RMSE (StdGP, SpTGP)

The population size was defined as either 50, 100 or 200 individuals. Ramped Half-and-Half was chosen for initialisation due to its capability of introducing diversity in the population (Koza, 1992). For SpTGP, the initialisation algorithm is the modified RHH with the parameter *forced_terminal* set to the terminal correspondent of the the CT (Algorithm 2). The number of generations was set to 200 for StdGP and 100 in CTGP and an additional 100 in SpTGP.

The maximum initial depth was set to 7 and the overall maximum depth to 17, as a straightforward approach to minimise bloat (Koza, 1992). The function set includes the operators of addition (+), subtraction (-), multiplication ($\times$) and protected division ($\div_p$) to avoid errors of division by zero, returning 1 in such case. The terminal set consisted solely of the input variables for StdGP, and of the pairs of features in CTGP. or of the original features with the additional created common feature. The selection operator chosen was tournament selection of size 2 and for genetic operators, standard mutation with probability 0.2 and standard crossover with probability 0.8 were used.

The number of elites is set to 1, meaning that the best individual at each generation is always passed unchanged to the next generation.

The experiments conducted are presented on Table 5.7. The SpTGP evolution was only performed after choosing between using RMSE or RSE in the fitness function of the CTGP evolution. It was then tested with every combination of population size of the chosen CTGP fitness function.

Table 5.7: List of experiments conducted.

Experiment Name	Details
StdGP50	StdGP with population size 50
StdGP100	StdGP with population size 100
StdP200	StdGP with population size 200
CTGP50_RMSE	CTGP with population size 50 with RMSE
CTGP100_RMSE	CTGP with population size 100 with RMSE
CTGP200_RMSE	CTGP with population size 200 with RMSE
CTGP50_RSE	CTGP with population size 50 with RSE
CTGP100_RSE	CTGP with population size 100 with RSE
CTGPT200_RSE	CTGP with population size 200 with RSE
SpTGP50	SpTGP with population size 50
SpTGP100	SpTGP with population size 100
SpTGP200	SpTGP with population size 200

RESULTS AND DISCUSSION

This section details the results obtained on the experiments and their discussion. The findings regarding the common part of the evolution are presented in Section 6.1, which covers the choice of fitness function, as well as the overall performance, overfitting, and program size of the obtained CTs. The results of the specific evolution are presented on Section 6.2.

6.1 Common Evolution

6.1.1 Fitness Function

During CTGP, two different metrics were tested for the fitness function of the CT: the RMSE and the RSE.

As expected, when using RMSE, the problems which start with lower RMSE values tend to not improve significantly during the evolutionary process. This occurs because reducing their already low error has a minimal impact on the average RMSE when paired with tasks that have higher initial error. As a result, these tasks often underfit, some even increasing their initial error if that corresponds to a decrease in the average error. Table 6.1 illustrates this phenomenon, showing how RMSE increases from the first generation ($G=0$) to the last ($G=100$) during the CTGP50_RMSE experiment in some dataset pairs. Although the table only shows this experiment, this was verified across all population sizes.

In contrast, using the RSE in the fitness function of the CT ensures that all tasks improve during evolution. This metric does not overly prioritize tasks with high initial errors, leading to a more balanced improvement across tasks. This makes the RSE a more robust choice for the fitness function of the common evolution. For example, as shown in Figure 6.1, the error of the Istanbul dataset when paired with the LD50 dataset increases when using RMSE (green curve), as the optimisation favours the minimisation of the error of the LD50 dataset. On the contrary, the decrease in error is more balanced when using the RSE (orange curve), with both tasks showing improvement. This is explained by the fact that the RSE normalises the error based on a naive model, the

mean of the target values, as opposed to the RMSE which is more sensitive to the scale of the problems.

Table 6.1: Median Train and Validation errors (Train : Validation) at the initial (G=0) and final (G=100) generations for datasets where the error increased when using RMSE in the common fitness function. Results are shown for **Task1** of the pair **Task1**/Task2 of the experiment CTGP50_RMSE.

Dataset	RMSE (G=0)	RMSE (G=100)
AT/LD50	2.06 : 2.05	6.65 : 5.22
FT/LD50	1.48 : 1.41	7.64 : 7.54
Istanbul /LD50	0.0162 : 0.0167	0.236 : 0.185
AT/PPB	2.91 : 2.71	6.21 : 6.30
FT/PPB	1.81 : 1.81	3.97 : 4.11
Istanbul /PPB	0.0171 : 0.0180	0.109 : 0.104
Istanbul /Bioavailability	0.0169 : 0.0180	0.117 : 0.107

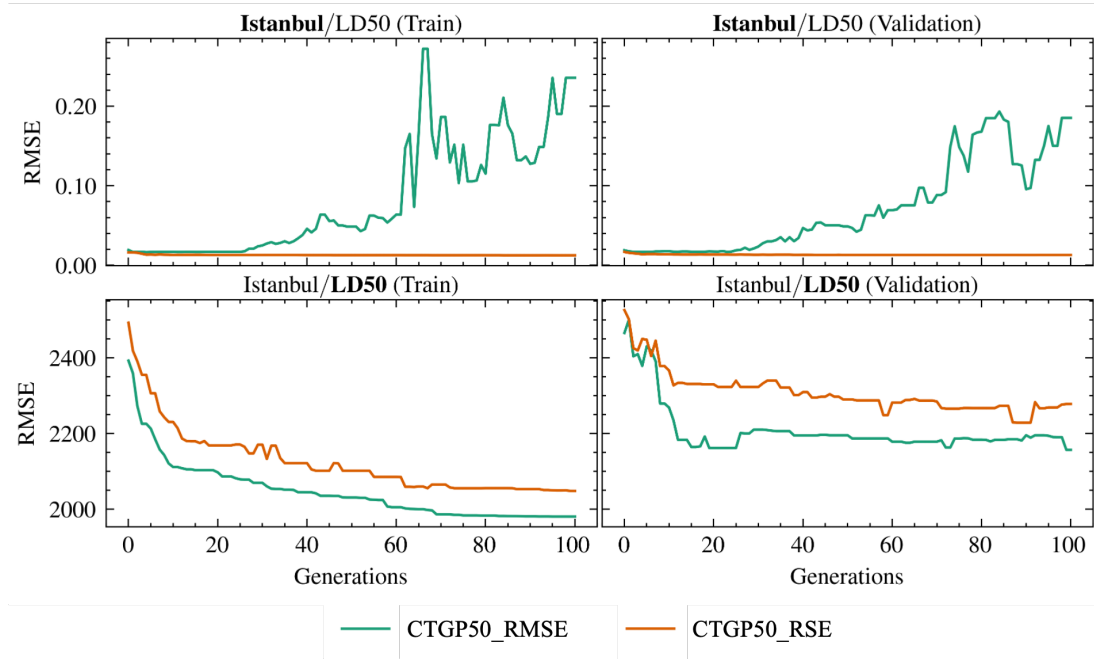


Figure 6.1: Median best RMSE of CTGP50_RMSE (green) and CTGP50_RSE (orange) of the pair composed by the Istanbul and LD50 datasets. The RMSE shown in each plot is of the dataset in bold.

For the reasons mentioned, the average RSE was chosen as the fitness function for the common evolution. However, the recorded fitness values throughout this study are of the RMSE to allow comparison with other experiments.

6.1.2 Performance

Table 6.2 summarises the best train and validation errors of the experiments of CTGP for each dataset, separated by population size. The criteria for choosing the best experiment was the error of the last generation in the validation set. Additionally, the bold values indicate the best fitness amongst the three population sizes. The table also shows, beneath the error values, the dataset pair used to obtain the result.

Table 6.2: Median Train and Validation errors (Train : Validation) of the best CTGP experiments. For each dataset on the left, the corresponding error values and dataset pair are shown. The bold values indicate the best-performing experiment among the three population sizes.

Dataset	CTGP50_RSE	CTGP100_RSE	CTGP200_RSE
AT	1.72 : 1.58	1.56 : 1.59	1.46 : 1.49
	FT	FT	UPDRS
Bioavailability	40.45 : 41.47	40.50 : 42.62	38.52 : 41.79
	UPDRS	UPDRS	UPDRS
FT	1.15 : 1.16	1.17 : 1.16	1.15 : 1.11
	LD50	UPDRS	UPDRS
Istanbul	0.0125 : 0.0128	0.0126 : 0.0128	0.0125 : 0.0129
	LD50	LD50	FT
LD50	2134 : 2230	2027 : 2236	1955 : 2217
	UPDRS	Istanbul	Istanbul
PPB	39.61 : 43.63	35.49 : 38.91	33.20 : 38.58
	UPDRS	UPDRS	UPDRS
UPDRS	11.23 : 11.22	10.93 : 11.00	11.00 : 11.05
	Istanbul	Istanbul	Istanbul

When looking at the best pair for each dataset, it is possible to observe that the relationship is not always symmetrical: a dataset being the best pair for one task does not mean the reverse is true. This asymmetry is the most common case, with the exception of the pair Istanbul/LD50.

Furthermore, in MTL and other knowledge-sharing paradigms, it is typically assumed that sharing information will naturally benefit more related problems than unrelated ones. This assumption was tested using the AT and FT datasets, given that both are QSAR toxicity models for predicting LC_{50} . However, despite their similarity in domain, AT is never the best pair for FT, nor is FT the best dataset pair for AT out of all of the experiments. This finding demonstrates that task relatedness is not a reliable

indicator to determine the most suitable pairings.

Another key observation from Table 6.2, is that only a subset of the datasets, namely the FT, LD50, UPDRS and Istanbul datasets, were the best pairs. Notably, there is a predominance of the UPDRS dataset as an optimal pair compared with the others. This suggests that certain datasets might possess characteristics that make them more suitable pairings than others.

These conclusions suggest that determining if CTGP will be successful is not straightforward. Similar findings have also been reported in a previous study on TL (Muñoz et al., 2020).

To provide a comparison, the StdGP errors of the best individual at generation 100 are shown in Table 6.3. It is important to mention that CTGP is expected to perform worse than StdGP, because it is being optimised for two problems instead of a single one. This is verified for all datasets with the exception of Istanbul, where CTGP outperforms StdGP in validation.

Table 6.3: Median Train and Validation errors (Train : Validation) of each StdGP experiment at generation 100.

Dataset	StdGP50	StdGP100	StdGP200
AT	1.41 : 1.51	1.38 : 1.47	1.34 : 1.40
Bioavailability	36.97 : 39.24	35.83 : 39.75	33.71 : 36.95
FT	1.07 : 1.07	1.04 : 1.06	1.05 : 1.09
Istanbul	0.0121 : 0.0139	0.0120 : 0.0138	0.0120 : 0.0143
LD50	1947 : 2239	1931 : 2166	1883 : 2188
PPB	32.98 : 35.82	31.57 : 35.98	29.46 : 35.68
UPDRS	11.09 : 11.05	10.82 : 10.87	10.80 : 10.89

However, it is necessary to know if the differences are statistically significant or not. Hence, a statistical test was performed to compare the best CTGP to the best StdGP, in both train and validation. The statistical test used to test was a Wilcoxon signed-rank test, with a significance level of 0.05. This is a robust non-parametric hypothesis test that compares two related samples. It was selected for its reliability, even when the data does not follow a normal distribution. From now on, whenever significance is mentioned it was verified through this test.

On the training set, all differences were statistically significant throughout all generations with StdGP having the best results. Yet, on the validation set, in the LD50 dataset during the last 50 generations the difference was not significant, and in Istanbul dataset, the difference was not significant in the last 88 generations. In these cases, failing to reject the null hypothesis does not provide sufficient evidence that the error

in validation of these two datasets is different between the two models.

Another observation regards the population size of the best experiments. In CTGP, five of the seven best results are from experiments with the highest value of population size, while this is only verified for three cases in StdGP. One possible explanation for this is that CTGP benefits from a higher population size because it allows for a better exploration of the search space of pairs of features, which is naturally larger than the search space of the features of a single dataset.

6.1.3 Overfitting Reduction

In order to verify the effectiveness of CTGP in preventing overfitting, the difference between the validation and train error was evaluated, as this is a common indicator of overfitting. Specifically, this difference was statistically compared with a Wilcoxon signed-rank test from generations 60 to 100 of CTGP and StdGP for all experiments. This interval was chosen because the initial generations are often not representative of the overall generalisation ability of the model. The results of this study are summarised in Table 6.4, which shows, for each dataset, the cases where overfitting was significantly reduced. The comparison was made across every population size.

The only datasets that exhibit a consistent behaviour independently of their pair are the UPDRS and Istanbul datasets. Overfitting is never significantly reduced in the UPDRS dataset, because it already generalises well in StdGP, so no further improvement in this aspect is required. In contrast, in the Istanbul dataset, overfitting reduction is significant in all combinations. It is also the only dataset where CTGP outperforms StdGP in validation. These results are further illustrated in Figure 6.2, which depicts the median error of the Istanbul and UPDRS datasets across generations when using CTGP and StdGP with different population sizes. The reduction of overfitting in the case of the Istanbul is evident, as the validation error of CTGP (thicker orange curve) is lower and closer to the train error (thinner orange curve) compared to the blue curves that represent StdGP. On the other hand, the absence of overfitting in the UPDRS plots is also noticeable, in both the CTGP and StdGP evolutions, justifying the reasoning behind the lack of significance between the two methods.

The next key observation concerns the LD50 dataset. As the population size increases, the number of cases where CTGP significantly reduces overfitting also increases. Specifically, when the population size is 50, no overfitting reduction is detected. At 100 individuals, two cases (with the AT and FT datasets) show reduced overfitting. Finally, with the largest population size, all the datasets tested with the LD50 dataset show overfitting reduction. This pattern suggests that adjusting population size in StdGP has a higher impact on overfitting, while CTGP is more robust when varying population sizes for this dataset.

For the remaining datasets, the results are inconsistent, so it is not possible to draw any definitive conclusions. However, the results are as follows:

- Bioavailability: The most overfitting reduction occurred when CTGP, with a population size of 50, was compared with larger StdGP populations. Overfitting was also reduced when paired with the AT dataset, upon comparing CTGP with a population of 100 individuals against StdGP with a population of the same size or larger. When paired with the UPDRS dataset, overfitting was also reduced when CTGP had a population size of 200 and StdGP of 100 individuals.
- AT: Overfitting was reduced when paired with the Istanbul dataset, with CTGP having a population size of 50 and StdGP of 200.
- FT: Overfitting was reduced when StdGP had a population size of 200 and CTGP of 100 and 200. The results were obtained with the dataset pairs PPB and AT, respectively.
- PPB: Overfitting was reduced when StdGP had a population size of 200 and CTGP of 50 and 200. The results were obtained with the dataset pairs FT and AT, respectively.

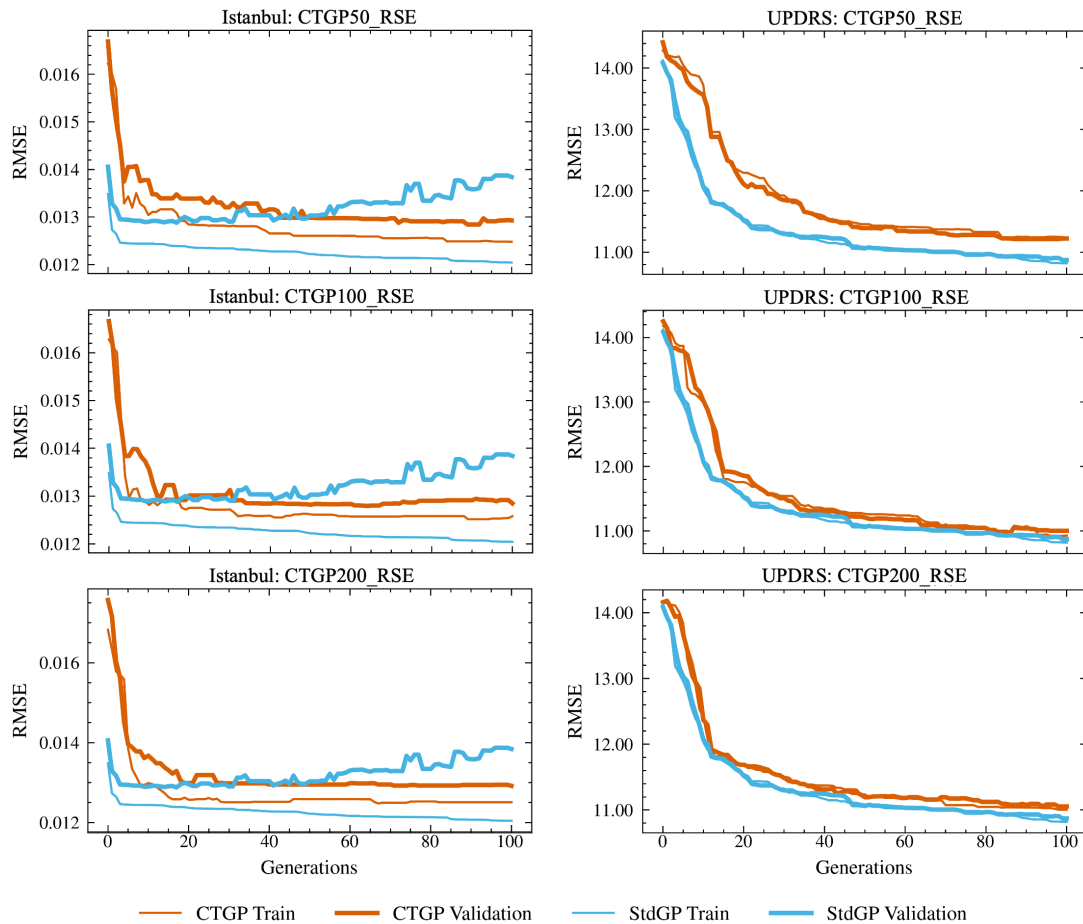


Figure 6.2: Median RMSE of Train (thicker curve) and Validation (thinner curve) across generations, comparing the best combinations of CTGP (orange) and StdGP (blue) for the Istanbul and UPDRS datasets across all population sizes.

Table 6.4: Comparison of overfitting reduction between CTGP and StdGP across datasets and population sizes. The left column indicates in which dataset the overfitting was reduced. The terms used in the table are defined in the legend below.

Dataset	CTGP50_RSE vs. StdGP50	CTGP50_RSE vs. StdGP100	CTGP50_RSE vs. StdGP200
AT	None	None	Istanbul
Bioavailability	None	AT, Istanbul, UPDRS	Istanbul, UPDRS
FT	None	None	None
Istanbul	All	All	All
LD50	None	AT, FT, UPDRS	AT, FT, Istanbul, UPDRS
PPB	None	None	FT
UPDRS	None	None	None

Dataset	CTGP100_RSE vs. StdGP50	CTGP100_RSE vs. StdGP100	CTGP100_RSE vs. StdGP200
AT	None	None	None
Bioavailability	None	AT	AT
FT	None	None	PPB
Istanbul	All	All	All
LD50	AT	AT, FT	AT, FT, UPDRS
PPB	None	None	None
UPDRS	None	None	None

Dataset	CTGP200_RSE vs. StdGP50	CTGP200_RSE vs. StdGP100	CTGP200_RSE vs. StdGP200
AT	None	None	None
Bioavailability	None	UPDRS	None
FT	None	None	AT
Istanbul	All	All	All
LD50	None	AT, FT	AT, FT, Istanbul, UPDRS
PPB	None	None	AT
UPDRS	None	None	None

Legend:

- **None** : No significant difference in overfitting between CTGP and StdGP.
- **All** : CTGP significantly reduced overfitting in all datasets.
- **X, Y, Z, ...**: CTGP significantly reduced overfitting in these datasets.

6.1.4 Program Size

Another important characteristic often studied in GP is program size — the number of nodes of a tree. Hence, an analysis of this aspect was also conducted. Figures 6.3, 6.4 and 6.5 display boxplots of the size of the best individual at generation 100 from each run across the different experiments conducted. In each plot, the non-shaded region corresponds to the CTGP experiments, while the shaded area to the StdGP, each with different population sizes. In the CTGP plots, the dataset pair is also shown. For ease of representation, Bioavailability is abbreviated as "Bio". All further observations will be based on the median value (orange line of each boxplot) and dispersion of values (size of the whiskers).

When focusing on CTGP across all figures, it is visible that some dataset combinations consistently produce smaller trees than others. Specifically, when one of the datasets in the pair is AT, FT or PPB, the produced trees tend to be smaller.

Additionally, when comparing CTGP and StdGP, the effect on tree size of increasing the population size (for example, from 50 to 100) is smaller in CTGP. This effect is more evident in pairs that include the LD50, Bioavailability and PPB datasets, which are also the datasets that have the highest number of features (Figure 6.4).

While model size is generally an important characteristic, as larger trees reduce computational efficiency, the most significant issue associated with it is bloat, described in Section 2.4.1.8. To investigate this, the best individual on validation at generation 100 of each run was plotted against its size. This is shown in Figures 6.6 and 6.7. The red triangles represent StdGP and the black dots represent CTGP. When the visualisation was affected by outliers, a logarithmic function was applied to the values to normalise them. In general, it is possible to observe that for the same values of error, StdGP has larger program sizes, indicating that StdGP bloats more than CTGP. This trend is observable across all population sizes, with the exception of the datasets Istanbul, UPDRS and PPB, where this is only evident when the population size is equal to 100 or 200 individuals. A possible explanation for this is that in the common evolution, for trees to bloat, an increase in size must reduce or maintain the average error across both problems, which is more restrictive than optimising a single task.

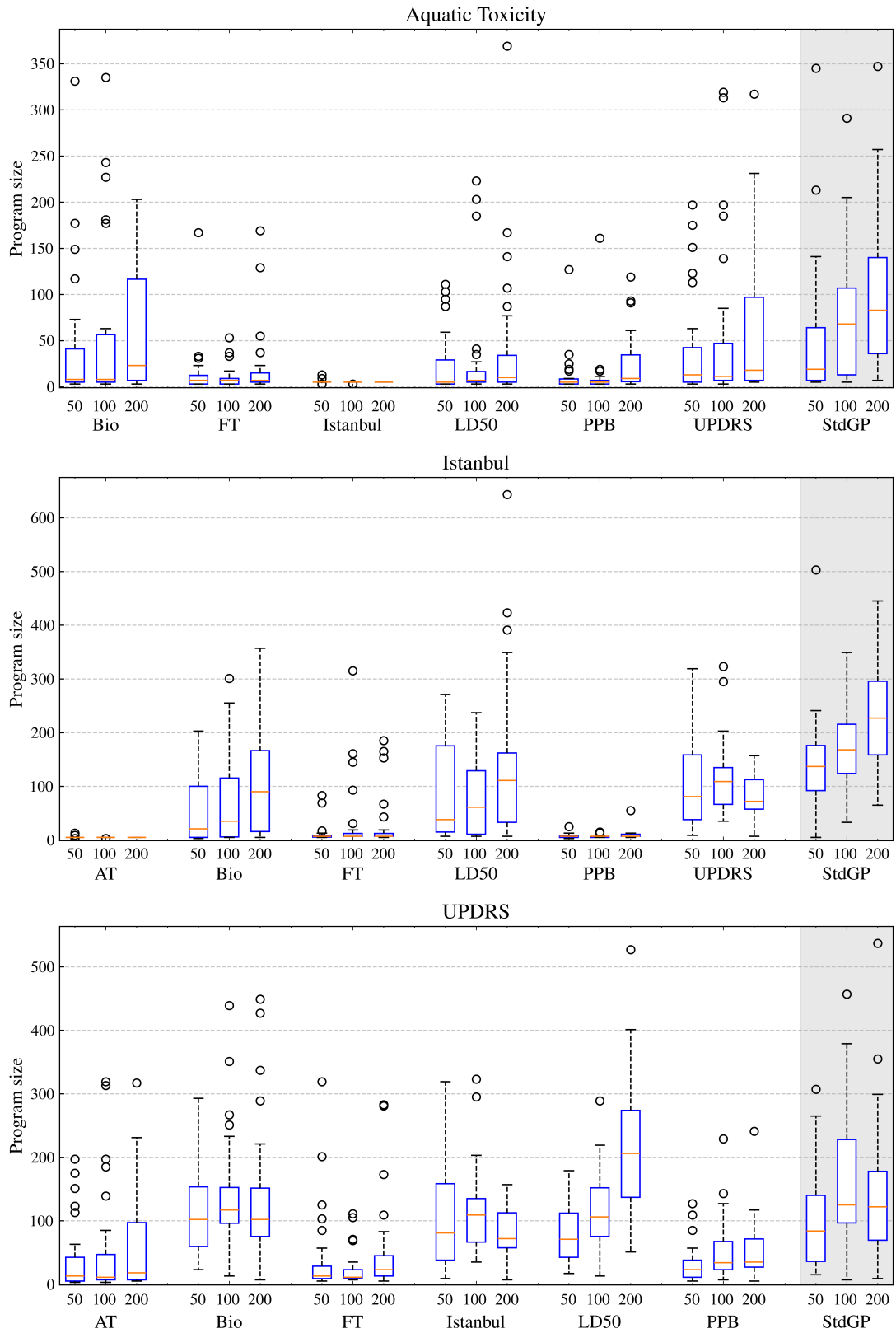


Figure 6.3: Boxplot of the best individual size at generation 100 over 30 runs of the AT, Istanbul and UPDRS datasets. The non-shaded part refers to CTGP and the shaded part to StdGP. The labels indicate the paired dataset if applicable, and population size.

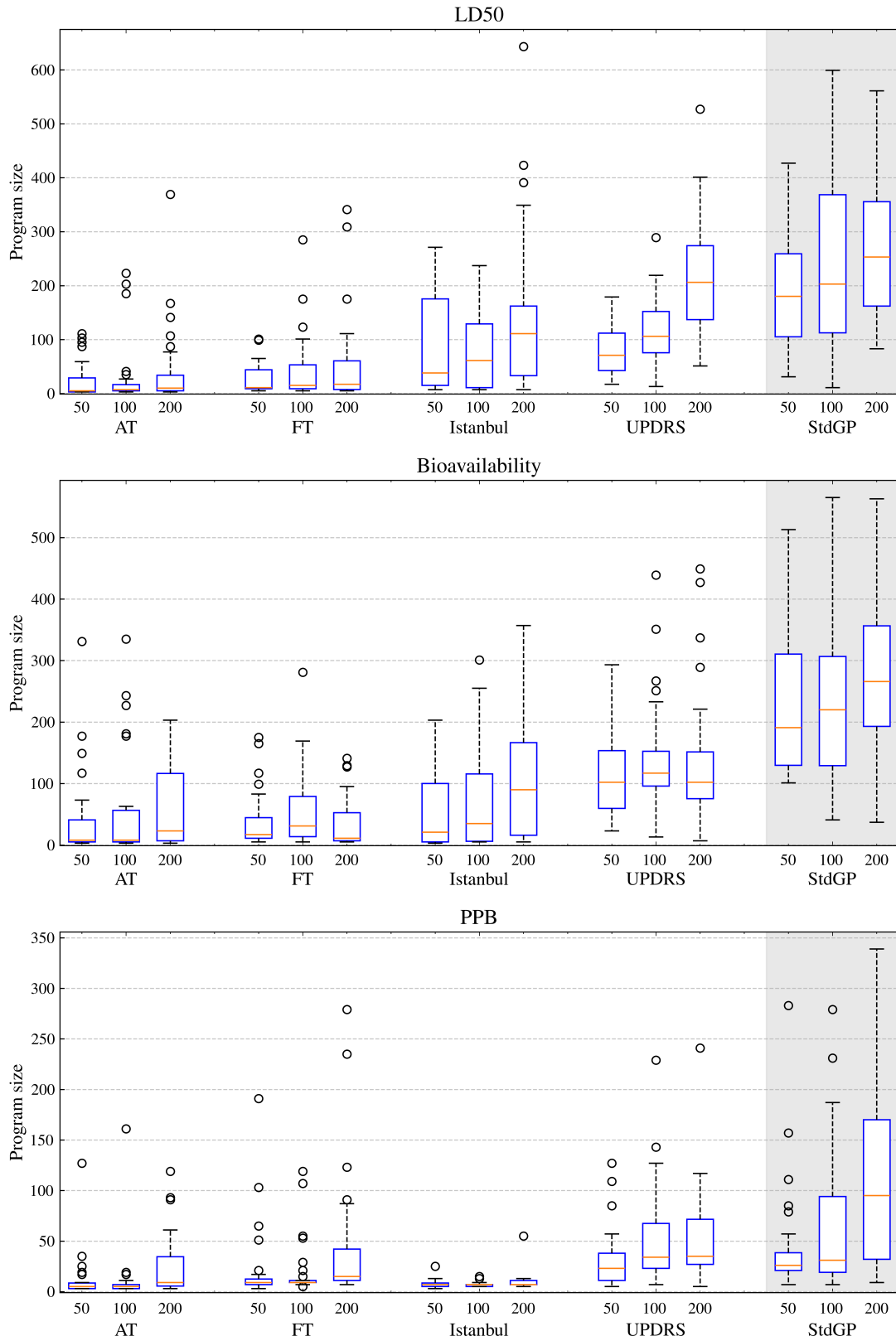


Figure 6.4: Boxplot of the best individual size at generation 100 over 30 runs of the LD50, Bioavailability and PPB datasets. The non-shaded part refers to CTGP and the shaded part to StdGP. The labels indicate the paired dataset if applicable, and population size.

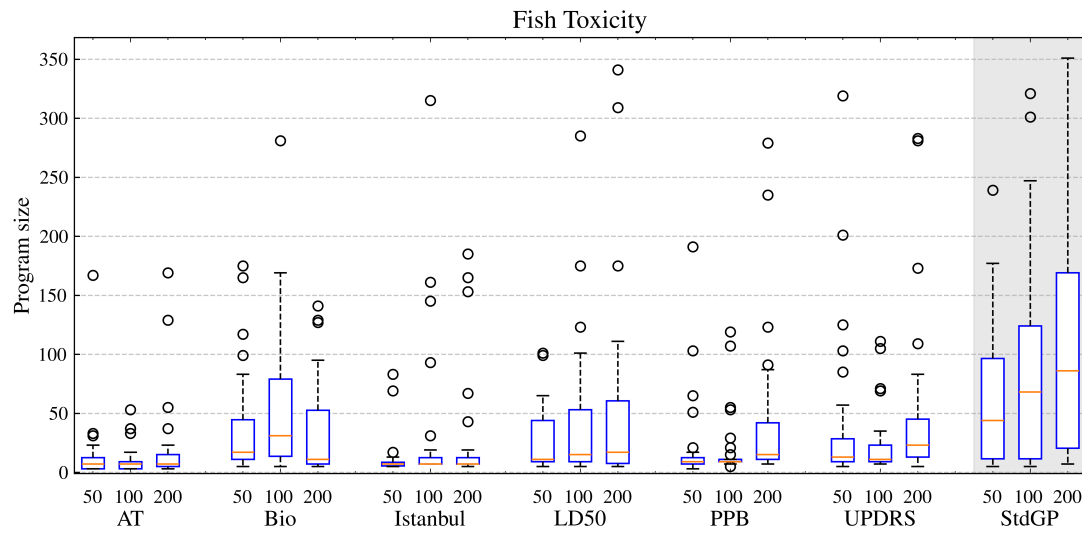


Figure 6.5: Boxplot of the best individual size at generation 100 over 30 runs of the FT dataset. The non-shaded part refers to CTGP and the shaded part to StdGP. The labels indicate the paired dataset if applicable, and population size.

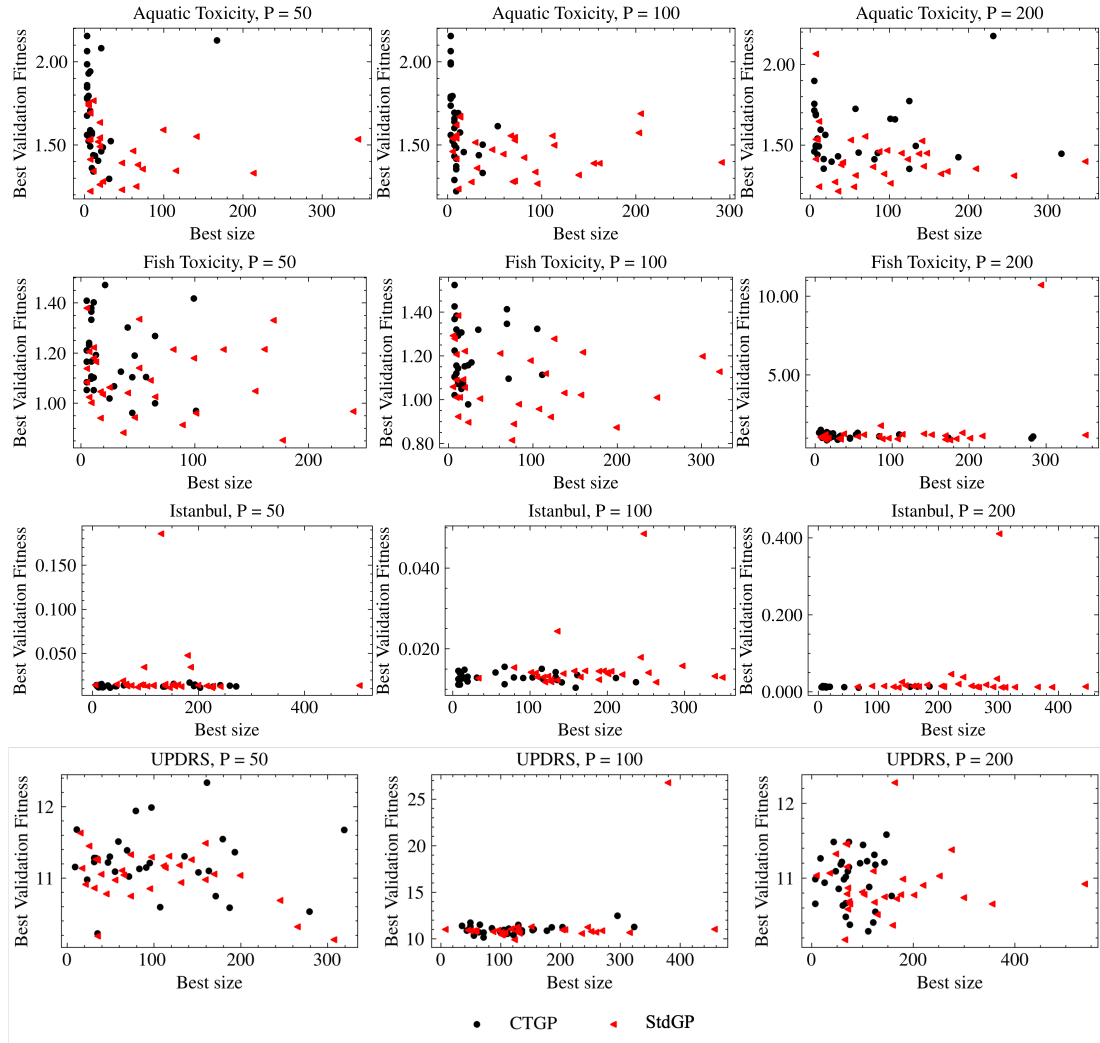


Figure 6.6: Comparison of validation fitness against program size of the best individuals between CTGP (black dots) and StdGP (red triangles) for the AT, FT, I and UPDRS datasets. Columns are of the same population (P) size and rows of the same dataset.

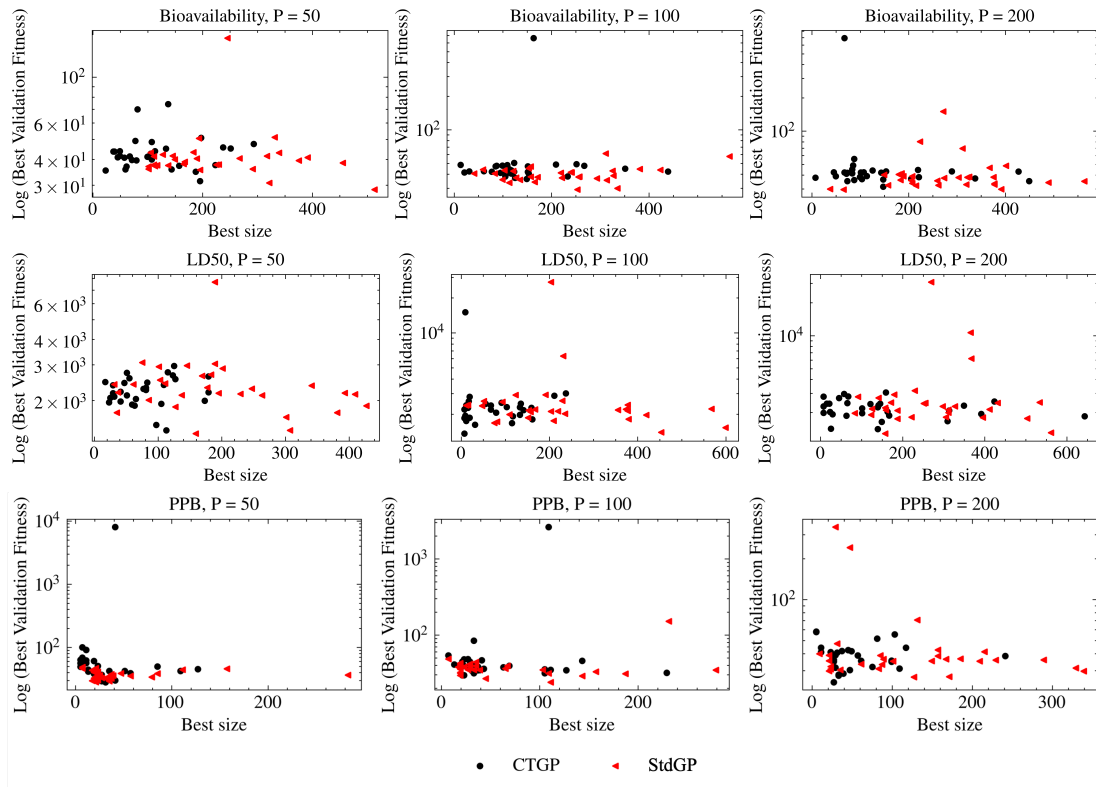


Figure 6.7: Comparison of validation fitness against program size of the best individuals between CTGP (black dots) and StdGP (red triangles) for the Bioavailability, LD50 and PPB datasets. Columns are of the same population (P) size and rows of the same dataset.

6.2 Specific Evolution

Tables 6.5 and 6.6 show the train and validation results of SpTGP. The specific evolution was performed using the best common tree combinations for each population size, shown previously in Table 6.2, for all datasets. The tables also display the results of StdGP after 200 generations. In addition, bold values indicate the lowest validation obtained for each dataset.

Comparing the SpTGP results with those in Table 6.2, it is possible to verify that most of the best-performing experiments during this stage of the evolution correspond to the same pairs observed in CTGP. It is expected for the pairs to remain the same because the CTs obtained during CTGP were the best predictors during CTGP, so naturally, they are the features that are expected to contribute the most in the second stage of the evolution. The only exception is the AT dataset, in which the best pair changed from the UPDRS dataset to the FT dataset.

To verify the model performance on truly unseen data, the test errors were calculated, with the criteria of choice being the best error on validation. These results are presented in Table 6.7. The Istanbul dataset was the only one where the specific evolution did not yield better results on validation. Hence, the best common evolution was kept as the final result. It is observed that the overall best performing model is not evident, as the results are not consistent.

In addition, the table includes the percentage of runs in which the best individual contains the added common feature. Since GP is a highly flexible system that can automatically perform feature selection, this is important to verify, because there's no guarantee that when a new feature is introduced in the system, like the case of the CT in the specific evolution, that this feature is not discarded. Nevertheless, it is verified that in almost all cases, the best individuals contain the CT feature. The Istanbul dataset's percentage is not shown because its solution is the CT itself.

To provide a more comprehensive view of the full evolution, the entire evolutionary process, including CTGP (orange), SpTGP (yellow) and StdGP (blue), was plotted in Figure 6.8. The shaded area corresponds to the p-values calculated from the statistical comparison of the test values between the two methods. The plot also includes a dashed line indicating the significance level of 0.05 for reference. It is evident that even if in some situations the SpTGP outperforms StdGP and vice-versa, that statistically, both methods do not differ, with the exception of Istanbul. This indicates that, while the CT was always used as a feature, it is not sufficient to regularise the specific evolution. This is the most evident in the Istanbul dataset. In Figure 6.9, the error across generations is plotted for the entire evolution of the proposed method. As it is possible to see by the thicker curve, which represents the test values, overfitting is highly increased a few generations after SpTGP starts. This suggests that transferring a common tree as a feature is not the best approach, and that the common fitness function was the key factor for preventing overfitting.

Table 6.5: Median train and validation error (Train : Validation) of the best SpTGP experiments for the AT, Bioavailability, FT and Istanbul datasets. For each dataset, the corresponding dataset pair is also shown above the error values. The bold values indicate the best performing experiment. The last row shows the performance of StdGP at generation 200.

	AT	Bioavailability	FT	Istanbul
CTGP50_RSE	FT	UPDRS	LD50	LD50
SpTGP50	1.41 : 1.50	34.91 : 39.58	1.07 : 1.13	0.0122 : 0.0129
SpTGP100	1.40 : 1.52	34.33 : 37.25	1.04 : 1.09	0.0119 : 0.0140
SpTGP200	1.39 : 1.43	34.17 : 39.15	1.03 : 1.07	0.0120 : 0.0138
CTGP100_RSE	FT	UPDRS	UPDRS	LD50
SpTGP50	1.40 : 1.46	35.66 : 39.46	1.05 : 1.08	0.0121 : 0.0129
SpTGP100	1.38 : 1.48	34.99 : 39.65	1.04 : 1.14	0.0120 : 0.0140
SpTGP200	1.36 : 1.46	34.24 : 38.45	1.02 : 1.10	0.0118 : 0.0136
CTGP200_RSE	UPDRS	UPDRS	UPDRS	FT
SpTGP50	1.37 : 1.52	34.06 : 38.10	1.08 : 1.11	0.0120 : 0.0134
SpTGP100	1.39 : 1.46	34.08 : 38.16	1.06 : 1.07	0.0120 : 0.0132
SpTGP200	1.36 : 1.48	33.30 : 37.09	1.02 : 1.04	0.0119 : 0.0137
StdGP				
StdGP50	1.38 : 1.40	33.34 : 37.23	1.04 : 1.08	0.0119 : 0.0143
StdGP100	1.34 : 1.46	32.80 : 38.64	1.02 : 1.05	0.0118 : 0.0140
StdGP200	1.29 : 1.38	30.41 : 34.36	1.01 : 1.09	0.0116 : 0.0142

Table 6.6: Median train and validation error (Train : Validation) of the best SpTGP experiments for the LD50, PPB and UPDRS datasets. For each dataset, the corresponding dataset pair is also shown above the error values. The bold values indicate the best performing experiment. The last row shows the performance of StdGP at generation 200.

	LD50	PPB	UPDRS
CTGP50_RSE	UPDRS	UPDRS	Istanbul
SpTGP50	1938 : 2305	34.03 : 40.95	10.78 : 10.74
SpTGP100	1915 : 2385	33.04 : 39.26	10.68 : 10.75
SpTGP200	1890 : 2244	31.46 : 37.57	10.69 : 10.70
CTGP100_RSE	Istanbul	UPDRS	Istanbul
SpTGP50	1917 : 2229	32.28 : 36.37	10.70 : 10.73
SpTGP100	1851 : 2248	31.74 : 40.15	10.54 : 10.62
SpTGP200	1809 : 2351	31.05 : 37.17	10.49 : 10.51
CTGP200_RSE	Istanbul	UPDRS	Istanbul
SpTGP50	1869 : 2238	30.66 : 39.58	10.64 : 10.64
SpTGP100	1807 : 2203	30.33 : 38.13	10.64 : 10.69
SpTGP200	1789 : 2340	29.51 : 40.81	10.55 : 10.64
StdGP			
StdGP50	1852 : 2394	30.25 : 37.02	10.82 : 10.77
StdGP100	1801 : 2160	28.78 : 36.87	10.52 : 10.57
StdGP200	1731 : 2271	26.04 : 38.34	10.63 : 10.64

Table 6.7: Train and test errors (Train : Test) of the proposed method and StdGP. CT(%) indicates the percentage of best performing individuals that contained the common tree as a terminal. The CT percentage for the Istanbul dataset is omitted, as the final result is the CT.

	StdGP	CTGP/SpTGP	CT (%)
AT	1.29 : 1.35	1.39 : 1.47	100%
Bioavailability	30.42 : 34.99	33.30 : 36.07	100%
FT	1.02 : 1.05	1.02 : 1.09	100%
Istanbul	0.0118 : 0.0141	0.0126 : 0.0128	-
LD50	1801 : 2304	1807 : 2213	100%
PPB	28.78 : 38.37	31.05 : 38.68	100%
UPDRS	10.52 : 10.65	10.49 : 10.55	87%

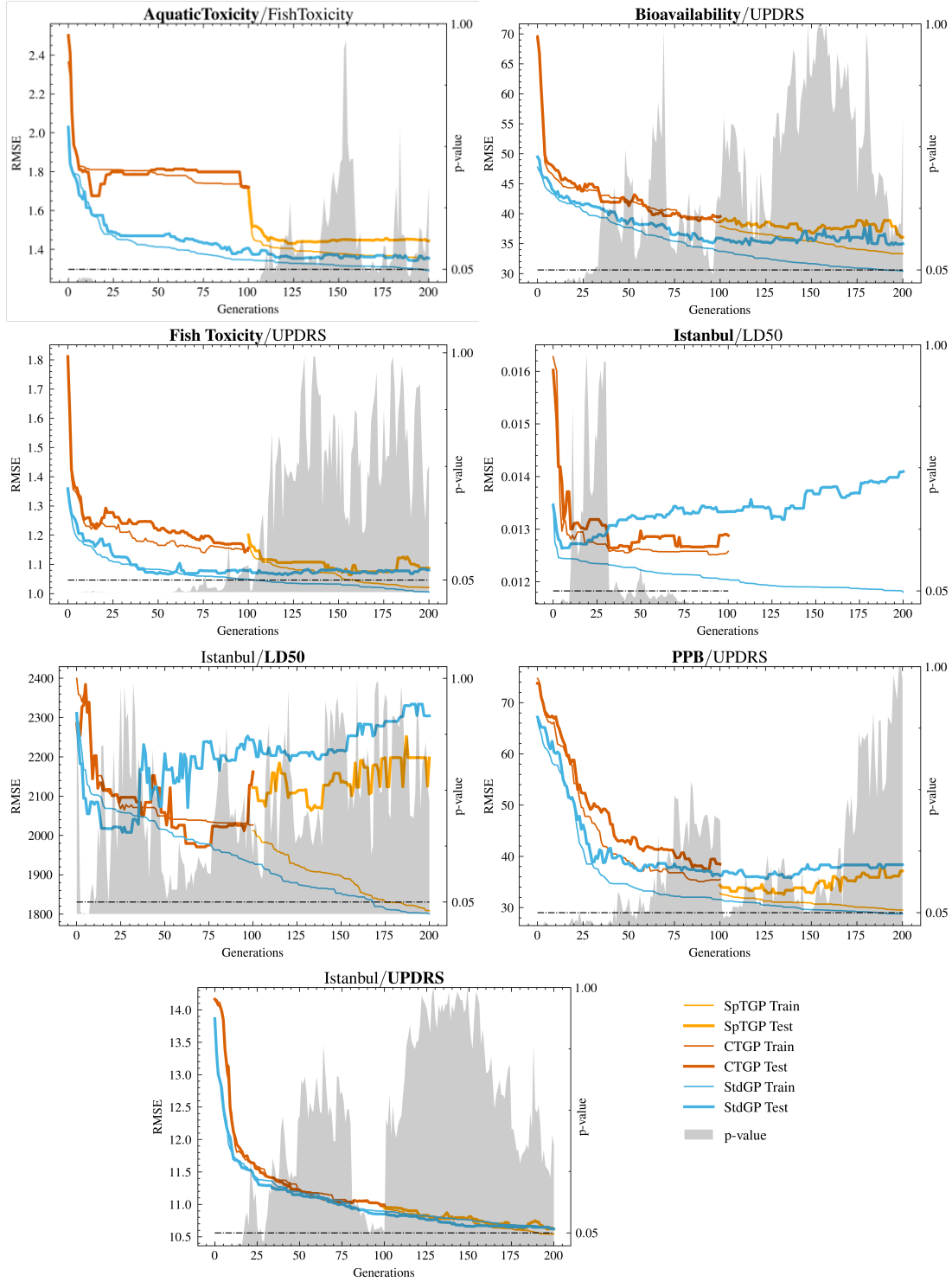


Figure 6.8: Train (thinner line) and Test (thicker line) RMSE of CTGP (orange) and SpTGP (yellow) against StdGP (blue) across generations. The shaded area refers to the p-value (right axis) of the comparison of the test scores between methods. The dashed line references the significance level of 0.05.

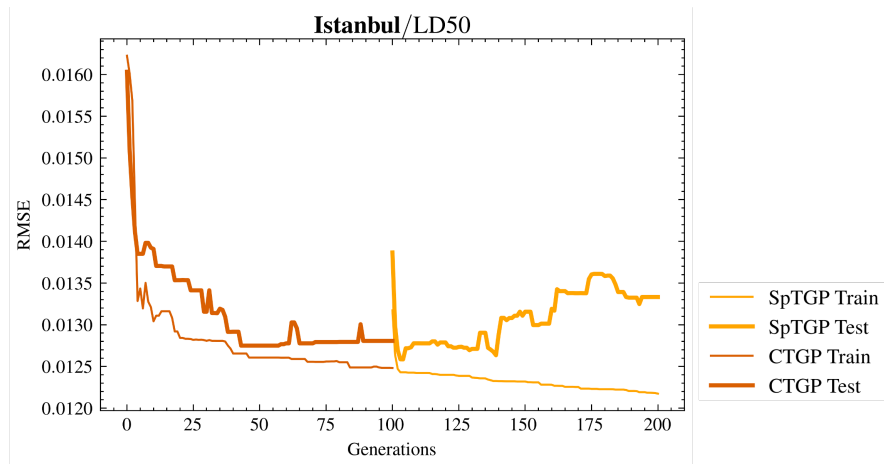


Figure 6.9: Train (thinner line) and Test (thicker line) RMSE of CTGP (orange) and SpTGP (yellow) of the Istanbul dataset.

CONCLUSIONS AND FUTURE WORK

This chapter presents an overview of the proposed study, highlighting the main conclusions related with joining MTL and GP. Moreover, it includes suggestions for future work.

In this thesis, a novel method that combines the area of MTL and GP is introduced. It evolves a tree that has a fixed shape for two different problems and whose terminals are pairs of features of both problems (CTGP). This evolution is guided by the average fitness of the two problems. After this step is complete, the output is inserted as a new feature in each dataset and a new evolution that is specific to each problem is conducted (SpTGP). All the individuals of the initial population of this second step are forced to have at least once the new feature as a terminal. The expected outcome was that the CT between problems would serve as a regularisation method, improving generalisation, but also increasing the efficiency of the algorithm.

The experimental work included studying the behaviour of pairing seven different benchmark real-world datasets, with the exclusion of some pairs due to their high dimensionality. The proposed method was compared with StdGP in terms of performance, generalisation ability on unseen data, and program size. This included several experiments with different population sizes on both methods.

Regarding the relationships between datasets, the analysis revealed that, contrary to what is usually assumed in MTL, the behaviour of CTGP did not depend on the relatedness of the domain of the tasks. This was proven with two related datasets, AT and FT, as AT was never the best pair for FT. It was also observed that the relationship between datasets is not symmetric, as the best pair of one dataset does not mean the reverse applies. In addition, the results suggest that some datasets are better pairs than others, with the UPDRS dataset being predominantly the best pair. All of these observations suggest that determining that two tasks are similar is more complex than just assessing if their domains are related. These conclusions align with previous studies on TL (Muñoz et al., 2020).

In terms of overfitting, CTGP significantly reduced overfitting in some cases, being Istanbul the dataset in which this is more evident. In this dataset, CTGP performed exceptionally well, even when sharing its fitness function with another task. The reason why this occurred for this case is not clear. It was also observed that in other combinations, if there is no overfitting whatsoever in StdGP, the difference between overfitting of the methods is not significant, as expected. Another observation was that CTGP is more robust to the overfitting when increasing population size in the LD50 dataset. Besides this, it was observed that the CT also produces, for the same values of fitness, trees with smaller sizes. This was verified across all datasets, specially when comparing higher population sizes, and is an indicator that using a CT regulates bloat. In addition, using a CT leads to overall smaller trees with the AT, FT and PPB datasets and that the tree size of CTs is more robust to increasing the population size in datasets with a high number of features.

In terms of training time, although it was expected for CTGP to have the shortest total training time, this was not verified in the majority of cases because for this to happen, the optimal pairs need to be symmetric. This reinforces the need to study heuristics on what defines a good pair.

When passing to the specific evolution, it was observed that even if the CT has less overfitting, it is not sufficient to regularize the specific evolution, leading to similar results as StdGP in most cases. This is evidence that the common fitness function was the most important factor for the reduction of overfitting. However, one conclusion that can be made is that the common feature is useful, since it is present in the best individual in almost all runs throughout the different datasets.

Like it was mentioned before, future research should focus on exploring which characteristics of the tasks, namely their datasets, influence the performance of CTGP. Namely, infer heuristics that determine what a dataset should have to be an optimal pair, or discover, for example, what makes the Istanbul dataset such a good problem for this method. Other potential lines of research are related with the common fitness function, such introducing as pre-defined weighting strategy dependent on the scale of the error of both problems, or other multiobjective methods that can be adapted to this system.

Besides this, given that the common tree is influenced by the two problems, it must contain information of both. So, another possible approach is to study the shape of the common tree and which pairs it chooses and see if it contains any relevant insights about the resolution and relatedness of the problems.

Furthermore, it would be relevant to investigate new ways to integrate CTs with task-specific trees besides inserting the CT as feature. Finally, future experiments in the context of classification tasks could also extend the applicability of the method.

BIBLIOGRAPHY

- Akbilgic, O., Bozdogan, H., & Balaban, M. E. (2014). A novel hybrid rbf neural networks model as a forecaster. *Statistics and Computing*, 24(3), 365–375. <https://doi.org/10.1007/s11222-013-9375-7> (cit. on p. 35).
- Al-Helali, B., Chen, Q., Xue, B., & Zhang, M. (2020a). Multi-tree genetic programming for feature construction-based domain adaptation in symbolic regression with incomplete data. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 913–921. <https://doi.org/10.1145/3377930.3390160> (cit. on pp. 2, 21).
- Al-Helali, B., Chen, Q., Xue, B., & Zhang, M. (2020b). Multi-tree genetic programming-based transformation for transfer learning in symbolic regression with highly incomplete data. *2020 IEEE Congress on Evolutionary Computation (CEC)*, 1–8. <https://doi.org/10.1109/CEC48606.2020.9185670> (cit. on pp. 2, 21).
- Al-Helali, B., Chen, Q., Xue, B., & Zhang, M. (2021). Multitree genetic programming with new operators for transfer learning in symbolic regression with incomplete data. *IEEE Transactions on Evolutionary Computation*, 25(6), 1049–1063. <https://doi.org/10.1109/TEVC.2021.3079843> (cit. on p. 21).
- Al-Helali, B., Chen, Q., Xue, B., & Zhang, M. (2024). Multitree genetic programming with feature-based transfer learning for symbolic regression on incomplete data. *IEEE Transactions on Cybernetics*, 54(7), 4014–4027. <https://doi.org/10.1109/TCYB.2023.3270319> (cit. on pp. 2, 21).
- Alzubi, J., Nayyar, A., & Kumar, A. (2018). Machine learning from theory to algorithms: An overview. *Journal of physics: conference series*, 1142, 012012 (cit. on p. 5).
- Archetti, F., Lanzeni, S., Messina, E., & Vanneschi, L. (2006). Genetic programming for human oral bioavailability of drugs. *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, 255–262. <https://doi.org/10.1145/1143997.1144042> (cit. on p. 36).
- Archetti, F., Lanzeni, S., Messina, E., & Vanneschi, L. (2007a). Genetic programming and other machine learning approaches to predict median oral lethal dose (ld50) and plasma protein binding levels (%ppb) of drugs. In E. Marchiori, J. H.

- Moore, & J. C. Rajapakse (Eds.), *Evolutionary computation, machine learning and data mining in bioinformatics* (pp. 11–23). Springer Berlin Heidelberg. (Cit. on p. 36).
- Archetti, F., Lanzeni, S., Messina, E., & Vanneschi, L. (2007b). Genetic programming for computational pharmacokinetics in drug discovery and development. *Genetic Programming and Evolvable Machines*, 8(4), 413–432. <https://doi.org/10.1007/s10710-007-9040-z> (cit. on p. 36).
- Back, T., & Schwefel, H.-P. (1996). Evolutionary computation: An overview. *Proceedings of IEEE International Conference on Evolutionary Computation*, 20–29 (cit. on p. 12).
- Bali, K. K., Gupta, A., Feng, L., Ong, Y. S., & Siew, T. P. (2017). Linearized domain adaptation in evolutionary multitasking. *2017 IEEE Congress on Evolutionary Computation (CEC)*, 1295–1302. <https://doi.org/10.1109/CEC.2017.7969454> (cit. on pp. 2, 21).
- Bali, K. K., Gupta, A., Ong, Y.-S., & Tan, P. S. (2021). Cognizant multitasking in multi-objective multifactorial evolution: Mo-mfea-ii. *IEEE Transactions on Cybernetics*, 51(4), 1784–1796. <https://doi.org/10.1109/TCYB.2020.2981733> (cit. on p. 21).
- Bali, K. K., Ong, Y.-S., Gupta, A., & Tan, P. S. (2020). Multifactorial evolutionary algorithm with online transfer parameter estimation: Mfea-ii. *IEEE Transactions on Evolutionary Computation*, 24(1), 69–83. <https://doi.org/10.1109/TEVC.2019.2906927> (cit. on pp. 2, 21).
- Bi, Y., Xue, B., & Zhang, M. (2022). Learning and sharing: A multitask genetic programming approach to image feature learning. *IEEE Transactions on Evolutionary Computation*, 26(2), 218–232. <https://doi.org/10.1109/TEVC.2021.3097043> (cit. on pp. 2, 22).
- Bi, Y., Xue, B., & Zhang, M. (2023). Multitask feature learning as multiobjective optimization: A new genetic programming approach to image classification. *IEEE Transactions on Cybernetics*, 53(5), 3007–3020. <https://doi.org/10.1109/TCYB.2022.3174519> (cit. on pp. 2, 22).
- Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Springer-Verlag. (Cit. on p. 6).
- Borchani, H., Varando, G., Bielza, C., & Larrañaga, P. (2015). A survey on multi-output regression. *WIREs Data Mining and Knowledge Discovery*, 5, 216–233. <https://doi.org/10.1002/widm.1157> (cit. on p. 11).
- Caruana, R. (1997). Multitask learning. *Machine Learning*, 28(1), 41–75. <https://doi.org/10.1023/A:1007379606734> (cit. on pp. 1, 11, 21).
- Cassotti, M., Ballabio, D., Consonni, V., Mauri, A., Tetko, I. V., & Todeschini, R. (2014). Prediction of acute aquatic toxicity toward daphnia magna by using the ga-knn method. *Alternatives to Laboratory Animals*, 42(1), 31–41. <https://doi.org/10.1177/026119291404200106> (cit. on p. 32).
- Chen, Q., Xue, B., & Zhang, M. (2019a). Differential evolution for instance based transfer learning in genetic programming for symbolic regression. *Proceedings*

- of the Genetic and Evolutionary Computation Conference Companion, 161–162. <https://doi.org/10.1145/3319619.3321941> (cit. on p. 20).
- Chen, Q., Xue, B., & Zhang, M. (2019b). Instance based transfer learning for genetic programming for symbolic regression. *2019 IEEE Congress on Evolutionary Computation (CEC)*, 3006–3013. <https://doi.org/10.1109/cec.2019.8790217> (cit. on p. 20).
- Chen, Q., Xue, B., & Zhang, M. (2022). Genetic programming for instance transfer learning in symbolic regression. *IEEE Transactions on Cybernetics*, 52(1), 25–38. <https://doi.org/10.1109/TCYB.2020.2969689> (cit. on p. 20).
- Darwin, C. (1859). *On the origin of species by means of natural selection* [or the Preservation of Favored Races in the Struggle for Life]. Murray. (Cit. on pp. 1, 12).
- De Jong, K. (1988). Learning with genetic algorithms: An overview. *Machine Learning*, 3(2), 121–138. <https://doi.org/10.1007/BF00113894> (cit. on p. 12).
- Dua, D., & Graff, C. (2017). Uci machine learning repository. <http://archive.ics.uci.edu/ml> (cit. on p. 32).
- Espejo, P. G., Ventura, S., & Herrera, F. (2010). A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(2), 121–144. <https://doi.org/10.1109/TSMCC.2009.2033566> (cit. on pp. 1, 13).
- Feng, L., Huang, Y., Zhou, L., Zhong, J., Gupta, A., Tang, K., & Tan, K. C. (2021). Explicit evolutionary multitasking for combinatorial optimization: A case study on capacitated vehicle routing problem. *IEEE Transactions on Cybernetics*, 51(6), 3143–3156. <https://doi.org/10.1109/TCYB.2019.2962865> (cit. on p. 22).
- Feng, L., Zhou, L., Zhong, J., Gupta, A., Ong, Y.-S., Tan, K.-C., & Qin, A. K. (2019). Evolutionary multitasking via explicit autoencoding. *IEEE Transactions on Cybernetics*, 49(9), 3457–3470. <https://doi.org/10.1109/TCYB.2018.2845361> (cit. on p. 22).
- Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13, 2171–2175 (cit. on p. 29).
- Fu, W., Xue, B., Zhang, M., & Gao, X. (2017). Transductive transfer learning in genetic programming for document classification. In Y. Shi, K. C. Tan, M. Zhang, K. Tang, X. Li, Q. Zhang, Y. Tan, M. Middendorf, & Y. Jin (Eds.), *Simulated evolution and learning* (pp. 556–568). Springer International Publishing. (Cit. on p. 20).
- Gill, P. E., Murray, W., & Wright, M. H. (1981). *Practical optimization*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers]. (Cit. on p. 11).
- Gupta, A., Ong, Y.-S., & Feng, L. (2016). Multifactorial evolution: Toward evolutionary multitasking. *IEEE Transactions on Evolutionary Computation*, 20(3), 343–357. <https://doi.org/10.1109/TEVC.2015.2458037> (cit. on pp. 2, 21).

- Gupta, A., Ong, Y.-S., Feng, L., & Tan, K. C. (2017). Multiobjective multifactorial optimization in evolutionary multitasking. *IEEE Transactions on Cybernetics*, 47, 1652–1665. <https://doi.org/10.1109/TCYB.2016.2554622> (cit. on pp. 2, 21).
- Haslam, E., Xue, B., & Zhang, M. (2016). Further investigation on genetic programming with transfer learning for symbolic regression. *2016 IEEE Congress on Evolutionary Computation (CEC)*, 3598–3605. <https://doi.org/10.1109/CEC.2016.7744245> (cit. on pp. 2, 19).
- Holland, J. H. (1975). *Adaptation in natural and artificial systems*. The University of Michigan Press. (Cit. on p. 12).
- Hossin, M., & Sulaiman, M. N. (2015). A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2), 1 (cit. on p. 7).
- Huong Dinh, T. T., Chu, T. H., & Nguyen, Q. U. (2015). Transfer learning in genetic programming. *2015 IEEE Congress on Evolutionary Computation (CEC)*, 1145–1151. <https://doi.org/10.1109/CEC.2015.7257018> (cit. on pp. 2, 19).
- Icke, I., & Bongard, J. C. (2013). Improving genetic programming based symbolic regression using deterministic machine learning. *2013 IEEE Congress on Evolutionary Computation*, 1763–1770. <https://doi.org/10.1109/CEC.2013.6557774> (cit. on p. 26).
- Iqbal, M., Al-Sahaf, H., Xue, B., & Zhang, M. (2019). Genetic programming with transfer learning for texture image classification. *Soft Computing*, 23(23), 12859–12871. <https://doi.org/10.1007/s00500-019-03843-5> (cit. on pp. 2, 20).
- Iqbal, M., Xue, B., Al-Sahaf, H., & Zhang, M. (2017). Cross-domain reuse of extracted knowledge in genetic programming for image classification. *IEEE Transactions on Evolutionary Computation*, 21(4), 569–587. <https://doi.org/10.1109/TEVC.2017.2657556> (cit. on pp. 2, 20).
- Iqbal, M., Xue, B., & Zhang, M. (2016). Reusing extracted knowledge in genetic programming to solve complex texture image classification problems. In J. Bailey, L. Khan, T. Washio, G. Dobbie, J. Z. Huang, & R. Wang (Eds.), *Advances in knowledge discovery and data mining* (pp. 117–129). Springer International Publishing. (Cit. on pp. 2, 20).
- Iqbal, M., Zhang, M., & Xue, B. (2016). Improving classification on images by extracting and transferring knowledge in genetic programming. *2016 IEEE Congress on Evolutionary Computation (CEC)*, 3582–3589. <https://doi.org/10.1109/CEC.2016.7744243> (cit. on pp. 2, 20).
- Jackson, D., & Gibbons, A. P. (2007). Layered learning in boolean gp problems. In M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, & A. I. Esparcia-Alcázar (Eds.), *Genetic programming* (pp. 148–159). Springer Berlin Heidelberg. (Cit. on pp. 2, 19).
- Jaśkowski, W., Krawiec, K., & Wieloch, B. (2007). Genetic programming for cross-task knowledge sharing. *Proceedings of the 9th Annual Conference on Genetic and*

- Evolutionary Computation*, 1620–1627. <https://doi.org/10.1145/1276958.1277281> (cit. on pp. 2, 21).
- Jaśkowski, W., Krawiec, K., & Wieloch, B. (2008). Multitask Visual Learning Using Genetic Programming. *Evolutionary Computation*, 16(4), 439–459. <https://doi.org/10.1162/evco.2008.16.4.439> (cit. on p. 21).
- Jaśkowski, W., Krawiec, K., & Wieloch, B. (2014). Cross-task code reuse in genetic programming applied to visual learning. *International Journal of Applied Mathematics and Computer Science*, 24(1), 183–197. <https://doi.org/doi:10.2478/amcs-2014-0014> (cit. on p. 21).
- Kennedy, T. (1997). Managing the drug discovery/development interface. *Drug Discovery Today*, 2(10), 436–444. [https://doi.org/https://doi.org/10.1016/S1359-6446\(97\)01099-4](https://doi.org/https://doi.org/10.1016/S1359-6446(97)01099-4) (cit. on p. 36).
- Kola, I., & Landis, J. (2004). Can the pharmaceutical industry reduce attrition rates? *Nat Rev Drug Discov*, 3(8), 711–715 (cit. on p. 36).
- Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. MIT Press. <http://mitpress.mit.edu/books/genetic-programming> (cit. on pp. 1, 12, 14, 15, 17, 37).
- Langdon, W. B., & Nordin, J. P. (2000). Seeding genetic programming populations. In R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, & T. C. Fogarty (Eds.), *Genetic programming* (pp. 304–315). Springer Berlin Heidelberg. (Cit. on pp. 1, 19).
- Lever, J. (2016). Classification evaluation: It is important to understand both what a classification metric expresses and what it hides. *Nature methods*, 13(8), 603–605 (cit. on p. 7).
- Lin, J., Chen, Q., Xue, B., & Zhang, M. (2023). Evolutionary multitasking for multi-objective feature selection in classification. *IEEE Transactions on Evolutionary Computation*, 1–1. <https://doi.org/10.1109/TEVC.2023.3338740> (cit. on p. 22).
- Lourenço, J. M. (2021). *The NOVAthesis L^AT_EX Template User’s Manual*. NOVA University Lisbon. <https://github.com/joaomlourenco/novathesis/raw/main/template.pdf> (cit. on p. i).
- Luke, S., & Panait, L. (2002). Fighting bloat with nonparametric parsimony pressure. In J. J. M. Guervós, P. Adamidis, H.-G. Beyer, H.-P. Schwefel, & J.-L. Fernández-Villacañas (Eds.), *Parallel problem solving from nature — ppsn vii* (pp. 411–421). Springer. https://doi.org/10.1007/3-540-45712-7_40 (cit. on p. 17).
- Luke, S., & Panait, L. (2006). A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3), 309–344. <https://doi.org/10.1162/evco.2006.14.3.309> (cit. on p. 17).
- M. Cassotti, R. T., D. Ballabio, & Consonni, V. (2015). A similarity-based qsar model for predicting acute toxicity towards the fathead minnow (*pimephales promelas*) [PMID: 25780951]. *SAR and QSAR in Environmental Research*, 26(3), 217–243. <https://doi.org/10.1080/1062936X.2015.1018938> (cit. on p. 33).

- Mei, Y., Ardeh, M. A., & Zhang, M. (2021). Knowledge transfer in genetic programming hyper-heuristics. In N. Pillay & R. Qu (Eds.), *Automated design of machine learning and search algorithms* (pp. 149–169). Springer International Publishing. https://doi.org/10.1007/978-3-030-72069-8_9 (cit. on p. 20).
- Miller, B. L., Goldberg, D. E., et al. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, 9(3), 193–212 (cit. on p. 15).
- Mitchell, T. M. (1997). *Machine learning* (1st ed.). McGraw-Hill, Inc. (Cit. on p. 5).
- Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). *Foundations of machine learning*. The MIT Press. (Cit. on pp. 5–7).
- Montana, D. J. (1995). Strongly typed genetic programming. *Evolutionary Computation*, 3(2), 199–230. <https://doi.org/10.1162/evco.1995.3.2.199> (cit. on p. 22).
- Muñoz, L., Trujillo, L., & Silva, S. (2020). Transfer learning in constructive induction with Genetic Programming. *Genetic Programming and Evolvable Machines*, 21(4), 529–569. <https://doi.org/10.1007/s10710-019-09368-y> (cit. on pp. 2, 20, 42, 57).
- Muñoz, L., Trujillo, L., Silva, S., Castelli, M., & Vanneschi, L. (2019). Evolving multidimensional transformations for symbolic regression with m3gp. *Memetic Computing*, 11(2), 111–126. <https://doi.org/10.1007/s12293-018-0274-5> (cit. on pp. 1, 2, 20).
- Murphy, K. P. (2012). *Machine learning : A probabilistic perspective*. MIT. <https://www.worldcat.org/title/machine-learning-a-probabilistic-perspective/oclc/781277861?referer=br&ht=edition> (cit. on p. 6).
- Naser, M. Z., & Alavi, A. H. (2023). Error metrics and performance fitness indicators for artificial intelligence and machine learning in engineering and sciences. *Architecture, Structures and Construction*, 3(4), 499–517. <https://doi.org/10.1007/s44150-021-00015-8> (cit. on p. 7).
- Niu, S., Liu, Y., Wang, J., & Song, H. (2020). A decade survey of transfer learning (2010–2020). *IEEE Transactions on Artificial Intelligence*, 1(2), 151–166 (cit. on p. 1).
- O'Neill, D., Al-Sahaf, H., Xue, B., & Zhang, M. (2017). Common subtrees in related problems: A novel transfer learning approach for genetic programming. 2017 *IEEE Congress on Evolutionary Computation (CEC)*, 1287–1294. <https://doi.org/10.1109/CEC.2017.7969453> (cit. on pp. 2, 20).
- Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 1345–1359. <https://doi.org/10.1109/TKDE.2009.191> (cit. on pp. 1, 9, 19).
- Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, 113, 54–71. <https://doi.org/https://doi.org/10.1016/j.neunet.2019.01.012> (cit. on p. 11).
- Park, J., Mei, Y., Nguyen, S., Chen, G., & Zhang, M. (2018). Evolutionary multitask optimisation for dynamic job shop scheduling using niched genetic programming. In T. Mitrovic, B. Xue, & X. Li (Eds.), *Ai 2018: Advances in artificial intelligence* (pp. 739–751). Springer International Publishing. (Cit. on p. 23).

- Perkins, D. N., Salomon, G., et al. (1992). Transfer of learning. *International encyclopedia of education*, 2, 6452–6457 (cit. on p. 8).
- Perry, J. (1994). The effect of population enrichment in genetic programming. *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, 456–461 vol.1. <https://doi.org/10.1109/ICEC.1994.349907> (cit. on pp. 1, 19).
- Qiao, K., Liang, J., Liu, Z., Yu, K., Yue, C., & Qu, B. (2023). Evolutionary multitasking with global and local auxiliary tasks for constrained multi-objective optimization. *IEEE/CAA Journal of Automatica Sinica*, 10(10), 1951–1964. <https://doi.org/10.1109/jas.2023.123336> (cit. on p. 22).
- Qiao, K., Yu, K., Qu, B., Liang, J., Song, H., & Yue, C. (2022). An evolutionary multitasking optimization framework for constrained multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 26(2), 263–277. <https://doi.org/10.1109/TEVC.2022.3145582> (cit. on p. 22).
- Ravisankar, P., Ravi, V., Raghava Rao, G., & Bose, I. (2011). Detection of financial statement fraud and feature selection using data mining techniques [Cited by: 345]. *Decision Support Systems*, 50(2), 491–500. <https://doi.org/10.1016/j.dss.2010.11.006> (cit. on p. 1).
- Silva, S., & Costa, E. (2009). Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines*, 10(2), 141–179. <https://doi.org/10.1007/s10710-008-9075-9> (cit. on p. 17).
- Silva, S., & Vanneschi, L. (2012). Bloat free genetic programming: Application to human oral bioavailability prediction. *International Journal Of Data Mining And Bioinformatics*, 6(6), 585–601. <https://doi.org/10.1504/IJDMB.2012.050266> (cit. on p. 36).
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (Second). The MIT Press. <http://incompleteideas.net/book/the-book-2nd.html> (cit. on p. 6).
- Tang, Z., Gong, M., Wu, Y., Liu, W., & Xie, Y. (2021). Regularized evolutionary multitask optimization: Learning to intertask transfer in aligned subspace. *IEEE Transactions on Evolutionary Computation*, 25(2), 262–276. <https://doi.org/10.1109/TEVC.2020.3023480> (cit. on pp. 2, 21).
- Torrey, L., & Shavlik, J. (2010). Transfer learning. In *Handbook of research on machine learning applications and trends* (pp. 242–264). IGI Global. <https://doi.org/10.4018/978-1-60566-766-9.ch011> (cit. on p. 8).
- Tsanas, A., Little, M. A., McSharry, P. E., & Ramig, L. O. (2010). Accurate telemonitoring of parkinson’s disease progression by noninvasive speech tests. *IEEE Transactions on Biomedical Engineering*, 57(4), 884–893. <https://doi.org/10.1109/TBME.2009.2036000> (cit. on p. 34).

- Vanneschi, L., & Silva, S. (2023). *Lectures on intelligent systems*. Springer, Cham. <https://doi.org/10.1007/978-3-031-17922-8> (cit. on pp. 12–14, 27).
- Wang, Z., Dai, Z., Poczos, B., & Carbonell, J. (2019). Characterizing and avoiding negative transfer. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 8).
- Wen, Y.-W., & Ting, C.-K. (2016). Learning ensemble of decision trees through multi-factorial genetic programming. *2016 IEEE Congress on Evolutionary Computation (CEC)*, 5293–5300. <https://doi.org/10.1109/CEC.2016.7748363> (cit. on pp. 2, 22).
- Zhang, F., Mei, Y., Nguyen, S., Tan, K. C., & Zhang, M. (2022). Multitask genetic programming-based generative hyperheuristics: A case study in dynamic scheduling. *IEEE Transactions on Cybernetics*, 52(10), 10515–10528. <https://doi.org/10.1109/TCYB.2021.3065340> (cit. on pp. 2, 23).
- Zhang, F., Mei, Y., Nguyen, S., Tan, K. C., & Zhang, M. (2023). Task relatedness-based multitask genetic programming for dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation*, 27(6), 1705–1719. <https://doi.org/10.1109/TEVC.2022.3199783> (cit. on p. 23).
- Zhang, F., Mei, Y., Nguyen, S., & Zhang, M. (2020). A preliminary approach to evolutionary multitasking for dynamic flexible job shop scheduling via genetic programming. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 107–108. <https://doi.org/10.1145/3377929.3389934> (cit. on p. 23).
- Zhang, F., Mei, Y., Nguyen, S., & Zhang, M. (2023). Multitask multiobjective genetic programming for automated scheduling heuristic learning in dynamic flexible job-shop scheduling. *IEEE Transactions on Cybernetics*, 53(7), 4473–4486. <https://doi.org/10.1109/TCYB.2022.3196887> (cit. on pp. 2, 23).
- Zhang, F., Mei, Y., Nguyen, S., Zhang, M., & Tan, K. C. (2021). Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation*, 25(4), 651–665. <https://doi.org/10.1109/TEVC.2021.3065707> (cit. on pp. 2, 23).
- Zhang, M.-L., & Zhou, Z.-H. (2014). A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8), 1819–1837. <https://doi.org/10.1109/TKDE.2013.39> (cit. on p. 11).
- Zhang, Y., & Rockett, P. I. (2009). A generic multi-dimensional feature extraction method using multiobjective genetic programming. *Evol Comput*, 17(1), 89–115 (cit. on p. 19).
- Zhang, Y., & Yang, Q. (2018). An overview of multi-task learning. *National Science Review*, 5(1), 30–43. <https://doi.org/10.1093/nsr/nwx105> (cit. on p. 1).
- Zhang, Y., & Yang, Q. (2022). A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12), 5586–5609. <https://doi.org/10.1109/TKDE.2021.3070203> (cit. on p. 11).

- Zheng, X., Qin, A. K., Gong, M., & Zhou, D. (2020). Self-regulated evolutionary multitask optimization. *Trans. Evol. Comp*, 24(1), 16–28. <https://doi.org/10.1109/TEVC.2019.2904696> (cit. on pp. 2, 21).
- Zhong, J., Feng, L., Cai, W., & Ong, Y.-S. (2020). Multifactorial genetic programming for symbolic regression problems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(11), 4492–4505. <https://doi.org/10.1109/TSMC.2018.2853719> (cit. on pp. 2, 22).
- Zhong, J., Ong, Y.-S., & Cai, W. (2016). Self-learning gene expression programming. *IEEE Transactions on Evolutionary Computation*, 20(1), 65–80. <https://doi.org/10.1109/TEVC.2015.2424410> (cit. on p. 22).
- Zhou, L., Feng, L., Tan, K. C., Zhong, J., Zhu, Z., Liu, K., & Chen, C. (2021). Toward adaptive knowledge transfer in multifactorial evolutionary computation. *IEEE Transactions on Cybernetics*, 51(5), 2563–2576. <https://doi.org/10.1109/TCYB.2020.2974100> (cit. on pp. 2, 21).
- Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., & He, Q. (2021). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1), 43–76. <https://doi.org/10.1109/JPROC.2020.3004555> (cit. on pp. 8–10).

