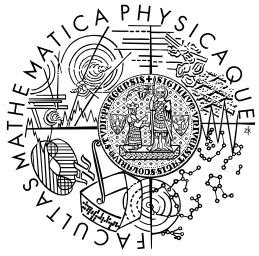

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

HABILITATION THESIS

Martin Pilát

Evolutionary Algorithms for Expensive Optimization

Department of Theoretical Computer Science and Mathematical Logic

Prague 2022

Acknowledgment

I would like to thank to the co-authors of the papers presented in this thesis for their work and for helping in the research. I would also like to thank to all my colleagues in the Department of Theoretical Computer Science and Mathematical Logic and at the whole Faculty of Mathematics and Physics for creating a great, stimulating working environment. The thesis would not be possible without them.

I would also like to thank to all the students I met during my years at the university for asking interesting questions and coming with interesting ideas for their own research and theses. They helped me to gain new insights into many different areas of evolutionary computation and artificial intelligence and, hopefully, also made a better teacher for those who will come after them.

Last, but not least, I am thankful to my wife, Petra, for providing me with all the support I needed during the writing of this thesis and for making my life better and happier.

Some of the work in this thesis was supported by a number of different Czech Science Foundation projects – 15-19877S, 17-10090Y, and 17-17125Y. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures and under the program "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042).

Contents

1 Introduction

With its recent fast development, artificial intelligence in general and machine learning in particular are gaining new applications every day. People and companies use machine learning to classify images and translate documents. Medical doctors use deep learning models to improve their diagnostic processes, and self-driving vehicles use machine learning, artificial intelligence and computer vision to navigate dynamic environments safely.

Many of these recent advances gain their power from mathematical optimization, ranging from the more low-level algorithms such as stochastic gradient descend and its variants used to train the parameters of numerous machine learning models to more high-level optimization algorithms based on evolutionary computing or reinforcement learning to optimize whole machine learning workflows in the areas of automated machine learning and neural architecture search.

In this thesis, we focus mostly on evolutionary algorithms – a nature-inspired population-based optimization technique that is inspired by Darwinian evolutionary theory. Evolutionary algorithms have obtained great results in the recent years both in the areas mentioned above (automated machine learning, neural architecture search), but also in a many different engineering applications. Every year at the GECCO conference, the best of these algorithms are awarded the Humies award for human-competitive results obtained by evolutionary computation techniques.

The list of the winners of the Humies competition is available at https://www.human-competitive.org/

However, evolutionary algorithms also have one strong disadvantage, and that is the fact that they typically need to make a lot of evaluations of the objective function we want to optimize – either minimize, or maximize. If the objective function is slow, which is often the case in the applications mentioned above, it also makes the whole algorithm slow. This, in turn, makes using evolutionary algorithms for these applications hard, or even impossible. In this thesis, we try to improve some of these shortcomings by implementing better parallel evolutionary algorithms and by using so called

surrogate models – cheap approximations of the slow or expensive objective function.

1.1 Thesis Structure

This thesis is divided into two parts. The first one contains the background and general context of the work, the second one contains eight selected papers that show our recent results in this area.

More specifically, the first part contains three chapters. Chapter 2 gives a general description of evolutionary algorithms and also some of their variants that are later discussed – differential evolution, genetic programming, and a multi-objective evolutionary algorithm – MOEA/D. This chapter also mentions the main challenges of evolutionary computation when dealing with optimization of objective functions that are slow or expensive to evaluate.

Chapter 3 discusses the ways, how evolutionary algorithms can be implemented in parallel, and also the problems that the parallel implementations face in case the evaluation time of the objective function is variable. This chapter thus gives a more detailed context to the papers included in Chapter 6.

Chapter 4 describes so called surrogate models and how they can be used to speed up evolutionary algorithms. Surrogate models are typically used in continuous optimization (optimization of functions $\mathbb{R}^n \rightarrow \mathbb{R}$). However, we also describe how these can be used in other areas, such as in genetic programming. This chapter provides introduction to the papers included in Chapter 7.

The second part of the thesis contains the selected papers themselves together with a single chapter (Chapter 5) that briefly describes the main results of the papers and their context within the wider field of work. This chapter also explains my contribution to the results of each of the papers and briefly discusses other works I published in related areas. The papers themselves are in four chapters divided by their topic – Chapter 6 contains two papers about the parallelization of evolutionary algorithms, Chapter 7 contains two papers about using surrogate models in genetic programming, Chapter 8 contains a paper about consideration of user preferences in multi-objective evolutionary algorithms, and,

finally, Chapter 9 contains three papers in which evolutionary algorithms are applied to various problems with expensive objective functions.

All the papers selected for this thesis were published either in journals, or as full papers at international peer-reviewed conferences. The only exception is the second paper on surrogate models in genetic programming that was presented as a poster at GECCO 2022, it was however still included in this thesis as it shows the current direction of our research on surrogate-based genetic programming.

The papers in this thesis are, however, not all the papers we published in the recent years. They were selected as representatives for each of the areas presented in this thesis. The reasons for selecting each of them and other closely related papers are mentioned in Chapter 5.

The full list of papers included in this thesis is below.

1.2 Included Papers

Martin Pilát, Roman Neruda (2017): 'Parallel Evolutionary Algorithm with Interleaving Generations'. In: *Proceedings of the Genetic and Evolutionary Computation Conference. GECCO 2017.* Berlin, Germany: Association for Computing Machinery, pp. 865-872. DOI: 10.1145/3071178.3071309

Štěpán Balcar, Martin Pilát (2020). 'Heterogeneous Island Models and Their Application to Recommender Systems and Electric Vehicle Charging'. In: *International Journal on Artificial Intelligence Tools* 29.03n04, World Scientific, p. 2060010. DOI: 10.1142/S0218213020600106

Martin Pilát, Roman Neruda (2016). 'Feature Extraction for Surrogate Models in Genetic Programming'. In: *Parallel Problem Solving from Nature - PPSN XIV.* Ed. by Julia Handl et al. Cham: Springer International Publishing, pp. 335-344. DOI: 10.1007/978-3-319-45823-6_31

Martin Pilát, Gabriela Suchopárová (2022). 'Using Graph Neural Networks as Surrogate Models in Genetic Programming'. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion. GECCO 2022.* Boston, Massachusetts: Association for Computing Machinery, pp. 582-585. DOI: 10.1145/3520304.3529024

Martin Pilát, Roman Neruda (2015). 'Incorporating User Preferences in MOEA/D through the Coevolution of Weights'. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. GECCO 2015.* Madrid, Spain: Association for Computing Machinery, pp. 727-734. DOI: 10.1145/2739480.2754801

Tomáš Křen, Martin Pilát, and Roman Neruda (2017). 'Automatic Creation of Machine Learning Workflows with Strongly Typed Genetic Programming'. In: *International Journal on Artificial Intelligence Tools* 26.05, World Scientific, p. 1760020. DOI: 10.1142/S021821301760020X

Martin Pilát (2018). 'Controlling the Charging of Electric Vehicles with Neural Networks'. In: *2018 International Joint Conference on Neural Networks (IJCNN 2018)*, IEEE, pp. 1-8. DOI: 10.1109/IJCNN.2018.8489027

Věra Kumová, Martin Pilát (2021). 'Beating White-Box Defenses with Black-Box Attacks'. In: *2021 International Joint Conference on Neural Networks (IJCNN 2021)*, IEEE, pp. 1-8. DOI: 10.1109/IJCNN52387.2021.9533772

Background

In this part, we provide the important background concerning most of the work in the areas covered by the papers in this thesis.

2 Evolutionary Algorithms

The thesis deals mostly with evolutionary algorithms, therefore, in this chapter, we give a short general introduction to the area of evolutionary computing and describe the concepts and versions of the algorithms that are most relevant to this thesis. A more comprehensive overview of the area is presented in the books by Eiben and Smith (2015) or Michalewicz and Fogel (2004).

2.1 Overview

Evolutionary algorithms are a population-based optimization technique inspired by natural evolution. An evolutionary algorithm works with a set (called *population*) of candidate solutions (*individuals*). The population is often first initiated randomly and then the algorithm runs in multiple iterations (*generations*). In each generation, the quality of all individuals is evaluated using so called *fitness function*. The fitness function should assign greater values to individuals with better (lower for minimization and greater for maximization) values of the objective function. Then, *genetic operators* are applied to individuals selected by so called *mating selection*. Individuals with better fitness have better chance of being selected. There are two types of genetic operators – *crossover* and *mutation*. Crossover takes two individuals as its input, combines the information from them and returns one or two offspring as its output. Mutation, on the other hand, takes only one individual as its input and returns a single offspring as its output. The individuals before the genetic operators are applied are often called *parents* and the newly created individuals are called *offspring*. At the end of each iteration, the populations of parents and offspring are combined and so called *environmental selection* is used to select which individuals will survive to the next generation. The number of parents and offspring is often the same, but there are also versions of the algorithm, where these number differ. In such cases, the number of parents is typically denoted as μ and the number of offspring as λ. This algorithm is also expressed in pseudo-code in Algorithm 1.

Algorithm 1: Evolutionary Algorithm

```
1  t ← 0
2  P_0 ← random initial population
3  Evaluate the fitness of all individuals in P_0
4  while not happy do
5      O = ∅
6      while |O| < λ do
7          p_1^i, p_2^i ← MatingSelection(P_t)
8          o'_1^i, o'_2^i ← Crossover(p_1^i, p_2^i)
9          o_1^i ← Mutation(o'_1^i); o_2^i ← Mutation(o'_2^i)
10         O ← O ∪ {o_1^i, o_2^i}
11     Evaluate the fitness of all new individuals in O
12     P_{t+1} ← EnvironmentalSelection(P_t, O)
13     t ← t + 1
14 return P_{t−1}
```

The description and pseudocode above give only the overall structure of (one version of) an evolutionary algorithm. There are many considerations that should be taken before evolutionary algorithms can be used to solve practical problems – we need to decide on the individual encoding (how the solution candidates can be encoded as individuals), on the fitness function (how to express the quality of the individuals), on the selection method (how to select the better individuals to which the genetic operators will be applied), and on the genetic operators itself.

In a more general and realistic case, we may want to find a bit-string that exactly matches an unknown one.

Consider, for example, the well-known OneMax problem, where the goal is to find a bit-string x of length n that contains all 1s. Therefore, the goal is to maximize

$$\textsc{OneMax}(x) = \sum_{i=1}^{n} x_i.$$

This is the so called one-point crossover.

This is the bit-flip mutation.

In this case, the individual can be a binary string of length n and the fitness can be directly the function OneMax defined above. The crossover operator typically chooses a random crossover point $1 < r < n$, the offspring are then created by copying values on indices $1..r$ from one parent and the rest of the values from the other parent. The mutation operator then randomly changes some values in the individual from 0 to 1 or vice versa. Both the mutation and crossover operators are typically performed with only some probability. The individuals, to which the operators are not applied are cloned

without changing them.

The individual encoding, fitness function, and genetic operators are problem specific. On the other hand, the mating and environmental selections and the termination condition tend to be one of the handful of options described below.

The mating selection of the individuals should ensure that better individuals have higher probability of being used in the genetic operators. There are two common ways, how this is performed – one of them is the *tournament selection*, the other is the *roulette wheel selection*. In tournament selection, two individuals are selected uniformly randomly from the population and the better one is then used with some high probability (typically around 80 percent). In roulette-wheel selection (also called fitness-proportionate selection), each individual is selected with probability directly proportionate to its fitness, i.e. the probability p_i of selecting individual i is defined as

This is a binary tournament, in k-ary tournament, the best of k individuals is selected with a high probability, if this one is not selected, the second best one is selected from the rest with the same high probability and so on.

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_n},$$

where f_i is the fitness of individual i and N is the number of individuals in the population. This type of selection assumes that the fitness is always positive. Tournament selection, on the other hand, does not have this assumption and, additionally, it is invariant with respect to any monotonically increasing transformations of the fitness functions, as the selected individuals depend only on the ranking of the individuals according to their fitness values and not on the specific fitness values.

The environmental selection decides which of the parents and offspring survive to the next generation. In the simplest case, the parents are discarded and only the offspring are kept to the next generation. However, this way has the disadvantage that the algorithm can lose the best solution found so far. Therefore, some form of *elitism* is often employed, where some number of the best parents are always kept to the next generation and the rest is replaced by the best offspring. There are also special types of environmental selection for cases where $\mu \neq \lambda$, i.e. the numbers of parents and offspring are different. For these the notation originally used in evolution strategies is often employed. The (μ, λ) selection (sometimes also called the *comma selection*) selects μ best individuals from the λ offspring, this case assumes that $\mu \geq \lambda$. In

case $\mu = \lambda$ this operation is equivalent to discarding the parents and keeping only offspring. The $\mu + \lambda$ selection (*plus selection*) selects the best μ individuals out of the combined populations of parents and offspring. In this case, there are no assumptions about the values of μ and λ an even extreme cases of $(1 + 1)$, $(\mu + 1)$, or $(1 + \lambda)$ are sometimes used.

The last part of the algorithm, we have not discussed yet, is the termination condition, expressed with "while not happy" in the pseudocode. In most cases the algorithm is limited by a specific number of generations (iterations of the loop), or a specific number of fitness evaluations. Other common termination conditions include reaching a specified quality of individuals (expressed by a target fitness value) or running for a pre-defined amount of time.

2.2 Differential Evolution

There are special types of evolutionary algorithms used for specific types of problems. For example, for continuous optimization, i.e. optimization of functions $\mathbb{R}^n \to \mathbb{R}$, evolution strategies (Hansen, Arnold, and Auger 2015) or differential evolution (Storn and Price 1997) are often used. In both cases, the individuals are encoded as vectors of floating point numbers with length n. We describe differential evolution in more detail here, as it is the algorithm that is used in some of the papers in this thesis.

The main idea of differential evolution is to replace the mutation operator by an operation that adds the difference of two randomly selected individuals to a third individual. More specifically, three different individuals a, b, and c are randomly chosen from the population and a new potential individual x' is generated as $x' = a + F(b - c)$, where F is a parameter, typically around 0.8. Then, another individual x is selected from the population and the new offspring is created by iterating over the values of x_i and x'_i and taking the value from x'_i with probability CR and the value from x otherwise. A single randomly selected value is always taken from x'_i. More specifically, first, values r_i for $i \in \{1, \ldots, n\}$ are generated from a uniform distribution between 0 and 1 and then a value j is randomly uniformly selected from the

set $\{1, \ldots, n\}$. Then, the offspring

$$y_i = \begin{cases} x'_i & r_i < \text{CR or } i = j \\ x_i & \text{otherwise,} \end{cases}$$

where CR is the crossover rate selected from interval $[0, 1]$. Finally, the fitness of y_i and x is compared and the better of these individuals is kept in the population. Typically, the individual x is not selected randomly, but the operation is run for each x in the population.

Differential evolution has some desirable features for continuous optimization. First, because of the way, how the individual x' is generated, the algorithm is robust with respect to rotation and scaling of the search space. Second, as the better individual is selected based on comparison of fitness values of both the new and original individuals, the algorithm is also invariant with respect to any monotonically increasing transformation of the fitness values.

2.3 Genetic Programming

A very popular variant of evolutionary algorithms is genetic programming. As its name suggests, the goal of genetic programming is to create programs using evolutionary algorithms. A very nice introduction to genetic programming was written by Poli, Langdon, and McPhee (2008) and is freely available online.

In its most common variant, the so called tree-based genetic programming, the individuals are encoded as trees. These trees can then represent mathematical expressions, simple programs or even describe analogue or digital circuits or structures of neural networks.

Formally, we need to define two sets – terminals and non-terminals. Terminals are the leaves of the trees and typically contain the inputs to the program and constants. Non-terminals are the internal nodes in the tree and they represent the elementary functions that can be used in the program. For each of the terminals, we need to specify its arity (number of arguments), this also specifies the number of sub-trees the non-terminal has. For example, if we want to use genetic programming to find polynomials in a single variable x, the

set of terminals can contain x and some common constants (typically small integers), the set of non-terminals can contain the binary functions for multiplication and addition (and potentially also powers).

The initialization is, as in other versions of evolutionary algorithms, done by randomly generating the initial population of trees. In this case, generally two options are used, either full trees with a given depth are generated, or trees with a given number of nodes are generated. A popular initialization scheme (*ramped half-and-half*), generates half of the individuals with the former approach and the other half with the latter one.

The genetic operators in genetic programming operate on the tree-based individuals directly. A popular choice for crossover is the sub-tree crossover that randomly selects a subtree in one individual and swaps it with a random subtree in the other individual. There are many options for mutation – we can remove a sub-tree and replace it with one of the terminals in that sub-tree, we can change one of the non-terminals to another one with the same arity, or we can replace a sub-tree with a randomly generated one.

In typed variants of genetic programming, terminals and non-terminals have types that affect how trees can be constructed. The type of the sub-tree must match the type of the input to the non-terminal. This is useful if we want to limit what trees are valid. For example, imagine we want to simulate an `if` condition in a tree that should otherwise return a number. The non-terminal for the condition can have tree inputs – one of them is boolean (either true or false), the other ones are numbers. If the condition is true, the first number if returned from this node, otherwise the second one is returned. We can then use other types of non-terminals, such as the less-then relation, that take numbers as inputs and return boolean variables. We can still use similar genetic operators to those mentioned above, they just need to take the types of the terminals and non-terminals into account.

Typed genetic programming is very useful if we need to consider multiple different types of data in the program or in the structure we try to evolve. In one of the papers that are part of this thesis (Křen, Pilát, and Neruda 2017b), we used typed genetic programming to ensure that the individuals represent valid machine learning pipelines.

2.4 Multi-Objective Optimization

Evolutionary algorithms are also very often used to solve multi-objective optimization problems. In multi-objective optimization, the goal is to optimize multiple functions at the same time. More formally, we want to minimize $f : D \rightarrow \mathbb{R}^n$, where D is some decision space. Such a problem typically does not have a unique solution, as some of the functions can be competing with each other. We are therefore looking for a set of *Pareto optimal solutions*, i.e. such solutions that there is no other solution that would be better in all objectives at the same time.

Evolutionary algorithms for multi-objective optimization differ in the way how selection (and especially environmental selection) is performed. Currently, two classes of multi-objective evolutionary algorithms that are commonly used – domination-based and decomposition-based ones.

Domination-based multi-objective optimization algorithms are based on the Pareto dominance relation. NSGA-II (Deb et al. 2002) is a popular example of this class of algorithms. It performs the environmental selection by first merging the parent and offspring populations and then dividing the merged population into so called non-dominated fronts. The first non-dominated front contains individuals that are not dominated by any other individual in the population, the second non-dominated front then contains individuals that are not dominated by any individuals in the population, if the first front is removed and so on. Then, individuals from the fronts with lower numbers are put into the population as long as the whole front fits, i.e. as long as the sum of the sizes of the fronts is at most equal to the population size. If more individuals are needed but the next non-dominated front does not fit whole, individuals are selected based on a secondary sorting criterion. In the original version of the algorithm this is the crowding distance that expresses how close the solution is to other solutions in the objective space. Solutions in less crowded areas are selected first.

While NSGA-II is still a very popular algorithm, especially for problems with two or three objectives, it does not work very well if the number of objectives increases, as in such a case most of the individuals belong to the first non-dominated front (Ishibuchi, Tsukamoto, and Nojima 2008). For these

problems, often called many-objective optimization problems, decomposition algorithms such as MOEA/D are more commonly used.

In MOEA/D (Q. Zhang and H. Li 2007), the multi-objective problem is decomposed into a set of single-objective ones that are all solved at the same time. One possible decomposition is using weighted sum of the objectives with different weights for each individual. More specifically, the fitness for individual k is computed as $f^k(x) = \sum_{i=1}^{n} w_i^k f_i(x)$, where f_i are the objective functions and w^k is the weight vector assigned to individual k. The weight vectors are also used to define neighborhoods of individuals – a number of individuals with the closest weight vectors are considered to be in the same neighborhood. The genetic operators are then performed with neighboring individuals with a high probability. Once a new offspring is generated, its fitness is compared to the fitness of its parent and the better one of these is kept in the population. This individual is also compared to other individuals in the neighborhood using their respective fitness functions, and if it is better it also replaces them. The sharing of the information in the neighborhood makes MOEA/D more effective in solving multi-objective problems than using independent runs of single-objective algorithms.

The weighted-sum decomposition works well for problems with convex Pareto fronts. An alternative working for all types of problems is Tchebycheff decomposition, where $f^k = \max_i \lambda_i |f_i(x) - z_i^*|$, where λ_i are the parameters of the decomposition, and z_i^* are the minima of f_i.

2.5 Challenges in Evolutionary Computation

All the evolutionary algorithms described above have one step in common – and that is the evaluation of the fitness function. This is also the step that is most affected by the optimization problem the algorithm tries to solve. This function, however, very often affects how long the algorithm will run and if it is practically usable. Commonly, evolutionary algorithms use tens or hundreds of individuals in population and run for hundreds, or even thousands of generations. The fitness of most of these individuals need to be evaluated in every generation, which leads to long running times with even relatively fast fitness functions. For example, with a population of 100 individuals and 1000 generations we need roughly 100,000 fitness evaluations. If each of these evaluations takes only 1 second, the whole run of the algorithm would take more than a day.

We actually do not have to re-evaluate fitness of individuals that were not changed by the genetic operators. Some implemetations may even use caching of the results of previously evaluated individuals and use these values if they are randomly created again.

There are multiple ways, how to speed up evolutionary algorithms. In some cases, the fitness function itself can be replaced by a faster fitness function with similar properties, or an approximation of the fitness function can be computed. This can work for example in symbolic regression, if the individuals are evaluated only on a subset of the training data. However, the potential to use these approaches is limited to specific problems. There are two more general ways, how to speed up evolutionary algorithms – parallel implementation or surrogate modelling.

Parallel evolutionary algorithms use multiple CPUs or even multiple computers to run the evolutionary algorithm. Surrogate modelling, on the other hand, aims to reduce the number of fitness evaluations by replacing the slow fitness function by a fast model thereof. We describe these two techniques in more details in the following two chapters.

3 Parallelization

Evolutionary algorithms are extremely easy to parallelize, which is one of the reasons, why their parallelization is an often considered way, how to make their computation faster. In this chapter, we describe the most common models of parallelization of evolutionary algorithms and some challenges parallel evolutionary algorithms face. A more detailed survey of parallel evolutionary algorithms was published by Sudholt (2015).

3.1 Classical Parallelization Models

The aforementioned survey mentions three different groups of parallel evolutionary algorithms based on the form of parallelization – in the *master-worker model*, only the evaluation of the fitness evaluation is parallelized, in the *island models* multiple sub-populations are executed in parallel, and in the *cellular models* each individual is assigned its own CPU. We describe all these types of parallelization below. Apart from these models, the survey also mentions the possibility to run a number of independent runs of the algorithm in parallel and a few other generalizations of the models above, these are however not really relevant for the rest of the thesis.

In the master-worker model, the main loop of the algorithm does not change and the whole structure remains the same. The only difference is that the fitness of all individuals is evaluated in parallel. This parallel evaluation can be done either on the same machine in case multi-core CPU is used or the individuals can be submitted to other machines over the network. With current CPUs, it is possible to evaluate tens or even hundreds of individuals in parallel making it often possible to evaluate the whole generation in parallel. In case all the fitness evaluations take the same amount of time, the speed up can thus be linear in the number of CPUs (ignoring any overhead for the parallelization itself).

The island model divides the population of the evolutionary algorithm into several sub-populations and each of these sub-populations run on a different computer. The

This is also called coarse-grained parallelization.

sub-populations themselves run in the same way a classical evolutionary algorithm does, the main difference is that there is a new genetic operator for migration of individuals. This operator sends some of the individuals to other islands and receives individuals sent by the other islands. Communication topology defines, between which pairs of island individuals can be exchanged. Commonly used topologies include ring topology, grid topology, torus topology, or a fully connected one. There are many ways to choose which individuals are sent, how often they are sent, and how they are incorporated into the population on the receiving island. This communication allows the islands to effectively cooperate with each other on solving the optimization problem at hand. If one of the islands converges to some local optimum, it can still receive better individuals from the other islands and continue in the optimization, hopefully finding the global optimum.

Sub-populations are also a good way to increase diversity in the population and can be used for this reason even without parallelization.

In cellular models, each individual is assigned to a computing node. Similarly to island models, there is topology defined among the computing nodes (often either grid or torus topology is used here), and the genetic operators are only performed on individuals which are neighbors in the topology. This limits the speed with which the information about new promising solutions can propagate in the population and increases the diversity of the solutions in the population. There are multiple ways, how the whole population can be updated, the simplest among them is the synchronous update where every node first generates new individual and then all of them are evaluated and replace the original one if they are better. There are, however, also asynchronous variants of this update.

This is also called the diffuse model, or fine-grained parallelization.

In both the island and the cellular models the topology defines which nodes can communicate with which other nodes. An important feature of the topology is the diameter of the graph as it affects how long it takes for the information to propagate to the whole population.

The graph diamater is the length of the longest of the shortest paths between any two nodes in the graph.

There are some less common variants of the island models with heterogeneous islands, i.e. such where each island uses different settings of the algorithm (Gong and Fukunaga 2011) or a different fitness function (Mambrini, Sudholt, and Yao 2012; Pilát and Neruda 2010).

3.2 Asynchronous Evolutionary Algorithms

The parallelization of evolutionary algorithms described above can speed up the computation significantly. Most of the techniques work best, if each of the fitness evaluations takes the same amount of time. However, there are problems where the fitness evaluation time is variable. Such problems exist for example in the area of automated machine learning (AutoML) (He, Zhao, and Chu 2021), where the goal is to find the best machine learning pipeline for a given dataset. In this case, evaluation of the fitness function typically includes training of the machine learning pipeline which is slow, and different machine learning pipelines can take even orders of magnitude different times to train.

In such cases using the master-worker parallelization with number of CPUs equal to the number of individuals in the population means that we have to wait for the slowest fitness evaluation in each generation. That in turn means that we waste a lot of the computational power by waiting. For this reason, so called asynchronous evolutionary algorithms are commonly used. Asynchronous evolutionary algorithms do not wait for the whole generation to finish fitness evaluation, but instead generate new offspring as soon as an individual is evaluated. The algorithm thus first generates a random initial population and submits all the individuals for evaluation. Once enough individuals are evaluated, the algorithm runs the selection to select parents, runs the genetic operators on them to create new offspring and submits the offspring for evaluation whenever a free CPU is available. This implementation ensures that all the CPUs are always utilized, however, the algorithm itself is biased towards the areas of the search space where the individuals evaluate faster (Scott and De Jong 2015).

3.3 GPU Implementation of Evolutionary Algorithms

Using graphic cards (or, more formally, graphic processing units, GPUs) for computation has led to significant advances

in machine learning and especially deep learning. Evolutionary algorithms can also benefit from their GPU implementation and such implementations started appearing as soon as GPU computing became available (Jaros 2012; Krömer et al. 2011). However, as we already mentioned above, in evolutionary algorithms the most time-consuming part is typically the fitness function. Therefore, if the fitness function is not evaluated on a GPU, the overall algorithm may not run much faster.

For GPU implementation, even finer grained parallelization than the one used in cellular models is often needed (Cheng and Gen 2019) and these implementations often use so called gene-level granularity, where a computing thread is assigned to each gene in every individual. This is possible thanks to the fact that GPUs typically have thousands of parallel computing threads.

A *gene* is a specific position in the individuals. For example, an individual encoded as a vector of n numbers consists of n genes.

There is a number of general-purpose libraries for GPU computing, such as pyTorch (Paszke et al. 2019) and Tensorflow (TensorFlow Developers 2022), these are however typically not used for implementation of evolutionary algorithms. Recently, we have shown (Valkovič and Pilát 2022) that such implementations are possible and quite simple using the pyTorch library, if the genetic operators are carefully selected and some tricks are used to make the implementation more suitable for GPU computing.

4 Surrogate Models

Surrogate models are another option, how to speed up evolutionary algorithms. They are a fast approximation of the original slow or expensive fitness function. These approximations are typically created by using the individuals evaluated in previous generations to fit a regression model of the fitness function. The main advantage of using surrogate models compared to parallelization is that they can reduce the number of the fitness evaluations in the first place. This can be important in cases, when the fitness evaluation is not only slow, but also expensive because, for example, some real-life experiments need to be performed to measure it. On the other hand, the most significant disadvantage of using surrogate models is the overhead caused by the need to train them. A very nice survey on the use of surrogate models in evolutionary computation was published by Jin (2011) and a more recent survey was published by Tong et al. (2021).

In this chapter, we describe, how surrogate models are used in evolutionary algorithms.

4.1 Building a Surrogate Model

In a typical surrogate-based evolutionary algorithm, we use the individuals evaluated during the optimization as a base for a training set for the model. Essentially, we have an archive of evaluated individuals that is sampled to provide the training set. The initial training set may be obtained in various ways, one of them is not using the surrogate model in the initial generations of the EA. Alternatively, the algorithms can use Latin Hypercube Sampling (McKay, Beckman, and Conover 1979) or other advanced sampling mechanisms to create more diverse initial population with the goal to train a better model.

Regarding the types of models, any regression model can be used, however, low order polynomials, support-vector regression, or RBF networks are commonly used based on our experience as well as the survey (Tong et al. 2021) mentioned above. Kriging is also quite often used, as it can provide

not only the prediction itself, but also the uncertainty of the model for the specific point. This allows for greater flexibility while selecting the individuals to be evaluated by the expensive objective function. Apart from regression models, some algorithms also use ranking models, such as rankSVM (Joachims 2002).

4.2 Using a Surrogate Model

Surrogate models can be used in multiple ways in evolutionary algorithms. The survey by Jin (2011) divides the types of uses into three groups – generation-based, individual-based, and population-based ones. Generation-based surrogates are used in some generations, while in others the real fitness function is used. In individual-based approaches some of the individuals are evaluated by the real fitness, while others are evaluated only by the surrogate. Finally, in the population-based ones the population is divided into sub-populations (similarly to the island models for parallelization) and each of the sub-populations uses a different surrogate model.

Surrogate models can also be used for pre-selection of individuals. In this case, after the offspring are generated, they are all evaluated using the surrogate models, then, only some of them are evaluated using the real fitness function. In this case, selection is based only on the real fitness function, in contrast to the individual-based technique described above, where some individuals may be evaluated only with the surrogate.

If we use kriging or other techniques that also return uncertainty in addition to the prediction, we have additional options, how to select the individuals for evaluation. For example, we can decide to evaluate not only the best individuals according to the model, but also individuals with high uncertainty of the prediction. Evaluating these and adding them to the training set may improve the quality of the model in areas that are not as well explored.

One of the criterions that combine both the quality of the individual and the uncertainty (variance) of the model is the expected improvement – the expected value of $\max(f^* - f, 0)$, where f is the value predicted by the model (viewed here as a random variable with a distribution given by the model) and

f^* is the current best value found by the algorithm. Another option that combines the quality with the uncertainty is the lower confidence bound.

Using the expected improvement is also the idea of the Efficient Global Optimization (EGO) algorithm (Jones, Schonlau, and Welch 1998). In this algorithm, the objective function is modelled using the kriging model, whenever a point is evaluated by the real objective function, the model is retrained. The next sampled point is then the one that maximizes the expected improvement according to the model.

This technique was also extended to multi-objective optimization (Knowles 2006).

Surrogate models can also be used in multi-objective optimization. Apart from the simple solution of using a different model for each of the objectives, it is also possible to create so called aggregated surrogate models. These models try to predict, whether a given solution is non-dominated in the current population. One of the first algorithms using this technique was described by Loshchilov, Schoenauer, and Sebag (2010), who used a combination of OneClass SVM and Regression SVM. We have also published an algorithm based on a similar technique (Pilát and Neruda 2011), however, we used a model based on the distance to the non-dominated front, and we also used the model in a mutation to generate new non-dominated individuals by running and internal evolutionary algorithm optimizing the model.

4.3 Surrogate Models Outside of Continuous Optimization

Surrogate models are most commonly used in continuous optimization, where the individuals are encoded as vectors of numbers. This makes training the surrogate models simple, as most regression models expect vectors of numbers as their inputs. However, surrogate models can also be useful in other areas, such as combinatorial optimization, or genetic programming.

While the problem of using surrogate models in combinatorial optimization was already mentioned as an open question in the survey by Jin (2011), there does not seem to be a lot of research done in this area. Bartz-Beielstein and Zaefferer (2017) published a survey on this use that divides the approaches into several categories based on the way how they

deal with the problem. Among these are approaches that use custom models, usable only for the specific problem at hand, similarity-based approaches, or feature extraction based approaches.

An interesting special case is using surrogate models in genetic programming. As we described in Chapter 2, in genetic programming, individuals are encoded as trees. For such individuals, Hildebrandt and Branke (2015) used so called phenotypic features. These features are obtained by using a fixed set of inputs for the genetic programs and using their outputs as the features. For each individual we thus get a vector of numbers and then any common regression technique can be used to build a model. In this paper, nearest neighbor regression was used.

The technique described above is usable, if the tree can be easily evaluated on some inputs. However, it is not possible to compute the phenotypic features in such cases, where the tree represents a complex structure, such as in automated machine learning, or any other application where the individual is not a program but rather a description of a structure. For such cases, we developed an algorithm based on feature extraction from the trees (Pilát and Neruda 2016). These features contain the size of the tree, the types of nodes in the tree and similar. Later, we also used graph neural networks that can work directly with trees as their inputs (Pilát and Suchopárová 2022). Both these papers are a part of this thesis and will be discussed in more detail in the next part.

4.4 Surrogate Models and Neural Architecture Search

Predicting the quality of individuals in evolutionary algorithms is closely related to the problem of performance estimation in the area of neural architecture search. In neural architecture search, the goal if to find a neural network architecture that maximizes the performance of the model on a given dataset. Such problems can be solved by evolutionary algorithms or by reinforcement learning. In both cases, evaluating the quality of the neural architecture is a complex and slow task that requires training the neural network. Elsken,

Metzen, and Hutter (2019) and Y. Liu et al. (2021) published recent surveys of the area.

Many techniques were created in the recent years to predict the quality of the architectures without the need to fully train them. One of the options in this area is to train the model only for a few epochs and predicting its final performance based on this short training (Rawal and Miikkulainen 2018; Baker et al. 2017). Another option is to train the model only on a subset of the data (P. Liu et al. 2019).

While these papers do not call the techniques they use "surrogate models", it is obvious that the idea is similar. Especially the training for only a few epochs and predicting the rest of the learning curve seems quite similar to the idea of using phenotypic features to build surrogate models in genetic programming, in the sense that we use data that are more easily obtainable to predict the performance. We also believe that some of the techniques used in surrogate-based evolutionary algorithms can be usable for performance prediction in neural architecture search, and vice versa, some of the techniques used to speed up neural architecture search can be adapted to evolutionary algorithms.

In evolutionary neural architecture search, parallelization and caching is also a popular technique to speed up the algorithm.

Selected Papers

This part first describes the main contributions of each of the papers comprising this thesis together with later developments in the area of the given paper. The rest of this part then contains the selected papers organized by topics.

5 Brief Overview

In this chapter, we give a brief overview of the papers selected as the main part of this thesis. The main goal of the chapter is to put these papers in a context, mention some of the circumstances that led to the writing of the papers, explain my contribution to the papers, and, finally, discuss further developments after these papers were published.

The part is divided into four chapters, dealing with different topics. Chapter 6 (Parallelization) contains two papers discussing parallelization of evolutionary algorithms, one of them describes the idea of interleaving generations, while the other uses heterogenous island model to implement online parallel portfolio optimization algorithm. Chapter 7 (Surrogate Modelling) contains two papers implementing surrogate modelling in genetic programming. The first one performs feature selection from the trees and uses these features to train the surrogate models, the other one uses two different types of graph neural networks to create the models. Chapter 8 (User Preferences) contains a single paper that discusses user preferences in multi-objective optimization, its goal is to adaptively change the decomposition weights in the MOEA/D algorithm with the goal to focus the search to the areas of the search space that contain solutions interesting to the user. Finally, Chapter 9 (Applications) contains three papers that use evolutionary algorithms for optimization problems with expensive objective functions. In one of these papers, we use genetic programming in the area of AutoML to create whole machine learning pipelines, including pre-processing and ensembles. Another one uses simple neuro-evolution to coordinate the charging of electric vehicles. Finally, the last paper in this chapter shows the use of differential evolution to create adversarial examples for image classification.

5.1 Parallelization

Parallelization of evolutionary algorithms is an important step to make them more usable, especially nowadays, when

evolutionary algorithms are used for complex problems, like neural architecture search. Chapter 6 (Parallelization) contains two paper related to this topic; these are briefly described here.

5.1.1 Evolutionary Algorithm with Interleaving Generations

The first paper on the parallelization topic proposes an evolutionary algorithm with interleaving generations (Pilát and Neruda 2017). This paper focuses on problems with slow objective functions whose evaluation additionally takes variable amount of time. We mentioned (cf. Section 3.2) that in such cases using asynchronous evolutionary algorithms is beneficial. This type of algorithms does not wait for the whole generation to be evaluated, but rather creates new individuals as soon as one is evaluated (assuming enough parents are available and a free CPU is also available). However, we also mentioned that these algorithms are biased towards areas of the search space that evaluate faster.

In the paper, we first briefly demonstrate this bias and then offer a solution – an evolutionary algorithm with interleaving generations. We first carefully analyze two special versions of evolutionary algorithms (namely the (μ, μ) and $(\mu + \mu)$ algorithms with tournament selection) and then implement them in such a way that individuals from multiple generations can be evaluated at the same time. This idea is simpler to explain in the (μ, μ) case. In this case, there is no environmental selection, therefore any generated offspring in one generation is directly used in the mating selection in the next generation, and can be used as a parent. In order to decide which of these will actually be used as parents, we run the tournament selection – for that we need to randomly select two of these offspring; the better of them is the parent in the next generation. The pairs of (indices of) individuals that will be compared by the tournament selection can be generated before the population is available. Once both the individuals in the pair are evaluated, we can run the tournament selection and select the parent. Once both parents are selected, we can run the genetic operators to create new offspring in the next generation. The algorithm thus does not have to wait for the slowest individuals in each generation to finish evaluation. It generates new individuals in the next

generation as soon as both their parents are available. This leads to much better utilization of CPU resources and, at the same time, ensures that the algorithm is still equivalent to the generational version and does not suffer from the evaluation time bias. Similar analysis is also done for the $(\mu + \mu)$ version in the paper.

This work was later extended by Noguchi, Sonoda, et al. (2021), who added tentative evaluation of individuals to the algorithm. If there are free CPUs available, they generate offspring even before it is known which of the two possible individuals in the previous generation will become the parent. They generate both possible offspring and submit both of them for evaluation. Once the correct parent becomes known, one of these tentative offspring will not be needed and its evaluation is suspended. The other offspring should however finish the evaluation sooner than if the algorithm waited for both parents to become available. In another paper (Noguchi, Harada, and Thawonmas 2021), similar group of authors also created a version of differential evolution based on the same idea.

Independently of this research, I have also supervised a master thesis (Záboj 2020) that also considers the case of speculative creation of new offspring for evaluation. Additionally, it also considers the possibility to use surrogate models to decide which of the potential parents will be better according to the tournament selection. This should improve the utilization of CPU resources further by increasing the chance that the speculative individual will be indeed the correct one. We have, however, observed some discrepancies between the implemented algorithm and the generational one that should be equivalent. So far, we were unable to explain what causes them – it may be some bias caused by the use of surrogate models or a simply a bug in the rather complicated implementation. This is also the reason, why these results have not yet been published in a paper, but we intend to work on them further and once the issues are resolved, the work will be published.

My Contribution In this case, I was the author of the idea of interleaving generations and implemented the algorithm itself. Roman Neruda provided valuable discussion and insights during the work and feedback during the writing of the paper itself.

5.1.2 Heterogeneous Island Models

The other paper on the topic of parallelization deals with heterogeneous island models. In this paper (Balcar and Pilát 2020), we used the idea of dividing the population of the evolutionary algorithm into multiple sub-populations, similar to the way how island models work. In this case, however, each of the sub-population uses a completely different optimization algorithm and, at the same time, the types of optimization algorithms used by each of the islands change dynamically during the run of the algorithm. This essentially uses the framework of island models to implement an online parallel portfolio of optimization algorithms.

The implementation of the islands themselves is the same as in the island model used for parallelization of evolutionary algorithms – each of the islands run an optimization algorithm. These algorithms can however be different on each of the islands, they only need to share the encoding of the individuals. The migration operation is also implemented in order to allow the exchange of individuals among the islands. Additionally, the method also implements a central planner that observes the quality of the individuals shared by each of the islands running specific optimization methods and based on these observations decides, how to change the optimization algorithm on some of the islands for another one with the goal to improve the overall convergence speed of the algorithm. We implemented a number of different planners based on the number of different criteria and the paper compares them to homogeneous island models and static heterogeneous island models, demonstrating the advantages of the later one.

The paper is one of a small series of publications we had with Štěpán Balcar on this topic. The idea was first presented as a poster at GECCO 2018 (Balcar and Pilát 2018a), then as a full paper at ICTAI 2018 (Balcar and Pilát 2018b), where it was selected for the special issue of the Journal on Artificial Intelligence Tools (Balcar and Pilát 2020). This last paper is included this as it contains most of the information from the former publications and also some additional experiments that were allowed by the fact that the journal publication has no page limits, it thus gives the most complete picture of this area of research.

My contribution I am the author of the idea to use different optimization algorithms on different islands and changing the types of algorithms adaptively. I also proposed some of the planner types described in the paper and wrote most of the paper. Štěpán Balcar implemented the method as part of his master thesis and continued to extend the idea during the first year of his Ph.D. studies. He also proposed some of the types of the planners.

5.2 Surrogate Modelling

Chapter 7 (Surrogate Modelling) contains two related papers on using surrogate models in the area of genetic programming. Use of surrogate models in this area is still relatively uncommon compared to their use, for example, in continuous optimization. The main reason is that creating models that use trees and similar complex structures as inputs is harder and less common.

In the first paper (Pilát and Neruda 2016), we approached the problem of constructing the surrogate model by first extracting some features from the trees. We only considered features that can be extracted without any evaluation of the trees, as our main intended application was for problems, where the trees encode some structure, whose evaluation is costly. This was mainly inspired by our previous work, where we used genetic programming in automated machine learning (Křen, Pilát, and Neruda 2017b). The extracted features consist mainly of various statistics about the size of the tree, the frequency of how often each of the terminals and non-terminals is used, and what are the values of constants. Additionally, we also included the fitness of parents of each individual as one of the features. We then use random forests for regression to predict the quality of the individuals. It turns out that such models give around 0.5-0.6 Spearman's correlation, which is not perfect, but seems to be enough to guide the evolutionary algorithm and reduce the number of fitness evaluations required.

The other paper in this chapter (Pilát and Suchopárová 2022) describes our latest results in this area. It replaces the feature extraction and random forests by using graph neural networks. Graph neural networks are models that can operate directly on graph structures. In the paper, we used two types

of graph neural networks – graph isomorphism network (Xu et al. 2019) and treeLSTM (Tai, Socher, and Manning 2015). For both of these models, we need to specify node features. These encode what each of the nodes in the graph represents – in this case, they are a simple one-hot encoding of the terminals and non-terminals used in each of the nodes. Additionally, for constants, we also encode their values. Compared to the previous paper, graph neural networks can also capture the structure of the tree and thus provide better prediction of its quality. This paper was presented as a poster at GECCO 2022, and it was included in this thesis to demonstrate the current direction of our research in this area.

The model proposed in the first paper was later used by Bi, Xue, and M. Zhang (2021b) for feature learning in image classification. The area of using surrogate models in genetic programming is also becoming much more active and multiple new methods appeared in the recent years (F. Zhang et al. 2022; Bi, Xue, and M. Zhang 2021a).

My contribution For the first paper (Pilát and Neruda 2016), I came up with the idea to use surrogate models based on feature extraction in genetic programming, implemented the methods and tested them. Roman Neruda helped with valuable discussion and insights, and also gave feedback on the paper itself.

The idea of the other paper (Pilát and Suchopárová 2022) came to my mind after I reviewed the master thesis of Gabriela Suchopárová (Suchopárová 2021), where she studied graph neural networks for performance prediction in neural architecture search. I realized that similar techniques could be also used in genetic programming. After Gabriela started her Ph.D. studies at our department, we tested how well graph neural networks perform as surrogate models in genetic programming. Gabriela provided the implementation she used in her master thesis, and I implemented the treeLSTM-based model and ran all the experiments.

5.3 User Preferences

The paper included in Chapter 8 (User Preferences) improves the effectiveness of evolutionary algorithms in a different

way than those in the previous chapters. It does not reduce the number of fitness evaluations, nor does it run them in parallel, but it makes sure that the algorithm is looking for the solutions the user is interested in. In the paper (Pilát and Neruda 2015) we deal with the problem of incorporating user preferences into the MOEA/D algorithm. As we mentioned in Section 2.4 MOEA/D uses a set of weight vectors to decompose a multi-objective problem into a set of single-objective ones. These weight vectors directly affect which solutions of the multi-objective problem will be found. There had been some existing work on tuning the distribution of the weights in order to provide more uniform distribution of the solutions (Siwei et al. 2011; Qi et al. 2014).

Our goal with the paper, however, was not to fine-tune the distribution of the solutions, but rather to change the distribution in such a way that the algorithm respects user preferences. We assume the user (decision-maker) specifies which areas of the Pareto front are interesting for them. The algorithm then changes the weights is such a way, that the solutions found by the algorithm lie in this area. We used co-evolution for this changing of the weights. The algorithm thus has a population of solutions and a population of weight vectors. The population of solutions is evolved the same as in MOEA/D, the population of weight vectors is changed by a random Gaussian mutation and every weight is compared to its parent. To this end, we first assign each weight to an individual in the population (the one for which is gives the best fitness) and then the fitness of the weight vector is the distance of its assigned individual to the closest individual in the preferred region (this is minimized by the algorithm). If the assigned individual is already in the preferred region, then the fitness is defined is such a way to provide as uniform distribution of the preferred individuals as possible.

The paper described above is among the first to consider user preferences in MOEA/D. The area of incorporating user preferences to MOEA/D and multi-objective optimization algorithms in general has grown significantly since the paper was published (L. Li et al. 2018; Zhu et al. 2018; Fernández et al. 2022).

My contribution I proposed the idea of using co-evolution of weights to incorporate the user preferences, designed and tested the algorithm. Roman Neruda provided valuable

insights in discussion and feedback during the writing of the paper itself.

5.4 Applications

The last chapter of the thesis (Chapter 9) contains three papers that use evolutionary algorithms in order to solve expensive optimization problem.

5.4.1 Evolution of Machine Learning Pipelines

In the first paper (Křen, Pilát, and Neruda 2017b) we used genetic programming in order to design whole machine learning pipelines. At that point in time, automated machine learning was a relatively new area and only a few algorithms existed. Most of them were unable to create more complex machine learning pipelines, they typically only combined a few methods. For example, auto-sklearn (Feurer et al. 2015) uses Bayesian optimization to find a pipeline consisting of data preprocessor, feature preprocessor and classifier. Multiple of these can also be automatically combined into an ensemble.

Our goal was to create a system, that could (at least in principle) generate any machine learning pipeline, combining any preprocessing and machine learning steps together and, at the same time, automatically creating ensembles of these methods. In such systems, it is essential to make sure that any pipeline created by the system makes sense from the machine learning point of view. For example, preprocessing should not be run after machine learning, but rather before, and if data are split into several groups, these groups should be combined again later. To this end, we used typed genetic programming, with the types ensuring that the pipelines make sense. In addition to selecting the methods, we also need to select their hyper-parameters – this could however be solved quite simply in the genetic programming.

A similar system to ours, called T-POT, was also created independently at roughly the same time (Olson et al. 2016). T-POT also uses genetic programming and creates pipelines with tree-like structure, where data flow from leaves to the root. In each internal node, the data are merged together,

potentially with predictions provided by machine learning methods in the sub-trees. Compared to our system, T-POT cannot split data and the pipelines produced by T-POT are not as expressive as the ones produced by our system. Since then, the field of automated machine learning has become one of the main fields in machine learning and especially deep learning (in the form of neural architecture search), with many new algorithms being proposed in the last few years.

The paper presented in this thesis is actually one of a series of four papers we published in this area. The first iteration of the method was presented at SSCI 2015 (Křen, Pilát, and Neruda 2015), later improvements were published at ICTAI 2016 (Pilát, Křen, and Neruda 2016) (this one added, among other things, asynchronous evolution to speed up the optimization). The ICTAI paper was selected for the special issue of the International Journal on Artificial Intelligence Tools (Křen, Pilát, and Neruda 2017b). Later, we also published a version of the paper using multi-objective optimization to also minimize the size of the pipelines or their training time (Křen, Pilát, and Neruda 2017a). I selected the journal paper for this thesis as it contains the most detailed description of the system thanks to the journal not having any page limits.

This research also strongly affected our other research. For example, the time-consuming nature of needing to train the machine learning pipelines in order to evaluate fitness led us to the work on surrogate models in genetic programming. At the same time, the training time for the pipelines is variable, which motivated us to explore the asynchronous evolutionary algorithms and led to the evolutionary algorithm with interleaving generations. Both of these directions were described above and form a part of this thesis.

My contribution In this work, I implemented the evaluation of the machine learning workflows and designed the asynchronous evolutionary algorithms. I also assisted Tomáš Křen with the design of the type system, mainly from the machine learning point of view (what must hold, so that the pipeline makes sense), but most of the work on the type system itself was done by him. Roman Neruda has been the supervisor of Tomáš during his Ph.D.

5.4.2 Coordination of the Charging of Electric Vehicles

Coordination of charging of electric vehicles is an important problem that whose importance only increases nowadays with the number of electric vehicles quickly growing. Given that most people have similar schedule, if everyone started charging their car once they come home in the afternoon, the electrical grid would be quickly overloaded. Therefore, we worked on algorithms that would coordinate the charging – the user would plug the vehicle in the charger and set a time, when the car needs to be fully charged (Pilát 2018a). The charger would then decide when and how quickly the vehicle would charge with the goal to smoothen the consumptions during the day and night.

Algorithms for this problem have existed, however, most of them either required a central planner (Clement, Haesen, and Driesen 2009), or they required some communication with third parties (Gan, Topcu, and Low 2013; Ma, Callaway, and Hiskens 2013). This could lead to loss of privacy for the users. Therefore, we wanted to create an algorithm that would solve the problem without any outside communication from the household. To this end, we designed a controller based on a small neural network. This network would get information about the current electricity consumption in the household (and in one version also in the whole neighborhood) and information about the charging request (how much and how quickly needs to be charged). The output from the network would then be the charging speed for the next 15 minutes. The weights in the neural network were set using either an evolutionary algorithm or a gradient-based optimization that used numerically computed gradient. The goal was to minimize the variance of the consumption of the whole neighborhood. The consumption during the day was computed using simulation of multiple households for several days.

The results of this work were actually first presented in an EvoApplications 2018 short paper (Pilát 2018b), but the main result is from IJCNN 2018 (Pilát 2018a). The ideas were also later extended by using imitation learning instead of direct optimization and present at ICTAI 2021 (Pilát 2021). In this paper, the networks were trained to imitate the optimal charging schedule to a relaxed version of the problem. In the

relaxation, we assumed that all the requests and electricity consumptions are known in advance. I selected the IJCNN 2018 paper as part of this thesis, as it contains full description of the problem and algorithm and in contrast to the newer ICTAI 2021 paper it uses evolutionary algorithms, and thus is closer to the main topic of this thesis.

5.4.3 Adversarial Examples in Image Classification

The last paper included in this thesis (Kumová and Pilát 2021) is an application of differential evolution to the problem of finding adversarial examples for image classification. Adversarial examples are an interesting phenomenon observed first in the area of deep learning. Szegedy et al. (2013) observed that the image classification models are sensitive to small, specially crafted, perturbations in the input images. These perturbations are typically invisible to humans when looking at the image, but they are able to cause the deep learning model to classify the input incorrectly, even though the original image was classified correctly. Over the years following this discovery, there were many attempts to create more robust machine learning models and at the same time to create adversarial examples that would be able to fool these models.

In the paper, we focused on finding adversarial examples for models that already used some form of defense. At that time, the defense proposed by Hu et al. (2019) showed that it can detect adversarial examples created by several white-box attacks. White-box attacks use the knowledge of the complete structure of the neural network, including its parameters to find the adversarial example. In our paper, we have shown that black-box attacks (those that rely only on the outputs of the network) can actually quite easily beat this specific defense and create adversarial examples that it cannot detect. The attack is based on differential evolution that directly finds the perturbation of the input image that causes the network to classify it incorrectly. In order to lower the dimensionality of the problem, we have used a tiling trick – instead of searching for a perturbation of every pixel, the image is divided into areas of $k \times k$ pixels and the same change is applied to all of them.

My Contribution I was the supervisor of the master thesis written by Věra Kumová (Kumová 2021). I helped mostly with consultations related to adversarial examples and differential evolution in general. I also prepared the paper for publication based on the experiments in her thesis. However, the idea to use the tiling trick is Věra's, she also implemented and tested the algorithm.

6 Parallelization

This chapter contains the papers:

Martin Pilát, Roman Neruda (2017): 'Parallel Evolutionary Algorithm with Interleaving Generations'. In: *Proceedings of the Genetic and Evolutionary Computation Conference. GECCO 2017.* Berlin, Germany: Association for Computing Machinery, pp. 865-872. DOI: 10.1145/3071178.3071309

Štěpán Balcar, Martin Pilát (2020). 'Heterogeneous Island Models and Their Application to Recommender Systems and Electric Vehicle Charging'. In: *International Journal on Artificial Intelligence Tools* 29.03n04, World Scientific, p. 2060010. DOI: 10.1142/S0218213020600106

7 Surrogate Modelling

This chapter contains the papers:

Martin Pilát, Roman Neruda (2016). 'Feature Extraction for Surrogate Models in Genetic Programming'. In: *Parallel Problem Solving from Nature - PPSN XIV.* Ed. by Julia Handl et al. Cham: Springer International Publishing, pp. 335-344. DOI: 10.1007/978-3-319-45823-6_31

Martin Pilát, Gabriela Suchopárová (2022). 'Using Graph Neural Networks as Surrogate Models in Genetic Programming'. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion. GECCO 2022.* Boston, Massachusetts: Association for Computing Machinery, pp. 582-585. DOI: 10.1145/3520304.3529024

8 User Preferences

This chapter contains the paper:

Martin Pilát, Roman Neruda (2015). 'Incorporating User Preferences in MOEA/D through the Coevolution of Weights'. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. GECCO 2015.* Madrid, Spain: Association for Computing Machinery, pp. 727-734. DOI: 10.1145/2739480.2754801

9 Applications

This chapter contains the papers:

Tomáš Křen, Martin Pilát, and Roman Neruda (2017). 'Automatic Creation of Machine Learning Workflows with Strongly Typed Genetic Programming'. In: *International Journal on Artificial Intelligence Tools* 26.05, World Scientific, p. 1760020. DOI: 10.1142/S021821301760020X

Martin Pilát (2018). 'Controlling the Charging of Electric Vehicles with Neural Networks'. In: *2018 International Joint Conference on Neural Networks (IJCNN 2018)*, IEEE, pp. 1-8. DOI: 10.1109/IJCNN.2018.8489027

Věra Kumová, Martin Pilát (2021). 'Beating White-Box Defenses with Black-Box Attacks'. In: *2021 International Joint Conference on Neural Networks (IJCNN 2021)*, IEEE, pp. 1-8. DOI: 10.1109/IJCNN52387.2021.9533772

10 Conclusion

We have discussed evolutionary algorithms for optimization problems with expensive objective functions. We have shown two techniques how to make them faster – parallelization and surrogate modelling, and we have presented our results in these areas.

Regarding parallelization, we presented an evolutionary algorithm with interleaving generations (Pilát and Neruda 2017) that allows for better parallelization of evolutionary algorithms while avoiding the evaluation time bias inherent to asynchronous evolutionary algorithms. We have also discussed heterogeneous island models that were used as parallel portfolios of optimization algorithms with automated algorithm selection (Balcar and Pilát 2020).

In surrogate modelling, we discussed two techniques for implementation of surrogate models in genetic programming. One of them was based on features statically extracted from the genetic programs (Pilát and Neruda 2016), the other was based on modern graph neural networks (Pilát and Suchopárová 2022). Both these techniques can predict the quality of the individual in genetic programming without the need to evaluate the program itself. This makes them useful in problems, where the program encodes a complex structure that cannot be easily evaluated, such as in automated machine learning.

Additionally, we have presented a multi-objective evolutionary algorithm that uses co-evolution in order to consider user preferences (Pilát and Neruda 2015). This makes the algorithm more effective, as computation is not wasted on individuals that are not interesting for the user.

We have also shown three different applications of evolutionary algorithms in problems with complex objective functions. In one of them, we used evolutionary algorithms in the area of automated machine learning (Křen, Pilát, and Neruda 2017b), in another, we used it to implement controllers for coordinated charging of electric vehicles (Pilát 2018a), and in the last one we use differential evolution to find adversarial examples in image classification with deep neural networks

(Kumová and Pilát 2021). These applications are important not only by themselves, but they also help to steer other parts of our research. This is most obvious with the one in automated machine learning. The problems we encountered while working on this task led us to create the algorithm with interleaving generations and also to the work on surrogate models in genetic programming.

10.1 Future Research

In the future, we would still like to focus on evolutionary algorithms for problems with expensive objective functions. We believe that such problems will become even more common with evolutionary algorithms finding new applications in automated machine learning and neural architecture search – two very quickly developing areas.

We will still work on surrogate models, and we are especially interested in using surrogate models in areas other than continuous optimization, especially in genetic programming. We have presented some of the newer results using graph neural networks in this thesis, and we would like to extend them further. We also believe that ideas from surrogate modelling are transferable to the area of performance prediction in automated machine learning and neural architecture search.

Parallel implementations of evolutionary algorithms are another area, we find interesting, especially in cases where the fitness evaluation time is long and variable. So far, we have explored mostly single-objective algorithms for problems of this nature, but we would like to extend some of these ideas to multi-objective optimization. In this case, the situation is even more interesting as some of the objectives may be fast to evaluate, while others are slow.

Finally, we will also focus on applications of evolutionary algorithms especially in areas with expensive objective functions. We are mostly interested in machine learning and problems such as automated machine learning, neural architecture search, and adversarial examples. We believe these applications will be able to guide our future research in evolutionary algorithms.

Appendix

Bibliography

Here are the references in alphabetical order.

Baker, Bowen et al. (2017). 'Accelerating Neural Architecture Search using Performance Prediction'. In: DOI: 10.48550/ARXIV.1705.10823 (cited on page 25).

Balcar, Štěpán and Martin Pilát (2018a). 'Heterogeneous Island Model with Re-Planning of Methods'. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '18. Kyoto, Japan: Association for Computing Machinery, pp. 245–246. DOI: 10.1145/3205651.3205786 (cited on page 32).

– (2018b). 'Online Parallel Portfolio Selection with Heterogeneous Island Model'. In: *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, pp. 757–764. DOI: 10.1109/ICTAI.2018.00119 (cited on page 32).

– (2020). 'Heterogeneous Island Models and Their Application to Recommender Systems and Electric Vehicle Charging'. In: *International Journal on Artificial Intelligence Tools* 29.03n04, p. 2060010. DOI: 10.1142/S0218213020600106 (cited on pages 32, 139).

Bartz-Beielstein, Thomas and Martin Zaefferer (2017). 'Model-based methods for continuous and discrete global optimization'. In: *Applied Soft Computing* 55, pp. 154–167. DOI: 10.1016/j.asoc.2017.01.039 (cited on page 23).

Bi, Ying, Bing Xue, and Mengjie Zhang (2021a). 'Instance Selection-Based Surrogate-Assisted Genetic Programming for Feature Learning in Image Classification'. In: *IEEE Transactions on Cybernetics*, pp. 1–15. DOI: 10.1109/TCYB.2021.3105696 (cited on page 34).

– (2021b). 'Random Forest-Assisted GP for Feature Learning'. In: *Genetic Programming for Image Classification: An Automated Approach to Feature Learning*. Cham: Springer International Publishing, pp. 207–226. DOI: 10.1007/978-3-030-65927-1_9 (cited on page 34).

Cheng, John Runwei and Mitsuo Gen (2019). 'Accelerating genetic algorithms with GPU computing: A selective overview'. In: *Computers & Industrial Engineering* 128, pp. 514–525. DOI: 10.1016/j.cie.2018.12.067 (cited on page 20).

Clement, Kristien, Edwin Haesen, and Johan Driesen (2009). 'Coordinated charging of multiple plug-in hybrid electric vehicles in residential distribution grids'. In: *2009 IEEE/PES Power Systems Conference and Exposition*. IEEE, pp. 1–7. DOI: 10.1109/PSCE.2009.4839973 (cited on page 38).

Deb, K. et al. (2002). 'A fast and elitist multiobjective genetic algorithm: NSGA-II'. In: *IEEE Transactions on Evolutionary Computation* 6.2, pp. 182–197. DOI: 10.1109/4235.996017 (cited on page 13).

Eiben, A E and James E Smith (2015). *Introduction to evolutionary computing*. 2nd ed. Natural computing series. Berlin, Germany: Springer (cited on page 7).

Elsken, Thomas, Jan Hendrik Metzen, and Frank Hutter (2019). 'Neural Architecture Search: A Survey'. In: *Journal of Machine Learning Research* 20.55, pp. 1–21 (cited on page 24).

Fernández, Eduardo et al. (2022). 'Preference incorporation in MOEA/D using an outranking approach with imprecise model parameters'. In: *Swarm and Evolutionary Computation* 72, p. 101097. DOI: 10.1016/j.swevo.2022.101097 (cited on page 35).

Feurer, Matthias et al. (2015). 'Efficient and Robust Automated Machine Learning'. In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc. (cited on page 36).

Gan, Lingwen, Ufuk Topcu, and Steven H. Low (2013). 'Optimal decentralized protocol for electric vehicle charging'. In: *IEEE Transactions on Power Systems* 28.2, pp. 940–951. DOI: 10.1109/TPWRS.2012.2210288 (cited on page 38).

Gong, Yiyuan and Alex Fukunaga (2011). 'Distributed island-model genetic algorithms using heterogeneous parameter settings'. In: *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE, pp. 820–827. DOI: 10.1109/CEC.2011.5949703 (cited on page 18).

Hansen, Nikolaus, Dirk V. Arnold, and Anne Auger (2015). 'Evolution Strategies'. In: *Springer Handbook of Computational Intelligence*. Ed. by Janusz Kacprzyk and Witold Pedrycz. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 871–898. DOI: 10.1007/978-3-662-43505-2_44 (cited on page 10).

He, Xin, Kaiyong Zhao, and Xiaowen Chu (2021). 'AutoML: A survey of the state-of-the-art'. In: *Knowledge-Based Systems* 212, p. 106622. DOI: 10.1016/j.knosys.2020.106622 (cited on page 19).

Hildebrandt, Torsten and Jürgen Branke (Sept. 2015). 'On Using Surrogates with Genetic Programming'. In: *Evolutionary Computation* 23.3, pp. 343–367. DOI: 10.1162/EVCO_a_00133 (cited on page 24).

Hu, Shengyuan et al. (2019). 'A New Defense Against Adversarial Images: Turning a Weakness into a Strength'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc. (cited on page 39).

Ishibuchi, Hisao, Noritaka Tsukamoto, and Yusuke Nojima (2008). 'Evolutionary many-objective optimization: A short review'. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, pp. 2419–2426. DOI: 10.1109/CEC.2008.4631121 (cited on page 13).

Jaros, Jiri (2012). 'Multi-GPU island-based genetic algorithm for solving the knapsack problem'. In: *2012 IEEE Congress on Evolutionary Computation*. IEEE, pp. 1–8. DOI: 10.1109/CEC.2012.6256131 (cited on page 20).

Jin, Yaochu (2011). 'Surrogate-assisted evolutionary computation: Recent advances and future challenges'. In: *Swarm and Evolutionary Computation* 1.2, pp. 61–70. DOI: 10.1016/j.swevo.2011.05.001 (cited on pages 21–23).

Joachims, Thorsten (2002). 'Optimizing Search Engines Using Clickthrough Data'. In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '02. Edmonton, Alberta, Canada: Association for Computing Machinery, pp. 133–142. DOI: 10.1145/775047.775067 (cited on page 22).

Jones, Donald R., Matthias Schonlau, and William J. Welch (Dec. 1998). 'Efficient Global Optimization of Expensive Black-Box Functions'. In: *Journal of Global Optimization* 13.4, pp. 455–492. DOI: 10.1023/A:1008306431147 (cited on page 23).

Knowles, J. (2006). 'ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems'. In: *IEEE Transactions on Evolutionary Computation* 10.1, pp. 50–66. DOI: 10.1109/TEVC.2005.851274 (cited on page 23).

Křen, Tomáš, Martin Pilát, and Roman Neruda (2015). 'Evolving Workflow Graphs Using Typed Genetic Programming'. In: *2015 IEEE Symposium Series on Computational Intelligence*. IEEE, pp. 1407–1414. DOI: 10.1109/SSCI.2015.200 (cited on page 37).

– (2017a). 'Multi-objective evolution of machine learning workflows'. In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, pp. 1–8. DOI: 10.1109/SSCI.2017.8285357 (cited on page 37).

– (2017b). 'Automatic Creation of Machine Learning Workflows with Strongly Typed Genetic Programming'. In: *International Journal on Artificial Intelligence Tools* 26.05, p. 1760020. DOI: 10.1142/S021821301760020X (cited on pages 12, 33, 36, 37, 139).

Krömer, Pavel et al. (2011). 'Many-Threaded Implementation of Differential Evolution for the CUDA Platform'. In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO '11. Dublin, Ireland: Association for Computing Machinery, pp. 1595–1602. DOI: 10.1145/2001576.2001791 (cited on page 20).

Kumová, Věra (2021). 'Creating Adversarial Examples in Machine Learning'. Supervisor: Martin Pilát. MA thesis. Charles University, Faculty of Mathematics and Physics (cited on page 40).

Kumová, Věra and Martin Pilát (2021). 'Beating White-Box Defenses with Black-Box Attacks'. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8. DOI: 10.1109/IJCNN52387.2021.9533772 (cited on pages 39, 140).

Li, Longmei et al. (2018). 'Integrating region preferences in multiobjective evolutionary algorithms based on decomposition'. In: *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, pp. 379–384. DOI: 10.1109/ICACI.2018.8377488 (cited on page 35).

Liu, Peng et al. (2019). 'Deep Evolutionary Networks with Expedited Genetic Algorithms for Medical Image Denoising'. In: *Medical Image Analysis* 54, pp. 306–315. DOI: 10.1016/j.media.2019.03.004 (cited on page 25).

Liu, Yuqiao et al. (2021). 'A Survey on Evolutionary Neural Architecture Search'. In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21. DOI: 10.1109/TNNLS.2021.3100554 (cited on page 25).

Loshchilov, Ilya, Marc Schoenauer, and Michèle Sebag (2010). 'A Mono Surrogate for Multiobjective Optimization'. In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. GECCO '10. Portland, Oregon, USA: Association for Computing Machinery, pp. 471–478. DOI: 10.1145/1830483.1830571 (cited on page 23).

Ma, Zhongjing, Duncan S. Callaway, and Ian A. Hiskens (2013). 'Decentralized Charging Control of Large Populations of Plug-in Electric Vehicles'. In: *IEEE Transactions on Control Systems Technology* 21.1, pp. 67–78. DOI: 10.1109/TCST.2011.2174059 (cited on page 38).

Mambrini, Andrea, Dirk Sudholt, and Xin Yao (2012). 'Homogeneous and Heterogeneous Island Models for the Set Cover Problem'. In: *Parallel Problem Solving from Nature - PPSN XII*. Ed. by Carlos A. Coello Coello et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 11–20. DOI: 10.1007/978-3-642-32937-1_2 (cited on page 18).

McKay, M. D., R. J. Beckman, and W. J. Conover (1979). 'A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code'. In: *Technometrics* 21.2, pp. 239–245. DOI: 10.2307/1268522 (cited on page 21).

Michalewicz, Zbigniew and David B Fogel (2004). *How to solve it: Modern heuristics*. 2nd ed. Berlin, Germany: Springer (cited on page 7).

Noguchi, Hayato, Tomohiro Harada, and Ruck Thawonmas (2021). 'Parallel Differential Evolution Applied to Interleaving Generation with Precedence Evaluation of Tentative Solutions'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '21. Lille, France: Association for Computing Machinery, pp. 706–713. DOI: 10.1145/3449639.3459337 (cited on page 31).

Noguchi, Hayato, Akari Sonoda, et al. (2021). 'Improving CPU utilization of interleaving generation parallel evolutionary algorithm with precedence evaluation of tentative solutions and their suspension'. In: *SICE Journal of Control, Measurement, and System Integration* 14.1, pp. 242–256. DOI: 10.1080/18824889.2021.1972386 (cited on page 31).

Olson, Randal S. et al. (2016). 'Automating Biomedical Data Science Through Tree-Based Pipeline Optimization'. In: *Applications of Evolutionary Computation*. Ed. by Giovanni Squillero and Paolo Burelli. Cham: Springer International Publishing, pp. 123–137. DOI: 10.1007/978-3-319-31204-0_9 (cited on page 36).

Paszke, Adam et al. (2019). 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035 (cited on page 20).

Pilát, Martin (2018a). 'Controlling the Charging of Electric Vehicles with Neural Networks'. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8. DOI: 10.1109/IJCNN.2018.8489027 (cited on pages 38, 139).

– (2018b). 'Evolving Controllers for Electric Vehicle Charging'. In: *Applications of Evolutionary Computation*. Ed. by Kevin Sim and Paul Kaufmann. Cham: Springer International Publishing, pp. 247–255. DOI: 10.1007/978-3-319-77538-8_18 (cited on page 38).

– (2021). 'Training Electric Vehicle Charging Controllers with Imitation Learning'. In: *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, pp. 674–681. DOI: 10.1109/ICTAI52525.2021.00107 (cited on page 38).

Pilát, Martin, Tomáš Křen, and Roman Neruda (2016). 'Asynchronous Evolution of Data Mining Workflow Schemes by Strongly Typed Genetic Programming'. In: *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, pp. 577–584. DOI: 10.1109/ICTAI.2016.0094 (cited on page 37).

Pilát, Martin and Roman Neruda (2010). 'Combining multiobjective and single-objective genetic algorithms in heterogeneous island model'. In: *2010 IEEE Congress on Evolutionary Computation*. IEEE, pp. 1–8. DOI: 10.1109/CEC.2010.5586075 (cited on page 18).

– (2011). 'ASM-MOMA: Multiobjective memetic algorithm with aggregate surrogate model'. In: *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE, pp. 1202–1208. DOI: 10.1109/CEC.2011.5949753 (cited on page 23).

– (2015). 'Incorporating User Preferences in MOEA/D through the Coevolution of Weights'. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO '15. Madrid, Spain: Association for Computing Machinery, pp. 727–734. DOI: 10.1145/2739480.2754801 (cited on pages 35, 139).

– (2016). 'Feature Extraction for Surrogate Models in Genetic Programming'. In: *Parallel Problem Solving from Nature – PPSN XIV*. Ed. by Julia Handl et al. Cham: Springer International Publishing, pp. 335–344. DOI: 10.1007/978-3-319-45823-6_31 (cited on pages 24, 33, 34, 139).
– (2017). 'Parallel Evolutionary Algorithm with Interleaving Generations'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '17. Berlin, Germany: Association for Computing Machinery, pp. 865–872. DOI: 10.1145/3071178.3071309 (cited on pages 30, 139).

Pilát, Martin and Gabriela Suchopárová (2022). 'Using Graph Neural Networks as Surrogate Models in Genetic Programming'. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '22. Boston, Massachusetts: Association for Computing Machinery, pp. 582–585. DOI: 10.1145/3520304.3529024 (cited on pages 24, 33, 34, 139).

Poli, Riccardo, William B. Langdon, and Nicholas Freitag McPhee (2008). *A field guide to genetic programming*. Freely available at `http://www.gp-field-guide.org.uk` and publised via `http://lulu.com`. (cited on page 11).

Qi, Yutao et al. (2014). 'MOEA/D with Adaptive Weight Adjustment'. In: *Evolutionary Computation* 22.2, pp. 231–264. DOI: 10.1162/EVCO_a_00109 (cited on page 35).

Rawal, Aditya and Risto Miikkulainen (2018). 'From Nodes to Networks: Evolving Recurrent Neural Networks'. In: DOI: 10.48550/ARXIV.1803.04439 (cited on page 25).

Scott, Eric O. and Kenneth A. De Jong (2015). 'Evaluation-Time Bias in Asynchronous Evolutionary Algorithms'. In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO Companion '15. Madrid, Spain: Association for Computing Machinery, pp. 1209–1212. DOI: 10.1145/2739482.2768482 (cited on page 19).

Siwei, Jiang et al. (2011). 'Multiobjective optimization by decomposition with Pareto-adaptive weight vectors'. In: *2011 Seventh International Conference on Natural Computation*. Vol. 3. IEEE, pp. 1260–1264. DOI: 10.1109/ICNC.2011.6022367 (cited on page 35).

Storn, Rainer and Kenneth Price (Dec. 1997). 'Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces'. In: *Journal of Global Optimization* 11.4, pp. 341–359. DOI: 10.1023/A:1008202821328 (cited on page 10).

Suchopárová, Gabriela (2021). 'Graph neural networks for NAS performance prediction'. Supervisor: Roman Neruda. MA thesis. Charles University, Faculty of Mathematics and Physics (cited on page 34).

Sudholt, Dirk (2015). 'Parallel Evolutionary Algorithms'. In: *Springer Handbook of Computational Intelligence*. Ed. by Janusz Kacprzyk and Witold Pedrycz. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 929–959. DOI: 10.1007/978-3-662-43505-2_46 (cited on page 17).

Szegedy, Christian et al. (2013). 'Intriguing properties of neural networks'. In: arXiv. DOI: 10.48550/ARXIV.1312.6199 (cited on page 39).

Tai, Kai Sheng, Richard Socher, and Christopher D. Manning (July 2015). 'Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks'. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long*

Papers). Beijing, China: Association for Computational Linguistics, pp. 1556–1566. DOI: 10.3115/v1/P15-1150 (cited on page 34).

TensorFlow Developers (May 2022). *TensorFlow*. Version v2.8.2. DOI: 10.5281/zenodo.6574269 (cited on page 20).

Tong, Hao et al. (2021). 'Surrogate models in evolutionary single-objective optimization: A new taxonomy and experimental study'. In: *Information Sciences* 562, pp. 414–437. DOI: 10.1016/j.ins.2021.03.002 (cited on page 21).

Valkovič, Patrik and Martin Pilát (2022). 'Implementing and Evaluating Parallel Evolutionary Algorithms in Modern GPU Computing Libraries'. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '22. Boston, Massachusetts: Association for Computing Machinery, pp. 506–509. DOI: 10.1145/3520304.3529000 (cited on page 20).

Xu, Keyulu et al. (2019). 'How Powerful are Graph Neural Networks?' In: *International Conference on Learning Representations* (cited on page 34).

Záboj, Petr (2020). 'Efektivní paralelizace evolučních algoritmů'. (in Czech), Supervisor: Martin Pilát. MA thesis. Charles University, Faculty of Mathematics and Physics (cited on page 31).

Zhang, Fangfang et al. (2022). 'Instance Rotation Based Surrogate in Genetic Programming with Brood Recombination for Dynamic Job Shop Scheduling'. In: *IEEE Transactions on Evolutionary Computation*, pp. 1–1. DOI: 10.1109/TEVC.2022.3180693 (cited on page 34).

Zhang, Qingfu and Hui Li (2007). 'MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition'. In: *IEEE Transactions on Evolutionary Computation* 11.6, pp. 712–731. DOI: 10.1109/TEVC.2007.892759 (cited on page 14).

Zhu, Xinqi et al. (2018). 'A decomposition-based multi-objective optimization approach considering multiple preferences with robust performance'. In: *Applied Soft Computing* 73, pp. 263–282. DOI: 10.1016/j.asoc.2018.08.029 (cited on page 35).