Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science

DIPLOMA THESIS AGREEMENT

Student: Gamec Ján

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: Symbolic Regression Using Compound Models

Guidelines:

1) Study current state-of-the-art symbolic regression methods. Focus on evolutionary-based approaches using generalized linear models, i.e. a linear combination of possibly non-linear basis functions.
2) Propose a symbolic regression algorithm producing final models as a composition of non-overlapping partial models.
3) Implement the proposed algorithm.
4) Experimentally evaluate a performance of the proposed algorithm on standard symbolic regression benchmarks as well as on datasets from the reinforcement learning domain (e.g. data sampled from a V-function).
5) Analyse achieved results and compare the proposed algorithm with other existing approaches.

Bibliography/Sources:

[1] Jackson, D.: A new, node-focused model for genetic programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 49–60. Springer, Heidelberg (2012). doi:10.1007/978-3-642-29139-5 5

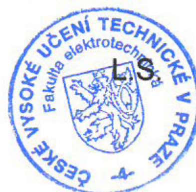
[2] Searson, D.P., Leahy, D.E., Willis, M.J.: GPTIPS: an open source genetic programming toolbox for multigene symbolic regression. In International MultiConference of Engineers and Computer Scientists, vol. 1, pp. 77–80 (2010)

[3] Arnaldo, I., O'Reilly, U.-M., Veeramachaneni, K.: Building Predictive Models via Feature Synthesis. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation GECCO '15, pp. 983–990. ACM, New York (2015)

[4] Jiří Kubalík, Eduard Alibekov, Jan Žegklitz, Robert Babuška: Hybrid Single Node Genetic Programming for Symbolic Regression. Transactions on Computational Collective Intelligence XXIV, Volume 9770, LNCS, pp. 61-82, 2016

Diploma Thesis Supervisor: Ing. Jiří Kubalík, Ph.D.

Valid until the end of the summer semester of academic year 2017/2018

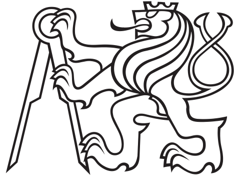


prof. Dr. Michal Pěchouček, MSc.
Head of Department

prof. Ing. Pavel Ripka,CSc.
Dean

Prague, January 16, 2017

Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Symbolic Regression Using Compound Models

Bc. Ján Gamec

Supervisor: Ing. Jiří Kubalík, Ph.D.
May 2017

Acknowledgements

First of all, I would like to thank my supervisor, Ing. Jiří Kubalík Ph.D. for his advice, supervision, endless patience and time he dedicated me while creating this thesis. I would also like to thank my consultant, Prof. Dr. Ing. Robert Babuška for valuable research advice. At last but not least, I would like to express my great gratitude to my family, friends and especially my parents, who supported me, advised me and helped me during whole study and preparation of this thesis.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used within the research and work. I have no objections to usage of this work in compliance with Act No 121/2000 Sb. (the Copyright Act), as amended, and in compliance with copyright-related rights currently in force.

Abstract

The aim of the work is to propose a robust Symbolic Regression method based on Genetic Programming, which will be capable of finding symbolic models for complex datasets containing nonlinearities, irregularities and other artifacts. The method is based on a principle of dividing the dataset into regions and finding a piecewise models locally for them. The proposed method is tested on synthetic datasets and also on datasets sampled from real tasks such as Reinforcement Learning problems. The performance of the method is also compared to the performance of standard Genetic Programming approach, which finds single global model for whole dataset.

Keywords: Symbolic Regressing, Piecewise Models, Genetic Programming, Hierarchical Learning

Supervisor: Ing. Jiří Kubalík, Ph.D.

Abstrakt

Cieľom tejto práce je navrhnúť robustnú metódu pre Symbolickú Regresiu založenú na princípoch Genetického Programovania, ktorá bude schopná nájsť symbolické modely zložitých datasetov obsahujúcich nelinearity, nepravidelnosti a ďalšie artefakty. Metóda je založená na princípe rozdelenia datasetu na oblasti, na ktorých sa hľadajú symbolické modely lokálne. Navrhnutá metóda je testovaná na syntetických datasetoch ako aj datasetoch pochádzajúcich z reálnych úloh ako je napríklad Reinforcement Learning. Metóda je taktiež porovnaná so štandardnou metódou založenou na Genetickom Programovaní, ktorá hľadá jeden globálny model pre celý dataset.

Kľúčové slová: Symbolická Regresia, Čiastočné modely, Genetické programovanie, Hierarchické učenie

Contents

1 Introduction 1

2 Evolutionary approaches in symbolic regression 3
2.1 Evolutionary Feature Synthesis . . 3
2.2 Multi-gene Genetic Programming 4
2.3 SNGP 5
2.3.1 SNGP Population 6
2.3.2 Fitness Evaluation 6
2.3.3 SNGP Operator 7
2.3.4 Evolutionary Search 7
2.3.5 Single-run SNGP with LASSO regression 8
2.4 Summary 8

3 Local Model Methods 9
3.1 Local Model Networks 9
3.2 Operating regime decomposition 10
3.3 Multi-Modal Symbolic Regression 11
3.3.1 Clustered Symbolic Regression 12
3.3.2 Transition Modeling 12
3.3.3 Problems in MMSR 12
3.4 Summary 13

4 Proposed Method 15
4.1 Clustering 16
4.1.1 Spectral clustering 18
4.1.2 Simple Clustering 19
4.1.3 Evolutionary Hierarchical Clustering (EHC) 20
4.2 Hierarchical learning 24
4.3 Piecewise models merging 27
4.3.1 Labeling unseen data 27
4.3.2 Merging method 1 28
4.3.3 Method 2 30
4.3.4 Method 3 31
4.3.5 Model merging in hierarchical models 34

5 Experiments 35
5.1 Setup 35
5.2 Clustering Experiments 37
5.3 Experiments with proposed method 38
5.3.1 Evaluation and Analysis 41

6 Conclusion 47

Bibliography 49

A User Guide 51
A.1 Requirements 51
A.2 Organization of appended CD . 51
A.3 Dataset 51
A.4 Clustering 51
A.5 Hierarchical Learning 52
A.6 Merging 53

Figures

2.1 The building blocks of the EFS algorithm. 4
2.2 EFS population example 5
2.3 MGGP individual example 6
2.4 SNGP population example 7

4.1 Block scheme of the proposed method 16
4.2 Example of GVF 18
4.3 Simple clustering example 21
4.4 The idea of EHC algorithm 21
4.5 Scheme of the EHC algorithm 22
4.6 Hierarchical function distribution 23
4.7 Example of simple neighborhood search tree 25
4.8 Example of 4-nearest grid neighbors 28
4.9 Merging 1 algorithm example 30
4.10 Merging 2 algorithm example 31
4.11 Merging 3 Case 1 example 33
4.12 Merging 3 Case 2 example 34

5.1 Visualization of 1DOF Swingup dataset (a) and Robot Arm Policy dataset (b). 36
5.2 Visualization of 1DOF Swingup Policy dataset 37
5.3 Custom analytic functions visualization 37
5.4 Visual comparison of clustering approaches on 1DOF Swingup problem 39
5.5 Visual comparison of clustering approaches on 1DOF Swingup Policy problem 39
5.6 Visual comparison of clustering approaches on Robot Arm Policy problem 40
5.7 Visual comparison of clustering approaches on custom Fun1 problem 40
5.8 Visual comparison of clustering approaches on custom Fun2 problem 41
5.9 Visual comparison of best runs of SNGP and HL on Fun1 dataset . . . 44
5.10 Visual comparison of best runs of SNGP and HL on Fun2 dataset . . . 44
5.11 Visual comparison of best runs of SNGP and HL on 1DOF Swingup dataset 45
5.12 Visual comparison of best runs of SNGP and HL on 1DOF Swingup Policy dataset 46
5.13 Visual comparison of best runs of SNGP and HL on Robot Arm Policy dataset 46

Tables

5.1 Experimental datasets overview. 36
5.2 SNGP Parameters setup 38
5.3 Results of experiments on Function 1. 43
5.4 Results of experiments on Function 2. 44
5.5 Results of experiments on 1 DOF Swingup problem. 45
5.6 Results of experiments on 1 DOF Swingup Policy. 45
5.7 Results of experiments on Robot Arm Policy. 46

Introduction

Symbolic Regression (SR) is a modeling technique with a wide range of applications in Mathematics, Physics, Artificial Intelligence and Reinforcement Learning (RL). A common scenario in SR problems is a discrete dataset of input points, which were sampled experimentally from problems like robot control task, or from some analytic function. The main difficulty with such problems is a complexity of dataset, which is represented by occurrences of various nonlinearities and irregularities.

Application of standard SR approaches, such as Genetic Programming (GP) showed, that the task of fitting datasets containing such artifacts is a really challenging problem, and can be hardly solved using only single run of standard GP. This is caused not only by insufficient expressiveness of basic functions set that are used to create a symbolic model in case of a GP, but also by selection of objective measure, which evaluates the candidate models. The problem with the objective measures such as Mean Square Error is that they are global, what makes it difficult to fit a model to local and possibly detailed artifacts. An example situation of this problem is when such artifact appears and covers only small area in dataset, and then is usually ignored by the learning process, which results in a sufficient model in the meaning of the MSE metric, but may result in an insufficient model in the meaning of usability.

The thesis proposed a SR method based on GP, which fits a local piecewise models on the divided dataset, and merges these models into a single compound model. The method utilizes state-of-the-art GP approach called Single Node Genetic Programming (SNGP). The idea of dividing the dataset is based on principles of Local Model Networks (LMN) and is used to divide the dataset into partial datasets, or better said clusters, which might be easier to fit. SNGP is used to find symbolic models of the partial datasets separately. In the final phase of the learning process, piecewise models are merged, to form a single compound model.

The first chapter of this thesis is dedicated to overview of some of the state of the art evolutionary-based approaches to Symbolic Regression, from which SNGP was selected as the most promising option, due to previous experimental results. The next chapter then overviews basic principles of

the Local Model Networks as well as current approaches, which solve the piecewise modeling in similar areas, such as control tasks.

Chapter 4 describes the proposed method, which is divided into 3 separate steps including clustering, hierarchical learning of piecewise models and final merging. It proposes 3 different approaches to clustering and 3 different approaches to merging. In chapter 5, all approaches are experimentally evaluated and tested on datasets from RL control problems as well as SR benchmark problems. Their performance is analyzed and compared to state of the art implementation of SNGP algorithm, which finds a single global model for the whole dataset.

Appendix A provides a guide for running the implementation of the method as well as scripts to visualize datasets, trained models and to replicate the experiments.

Chapter 2

Evolutionary approaches in symbolic regression

Symbolic Regression (SR) can be described as a type of regression analysis, where the search space consists of mathematical expressions and functions, which are combined in order to find a complex model that fits the target dataset with the lowest possible error, with respect to given error metric. This is a problem, that can be solved by *evolutionary search* and particularly by a technique called Genetic Programming (GP)[1]. Lately, there have been proposed several approaches based on the standard approach such as Grammatical Evolution [2] or Gene Expression Programming [3], which worked well on SR as well. In this chapter, I review some of the state-of-the-art methods used to solve this problem.

2.1 Evolutionary Feature Synthesis

Evolutionary Feature Synthesis (EFS) is a regression method based on evolutionary computation that generates readable nonlinear models. According to [4] , EFS is one of the fastest regression tools reported to date. It utilizes a state-of-the-art implementation of regularized linear regression. The method combines principles of evolutionary computation and regularized linear regression, but differs from other population-based approaches in a representation of individuals in a population. Where similar population-based methods evolve a set of candidate solutions represented as individuals, the EFS evolves just single solution (expression), where individuals in the population represent features usually in a form of mathematical functions. An example population of EFS is depicted in Figure 2.2.

Figure 2.1 shows the basic building block of the EFS algorithm. In the initialization step of the algorithm, the population is seeded with functions $f_i(X) = X_i$, where $i = 1...p$, where p is number of original variables. The algorithm then creates a linear combination of current population, which consists of original variables in the beginning. This step corresponds to the Model Generation block in Figure 2.1. EFS utilizes a method called LASSO (Least Absolute Shrinkage and Selection Operator described in [5]) to obtain coefficients of the linear combination. If error of the linear combination of

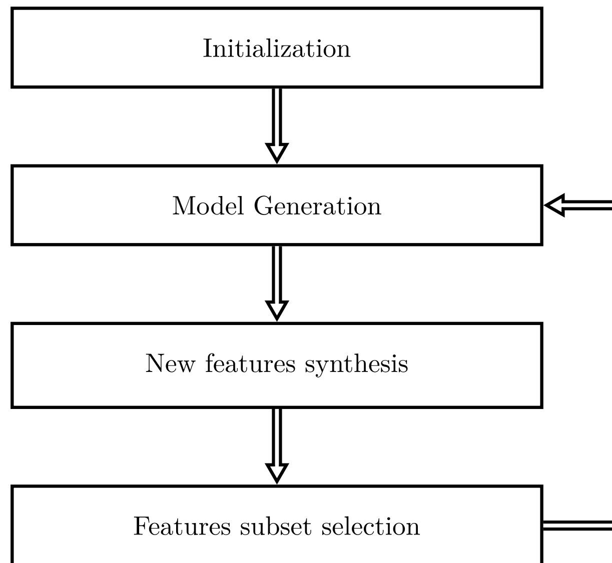


Figure 2.1: The building blocks of the EFS algorithm.

current population is reduced in comparison with the best model found so far, the newly generated model is stored.

In next step denoted as New feature synthesis in figure, the population is extended by new features, which are created similarly as in tree-based Genetic Programming approaches that utilize crossover and mutation operators. In EFS new features are created applying unary and binary operators like $+, *, sin()$ on existing features.

The last step of the EFS optimization loop is called Features subset selection and is meant to reduce number of features in the current population according to their importance in the model. To achieve this, the second run of LASSO regression is utilized with current population extended by new features, which estimates the importance of features. In final, there are selected $p+q$ features, which are stored as a model and μ features that are discarded. This process is also shown in Figure 2.2.

If the EFS search process does not find a better model for a preset number of iteration, or maximum number of iterations is reached, the search is stopped. Else, the search process goes back to Model Generation

2.2 Multi-gene Genetic Programming

In standard GP approaches, there is evolved a population of trees, where each of the trees encodes some mathematical equation. Each tree (equation) in the population is called an individual and is evaluated separately. At the end of search process, there is selected the best single individual, which minimizes the prediction error on the input.

In contrast with standard GP approach, Multi-Gene Genetic Programming (MGGP) [6] individual is a combination of several GP trees, where each

Function	$f_1(X)$	$f_2(X)$	$f_3(X)$	$f_4(X)$	$f_5(X)$	$f_6(X)$	$f_7(X)$	$f_8(X)$	$f_9(X)$	$f_{10}(X)$	$f_{11}(X)$
Score	11	5	4	1	5	6	4	-	-	-	-
Size	1	1	1	1	2	3	2	4	2	5	2
Expression	x_1	x_2	x_3	x_4	$(x_4 * x_2)$	$\tan(x_3)$	$\ln(x_1)$	$x_4\sin(x_2)$	$\cos(x_1)$	$\frac{x_1}{(x_3+x_2)}$	$(-x_3)^3$

p q µ

M

Figure 2.2: EFS population example - every generation, μ new features are composed by applying unary or binary operators to the current population of features. The size of the current population is given by $p + q$, where p is the dimensionality of the data and q is the number of composed features surviving from the previous generation.

tree represents some symbolic function. Single GP tree within an MGGP individual is called a *gene*. The whole MGGP individual is then evaluated as a linear combination of outputs of its genes. This can be formally described by the equation:

$$y = a_0 + a_1 f_1 + a_2 f_2 \dots a_n f_n,$$

where $f_1, \dots, f_n$ are single trees evolved by evolutionary search using common GP operators and $a_0, \dots, a_n$ are coefficients calculated using ordinary least squares method.

The initial population of the MGGP is constructed at random, populating individuals with 1 to G_{max} random trees of variable depth. During evolutionary process, nodes in trees are added, modified and deleted using crossover and mutation operators, known from standard GP approach [1]. An example individual of the MGGP population can be seen in Fig. 2.3.

2.3 SNGP

Single Node Genetic Programming (SNGP) [7, 8] is a graph-based GP system, that handles individuals in a similar way that the EFS does. The individual in SNGP is represented not as a GP tree, but as a feature or single program node. The SNGP individuals are interlinked using predefined operators, which results in a graph structure such as the one in Figure 2.4. The population is ordered, so that each individual has its position indexed with number from 0 to $N-1$, where N corresponds to the maximum number of individuals. Each individual can be connected only to individuals which have lower position

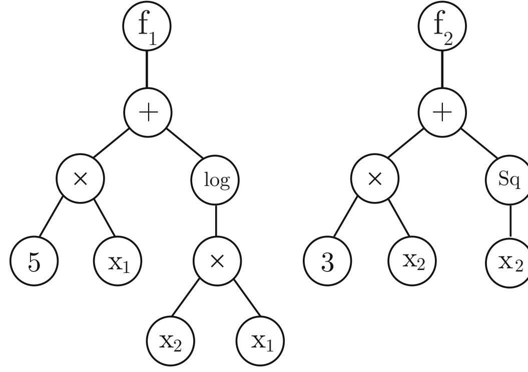


Figure 2.3: MGGP individual example - there are evolved multiple tree, where each tree represents single function. These trees represents symbolic model $y = a_0 + a_1(5x_1 + log(x_2 + x_1)) + a_2(3x_2 + x_2^2)$

index than the current individual.

2.3.1 SNGP Population

SNGP population is an ordered sequence of N individuals

$$M = \{m_0, m_1, ..., m_{N-1}\}$$

where each individual is a tuple in form

$$m_i =< u_i, r_i, S_i, P_i, O_i >$$

where:

$u_i \in \{T \bigcup F\}$ is a single graph node taken from either the function set F or the terminal set T defined by a user;

r_i is the rating of fitness for the individual;

S_i is a set of successors of this node;

P_i is a set of predecessors of the node;

O_i is a vector of outputs generated when this node is evaluated.

A SNGP population is partitioned, so that first K individuals are terminals, what in case of SR means variables and constants. The other $N-K$ individuals are function nodes, which take individuals with lower indexes as operands. This means that each function node can be seen as a root of a tree, or a subtree, that is constructed traversing its successors (operands) recursively.

2.3.2 Fitness Evaluation

In SNGP, fitness of the individual is evaluated using output of a tree rooted in the individual's node. The evaluation process of SNGP starts at terminal nodes, which are placed at lower indexes of the population. During initialization, each terminal node is evaluated on all data points and outputs are stored in O_i vector. Function nodes, which precede terminal nodes in graph

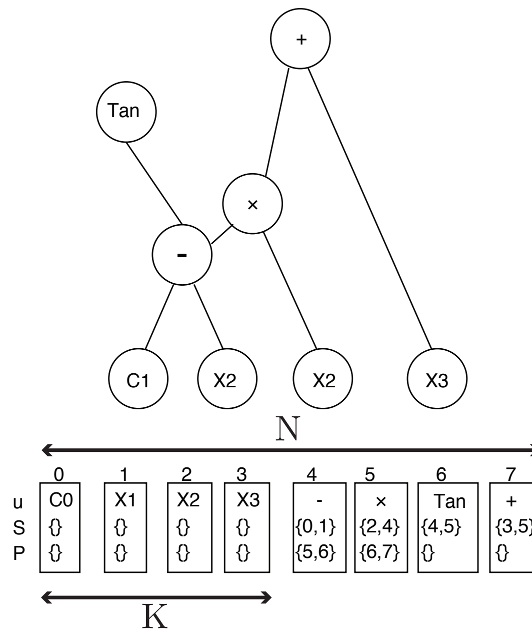


Figure 2.4: SNGP population example - 0th node of the population is generated constant, X1-3 are input variables and other nodes represent function symbols.

representation then utilize these precalculated outputs to make the evaluation process more efficient. The outputs of an individual on each data point is always stored in a vector, so that preceding individuals can utilize them, and don't need to reevaluate whole tree again. Note, that outputs on terminal nodes are calculated only in the initialization phase, because they don't change over time. An example of a SNGP population can be seen in Fig. 2.4. The graph representation corresponds to sequence of individuals below graph. The population is partitioned into $K = 4$ terminals and 4 function nodes.

2.3.3 SNGP Operator

SNGP uses only single evolutionary operator called *successor mutate (smut)*. It selects an individual at random and then exchanges one (or more) of its operands for another individual in the population.

2.3.4 Evolutionary Search

The evolutionary process is driven using a hill-climbing principle, which means that the previous population is replaced by a new only if it is not worse than the previous one. In [7] population evaluation is based on fitness measurements across whole population, rather than on individuals separately. The same author also modifies the approach in [8], so that the population is evaluated according to the best individual. There have also been proposed several enhancements to the original approach such as [9], where authors discuss various selection strategies. Authors also mention combination of

SNGP with other approaches such as local search method, which can be classified as a memetic algorithm. The most promising approach seems to be a combination with already mentioned LASSO regression [5], which is discussed in the following section.

2.3.5 Single-run SNGP with LASSO regression

A Single-run SNGP with LASSO regression is an enhancement of the original SNGP approach proposed in [9]. The main difference between this approach and the original SNGP lies in the way how a population is evaluated. In contrast with SNGP described in [8], where population's fitness corresponds to the fitness of the best individual, in this approach, fitness of the population is calculated using outputs of a subset of individuals.

The first step of the evaluation process is calculating the quality of each individual using Pearson product-moment correlation coefficient between the individuals's output and the desired output value.

In the next step, a generalized linear regression model is created, using only a subset of features from the population. The model is created using LASSO regression [5]. The subset of features for constructing it is called set of predictors. The maximum number of predictors is a preset parameter of the method. The subset of predictors is selected according to a quality measure calculated in the first step.

The created model consists of a limited number of features, which are called regressors and their maximum number is also preset as a parameter. The fitness of the whole population is then calculated as the **sum of absolute errors** of the LASSO model outputs and desired values.

2.4 Summary

This chapter describes few of the state-of-the-art evolutionary approaches utilized in SR. Even though all of these methods are capable of producing symbolic models, that are much better than the original approach proposed in [1], all of them have one drawback, which limits them in better performance on more complex datasets. The drawback is that even if there are evolved several features in a population, all of them are evaluated globally on whole dataset. This means that even if the method succeeded in evolving a feature, which performs well only on a local region of the dataset, the positive outcome of such feature would be suppressed due to the linear combination of features, which is global in final. This effect is however undesirable, because I assume that complex datasets require an approach, which can develop features with a local domain and output not influenced by concurrent features. I assume that such features may fit complex areas more easily and result in a more precise model. In this work, I propose an evolutionary approach based on SNGP, which evolves features locally and merges them together into a single model.

Chapter 3

Local Model Methods

The biggest problem with SR via GP seems to be the complexity of the original function. As far as we have just the limited amount of examples, the method tends to overfit certain areas but also underfit the others. Possible solution concept to this problem might be to split the state space into several subregions in which original function behaves simpler and less complex. The learning algorithm would then try to adapt on these subregions and the original function should come up from joining these sub-functions, which correspond to each of these subregions. This chapter will thus overview some existing approaches that deal with such problems.

3.1 Local Model Networks

The Local Model Networks (LMN) [10] are the generalization of the Radial Basis Functions (RBF), which were introduced by Broomhead and Lowe in [11] and Poggio and Girosi in [12]. RBFs are three layered feed-forward neural networks, which use linear transfer function for the output neurons and nonlinear (usually Gaussian) transfer function for neurons in the hidden layer. The training process of the network then adjusts the parameters of the transfer functions (like centroids in Gaussians) on the hidden neurons, which leads (in case of Gaussians) to dividing the input space into subregions.

LMNs replaces the hidden neurons in the RBFs with some local submodels (or local functions in other words), which output is fed into their basis or validity functions. The validity function, in this case, defines an area in state space, to which corresponding local model contributes. Authors of a method in [13] use terminology of operating regimes for these areas, to design a control system using a global model. The global model is created of multiple local models according to the domain of corresponding operating regimes. Formally speaking, the output of the network can be described as:

$$y = \sum_{i=1}^{m} \rho_i(x, \eta_i) h_i(x, a_i),$$

where h_i is the local model, ρ_i is the validity function, x an input vector and η_i/a_i parameters of validity/local model function. The local model can be

arbitrary linear or nonlinear function, but the validity functions are restricted to satisfy the constraint

$$\sum_{i=1}^{m} \rho_i(x, \eta_i) = 1 \qquad \forall x \in X$$

The main problem with LMNs is thus not only finding appropriate (and simple) local models, but also a validity functions, which satisfies the *partition of unity* property. There were proposed several techniques to learning and estimating the parameters of the validity functions.

3.2 Operating regime decomposition

The process of creating the local model networks can be divided into 2 tasks, where the first one deals with the identification of local models and the other one deals with the estimation of parameters of the validity function. Usually, the local models identification task utilizes some *a priori* knowledge to create models such as the method proposed in [14], which constructs Takagi-Sugeno (TS) fuzzy models. A general approach to estimation of parameters of the validity function is an optimization problem, which minimizes some (objective) measure. Most of the current approaches to solving these tasks follow the so-called **Divide and Conquer** strategy. According to the authors of the LMN [13], the task of estimating such validity functions may also be called an *operating regime decomposition*.

According to [15], the techniques for regime (local subregions) decomposition can be divided into following classes:

1. **Fixed Selection**
   Centers of the regimes are selected randomly and the dimensions of the regions are calculated based on some *a priori* information. This approach is not suitable for more complex problems.

2. **Self-organizing and Clustering**
   This approach utilizes unsupervised learning technique to estimate the centers of the regimes. Authors of [16] are using well-known Expectation-Maximization (EM) algorithm to identify centers and parameters of the operating regimes. The main disadvantage of this approach is that it does not take into account the complexity of the problem (complexity of original function in symbolic regression), but only the density of the function.

3. **Non-optimal construction algorithms with heuristic growing strategies**
   This family of approaches starts with a single regime and iteratively splits it into smaller subregimes. An example of such approach is the original algorithm proposed in [13].

4. **Splitting and merging**
   This approach is more oriented on complexity than previous approaches, because it again starts from a single operating regime and splits it into two, but then studies the behavior of the subregions and only if their behavior is satisfactory (less complex than the original regime), the split is accepted. Otherwise, the regimes are put back together.

5. **Fine-to-course learning**
   This approach, in opposite to the previous, begins with a large amount of simple local models and merge them together to get simpler decomposition.

It's important to emphasize that all approaches mentioned so far were strongly related to the control problems and usually utilized in the design of nonlinear controllers. Symbolic regression problems, in contrast with control systems, have hardly any *a priori* knowledge about the problem. Where in control systems the problem is described and limited by physical laws (which can also be very hard to model), the symbolic regression on discrete data does not have any objective measure, which could evaluate the complexity of the original function. This also limits the usage of the mentioned principles in symbolic regression, because most of the approaches require some criterion to evaluate the performance of the regimes. E.g. in case of clustering, the input sample would be split into clusters, but their performance could be evaluated only after approximating the local models, which is not feasible in the means of computational complexity. The regime decomposition in symbolic regression thus requires some information about the complexity of the original function, which could be utilized to split the input sample into regions, where original function behaves less complex.

3.3 Multi-Modal Symbolic Regression

Lipson and Ly algorithm [17] constructs a nonlinear symbolic representation of a discrete dynamic multi-modal system, using an unlabeled time-series data. The algorithm is divided into 2 sub-algorithms which are called **Clustered Symbolic Regression** and **Transition modeling**. The problem, which it solves can be formally described as

$$m_n = T(m_{n-1}, u_n)$$

$$y_n = F(m_n, u_n) = \begin{cases} F_1(u_n) & if \;\; m_n = 1 \\ \vdots & \\ F_K(u_n) & if \;\; m_n = K \end{cases}$$

where u_n is an input vector at time n, y_n is the output vector and $m_n = \{1, .., K\}$ is a current mode of the system at time n. $F_{1..K}$ are the piecewise functions that describe a local behavior of the global function. Function $T(m_{n-1}, u_n)$ is a transition function, which defines conditions for transitioning

between 2 system modes. It depends on the mode in which system was in previous time step and also current input vector. The current mode of the system can also be understood as an analogy to operating regime or subarea, in which input vector lies.

3.3.1 Clustered Symbolic Regression

The role of the first sub-algorithm **Clustered Symbolic Regression**(CSR) is not only to find symbolic representations of the piecewise functions provided a discrete dataset, but also to find parameters of membership functions, which later determine the membership of input points to these piecewise functions. The algorithm utilizes the generalized EM algorithm. The expectation step takes current symbolic representations of all piecewise functions and estimates the membership of the training input-output pairs to the piecewise functions $F_{1..K}$. The maximization step then adjusts current symbolic representations so that they best fit the training samples, according to their membership to piecewise functions.

3.3.2 Transition Modeling

It is a supervised learning technique, which determines a symbolic representations of transition events. Informally speaking, the algorithm tries to find a set of expressions, which describes when a system moves from one mode (operating regime) to another. The resulting dynamical system (from both algorithms) can be described as tuple

$$H =< W, M, F, T >$$

, where W is input data, M represents operation modes, F set of piecewise functions (local behaviors) and T set of transitions.

In the meaning of this tuple, the transition modeling algorithm takes W, M, F and finds a symbolic representation of T, which describes the transition system between modes $m_n \in M$.

3.3.3 Problems in MMSR

Even though this algorithm generates a piecewise solution of even more complex problems, it still does not consider the complexity of the original function. There exist problems, where maintaining a shape of the original function is more important than a numeric metric (like mean absolute error), e.g. we need to preserve gradients of the original functions. In this case, common metrics (MSE, MAE etc.) or criteria (AIC, BIC etc.) might not be relevant for splitting the original functions into regions, and some other mechanism for splitting the original function surface into simpler shapes may be helpful.

Another principal insufficiency of this approach is a requirement for a time series data. This requirement is needed especially in a Transitioning system,

which depends on the system mode in previous time step. This, unfortunately, limits the domain of application of this method. Nevertheless, the idea of evolving a symbolic transition system can be generalized for any dataset.

3.4 Summary

The main drawback of the Local Model Methods (LMMs) described in this chapter is their orientation primary on control tasks. This usually means, that there is a requirement for a priori knowledge of the data or an ordered dataset, like the time series data in MMSR. The requirements unfortunately limit the application of these methods generally in SR problems. The LMMs, however, contains few principles, that can be generalized under some conditions. An example is a transitioning system designed in MMSR. This system is a motivation for designing an evolutionary clustering approach, proposed in next section, which divides the dataset into regions according to the complexity of the dataset.

Chapter 4

Proposed Method

Preliminary experiments with standard GP methods for SR showed, that finding a sufficient symbolic model with minimal error is strongly dependent on the complexity of input dataset. Datasets sampled from functions with complex surfaces, containing different kinds of *artifacts* are almost impossible to fit with a precise model using GP approaches described in Chapter 2. Having the dataset sampled from some function surface, such *artifacts* are areas represented for example by jagged slopes, very narrow ridges or just a plateau surrounded by steep slopes. This problem could be partially overcome cutting such problematic areas out of the dataset and fitting them separately. As the right approach could serve the Local Model Methods(LMN), which are well-known approach from control systems area. The disadvantage with the LMNs is however their strong orientation on control tasks, which make them hardly applicable in the area of SR. The LMNs usually required *a priori* knowledge about the problems or were not feasible for application due to their computational complexity.

I propose a method which enhances the original GP approach in SR with a utilization of some principles of local models methods. In the beginning, it's important to mention that the method requires that the finite training dataset is topologically organized into a regular rectangular grid. This means that the training points are equidistantly sampled along all input dimensions. The method is not limited in the number of input dimensions, although the descriptions and examples are mostly presented in 2D.

The method divides the training dataset into areas which are likely to be less complex and fit the symbolic models on these piecewise datasets. Supposing that the divided regions are less complex, the fitting process should also produce more precise models. Fitting the separated areas containing different kinds of artifacts may also be useful in interpretation and understanding of these artifacts.

Figure 4.1 outlines the basic block scheme of the proposed method. It is distributed into 3 main steps. The first one, clustering, divides the training dataset into smaller subareas (partial datasets), which could be potentially easier to fit. The learning step then takes the individual partial datasets and fits a symbolic model on them. During the learning phase, some of the partial datasets may also be joined together, if the compound model of these partial

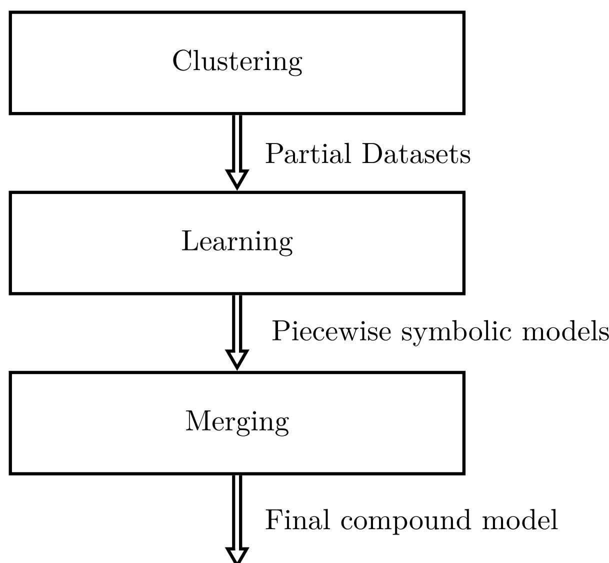


Figure 4.1: Block scheme of the proposed method - The dataset is divided in the clustering step. After then, piecewise models are learned on divided dataset. In the merging step, piecewise models are joined together to form a single model.

datasets can be fitted with a reasonable error. If the two partial datasets are joined together, they form a bigger partial dataset, which contains data points from both of these partial datasets. The compound model is then fitting this bigger partial dataset. The merging step then takes all of the learned models and join them, so that they behave as a single model. The steps of this method are described in following sections of this chapter.

4.1 Clustering

In the beginning, I consider necessary to define few terms, which may have various meanings and will often be used in the following text. The first important term, which has already been mentioned is *training dataset* or also *original dataset*, where both terms have equal meaning and are usually denoted as D. A $n-$dimensional *training dataset* is a predefined set of training points in form

$$D = \{(x_1, y_1), \dots, (x_i, y_i), \dots, (x_m, y_m)\}$$

where $x_i \in R^n$ is an input vector in form $< x_{i_1}, \dots, x_{i_n} >$ and $y_i \in R$ is a target value. As mentioned in the introduction of this chapter, training points are organized into a regular rectangular grid, what is also a requirement for this method.

During the clustering phase of the algorithm, the training dataset is divided into several subsets - clusters $c_1, \dots, c_k$, which cover areas that might be easier to fit rather than fitting whole dataset at once. These clusters are subsets of training points from training dataset D, such that each point belongs to one

and only one subset c_i and $\bigcup_{i=1}^{k} c_i = D$.

Regarding the training points, it also necessary to define term **neighborhood** of a training point, whereas this method usually works with an *8-neighborhood* also called a *Moore neighborhood* and a grid neighborhood. Having the training points organized into regular grid, a *8-neighborhood* is set of all points, which have *Chebyshev distance* to the point equal 1. The grid neighborhood, which is used in merging methods, is described later and example can be seen in Figure 4.8. For the purposes of clustering phase, we will consider only *8-neighborhood*.

As mentioned in the previous text, I described a complexity of the dataset as the main problem of GP in SR and the main motivation for designing this method. In order to find areas, which might be less complex, some objective complexity measure has to be defined. As far as there does not exist any conventional approach to evaluate the complexity of a dataset, I decided to utilize a Gradient Vector Field (GVF) for dataset dividing purposes. As known, GVF tells us how the function surface changes in every direction and is calculated as partial derivate along each dimension. This also holds for discrete datasets, where such numerical gradient in 2D training dataset is calculated according to following formula

$$G_{i,j} = 0.5(D_{i,j+1} - D_{i,j-1})$$

Having a training point on i-th row and j-th column of 2D training dataset organized into a rectangular grid, numerical gradient along the second axis is calculated as a difference between output values of the training points lying on the left and right of the current position. Using this formula, we can calculate gradients for every training point in dataset and get the GVF. Figure 4.2b shows an example of GVF calculated from the dataset sampled from function in Figure4.2a . As we can see, points on the same slopes have similar direction of the gradient. This also holds for points which lie on same plateaus. In contrast, points in more complex areas such as areas with local extrema have usually colliding directions of gradients. The idea of first 2 clustering approaches, described in this section uses this fact, and groups neighboring points of similar gradient attributes like its direction or size, in case of plateau, where gradient is equal 0.

In next sections, I will describe and propose 3 types of clustering approaches used to divide the training dataset into subsets, which cover an area that could be easier to fit. The first one is called spectral clustering and is based on using well-known k-means algorithm in space of first **K** eigenvectors of the similarity matrix.

The second clustering approach is more straightforward and tries to search for connected components in the similarity graph, which is constructed from similarity matrix.

The last approach utilizes evolutionary search using SNGP [7], which tries to find a combination of nonlinear functions, which are used as a decision boundaries to separate training points into distinct clusters.

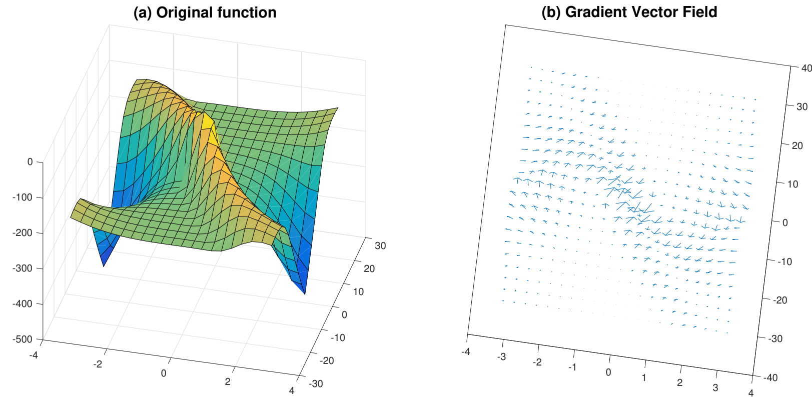


Figure 4.2: Example of GVF. In the left image, there is a surface of the original function, from which was training dataset sampled.

4.1.1 Spectral clustering

As mentioned, the idea is to split the dataset according to the GVF into subsets, where training points in the same subset have similar gradient. I assume that continuous area covered by training points with similar gradient, might be easier to fit. The GVF could be translated into some similarity graph, where nodes represent training points and edges represent similarities between nodes. More specifically, weights of edges can numerically represent similarity between 2 nodes. Such a graph can then be cut into several connected components, so that nodes in the graph have high similarity. One approach for finding such cuts and also components is called Spectral Clustering[18]. The only problem, which needs to be solved is a translation of the GVF into graph structure. To achieve this, I create a similarity matrix storing angles between points, which serve as a similarity measure. Process of creating such similarity matrix is described in next paragraph.

The main advantage of the Spectral Clustering over other clustering approaches, is that it does not take any assumptions on the form or shape of the clusters. As opposed to classic K-means algorithm, where clusters form a convex sets, Spectral Clustering should be able to solve very general problems such as Intertwined spirals. According to [18] the normalized spectral clustering algorithm can be described by the following algorithm

Algorithm 1 Spectral Clustering Algorithm

Input: Similarity matrix S, K number of clusters
Output: Set of clusters $C = \{c_1, c_2, ..., c_K\}$, where c_i is set of points from S
1: Construct similarity graph G from S
2: Compute (normalized) Laplacian L_{sym}
3: Compute **first k** eigenvectors of L_{sym}
4: Cluster the points in the space of eigenvectors using k-means algorithm
5: **return** C

Similarity matrix and graph

Given the dataset with m points, we can compute GVF using numerical formula mentioned before. For *n-dimensional* input space, we get n components of gradient vector for each point. The similarity between 2 points (or nodes) is then calculated as an angle between these gradient vectors in these 2 points. Having the m training points forming a rectangular grid, a similarity matrix S, will be a matrix of $m \times m$ points, where:

$$S_{ij} = \begin{cases} 0 \quad if \;\; j \textbf{ is not in 8-neighbourhood of } i \\ 180 - \alpha_{g_i,g_j} \quad otherwise \end{cases}$$

where S_{ij} is an element of *similarity matrix* on i-th row and j-th column, α_{g_i,g_j} is an angle between gradients in points i and j, calculated using a dot product of vectors. Subtracting the angle from 180 means that similar neighboring gradients have high similarity.

The similarity graph is constructed from a similarity matrix, so that each point in dataset corresponds to individual node in a graph. Nodes i and j are connected, if value in a similarity matrix $W_{ij} > threshold$, where the threshold parameter is arbitrarily selected. An example of such similarity graph can be seen in Fig. 4.3a, which correspond to function surface displayed in Figure 4.2a.

First K eigenvectors

The main idea of spectral clustering is to utilize eigenvectors and eigenvalues (spectrum) of a similarity matrix to reduce the input space dimension and cluster the dataset in the reduced space. The eigenvectors and eigenvalues can be calculated using similarity matrix and graph Laplacian. Selecting the proper **K** for a number of clusters is a tricky task and there were proposed several methods to solve this problem such as methods based on stability approaches [19], information-theoretic perspective [20] or eigengap heuristic [18]. The proper number of clusters can also be estimated visually, from the shape of a function, in case of space with lower dimension.

4.1.2 Simple Clustering

Having the similarity graph created in the same way as in Section 4.1.1, this approach searches the whole graph for connected components using a Breadth-first search (BFS). The algorithm looks for nodes with highest degree (connectivity), which are perspective to be a part of bigger connected component (line 4 in Algorithm 2). The algorithm then recursively traverses all nodes (in similarity graph) that are connected to the selected node and adds them to the same cluster (lines 6-12). When there is no other nodes connected to currently expanding cluster, the search is restarted, nodes in cluster removed from a graph and a new empty cluster is initialized. When $K-1$ clusters are created, the remaining nodes are all put into the last cluster (lines 15-17), which ensures that all nodes are assigned to cluster. The output

of the algorithm is set of at most (K-1) connected components in similarity graph and single cluster merging isolated areas.

Algorithm 2 Simple Clustering Algorithm

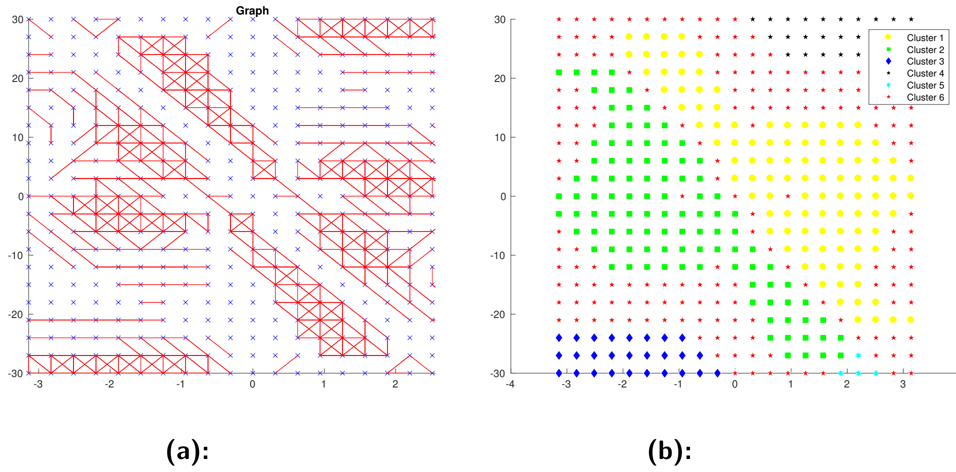
Input: Similarity graph G, K number of clusters
Output: Set of clusters $C = \{c_1, c_2, ..., c_K\}$, where c_i is set of points from G
Initialization :
1: $C = \emptyset$
2: Queue $Q = \emptyset$
BFS Search
3: **while** $C.size \leq (K-1)$ and G is not empty **do**
4: Find node with highest degree in G and add to Q
5: currCluster = $\emptyset$
6: **while** Q is not empty **do**
7: current = $Q.dequeue()$
8: currCluster.add(current)
9: N = neighbors(current) - neighbors according to G
10: $Q.enqueue(N)$
11: Remove current from G
12: **end while**
13: $C.add$(currCluster)
14: **end while**
15: **if** G is not empty **then**
16: $C.add(G)$
17: **end if**
18: **return** C

The outcome of the procedure applied on the similarity graph in Fig.4.3a can be seen in Fig.4.3b. It is important to add, that similarity graph and also resulting clustering is thus strongly dependent on the choice of the threshold value of an angle between 2 gradient vectors.

4.1.3 Evolutionary Hierarchical Clustering (EHC)

This evolutionary clustering approach does not only divide the data into separate clusters, but also constructs a splitting function, or better said a combination of functions, which split the input space into smaller subareas, where the fitting of the original function should be easier. This approach is analogical to [13], where authors look for hierarchical decomposition of the input space into operating regimes. The difference between our approach and [13] is in a usage of nonlinear function (such as Gaussian, Hyperbolic Tangent etc.) used to find a boundary between clusters.

The idea of this algorithm is also captured in Figures 4.4 and 4.6. In first figure, we can see a dataset organized into a rectangular grid and functions $f1, f2, f3$ which are dividing or splitting functions mentioned also in Figure 4.6. Original dataset is first divided by function $f1$ into positive and negative



(a): **(b):**

Figure 4.3: **(a)**Similarity graph based on angles between gradient vectors. The training dataset and gradient vector field is the same as in Figure 4.2.
(b) Outcome of simple clustering algorithm.

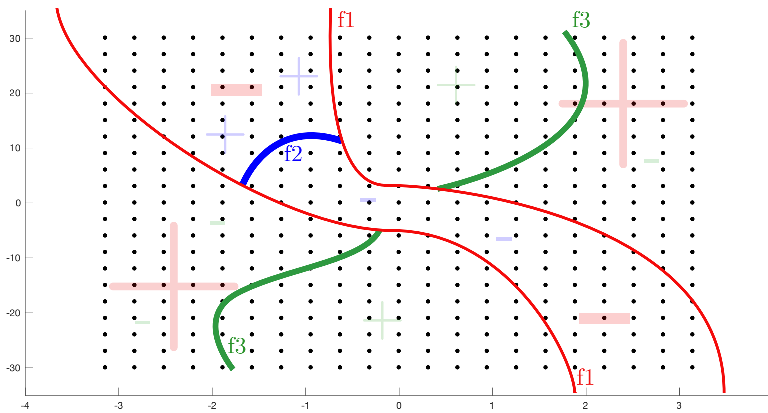


Figure 4.4: This figure visually captures the idea of EHC algorithm. Training dataset is hierarchically split into multiple partial datasets. Splitting function f1 divides the dataset into positive (marked with red +) and negative (-) partial datasets. Functions f2 and f3 does the same recursively on partial datasets from f1.

areas marked with plus and minus signs respectively. These parts are next divided by functions $f1$ and $f2$, which split the dataset into smaller partial datasets. These partial datasets represent clusters in final.

In order to find such functions, I utilized a recent genetic programming approach called Single Node Genetic Programming (SNGP) [7, 8] also described in Section 2.3. The main idea of the evolutionary cycle in this approach is to evolve new population of candidate functions, split the dataset according to these functions and try to fit both parts separately, using only simple operators (features) like plus and multiplication. If the error on the divided

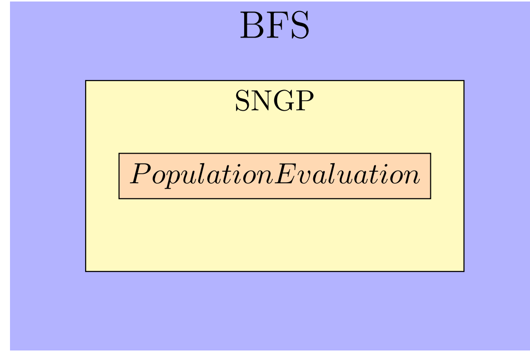


Figure 4.5: Scheme of the EHC algorithm. The BFS search models generated by SNGP. SNGP uses custom Population Evaluation method to evaluate candidate solutions.

datasets is smaller than error on complete dataset, the splitting function is accepted. A lesser error using only primitive operators also means, that the shape of divided surfaces should be less complex than the original one and may be easier to fit.

However, this basically means running 2 nested instances of genetic programming at once. The first one would search for splitting function and the second one would be used for evaluation of each splitting function in population. This approach is too computationally expensive, and I decided to simplify an evaluation step a bit. This simplification is described later in this chapter.

Because of the complexity, it would be problematic and confusing to describe whole algorithm in a single block. Due to this, Figure 4.5 displays the organization of the algorithm in a block scheme. At the top, there is a breadth-first search (BFS) utilized, which hierarchically distribute dividing functions and training dataset. Each step of the BFS contains a single SNGP run, which evolves new populations of dividing functions and is described in detail in Section 2.3. The only difference from original SNGP approach here is a custom fitness function, which is described by Population Evaluation step. In this step, instead of directly calculating fitness of the nodes, training dataset is divided according to dividing function encoded in the node and divided parts are fitted using previously mentioned LASSO regression. The precision of this fit estimates the complexity of partial datasets and evaluates fitness of the dividing function.

Following paragraphs describe subtasks of this algorithm.

BFS

The idea of this subtask is simple, we want to find a hierarchically ordered functions $f_1, ..., f_n$, such that f_1 will split input space into 2 not necessarily connected areas called positive and negative areas. Function f_2 and f_3 then recursively splits the positive and negative areas, respectively, into 2 subareas. The idea is illustrated in Fig. 4.6. The search continues until the desired

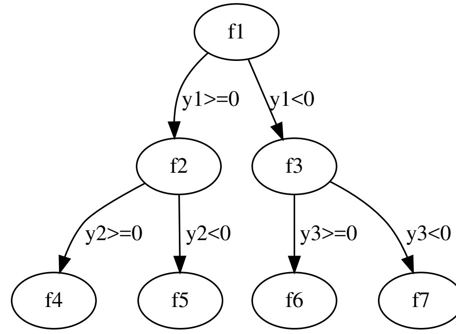


Figure 4.6: Hierarchical function distribution

number of splitting functions, or the number of clusters is found, which is equal to the number of leaf nodes in a search tree.

Algorithm 3 EHC-BFS

Input: Dataset D, maximal level M
Output: Hierarchically distributed functions $\{f_1, ..., f_n\}$

Initialization

1: i = 1
2: Queue = ∅
3: root = new SNGP population with complete dataset D
4: Queue.enqueue(root)

Search

5: **while** Queue is not empty and current level $< M$ **do**
6: current = Queue.dequeue()
7: f_i = current.solve() - complete SNGP search
8: D_{pos}, D_{neg} - split current dataset according to evolved function f_i
9: Create new SNGP nodes where $node_{pos}$ receives only D_{pos} and $node_{neg}$ only D_{neg}
10: Queue.enqueue($node_{pos}$, $node_{neg}$)
11: i += 1
12: **end while**

Algorithm 3 (EHC-BFS) thus starts the search in a root node corresponding to function f_1. The training dataset is then split into 2 parts corresponding to positive ($f_1(X) >= 0$)) and negative (($f_1(X) < 0$)) examples on line 8. After that, new search node are created and initialized with only the positive, or negative dataset part respectively (lines 9,10). The search process continues as standard BFS search until all leaf nodes at level M are created. Looking at the Fig. 4.6, we can see 4 splitting functions at the last level of the tree. Each of these functions also splits the corresponding partial dataset, that was propagated to it from previous nodes, so at final, we get 8 partial datasets corresponding to 8 clusters.

Population Evaluation

The main idea behind clustering in our approach is to split the training dataset and its surface into separate areas, which could be easier to fit using symbolic regression. In the previous clustering approaches, I utilized function GVF and separated areas with colliding vectors. In evolutionary clustering approach, I try to fit the divided areas with simple features, which usually have simpler surface. Finding such combination of features could be done using SNGP, but it is too complex. Because of this complexity, the population evaluation is limited to search only a small set of linear and quadratic features, which in 2-dimensional input space means $\{x_1, x_2, x_1x_2, x_1^2, x_2^2\}$. More formally, we are looking for a function

$$g(x_1, x_2) = a_1x_1 + a_2x_2 + a_3x_1x_2 + a_4x_1^2 + a_5x_2^2 + a_0$$

and the task is to estimate the coefficients $a_1, .., a_n$ such that MSE of this function on current part of dataset is minimal. In order to find such coefficients, I utilize a Lasso Regression search [5].

To evaluate a population of SNGP splitting functions, we thus first split the dataset into positive and negative regions according to each candidate function, where output of the function is $f_i(x_1, x_2) >= 0$ in positive region. We estimate parameters of function g on both parts of dataset and a final fitness is calculated according to

$$fitness = \frac{E^2_{positive} + E^2_{negative}}{|dataset|}$$

, which is basically the MSE of the dataset.

4.2 Hierarchical learning

The first principal problem with clustering is a selection of the proper number of clusters, which is closely related to solved problem. There are some approaches to estimate the optimal number of clusters, like eigengap heuristic [18], but these are hardly applicable in this problem. I thus decided to try another approach, that may start with a relatively higher amount of smaller clusters and merge compatible clusters together, if performance of the merged supercluster is not worse than the mean performance of single clusters alone. This approach is described by Algorithm 4 (Hierarchical Learning).

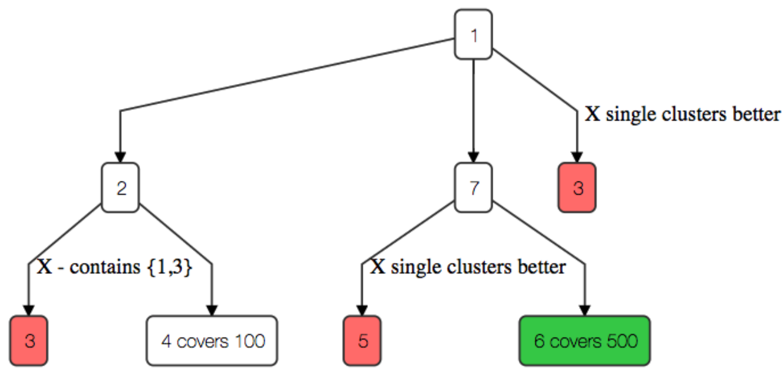


Figure 4.7: Example of simple neighborhood search tree

Algorithm 4 Hierarchical Learning

Input: Set of clusters $C_{start} = \{c_1, c_2, ..., c_K\}$, Maximum supercluster size max_depth

Output: Set of clusters and superclusters $C_{end} = \{c_1, c_2, ..., c_R\}$, Set of models $M = \{m_1, m_2, ..., m_R\}$

Initialization :

1: $C_{end} = \emptyset$
2: Queue $Q = \emptyset$
3: TABU$_{global} = \emptyset$
Preparation
4: Train models for all simple clusters $c_i \in C_{start}$
BFS Search
5: **while** C_{start} is not empty **do**
6: root = Random starting cluster $c_i \in C_{start}$
7: winner = root
8: $Q.enqueue$(root)
9: **while** Q is not empty **do**
10: current = $Q.dequeue()$
11: N = neighbors(current) - neighboring clusters to current, such that each of neighbors $c_i \in C_{start}$, and $current \cup \{n_i \in N\} \notin TABU_{global}$
12: **for all** $n_i \in N$ **do**
13: Create joint cluster C_{joint} of all simple clusters $\{c_i \in current\} \cup n_i$
14: model = train joint model for C_{joint}
15: **if** $mse_{model} <= (1 + \epsilon)\frac{\sum_{c_i \in C_{joint}} mse_{model_{ci}} |c_i|}{|C_{joint}|}$ **then**
16: $Q.enqueue(C_{joint})$
17: **if** $|C_{joint}| > |winner|$ **then**
18: winner = C_{joint}
19: **end if**
20: **else**
21: TABU$_{global}$.add(C_{joint})
22: **end if**
23: **end for**
24: **end while**
25: $C_{end}.add$(winner)
26: C_{start}.remove($winner$) - winner is a superset of simple clusters
27: **end while**
28: **return** C_{end}

On the input of the algorithm, we have K clusters, represented by partial dataset files created from original dataset, and also selected limit for a depth of a search tree. The output of the algorithm will then contain an assignment of input clusters to superclusters, which in general can contain from one to K input clusters. In addition, each of these superclusters has a learned symbolic model, evolved using SNGP (Section 2.3). Lines 4 in the

algorithm 4 is important because of the comparison criteria on line 15, where we compare MSE of currently evolved supercluster and the MSE of single clusters contained in the supercluster, but trained separately. There is also some little threshold value ϵ, accepting also not strictly better superclusters. To save resources, we thus precalculate the errors on the single clusters in the initialization phase. The loop starting on line 9 is a BFS search traversing a tree such as the one in Fig. 4.7. The most important part of the algorithm takes part on lines 11-21. For a currently selected cluster or supercluster, we create a new node for each neighboring clusters, which corresponds to the union of the current node and a neighbor. After that, a symbolic model is trained for every new node, which does not contain a combination of clusters contained in a global TABU list. The global TABU list important, because it allows us to prune not perspective combinations of clusters early, without actually learning them, what saves computational resources.

When the model for a supercluster is trained, its performance is compared to the performance of a model created of separately trained models on single clusters, and if it's worse, corresponding cluster combination is appended to the global TABU list. The best supercluster is then not selected according to its MSE, which must be better than single clusters, but according to the size of a dataset which it covers.

When a winning supercluster is found, or a search tree depth limit is reached, clusters (and also data points) contained in supercluster are removed from the search, and it's restarted. The search process continues until no cluster is left in input cluster set. A simple example is seen in Fig. 4.7.

4.3 Piecewise models merging

Suppose we have training data distributed into clusters (or superclusters) and each cluster has learned corresponding symbolic model using SNGP. The task now is to merge models together, so that they work as a single model. The first appearing problem seems to be a classification of unseen data and assignment of models, which evaluate it. Another problem seems to be data points which lie on the borders of 2 or more cluster (models). In this situation, we cannot simply pick a single model randomly, because it would cause a discontinuous transition on borders. This problem can be partially overcome including these bordering areas in training datasets, but this solution may not work perfectly. Following sections propose and discuss some approaches which are suitable for solving these issues.

4.3.1 Labeling unseen data

As mentioned, there is a principal problem with assigning labels to unseen data. For example, in Section 3.3.1, authors of CSR utilize the transitioning system, which detects when time series input switches between 2 clusters (models). Such a transitioning system would be unfortunately hardly applicable with multidimensional or unordered data. I've partially overcome this issue in

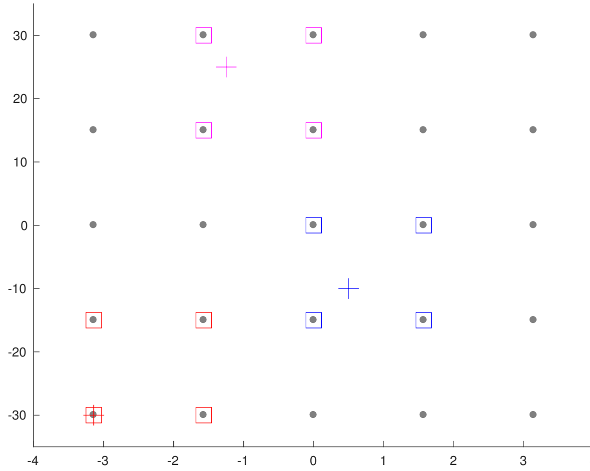


Figure 4.8: Example of 4-nearest grid neighbors

Section 4.1.3, where the symbolic representation of splitting functions are evolved. The principle is a generalization of the CSR transitioning system. Classification of unseen data can be then found analogically to splitting the dataset with EHC.

Despite that, there is still need for a solution because of spectral a simple clustering.

As the most straightforward approach, we can utilize slightly modified K-nearest neighbors search. Having the training data forming a grid in a 2D space, we are not looking for 4-nearest neighbors in the means of the distance function, but 4-nearest points enclosing a grid field, in which the point lies. The idea is visualized in Fig. 4.8, where the + sign is a position of input point and same color squares are nearest neighbors. The distance between 2 neighboring points in the grid is normalized to value 1. The input point is then classified as the majority of its neighbors. In case 2 neighbors are from the first cluster and the other two from the second, final cluster is assigned randomly between the two.

4.3.2 Merging method 1

The main idea of the first merging approach is to evaluate input points lying in bordering areas (between cluster) proportionally, according to the precision (error) of bordering models. This method requires calculating the errors on each of the training points in advance.

The input of the algorithm contains the training dataset points, trained models, errors of these models and the input which need to be evaluated. The training dataset points are necessary to calculate nearest neighbors and assign corresponding models. In the initialization phase of the algorithm on line 1 of Algorithm 5, the errors are normalized, so that they fulfill *partition of unity* property. These normalized errors are then converted to precisions and used

for a weighting output of each model to which input point corresponds. It's also important to mention, that error, normalized error and the precision of model, to which the input point does not belong **should be** 0. The conversion from error to precision means, that models with higher error should have lower precision and vice versa.

Line 4 of the algorithm utilize precision on the nearest training point for output weighting. It can be easily extended to utilize precisions of multiple nearest neighbors, so that we average precisions on all of these neighbors and normalize it, so that it satisfies *partition of unity* property.

Algorithm 5 Merging method 1

Input: Training points D size $m \times n$, Trained Models $\{f_1, ..., f_k\}$, Errors err_D size $m \times k$, Input point(s) Inp size $r \times n$
Output: Output Out size $r \times 1$
Initialization
1: normalize and convert err_D to $precision_D$, so that $\sum_{j=1}^{k} precision_{ij} = 1, \text{for } i = 1..m$
Calculate output
2: **for** i = 1 ... r **do**
3: y= empty $1 \times k$ array
4: $prec = precision_i$, where i is nearest neighbor in D
5: **for** j = 1 ... k **do**
6: $y_j = f_i(Inp_i)$
7: **end for**
8: $Out_i = prec \times y$
9: **end for**

Algorithm 5 can be better understood on following example, which is also visualized in Figure. We have random input point (red plus in figure), original training data points (black dots) and errors of the models on these training points. We first get the nearest neighbor(s) of input point in a set of training points, which is marked with a red square in the figure. After that, we calculate output value as a weighted sum of outputs of all models this training point. The weights of the models are calculated so that model that has better precision, has higher value. The figure shows that difference between desired output and output of the first model, on this training point, was 90. The second model had error equal 10. This thus means, that model 2 is more precise and will have 90% contribution in new input point. If nearest training data point does not belong to some *cluster(model) i*, and also does not border with this cluster, the error and also weight corresponding to *model i* is 0. Also node that $\sum_{i=1}^{numClusters} precision_i = 1.0$. Due to this, the precisions will be $precision_{f_1} = 0.1$, $precision_{f_2} = 0.9$.

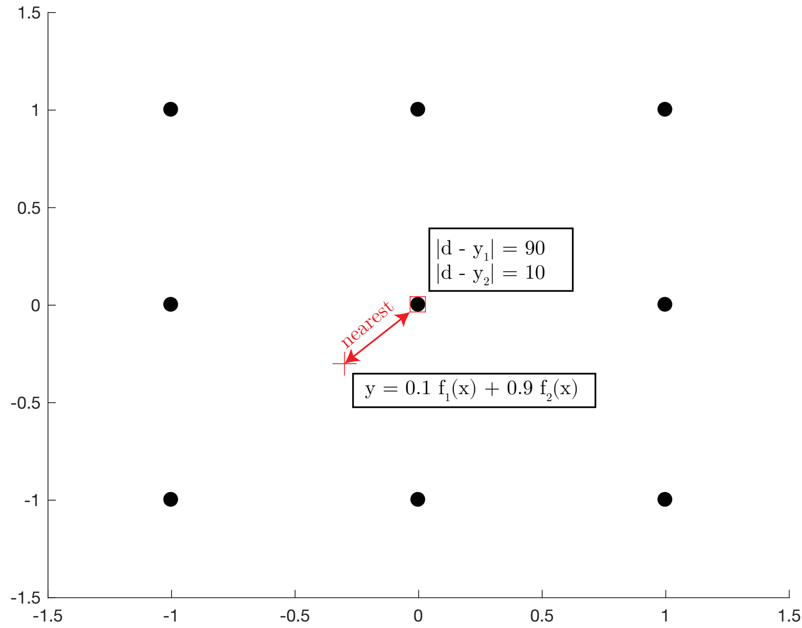


Figure 4.9: Example of merging 1 algorithm. Algorithm first finds the nearest training point and then weighten outputs of all models on current input point according to precision of these models on nearest training point.

4.3.3 Method 2

In this clustering approach, all input points receive only single cluster assignment, and the output is calculated according to the model corresponding to the cluster. The cluster assignment is not dependent on distance to training points, which are already classified, but on their cluster assignment.

The input of the algorithm contains training data points, trained models, input and K, which is selected number of neighbors, included in cluster assignment. The algorithm goes through all input points and for each one finds K nearest neighbors in training dataset, which can be found according to the distance or grid neighbors method mentioned earlier. All of these neighbors already have at least one cluster assignment from preprocessing phase. In case of points lying on borders of 2 clusters, single point is assigned both of these clusters. In final, the algorithm selects the cluster, which has the most occurrences within neighbors and input point is evaluated according to model corresponding to this cluster. In case of 2 clusters, which has the same number of occurrences, the final cluster is picked randomly from these two.

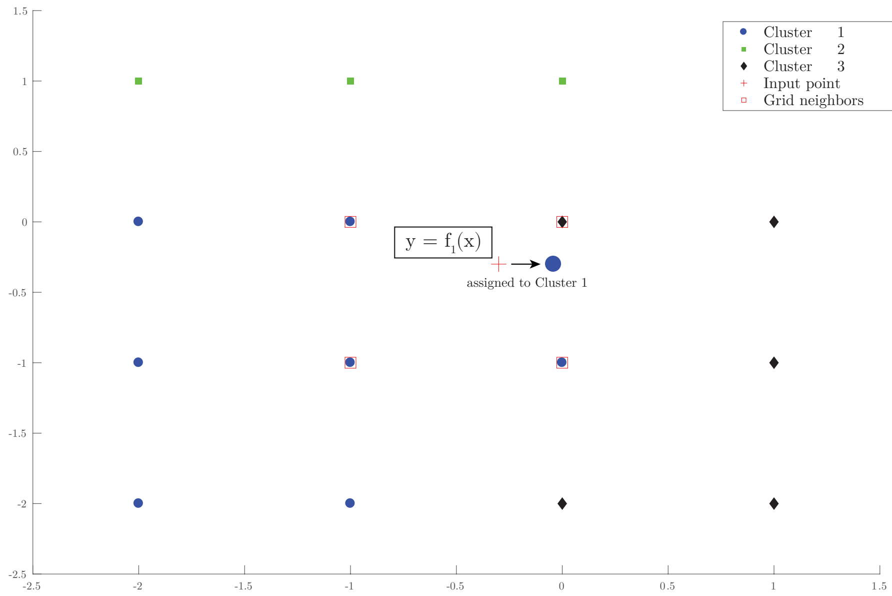


Figure 4.10: Example of merging 2 algorithm. Algorithm first finds grid neighbors of input point and then assigns a cluster, which is dominant among the grid neighbors.

Algorithm 6 Merging method 2

Input: Training points D size $m \times n$, Input point(s) Inp size $r \times n$, K, Trained Models $\{f_1, ..., f_k\}$
Output: Output Out size $r \times 1$
1: **for** i = 1 ... r **do**
2: $neigh$ = get K nearest neighbors of Inp_i in D
3: get cluster assignments of $neigh$
4: $Out_i = f_j(Inp_i)$, so that j is cluster, with max occurrences in $neigh$
5: **end for**

This merging algorithm is also visualized in Figure 4.10. The training points form a square grid and are assigned to clusters, which are visualized by special symbols like dots and diamonds. Next, there is a random input point marked with red plus sign and its grid neighbors, which are marked with red squares. As we can see, blue dots are dominant among neighbors, so input point is assigned to the first cluster. The output is then calculated according to model f_1, which corresponds to cluster 1.

4.3.4 Method 3

The last merging method is based on utilizing standardized euclidean distance as a weighting factor in calculating output from multiple models. The standardized euclidean distance measure normalizes distances between 2 neighboring grid nodes to value 1 in all dimensions. The main idea of the

algorithm in opposite to previous approaches is not to assign a cluster to input point, and calculate output according to specific model, but to make a weighted average of the outputs in neighboring points, so that transitions between 2 models will be more continuous.

On the input, there is again an original training data with already assigned clusters from preprocessing phase, input points and trained models. In the initialization phase, there is a constant $MAXDIST$ for a maximal distance of an input point to any grid neighbor. Because all examples are held in 2D and distances are normalized, this constant is equal to square root of 2, what is the maximal distance in square with unit side length. This constant can however differ in higher dimensions.

There can basically occur 2 situations which are treated separately, in opposite to previous approaches. The first one happens, when all grid neighbors belong to the same cluster. In this situation the input point is evaluated according to the same model as the grid neighbors are. On the other hand, when there are neighbors from different clusters the output is not calculated directly from the input, but from the output of the neighbors as seen on lines 8-13 in Algorithm 7. The output of the neighboring grid nodes is weighted according to the distance of the input point to these neighbors. This should ensure, that final compound model will contain continuous transitions between 2 submodels. There can occur one exception, when the input point lies too close to some grid point (training point). In this situation, when the distance is smaller than some threshold value, the output of the model is calculated directly from the input point according to the cluster assignment of the neighboring point.

In Figures 4.11 and 4.12, you can see visual examples of the two mentioned cases, which can occur. In Figure 4.11, the output is calculated directly from the input, because all neighbors belong to the same cluster. The second case in Figure 4.12 is more complicated. Output is a weighted sum of models, corresponding to cluster assignment of each grid neighbor. Weights in this example are calculated according to following steps:

1. Calculate distances between input point and grid neighbors $dist_{p_1}, \dots, dist_{p_4}$
2. Distances are subtracted from $MAXDIST = \sqrt{2}$, and we get vector with 4 components: $W = [\sqrt{2} - dist_{p_1}, \sqrt{2} - dist_{p_2}, \sqrt{2} - dist_{p_3}, \sqrt{2} - dist_{p_4}]$
3. Vector is normalized $W^i_{norm} = \dfrac{W_i}{\sum_{i=1}^{4} W_i}$, so that $\sum_{i=1}^{4} W^i_{norm} = 1.0$

At the end, outputs of grid neighbors (using corresponding models) are weighted and summed, giving output value to the input point. The output is characterized by equation $y = 0.3f_1(p_1) + 0.3f_2(p_2) + 0.2f_3(p_3) + 0.2f_2(p_4)$ in Figure 4.12, where $f_1, \dots, f_4$ are trained models and $p_1, \dots, p_4$ are grid neighbors from training dataset.

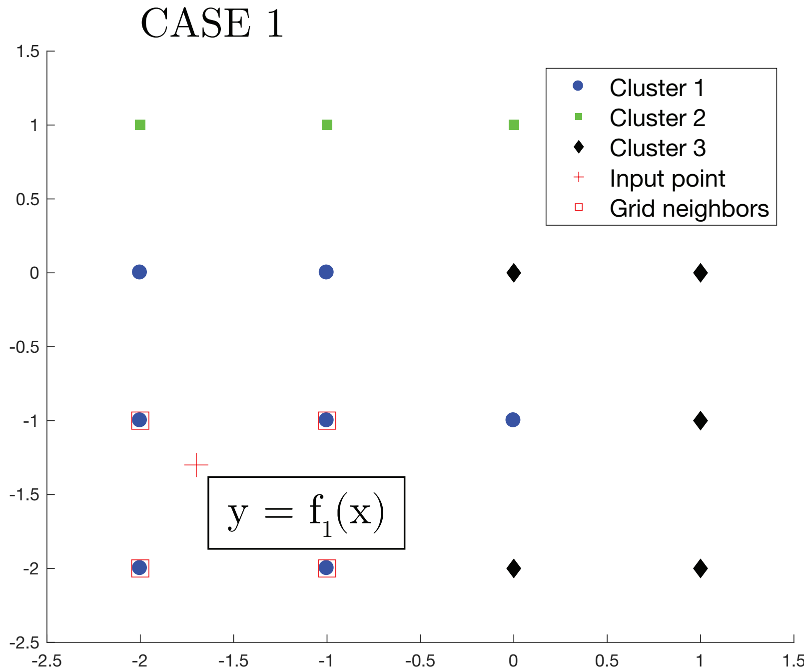


Figure 4.11: Visualization of the first case of Merging 3 algorithm. All grid neighbors belong to the same cluster, co output of input point is calculated according to corresponding model.

Algorithm 7 Merging method 3

Input: Training points D size $m \times n$, Input point(s) Inp size $r \times n$, Trained Models $\{f_1, ..., f_k\}$
Output: Out size $r \times 1$

Initialization

1: W = empty matrix of weights size $rx4$
2: $MAXDIST = \sqrt{2}$
3: **for** i = 1 ... r **do**
4: w - empty vector of weights for each neighbor
5: $neigh, dist$ = get 4 nearest grid neighbors (in 2D) of Inp_i in D and its distance to them
6: **if** all neighbors are from same cluster **then**
7: $Out_i = f_n(Inp_i)$, where f_n is model corresponding to cluster of neighbors
8: **else**
9: y = calculate output on $neigh$ according to their cluster assignment
10: **for** j in $neigh$ **do**
11: $w_j = MAXDIST - dist_j$
12: **end for**
13: normalize w so that $\sum_{i=1}^{4} w_i = 1$
14: $Out_i = y \times w$ - dot product of y and w
15: **end if**
16: **end for**

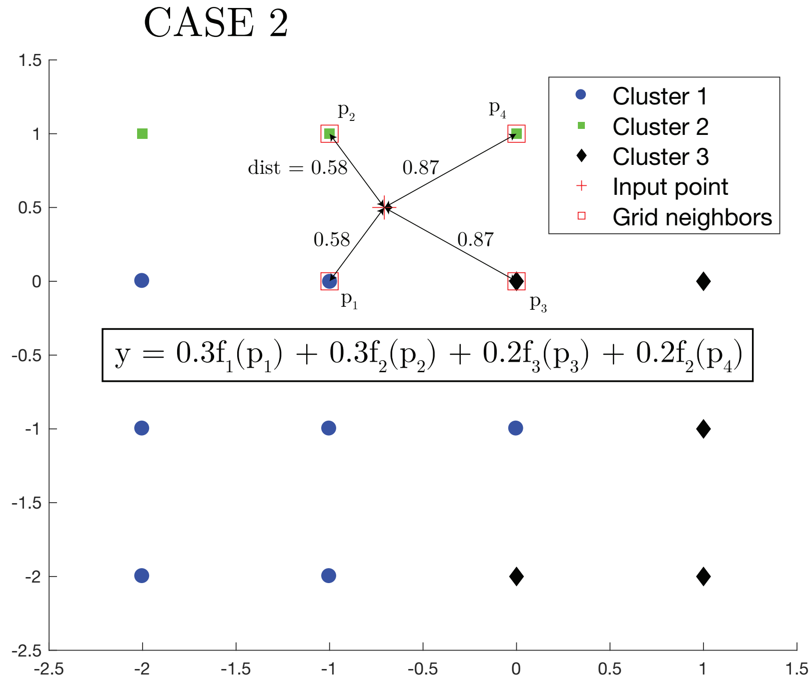


Figure 4.12: In the second case of Merging 3 algorithm, the input point lies on border of multiple clusters. At first, euclidean distances to grid neighbors are calculated. After that, distances are translated to weights and output is calculated as a weighted sum of outputs of grid neighbors.

4.3.5 Model merging in hierarchical models

In merging models from hierarchical learning method mentioned in Section 4.2 the situation is a bit different. All of the previous merging approaches have to be adjusted, so that in the initialization phase of the algorithm, they re-label training data according to hierarchically learned partition. Term partition here means the grouping of the elements (clusters) of some set into non-empty subsets (superclusters), so that each element occurs in one and only one subset. The superclusters are then handled as normal clusters with the learned model.

Chapter 5

Experiments

To test the performance of the proposed method and compare it with existing approach, I have selected few practical and also artificial Symbolic Regression problems. Since the method is more complex, and consists of multiple parts which are clustering, hierarchical learning and merging, I considered important to test these parts separately. Hence I will propose 2 experiments, where I first compare mentioned clustering approaches and then evaluate the performance of the whole method with the state-of-the-art implementation of the Single-run SNGP with LASSO regression.

5.1 Setup

For the experimental purposes, I selected few practical problems related to reinforcement learning and also datasets sampled from artificial benchmark functions. Table 5.1 overviews parameters of the experimental datasets and also sources, from which they were created. First three datasets were generated from reinforcement learning problem, sampling the V function and Policy respectively. Data of Fun1 and Fun2 were sampled from analytic function described by equations 5.1 and 5.2 respectively. All datasets in experiments have 2 input variables. Ranges of the input and output variables are included, to help the reader make a rough estimate of an error magnitude on these datasets.

Note that all datasets are evenly sampled, which means they are organized into a regular rectangular grid, as mentioned in previous chapter. This data organization is required by some of the clustering and merging approaches.

Figure 5.1 shows the 1DOF Swingup and Robot Arm Policy datasets. Figure 5.2 visualizes 1DOF Swingup policy dataset, which is obviously very challenging for original GP approaches due to many surface irregularities it contains. At last, Figure 5.3 visualizes surfaces of custom functions generated by formulas 5.1 and 5.2. I selected these problems, because all of these datasets contains artifacts that are very challenging for common GP approaches. Such artifacts are for example transitions between steep slopes and saturated plateaus as seen in 5.2, or the irregular shapes of plateaus in the same Figure. Robot Arm Policy contains also very challenging artifact in form of jagged slope seen in Figure 5.1, which is almost unsolvable for approaches learning

Name	#Vars	#Samples	X_1 range	X_2	Y
1DOF Swingup	X_1, X_2	81×81	$< -\pi, \pi >$	$< -30, 30 >$	$< -43, 0 >$
1DOF Policy Swingup	X_1, X_2	61×61	$< -\pi, \pi >$	$< -30, 30 >$	$< -2, 2 >$
Robot Arm Policy	X_1, X_2	37×37	$< -\pi, \pi >$	$< -30, 30 >$	$< -10, 10 >$
Function 1	X_1, X_2	31×31	$< 0, 1 >$	$< 0, 1 >$	$< -1, 1 >$
Function 2	X_1, X_2	31×31	$< 0, 1 >$	$< 0, 1 >$	$< -2, 2 >$

Table 5.1: Experimental datasets overview.

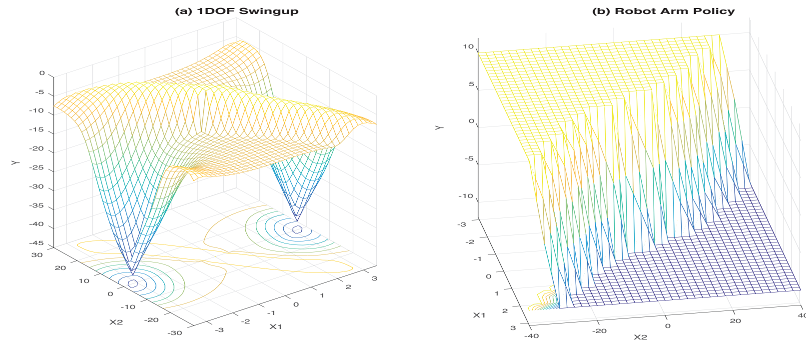


Figure 5.1: Visualization of 1DOF Swingup dataset (a) and Robot Arm Policy dataset (b).

on complete dataset. I assume that HL approach proposed in the previous chapter could be capable of learning these artifacts, because of the learning process, which is focused on local domains.

$$Fun1 = x_1^2 cos(10x_1 - 15x_2) \tag{5.1}$$

$$Fun2 = (x_1-1)^2 cos(10(x_1-1)-15(x_2-1)) - (x_2-1)^2 sin(10(x_1-1)+15(x_2-1)) \tag{5.2}$$

As explained in previous chapter, the proposed method utilizes 2 different instances of SNGP learning. First one in case of Evolutionary Hierarchical Clustering and the second one in case of Hierarchical Learning. Table 5.2 overviews unique properties of both SNGP runs, which are number of independent runs, maximal number of generations within each run and also basic functions sets. Maximal feature size limits each new feature to contain at most 5 basic functions. Properties Predictors and Regressors are described in Section 2.3.5.

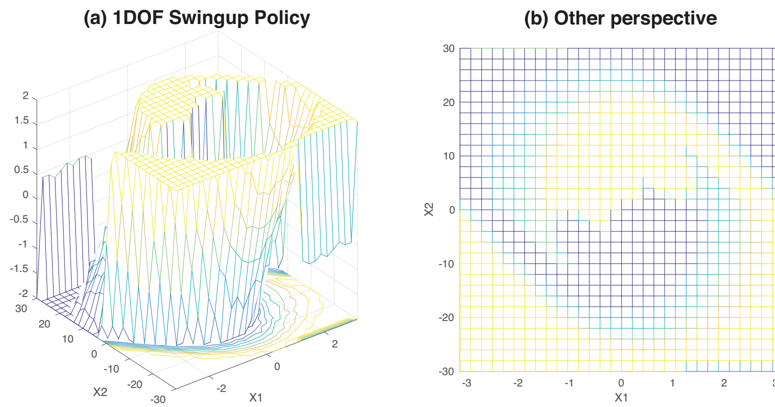


Figure 5.2: Visualization of 1DOF Swingup Policy dataset. Right image provides another perspective, to show irregularities in the dataset surface.

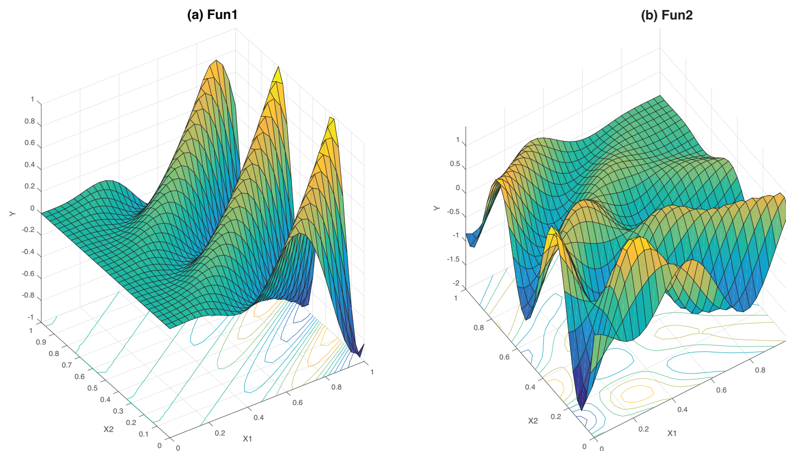


Figure 5.3: Custom analytic functions visualization. Original formulas are described in the text.

5.2 Clustering Experiments

The goal of the experiments with clustering is to compare approaches described in Chapter 4.1. I have performed 10 runs with all clustering approaches. In case of EHC, the setup parameters are shown in Table 5.2. All experimental datasets are clustered into 8 clusters. A higher number of clusters also influence computation complexity of the HL method. Each experimental dataset also requires custom setting of threshold parameter in Simple Clustering. This parameter, as mentioned in method description in Section 4.1.1 is used for construction of similarity graph.

The results showed, that the outcome of simple clustering algorithm is very consistent and results differ only in few data points. The spectral clustering algorithm usually results in smaller clusters as can be seen for example in

	EHC	**HL**
#Runs	10	30
#Generations	15000	50000-10000
Basic functions set	$\{+, -, *, pow2, pow3, pow4, pow5, Bent\}$	$\{+, -, *, pow2, pow3, pow4, Gauss\}$
Max feature size	5	5
Population Size	200	300
#Predictors	50	50
#Regressors	5	5

Table 5.2: SNGP Parameters setup

figure 5.4, but their position differs minimally in distinct runs. The EHC, similarly to Simple clustering usually evolves very similar results in all runs. Figures 5.4, 5.5, 5.6, 5.7 and 5.8 show clustering on the experimental datasets, where the specific run was selected randomly. This because the differences between outputs were not so significant.

As far as there is no objective measure to compare clustering approaches, I only provide visual comparison of these methods. I will return to this topic and analyze these methods in the end of this chapter, because an impact of these methods on the final outcome of the proposed method can be evaluated only after running all phases of the method.

5.3 Experiments with proposed method

In order to compare the performance of the proposed method against standard SNGP approach, I performed 30 independent runs of Hierarchical Learning method on each type of clustering, mentioned previously. To test SNGP, I performed 30 independent runs of the SNGP implementation [9], on complete dataset, which was used for clustering. As described, I have 5 different types of datasets of various properties. All HL instances had limited maximal tree depth to 3. This basically means, that in each run, there could be joined at most 3 simple clusters in a single supercluster. The number of generation also varied, depending on the complexity and size of a dataset. In case of custom functions (Function 1 and 2) there were only 50000 generations. 1 DOF Swingup required 100 000 generations per run. This however does not mean, that method wouldn't produce reasonable results in fewer generations.

After running Hierarchical Learning, I utilized merging approaches from Section 4.3 to create a single model and measured the statistical values on this model. Outcomes of the experiments can be seen in Tables 5.3, 5.4,

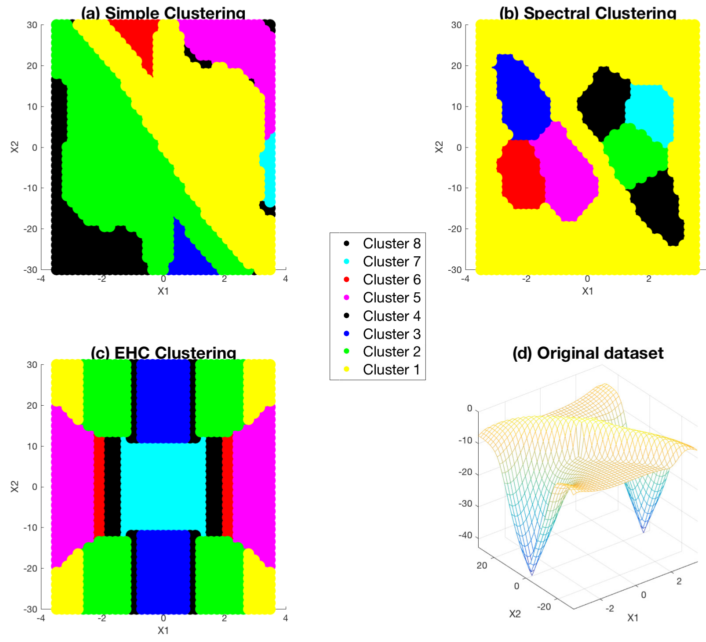


Figure 5.4: Visual comparison of clustering approaches on 1DOF Swingup problem. The threshold parameter for Simple Clustering is set to 172.

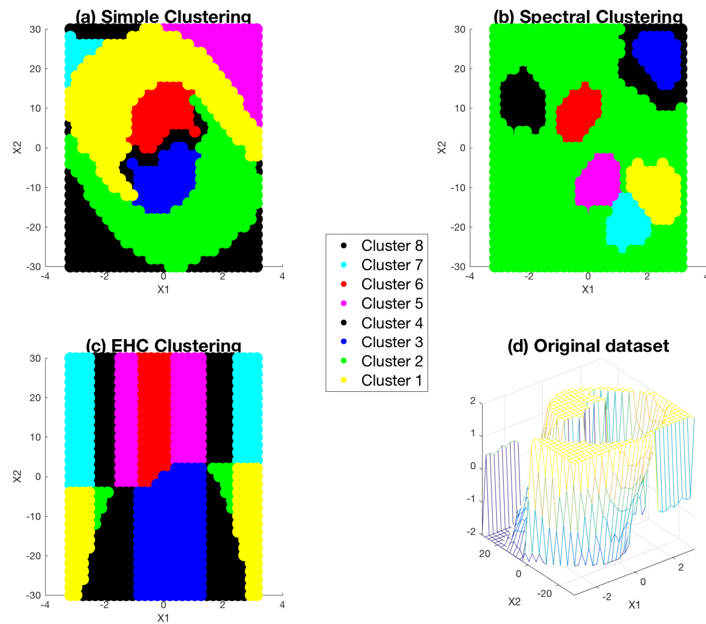


Figure 5.5: Visual comparison of clustering approaches on 1DOF Swingup Policy problem. The threshold parameter for Simple Clustering is set to 110.

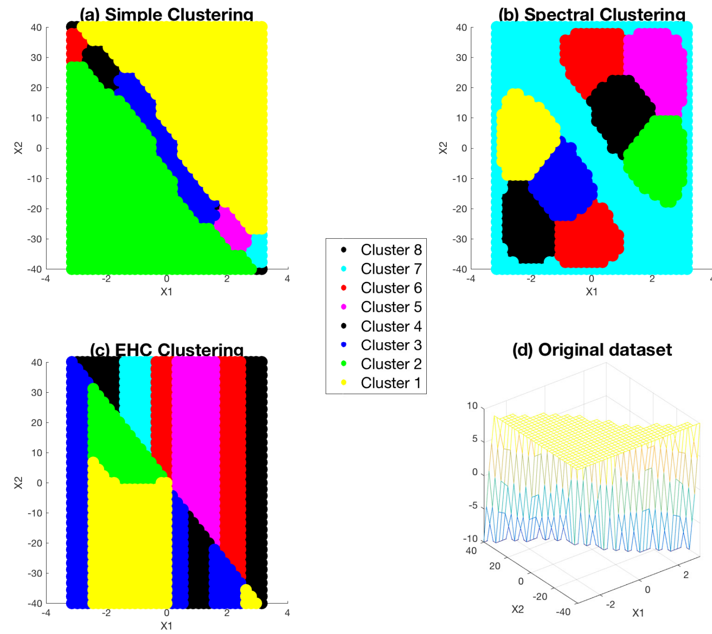


Figure 5.6: Visual comparison of clustering approaches on Robot Arm Policy problem. The threshold parameter for Simple Clustering is set to 175.

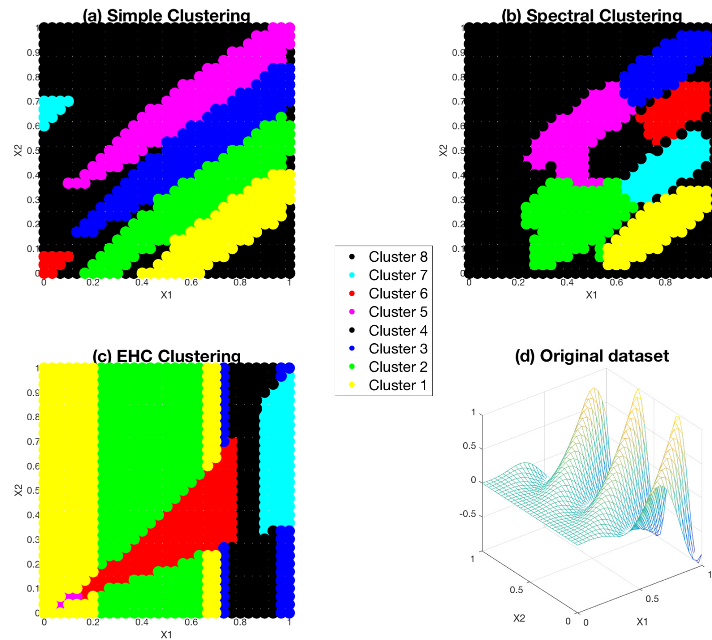


Figure 5.7: Visual comparison of clustering approaches on custom Fun1 problem. The threshold parameter for Simple Clustering is set to 170.

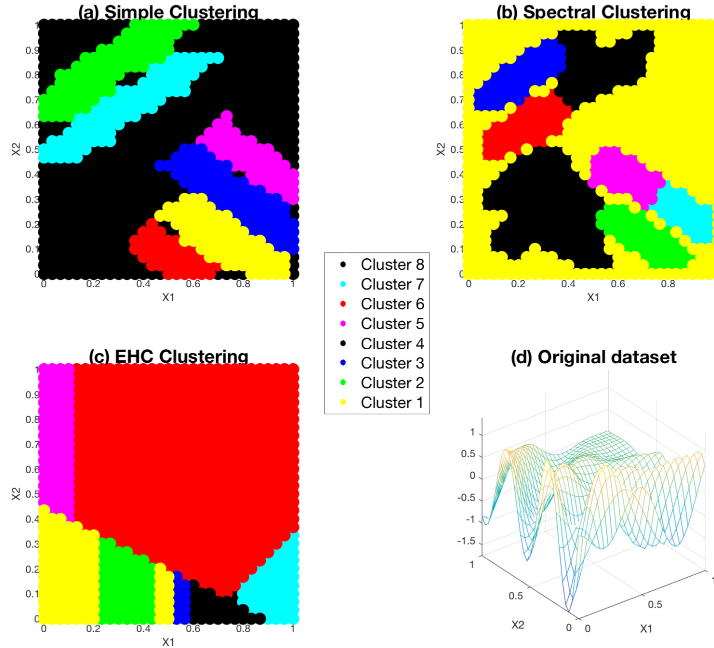


Figure 5.8: Visual comparison of clustering approaches on custom Fun2 problem. The threshold parameter for Simple Clustering is set to 170.

5.5, 5.6 and 5.7. The tables are organized, so that rows present individual merging approaches and columns present clustering methods. Value of the cell corresponds to mean and best MSE, respectively. In case of standard SNGP approach, there is only single value for all 3 merging methods, because the merging step was skipped. In each table, the value written in bold is considered the best. Values marked with * are considered significantly better than values of all other approaches, according to the two-sample t-test with 5% significance level.

To objectively evaluate these runs, I measured mean and minimal values of the Mean Square Error (MSE) on these runs. Figures 5.9, 5.10, 5.11, 5.12 and 5.13 show the visual comparison of the best run of standard SNGP and HL. The best run is selected according to the best combination of clustering and merging approach in results tables. The figures are organized so that Figure a) shows the original dataset, b) shows the model produced by SNGP and c) model produced by HL.

5.3.1 Evaluation and Analysis

As can be seen in tables, proposed method successfully outperform the standard SNGP approach in all experiments. To support this statement, I performed the two-sample t-test between results of the best approach with respect to mean MSE, and all other approaches. In case of Fun2 dataset, the result is not significantly better than the other merging approaches, but EHC

in general performed significantly better than all other approaches. In case of 1 DOF Swingup policy, there is no merging approach better than any other in simple clustering, but again, all merging approaches are better than the SNGP.

The only case, where mean MSE of SNGP wasn't the worst is the experiment with Robot Arm Policy, where first merging approach produced excessively bad results in case of Spectral Clustering algorithm. There has to be considered the fact, that the surface of the dataset (Figure 5.1) is not that complex in the meaning of MSE, which means that the standard SNGP performs well. Closer examination of the surface however reveals several *artifacts*, such as jagged slopes and steep transition between plateau and slope. These artifacts are ignored by standard SNGP. The problem with the combination of Spectral Clustering and Merging 1 technique in case of this dataset is the distribution of clusters, which created simple clusters in the area of plateaus and left the complex parts in single cluster. In addition to this, there occur uneven transitions between the clusters covering plateaus and cluster covering complex area, which result in an additional error.

Despite the Robot Arm Policy dataset, the proposed method was able to beat SNGP in all other instances with all combinations of clustering and merging approaches. One more interesting case occurs in the 1 DOF Swingup Policy, which seemed the most challenging for fitting. As can be seen in Table 5.6, the best approach seems to be Simple clustering, which performs well with any merging method. The MSE of this approach is however only twice as good as the standard SNGP is. Such a result suggest that using standard SNGP might be reasonable in the meaning of computation time and complexity, however visual evaluation of the two approaches displays huge difference. Figure 5.12 displays the best runs of both approaches. As can be seen, standard SNGP succeeded in fitting the basic shape of surface, so that local optima are placed well, however provides much less detail than Hierarchical learning. It also ignores specific artifacts of this dataset, such as slopes, which are not smooth but contains several jags. The standard SNGP also ignores the irregular shape of plateaus.

The same situation can also be seen in Figure 5.13, comparing Robot Arm Policy experiment, which contains artifact in form of jagged slope. The ignorance of such types of artifacts suggests, that standard objective measures like MSE and MAE might be insufficient for fitting complex datasets, due to global evaluation. This means that when an artifact is covered by few samples, or range of its function values is unimportant compared to global range, it is usually ignored. This also means that when one model fits the artifact better but also fits other areas worse than the other model, it is usually rejected. The effect of this problem is the most visible on problems, which require much better detail of fitting.

We can see, that the results (visual and also numerical) are strongly dependent on the selection of the clustering and merging approach. Where in case of merging, the results are usually similar, so that Merging 3 approach usually slightly outperforms Merging 2 and more significantly the Merging 1

approach, the clustering seems to be very problem dependent. Looking at the 1 DOF Swingup experiment, we can see that Simple Clustering is significantly outperformed by EHC. One possible reason for this situation might be the way how Simple Clustering handles local optima and in case of 1 DOF Swingup also a long ridge crossing the surface. Where the EHC always looks for function, which "simplifies" and divides the dataset, Simple Clustering looks for the maximal possible cluster (in number of points), looking only at the gradient map. The problem however occurs, when there is a ridge, which is too narrow, and contains only few data points. This usually causes, that there are 2 different clusters meeting right on the edge of the ridge and also a problematic situation for smooth transitioning between these 2 clusters. Such problematic transition then might lead to a significant increase of the error, which is the case of the 1 DOF Swingup dataset.

Generally speaking, we can say that Simple Clustering approach performed the best (best in 3 of 5 experiments) and is also much more computationally efficient than EHC. The Spectral Clustering, on the other hand, performed the worst. The reason might be, that it usually selects much smaller clusters than Simple Clustering and oriented more on not important details. This is seemingly caused by a selection of similarity matrix, which is created from Gradient Vector Map.

Back to the merging approaches, we can see that the performance is usually the same. Merging 1 approach stays behind the Merging 2 and 3, which is usually caused by handling of transitions. The problem is, that even when one piecewise model is providing satisfactory results on the input points lying in the bordering area, the error is influenced by the other model, which might be very bad. A possible adjustment here could be the utilization of more neighboring points.

All of the source codes, training datasets and also trained models from all experiment can be seen on appended CD.

Fun1				
	Simple Clustering	Spectral Clustering	EHC	SNGP
	Mean MSE			
Merging 1	0.0146	0.04	0.0228	0.0668
Merging 2	**0.0071***	0.0466	0.0232	
Merging 3	0.0072	0.0447	0.0201	
	Best MSE			
Merging 1	0.0121	0.0215	0.0131	0.0399
Merging 2	0.0032	0.0072	0.0079	
Merging 3	**0.003**	0.0139	0.0074	

Table 5.3: Results of experiments on Function 1.

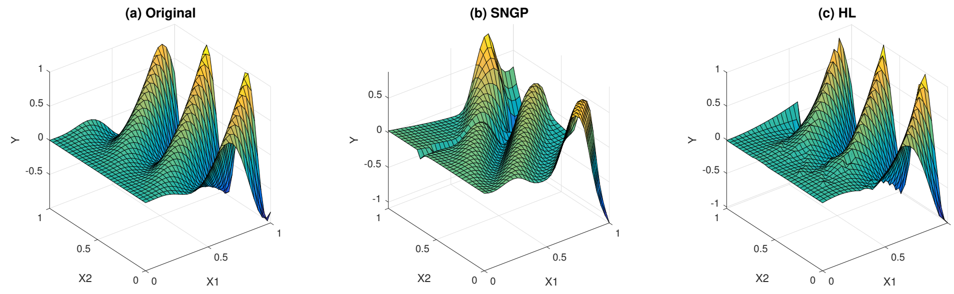


Figure 5.9: Visual comparison of best runs of standard SNGP and Hierarchical learning on Fun1 dataset. Figure c) shows a HL algorithm run which used Simple Clustering method in combination with Merging 2 method. Figure b) shows output of standard SNGP approach. Figure a) shows the surface of the original dataset.

Fun2				
	Simple Clustering	Spectral Clustering	EHC	SNGP
	Mean MSE			
Merging 1	0.0541	0.0643	0.039	0.1317
Merging 2	0.0682	0.0817	**0.0358**	
Merging 3	0.064	0.0767	0.0384	
	Best MSE			
Merging 1	0.0433	0.0506	0.0294	0.1026
Merging 2	0.0352	0.0431	**0.0245**	
Merging 3	0.0493	0.0622	0.0254	

Table 5.4: Results of experiments on Function 2.

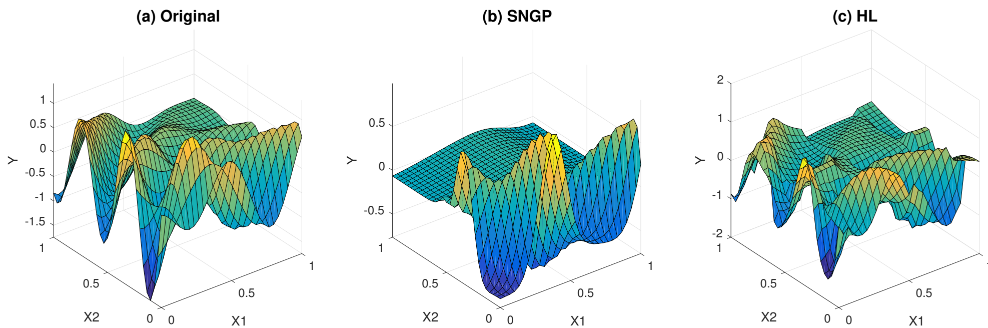


Figure 5.10: Visual comparison of best runs of standard SNGP and Hierarchical learning on Fun2 dataset.

1 DOF Swingup				
	Simple Clustering	Spectral Clustering	EHC	SNGP
	Mean MSE			
Merging 1	8.7498	5.281	1.2346	13.1308
Merging 2	3.1035	2.7568	0.8841	
Merging 3	3.0657	2.6204	**0.8493***	
	Best MSE			
Merging 1	2.1473	2.0431	0.7725	1.744
Merging 2	1.7924	0.7	0.4027	
Merging 3	1.8874	0.6702	**0.3996**	

Table 5.5: Results of experiments on 1 DOF Swingup problem.

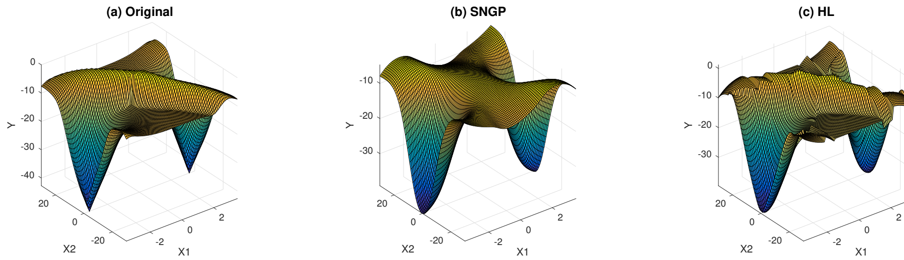


Figure 5.11: Visual comparison of best runs of standard SNGP and Hierarchical learning on 1 DOF Swingup dataset.

1 DOF Swingup Policy				
	Simple Clustering	Spectral Clustering	EHC	SNGP
	Mean MSE			
Merging 1	0.2859	0.6983	0.3717	0.749
Merging 2	**0.2793**	0.6639	0.4261	
Merging 3	0.4230	0.6594	0.3997	
	Best MSE			
Merging 1	0.2116	0.3993	0.2695	0.5006
Merging 2	**0.2099**	0.4125	0.2644	
Merging 3	0.2250	0.4341	0.2570	

Table 5.6: Results of experiments on 1 DOF Swingup Policy.

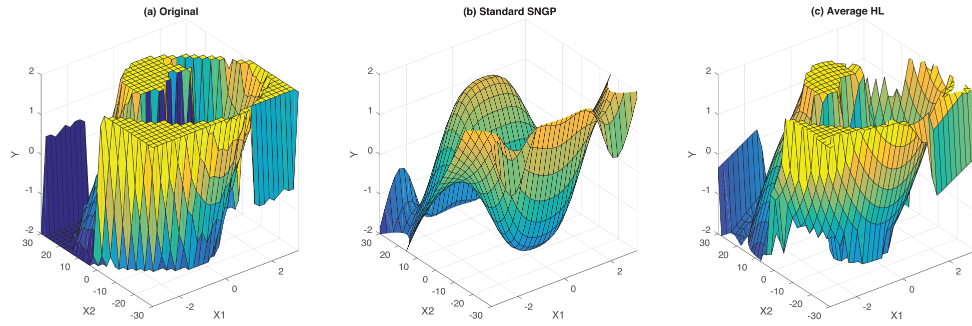


Figure 5.12: Visual comparison of best runs of standard SNGP and Hierarchical learning on 1 DOF Swingup Policy dataset.

Robot Arm Policy				
	Simple Clustering	Spectral Clustering	EHC	SNGP
	Mean MSE			
Merging 1	2.6276	30.5225	3.0009	10.8049
Merging 2	**0.8088***	9.8090	3.8743	
Merging 3	0.8186	9.3224	3.3091	
	Best MSE			
Merging 1	1.7107	23.0743	1.6867	7.5838
Merging 2	**0.4855**	6.0943	1.3450	
Merging 3	0.5321	5.9233	1.4087	

Table 5.7: Results of experiments on Robot Arm Policy.

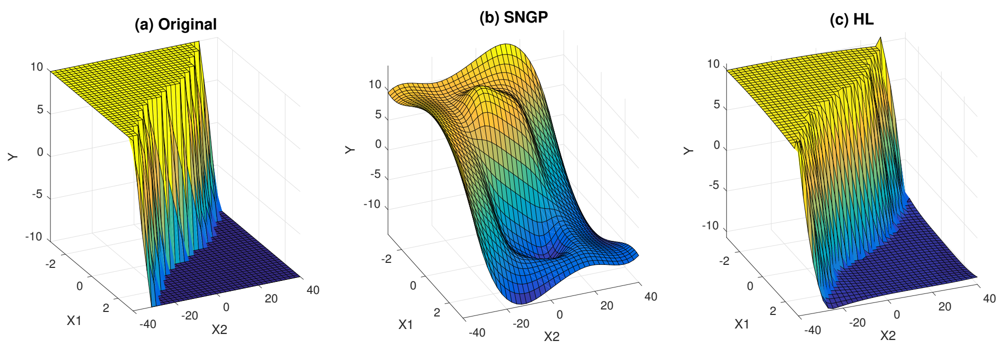
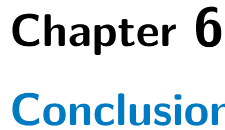


Figure 5.13: Visual comparison of best runs of standard SNGP and Hierarchical learning on Robot Arm Policy dataset.



Chapter 6
Conclusion

In this work, an evolutionary-based framework for solving Symbolic Regression problems was proposed, implemented and tested. For the testing purposes, there were selected 5 different datasets, each containing several problematic artifacts, which seemed difficult to fit using standard approaches. The datasets originated from real Reinforcement Learning problems like V-function approximation for 1 DOF Swingup problem, but also benchmark analytic functions.

To compare the proposed solution to standard SR approach, the state-of-the-art implementation of Single Node Genetic Programming (SNGP) was selected. To objectively evaluate and compare these approaches, there were performed several independent runs on the selected datasets and statistical measures like MSE and MAE were tracked. As results in the previous chapter showed, proposed method was able to outperform the SNGP approach in all experiments.

Even though the proposed method performed minimally 2 times better with respect to the average MSE, the experiments also pointed out at one important fact, that the objective measure like MSE and MAE might not be sufficient in comparing two approaches. This is the most significant in problems from real applications, such as Reinforcement Learning problems, where solution requires a better precision of surface details in specific areas of input space. The proposed method however succeeds in fitting such problematic areas, where it utilizes suitable cluster distribution.

The experiments also showed that the method is sensitive to the selection of clustering approach used to preprocess the input dataset. Even thought the Simple Clustering approach worked well on most of the datasets, there were cases when performed worse and produced higher error than the other two mentioned clustering approaches. The error was however still smaller than the standard SNGP approach. This problem could be solved in future work.

In case of merging, the results showed that Merging 2 and 3 dominated the Merging 1 in most cases. The topic of smoother transition between two neighboring clusters could also be a topic for future work.

Bibliography

[1] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection.* MIT press, 1992, vol. 1.

[2] C. Ryan, J. Collins, and M. O. Neill, “Grammatical evolution: Evolving programs for an arbitrary language,” in *European Conference on Genetic Programming.* Springer, 1998, pp. 83–96.

[3] C. Ferreira, “Gene expression programming in problem solving,” in *Soft computing and industry.* Springer, 2002, pp. 635–653.

[4] I. Arnaldo, U.-M. O’Reilly, and K. Veeramachaneni, “Building predictive models via feature synthesis,” in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation.* ACM, 2015, pp. 983–990.

[5] J. Friedman, T. Hastie, and R. Tibshirani, “Regularization paths for generalized linear models via coordinate descent,” *Journal of statistical software*, vol. 33, no. 1, p. 1, 2010.

[6] A. H. Gandomi and A. H. Alavi, “A new multi-gene genetic programming approach to nonlinear system modeling. part i: materials and structural engineering problems,” *Neural Computing and Applications*, vol. 21, no. 1, pp. 171–187, 2012.

[7] D. Jackson, “A new, node-focused model for genetic programming,” in *European Conference on Genetic Programming.* Springer, 2012, pp. 49–60.

[8] ——, “Single node genetic programming on problems with side effects,” in *International Conference on Parallel Problem Solving from Nature.* Springer, 2012, pp. 327–336.

[9] J. Kubalík, E. Alibekov, J. Žegklitz, and R. Babuška, “Hybrid single node genetic programming for symbolic regression,” in *Transactions on Computational Collective Intelligence XXIV.* Springer, 2016, pp. 61–82.

[10] R. Murray-Smith, “Local model networks and local learning,” *Fuzzy Duisburg*, vol. 94, pp. 404–409, 1994.

[11] D. S. Broomhead and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," DTIC Document, Tech. Rep., 1988.

[12] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1481–1497, 1990.

[13] T. A. Johansen and B. A. Foss, "Identification of non-linear system structure and parameters using regime decomposition," *Automatica*, vol. 31, no. 2, pp. 321–326, 1995.

[14] J. Abonyi, R. Babuska, H. B. Verbruggen, and F. Szeifert, "Incorporating prior knowledge in fuzzy model identification," *International Journal of Systems Science*, vol. 31, no. 5, pp. 657–667, 2000.

[15] J. Novák, "Nonlinear system identification and control using local model networks," 2007.

[16] J. A. J. Tar and F. Szeifert, "Identification of mimo processes by fuzzy clustering," in *the Proc. of the 2001 IEEE International Conference on Intelligent Engineering Systems (INES 2001), Helsinki, Finland*, 2001, pp. 65–70.

[17] D. L. Ly and H. Lipson, "Learning symbolic representations of hybrid dynamical systems," *Journal of Machine Learning Research*, vol. 13, no. Dec, pp. 3585–3618, 2012.

[18] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[19] U. Von Luxburg *et al.*, "Clustering stability: an overview," *Foundations and Trends® in Machine Learning*, vol. 2, no. 3, pp. 235–274, 2010.

[20] S. Still and W. Bialek, "How many clusters? an information-theoretic perspective," *Neural computation*, vol. 16, no. 12, pp. 2483–2506, 2004.

Appendix A

User Guide

A.1 Requirements

1. **Java** version 8
2. **Matlab** 2013 and later

A.2 Organization of appended CD

The source codes are organized into following categories:

1. Matlab sources - */matlab/*
   a. Sources of the simple and spectral clustering - */matlab/clustering/*
   b. Sources of the merging methods - */matlab/merging/*
2. Java sources of SNGP, EHC and HL implementation - */java/src/*
3. Java distributable of SNGP, EHC and HL implementation - */java/dist/*
4. Experimental datasets and results - */experiments/*

A.3 Dataset

The training dataset has to keep following format:

```
1 <number of variables>
2 <number of datapoints>
3 <X1> <X2> <Y>
4 <X1> <X2> <Y>
5 <X1> <X2> <Y>
```

A.4 Clustering

In order to test the clustering you have to run matlab script file **compareClustering.m** located in */matlab/clustering/*. Before executing the

script, you have to specify few initialization variables. In the script editor, specify following variables located at the top of the file:

1. **datafile** - path to the file with input data
2. **numClusters** - number of clusters in final
3. **threshold** - threshold value for simple clustering algorithm
4. **basename** - readable name of the input dataset
5. **split_fun_dir** - path to the folder, where EHC splitting functions are saved
6. **split_fun_seed** - random seed of EHC splitting functions on which they were trained

After running the script, there are created 3 folders in current folder, which contain datasets for each type of clustering methods. These folders serve as a input for Hierarchical Learning algorithm.

To run a EHC learning and get the splitting functions for script described above, you have to run terminal script **SNGPClustering.jar** located in folder */java/dist/* using following command:

$ java -jar SNGPClustering.jar

The script also requires a configuration file *clustering.properties*, which is also included in the same folder. User has to specify following properties in file:

1. **dataset** - path to training dataset
2. **nbOfRuns** - number of separate EHC runs
3. **seed** - random seed for PRNG
4. **numRestarts** - number of random restarts per run, to avoid local optima
5. **maxTreeDepth** - maximal search tree depth, which influence number of clusters $2^{maxTreeDepth} = numClusters$

After running the script, folder *results/* is created and filled with splitting functions. The path to these functions should then be specified as a **split_fun_dir** parameter in **compareClustering.m** script.

A.5 Hierarchical Learning

To run the Hierarchical Learning script, go to folder */java/dist/* and first, customize following properties in file **hierLearning.properties**:

1. **maxGenerations** - maximal number of generations

2. **datasetDir** - path to folder containing the split dataset files, generated in clustering phase

3. **maxTreeDepth** - maximal search tree depth

4. **numSimpleClusters** - specifies on how many starting clusters is algorithm learning

5. **epsilon** - epsilon in line 15 in Algorithm 4 (Hierarchical Learning)

6. **resultsDir** - path to folder, where learned models are stored

7. *nbOfRuns* - number of separate runs of the algorithm

8. *tailFunctionSet* - name of functions which can be used as a starting features (function nodes)

9. *populationSize* - size of population of new nodes

Properties in italics are optional, and you can leave default values for them. In case of *tailFunctionSet* you can pick from following functions: *Multiply, Plus, Minus, Pow2, Pow3, Pow4, Pow5, GeneralGauss2D, GeneralTanh, BendIdentity, Sine, Cosine*

The script is executed using command

```
$ java -jar SNGPHierLearning.jar
```

from the command console or terminal. Running the command again requires preinstalled Java environment. When the learning process is finished, matlab model files and also final partition of clusters are created in the results folder. These are used in the merging phase, which visualizes the outcome.

A.6 Merging

Merging scripts are found in folder */matlab/merging/*. To simply compare outcomes of merging methods, you can run script **compareMerging.m**. You also have to specify few variables, before running the script. In the script editor, open file **compareMerging.m**, and edit following variables at the top of the file:

1. **basename** - readable name of dataset, which is same as in clustering phase

2. **clustering_method** - type of clustering, which generated dataset, options are *{simple, spectral, ehc}*

3. **tag** - must be same as random seed of HL algorithm run

The script expects, that current working folder of matlab contains following folders:

1. **dataset folder** - the same folder, which was generated during clustering phase, format *"dataset_<basename>_<clustering_method>/"*
2. **model folder** - containing learned models from HL algorithm, format *"results_<clustering_method>/"*

The outcome of the script is a figure with 3 images, comparing all three merging methods. MSE (MAE) of these methods is stored in vector variable **mse** (**mae** respectively).