**FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE**

# ASSIGNMENT OF BACHELOR'S THESIS

| | |
|---|---|
| **Title:** | Grammar interactive evolution of graph interfaces |
| **Student:** | Petr Hanzl |
| **Supervisor:** | doc. Ing. Pavel Kordík, Ph.D. |
| **Study Programme:** | Informatics |
| **Study Branch:** | Knowledge Engineering |
| **Department:** | Department of Applied Mathematics |
| **Validity:** | Until the end of winter semester 2019/20 |

## Instructions

Explore the field of interactive evolution and grammar evolution. Adapt the grammar evolution to evolve description of visual interfaces. The fitness should be computed from user feedback (see. the Picbreeder project). Implement a prototype and demonstrate its functionality using at least three synthetic and real world use cases.

## References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 16, 2018

Czech Technical University in Prague

Faculty of Information Technology

Department of Applied Mathematics

Bachelor's thesis

# Grammar interactive evolution of graph interfaces

***Petr Hanzl***

Supervisor: doc. Ing. Pavel Kordík, Ph.D.

16th of May, 2019

# Acknowledgements

I wish to express my sicere thanks to my supervisor doc. Ing. Pavel Kordík, Ph.D. for the continuos encouragement.

I also thank my whole family, especially my parents for the support and attention.

# Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on 16th of May, 2019 ....................

Czech Technical University in Prague

Faculty of Information Technology

© 2019 Petr Hanzl. All rights reserved.

*This thesis is a school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

## Citation of this thesis

HANZL, Petr. *Grammar interactive evolution of graph interfaces*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019. Available also from WWW: ⟨`https://github.com/Lznah/thesis`⟩.

# Abstrakt

Tato bakalářská práce se zabývá možnostmi využití evolučních algoritmů pro generování grafických rozhraní na základě uživatelových preferencí. Tato grafická rozhraní mohou být popsána gramatikou, což je sada pravidel, která umožňuje popsat všechna jejich možná nastavení.

Dalším cílem je využítí zjištěných znalostí k vytvoření prototypu, který bude generovat grafové vizualizace nad danými datasety.

Na vytvořeném prototypu byly provedeny měření počtu iterací na vytvoření požadované vizualizace. Jelikož jsou interaktivní evoluční algoritmy závislé na náhodě a na uživatelových preferencích, nelze jasně říci, že by počet dosažených iterací měl vyšší vypovídající hodnotu. Tudíž měření slouží spíše jako proof of concept.

**Klíčová slova** Interaktivní evoluční výpočetní technika, Interaktivní tvorba rozhraní, Interaktivní genetické programování

# Abstract

This bachelor thesis investigates the utilization of evolutionary algorithms for generating graphical interfaces, from a user's preferences. These graphical interfaces can be described by grammar. Grammar is a set of rules that describes all of the feasible settings of the graphical interfaces.

Additionally, a prototype was created from the information obtained in the initial investigation. The prototype was designed to generate graph visualizations for given datasets. The prototype measures the number of iterations that were used to produce the desired visualization.

The number of iterations does not have any higher meaning as the interactive evolutionary algorithms are dependent on randomness and the user's preferences. Although, the measurements act as a proof-of-concept, indicating that the prototype is functioning as intended.

**Keywords** Interactive evolutionary computation, Interactive interfaces generation, Interactive Genetic Programming

# Contents

**Introduction** 23

**1 State of the art** 25

**2 Evolutionary Computation (EC)** 27

2.1 Evolutionary Algorithm (EA) . . . . . . . . . . . . . . . . . . . . . 27
2.2 Types of Evolutionary Algorithms . . . . . . . . . . . . . . . . . . 28
2.3 Individual's Representation . . . . . . . . . . . . . . . . . . . . . . 30
2.3.1 String . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
2.3.2 Tree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
2.4 Evolutionary Operators . . . . . . . . . . . . . . . . . . . . . . . . 31
2.4.1 Initialization . . . . . . . . . . . . . . . . . . . . . . . . . 31
2.4.2 Selection . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
2.4.3 Mutation . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
2.4.4 Recombination . . . . . . . . . . . . . . . . . . . . . . . . 35

**3 Interactive Evolutionary Computation (IEC)** 37

3.1 Differences of IEC to EC . . . . . . . . . . . . . . . . . . . . . . . 38
3.2 User fatique . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39

**4 Grammar evolution of graph interfaces** 41

4.1 Formal grammar . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
4.1.1 Definition . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
4.1.2 Vocabulary . . . . . . . . . . . . . . . . . . . . . . . . . . 42

4.1.3 Chomsky hiearchy . . . . . . . . . . . . . . . . . . . . 42
4.1.4 Backus–Naur Form . . . . . . . . . . . . . . . . . . . . 43
4.2 Evolutionary algorithms for evolving grammars . . . . . . . . . 44
4.2.1 Genetic Programming . . . . . . . . . . . . . . . . . . . 44
4.2.2 Grammatical Evolution . . . . . . . . . . . . . . . . . . 44
4.3 Grammar of graph interfaces . . . . . . . . . . . . . . . . . . . 46
4.3.1 Vega-Lite . . . . . . . . . . . . . . . . . . . . . . . . . 46

**5 Implementation of prototype** **53**
5.1 Goals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53
5.2 Implemented EA . . . . . . . . . . . . . . . . . . . . . . . . 54
5.2.1 Encoding of grammar . . . . . . . . . . . . . . . . . . . 54
5.2.2 Initialization . . . . . . . . . . . . . . . . . . . . . . . 57
5.2.3 Mutation . . . . . . . . . . . . . . . . . . . . . . . . . 57
5.2.4 Recombination . . . . . . . . . . . . . . . . . . . . . . 57
5.2.5 Selection . . . . . . . . . . . . . . . . . . . . . . . . . 58
5.3 Heuristic approaches . . . . . . . . . . . . . . . . . . . . . . 58
5.3.1 Datatypes . . . . . . . . . . . . . . . . . . . . . . . . . 58
5.3.2 Number of mapping in a graph . . . . . . . . . . . . . . 58
5.3.3 Forbidden transformation of nominal features . . . . . . 59
5.3.4 X and Y axes . . . . . . . . . . . . . . . . . . . . . . . 59
5.4 Description of GUI . . . . . . . . . . . . . . . . . . . . . . . 59
5.4.1 Left Column Window . . . . . . . . . . . . . . . . . . . 59
5.4.2 Main Window . . . . . . . . . . . . . . . . . . . . . . . 60

**6 Experiments** **63**
6.1 Experiment 1: Students performance in Exams . . . . . . . . . . 63
6.1.1 Goal . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
6.1.2 Evolution . . . . . . . . . . . . . . . . . . . . . . . . . 64
6.1.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . . 67
6.2 Experiment 2: Heart Diseases . . . . . . . . . . . . . . . . . . 68
6.2.1 Goal . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
6.2.2 Evolution . . . . . . . . . . . . . . . . . . . . . . . . . 68
6.2.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . . 68
6.3 Experiment 3: Graduate Admission . . . . . . . . . . . . . . . . 71
6.3.1 Goal . . . . . . . . . . . . . . . . . . . . . . . . . . . 71

6.3.2 Evolution . . . . . . . . . . . . . . . . . . . . . . . . . 71
6.3.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
6.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71

**Conclusion** **75**

**Bibliography** **77**

**A List of Acronyms** **81**

**B Supplemental Material** **83**

# List of Listings

41 A structure of a single view specification in Vega-Lite [27] . . . . . 48
42 Structure of the encoding property for a single view specification [28] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51

51 The grammar encoded into JSON syntax. . . . . . . . . . . . . . . . . 56

61 The final visualization's specification from the experiment 1. . . . . 66
62 The final visualization's specification from the experiment 2. . . . . 70
63 The final visualization's specification from the experiment 2. . . . . 73

# List of Tables

4.1 Chomsky hiearchy . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
4.2 Description of properties for single view specification[27] . . . . . . 49
4.3 Supported field properties[8] . . . . . . . . . . . . . . . . . . . . . . 50

5.1 Properties of the JSON definitios of the grammar with analogy for formal grammars. . . . . . . . . . . . . . . . . . . . . . . . . . . 55
5.2 Equivalent constructs for the definition of production rules used in the JSON definition of the grammar used in the prototype. . . . . . 55

6.1 Measurings of needed interations to reach sufficient visualization for experiment 1. . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
6.2 Measurings of needed interations to reach sufficient visualization for experiment 2. . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
6.3 Measurings of needed interations to reach sufficient visualization for experiment 2. . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
6.4 Summary of needed iterations for all experiments . . . . . . . . . . 73

# List of Figures


2.1 Evolutionary Algorithm . . . . . . . . . . . . . . . . . . . . . . . . 29
2.2 Fitness proportiate selection. [24] . . . . . . . . . . . . . . . . . . . . 32
2.3 Mutation of the string genotype. . . . . . . . . . . . . . . . . . . . . 34
2.4 Mutation of the tree-like genotype. . . . . . . . . . . . . . . . . . . . 34
2.5 Recombination of the string genotype. . . . . . . . . . . . . . . . . . 36
2.6 Recombination of the tree-like genotype. . . . . . . . . . . . . . . . . 36

3.1 General IEC system – genotypes are decoded into phenotypes, that are retrieved to the user and user provides his feedback on them. . 38

4.1 A derivation tree of a string. . . . . . . . . . . . . . . . . . . . . . . 43
4.2 Comparison between the GE system and a biological genetic system [15]. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45

5.1 Graphical User Interface of implemented prototype . . . . . . . . . 61

6.1 The desired visualization for the experiment 1. . . . . . . . . . . . . 64
6.2 The diagram of chosen individuals during iterations for the experiment 1. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
6.3 The diagram of chosen individuals during iterations for the experiment 2. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
6.4 The diagram of chosen individuals during iterations for the experiment 2. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72

# Introduction

Evolutionary process, which is one of the pillars of evolutionary biology, has proven, over almost five billions of years, to be an efficient approach to get complex and successful individuals in various environments with enormous diversity of individuals and with just a small changes of heritable characteristic over successive generations [4].

Many methods in machine learning and artificial intelligence are actually inspired by processes seen in nature [6] and as it is evolutionary algorithms are one of them. More specifically, evolutionary algorithms emulate mechanisms of evolutionary process such as gene mutation and allelic combination (genotype). Traditionally evolutionary algorithms are usually used for finding a suitable solution (not necessarily the best solution) for difficult optimization problems, which are not possible to be solved in polynomial time, but quality of the solutions is easily numerically expressible so searching over space of feasible solutions is completely automated. In addition, these problems that satisfy the requirement are based on mathematical evaluation of their solutions, which means the evaluation is based on an objective approach. However, quality of solutions of some problems are hard to be described mathematically e.q. music and art, because quality of their representative solutions is based on subjective preferences of an observer. Not only a human-based evaluation is needed for the problems described above, but it is also way more efficient, and, besides, it can be used in combination with traditional mathematically described evaluation for very complex tasks to help directing the space search.

The aims of this thesis is to explore the field of existing methods of interactive evolutionary computation used on graphical interfaces and use given knowledge to create a prototype for generating visualizations. For this purpose a high-level grammar of graph visualizations **Vega-Lite** [26] is chosen. The implemented prototype should help data scientist to display visualization much more easily if they do not have knowledge about defining visualization or they want to you Vega-Lite, for creating, saving and sharing their visualizations.

Chapter 1

# State of the art

Unlike traditional evolutionary computation (EC), which is widely studied and is used in various applications, interactive evolutionary computation (IEC) do not typically get the same attention, because of its non-mathematical based evaluation and its application is mostly used for art, music and design, which are not very practical uses if people can do it better by themselves. For example, well-known is a project Picbreeder [16] from 2006, which is collaborative art application that allows pictures be bred like animals and shared these pictures with community, so everyone can continue with breeding their own pictures. Another interesting application of IEC for art creation is Eletric Sheep [5], that creates beautiful psychedelic visualizations that could be used as a screensaver. The evaluation is based on community rankings of newly created visualizations in a single iteration. Electric Sheep project runs since 1999 [21] and in its archive of created visualization [7], an obvious progress can be seen after almost 250 iterations. Both examples uses Interactive Genetic Programming to map mathematical representations of images as tree-like structures.

As it was said before, IEC is also used for music. For example, GenJam [10] project, that is developed since 1993 and that can play jazz alongside a jazz musician.

Another use case of IEC, that is not described in the paper above is a generation of GUI from a paper **Interactive Genetic Algorithms for User Interface Design** [18].

Of course, there are other use cases of IEC, for example industrial design, face image generation or database retrieval and they are well summarized in Hideyuki Tagaki's paper **Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation** [23].

IEC are not so different from traditional EC. Additionaly, the structure of the visualizations has a format of a a tree, so traditional genetic programming (GP), that was introduced in 1990 by John R. Koza in an article **Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems** [12] is apparently a good example of an inspiration, because GP is used exactly for evolving tree-like structures. Throughout the years new ways of tree-like structure evolution were discovered. For example relatively new Grammatical Evolution (GE) [15] by Michael O'Neill and Connor Ryan from 2001, that is based on Koza-style GP, but allows user to define grammar. The grammar provides control over the creation of feasible solutions in such way, that every rule of the tree structure can not be expanded into arbitrary rule from a set of available rules, but has to follow given grammar. Traditionally Koza-style GP is still able to use, but it has to be bended to support this functionality.

User fatigue is a big problem for all IEC systems and using only an interactive evolutionary algorithm (IEA) might not be efficient enought in seaching desired solutions in a state space. In term of user fatique, IEA has to search as much efficiently as possible. A normal IEC process lasts 10–20 iterations, before user gets bored [23]. It is necessary to implement mechanisms to prevent evolutionary algorithm search in evidently wrong regions. This could be accomplished with additional information provided by user [19].

Chapter 2

# Evolutionary Computation (EC)

In computer science, evolutionary computation is an umbrella term for algorithms, that are inspired by biological evolution with similar mechanisms seen in nature and used on computer for automated global optimization. A goal of evolutionary computation is to produce highly optimized solutions from initial set by repeteadly applying similar mechanisms from the biological evolution in an attempt, that each iteration of applying these mechanisms over big set of solution, produces similar solutions to continuesly get better optimized solutions. Evolutionary computation has numerous applications in problem solving, designing, planning, machine learning an more [3].

## 2.1 Evolutionary Algorithm (EA)

Evolutionary algorithms are a subgroup of algorithms from evolutionary computation using the same mechanisms to breed higher optimized solutions. These mechanism are:

- initialization
- selection
- mutation
- recombination (crossover)

They are called evolutionary operators [17] and ways how they are applied on an individual from a population, differ from one algorithm to another, but all of them simulates the same principle from the biological evolution. The order how these operators are applied is shown in figure 2.1. Even same called EAs can use a bit different evolutionary operators. What defines EAs more than evolutionary operator is a representation of an individual [2].

Usually, scientists distinquist two basic structures representing the individual, tree-like structures and string structures. These structures can not be mixed in population. That means, the evolutionary algorithm can not use tree-like structure and string structure at the same time, in the same population, because there is no way how to combine the solutions, which is a pillar of evolutionary computation, but a tree-like structure can be encoded into a string [9]. Additionaly, these structures use different representation of a single gene in its genetic code. It can be integer, binary or real number representation of a gene for string structured chromosomes [12].

## 2.2 Types of Evolutionary Algorithms

Different types of evolutionary algorithms use similar mechanisms, they differ in a representation of the genotype and implementation details.

**Genetic Algorithm (GA)** is the most popular type of EA and usually easiest to implement. GA uses the string representation of the genotype of the individual. GA is usually used for problem solving. By applying evolutionary algorithms described above, the solution of the given problem is being sought. However, definitions of the GA do not always match. In more cases, GA is EA that uses only string genotypes for representing the individual, but in some literature, GA is defined as the generic evolutionary algorithm, so GA an EA are defined as an exactly same algorithm [2].

**Genetic Programming (GP)** is the type of EA, that uses tree-like structures as the representation of the genotype of the individuals. GP is typically used to find a computer program, that efficiently imitates a computational problem. It is also used to find a mathematical function that describe behaviour of given data.

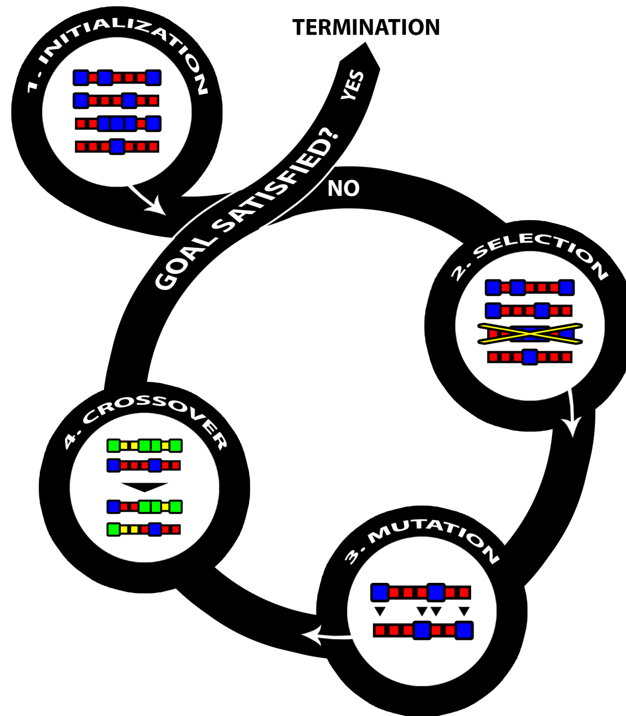



**Figure 2.1:** Evolutionary Algorithm

**Gene expression programming (GEP)** is used for the same cases as GP, but it uses the string representation of the genotype by introducing genotype-phenotype system of encoding.

**Evolutionary Programming (EP)** is also similar to GP, but uses only a fixed structure of trees with numerical variables. EP evolves these numerical variables instead of the tree-like structure.

**Grammatical Evolution (GE)** is also related to idea of GP, but uses string representation of genotype instead of tree-like structure. It allows to evolve solutions accordinly to grammar, that is provided by user.

**Differential Evolution (DE)** uses the string representation of the genotype, but genes are in the form of real numbers or a vector.

## 2.3 Individual's Representation

In biological evolutions, a genotype and a phenotype are very-similar sounding words. These two terms are related, but they still mean different things.

The genotype is a genetic code, a set of genes, that is inherited from ancestors on an organism. For the organism, its DNA or RNA is its genotype.

The phenotype is a physical expression of these genes, this genotype, and an environmental influence of the organism.

The genotype can not be determined by simple observation, but the phenotype can be. For example, an eye color is the inherited trait from parents. The eye color is written in DNA by the same sequence of genes, but its actual appereance is the phenotype. Furthermore, two people can have same eye color, they share the same phenotype, even though they are not relatives and they do not share the same genotype, same genes [29].

Evolutionary algorithm are also distinguished by their ability how acurately they simulate coding and processes from biological evolution. The basic ones, like genetic algorithm or genetic programming, simulate only simple encoding of genes in genotype and simple translation of the genotype into phenotype. The more complex EAs simulate much complex encoding and translation. These more complex algorithms typically come up with more terms from biological evolution. For example codons, alleles and so on, that desribe substructures in the individual's genotype. Evolutionary operators of these more complex EAs are able to handle these more sophisticated structures, thus descriptions of their functionality are also more complex. Grammatical Evolution is one of these more complex EAs [15].

### 2.3.1 String

A string representation of the individual is a sequence of genes. These genes can be represented with boolean, integer or real number value. The

main characteristics of string representation is that two parents can be easily combined into an offstring by a simple cut in the. Mutation is just a change of a value in the gene. Representative example of EA with the string representation of the genotype is Genetic Algorithm (GA) [2]. EAs with the string representation are usually much more easy too implement than EAs with tree-like structure, but it is not a rule. As it was said before, Grammatical Evolution is a complex EA, even though it uses the string representation of the genotype. Typically, EAs with this representation of the individual are used for problem solving.

### 2.3.2 Tree

Tree representation of the individual is a connected graph that contains no cycles [13]. This representation is often used in EAs, that are used to find and optimize mathematical equations or programs.

## 2.4 Evolutionary Operators

Each EA has a bit different evolutionary operators and how they are applied on the genotype of the individual from the population. A functionality of the evolutionary operators, mostly depends on representation of the genotype. The functionality also depends on representation of a single gene. Regardles of different applications, the evolutionary operators still simulate the same thing from the biological evolution.

### 2.4.1 Initialization

The first step that each EA does is an initialization of a population of individuals. How well this step is done, can significantly increase convergence of the evolutionary algorithm to satisfy its goal.

### 2.4.2 Selection

Each iteration in EA, all individuals are measured by fitness function, that gives every individual a score called fitness. Fitness of an individual says how good is the individual, how well it is optimized for a given problem and what is its

quality. The best solutions with highest scoe are selected for breeding. This is only one possible way of selection. There are actually more ways how to select these solutions to prevent EA to stuck in a local optimum. Selecting only top $n$ individuals for breeding can correspond in production of very similar individuals with the same score and without bonuses from breeding of these solution. EA that stuck in the local optimum is not able to find better quality solutions in a state space throughout iteration.

Again, there are much more algorithms for selection of individuals for breeding, but the most used ones are tournament selection and fitness proportiate selection.

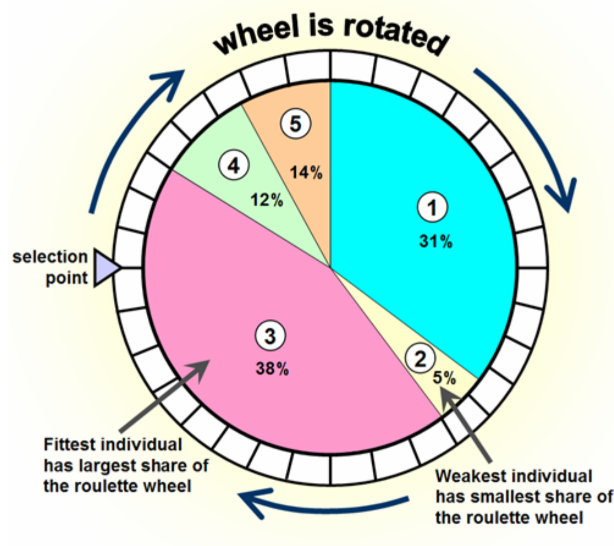



**Figure 2.2:** Fitness proportiate selection. [24]

**Tournament Selection** iteratively creates a pool(tournament) of $n$ random individuals from population and from this tournament top $m$ individuals with highest fitness are selection for breeding [2].

**Fitness proportionate selection** could be imagined like a roulette wheel in a casino that is shown in figure 2.2. A proportion of each individual on the wheel is assigned by its fitness. Usually, fitness of each individual is normalized to 1 or divided by total sum of all fitness values. Then, algorithm starts to randomly rotate with the wheel that leads to selecting the individual and lowering its proportion on the wheel [2].

### 2.4.3 Mutation

In biology, mutation is a biological process, that changes random genes in the genotype of the individual, in an attempt to find new characteristics, that can be very useful in a life of a new born creature. Exploration of these new characteristics can still create weaker creatures, that are not able to spread their genes in following generation and natural selection can deal with them, but characteristics discovered by mutation have not been discovered in creature's parents yet [2]. Not-yet discovered features can help the creature to survive in the changing environment and this is the reason, why it is so important.

In evolutionary algorithms, mutation works exactly the same way. It can also create worse-optimized individuals, but its biggest benefit is that it find new ways and helps EA to not stuck in local optimum [2].

**Mutation of the string genotype** is a random change to different value of a random gene in the individual's genotype. It is shown in figure 2.3, where 4 genes out of 7 of the individual's genotype are changed. There are two blue big squares, that are mutated into red small squares and there are also two red small squares, that are mutated into big blue squares. This created a new individual, that might be better optimized than the previous one [2].

**Mutation of the tree-like genotype** is a random expansion of a random node in the tree to a different node or a different subtree. It is shown in figure 2.4, where is the arrow pointing at the red node. This whole subtree of the red node with its two sons is replaced by a different subtree [12].

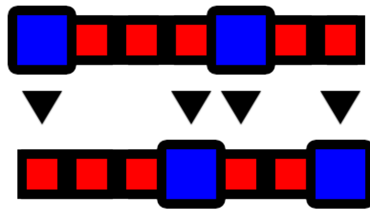


**Figure 2.3:** Mutation of the string genotype.

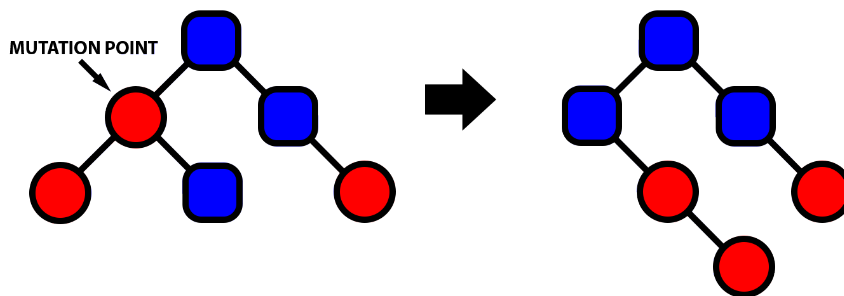



**Figure 2.4:** Mutation of the tree-like genotype.

### 2.4.4 Recombination

Also known as crossover, recombination is a method that uses genotype of one or two parents to produce genotypes of one or two individuals. Two basic types of recombination are asexual and sexual recombination and they are distinguished by number of included parents in the newly created genotype and procedure that makes these descendants. Traditionally, asexual recombination is not possible for same length string genotypes, but it is possible for variable length string genotypes and for tree-like ones. Asexual recombination is not possible for same length string genotypes, because of their representation. A new individual with different genotypes can not be created from the same length sequence of values, because it is always identical to its parent. Tree-like genotypes can always change at least their leaves of subtrees. Variable length sequences can shrink .

**Recombination of the string genotype** concatenates two subsequencies from genotypes of two parents to create an offspring as it is shown in figure 2.5. As it was said above, asexual recombination is possible only for genotypes with the variable length and not for genotypes with the same length, bacause it leads to creation of the same genotype of individual as individual's parent [2].

**Recombination of the tree-like genotype**, also called crossover, is a mechanism, which randomly chooses two nodes in trees of one or two parents. These nodes then represent two subtrees that are exchanged to create one or two individuals as it is shown in figure 2.6. The number of created individuals depends on an implementation of the recombination algorithm [12].

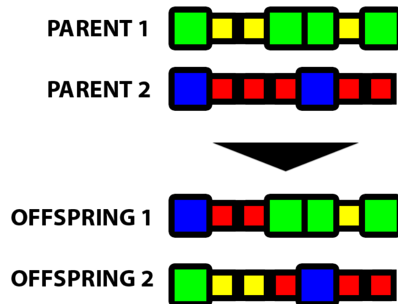



**Figure 2.5:** Recombination of the string genotype.

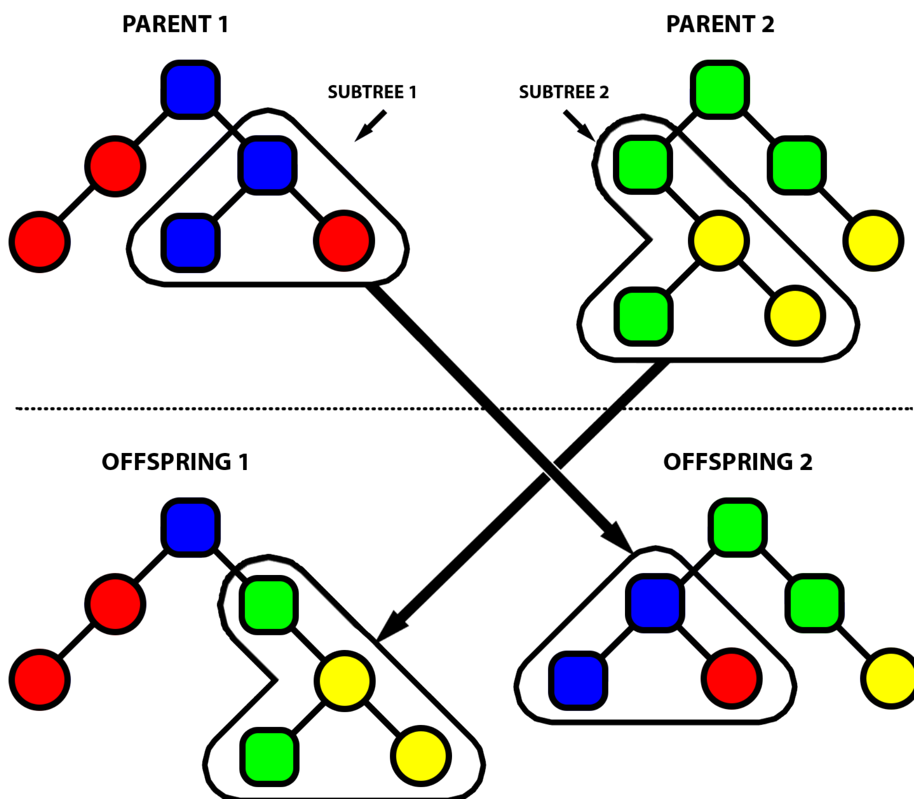



**Figure 2.6:** Recombination of the tree-like genotype.

Chapter 3

# Interactive Evolutionary Computation (IEC)

In evolutionary computation, algorithms search a space of parameters and use gradient information of the space to optimize parameters to get solutions of higher quality. Since gradient information of user's feelings, knowledge and preference cannot be used to determine a quality of the solution, it is needed to evalute solutions in different approach that is different from conventional optimization methods [23].

In other words, algorithms of the traditional evolutionary computation optimize solutions of problems, whose performace (fitness) is numerically described and an algorithm is able to compute this performace.

However, some problems do not have the performance evalution of their solutions numerically described, because the description is too difficult or even impossible to specify. These problems typically require a human evaluation, that is also difficult to implement and sometimes, the solution of the given problem requires subjective preferences of an observer. Algorithms that require the human evalution as a replacement of the fitness function are algorithms of **Interactive Evolutionary Computation (IEC)**. Usually, algorithms of IEC retrieve graphics or music, such outputs must be subjectively evaluated [23]. In terminology of EC, algorithms work with genotypes and at the same time with phenotypes. Genotypes are evolved by evolutionary operators, but they are not presented to the user. User gets their phenotypes and expresses his

interests and preferences on these phenotypes in the same way as it is shown in figure 3.1.

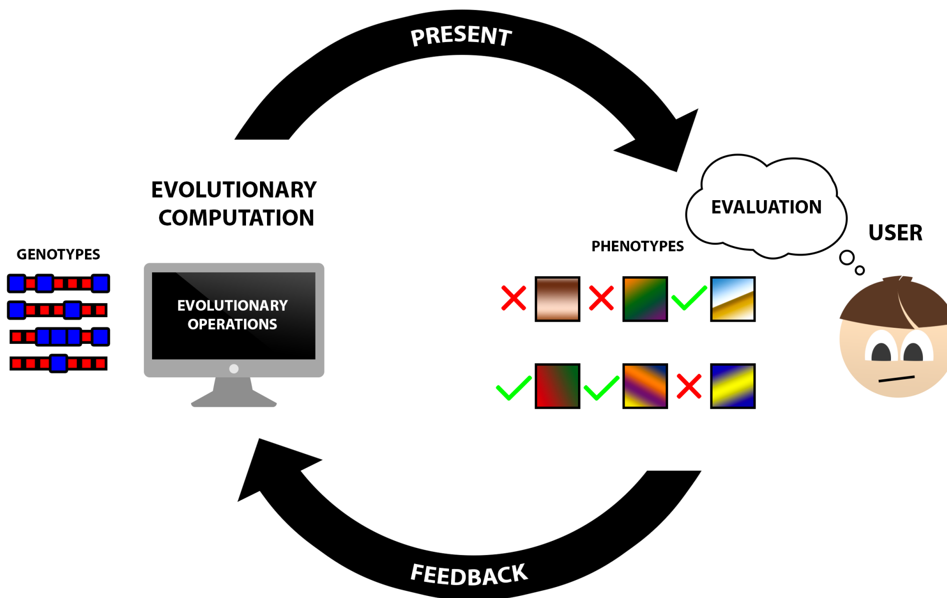



**Figure 3.1:** General IEC system – genotypes are decoded into phenotypes, that are retrieved to the user and user provides his feedback on them.

Algorithms of the subgroup evolutionary algorithms can be used with human-based evalution. They are called the same, but with the word 'interactive' in the beginning of the original name of the algorithm, to show that the evaluation is human-based. For example, **Interactive Genetic Algorithm** and **Interactive Genetic Programming** [23].

## 3.1 Differences of IEC to EC

The main difference is the human evalution of the fitness of individuals, but there is more to deal with. People are different to computers and unlike to computers, they are not capable of making hundreds or even more calculations and computations each seconds. Their are also biased because of their preferences and this means that each person can evaluate one individual

differently. The user's preferences can be desired in other applications [23].

## 3.2 User fatique

User fatique is an issue, that all IEC systems have to deal with. If user has to evaluate a lot of individuals from population, he usually gets easily exhausted in a few steps of the evolution. Also, he gets exhausted, if he has to do many steps in the evolution to get desired result. A normal IEC process lasts 10–20 iterations, before user gets exhausted [23].

To prevent the user to get easily exhausted, the IEC system should provide user a small amount of individuals to get evaluated. These individuals should has a high quality, to prevent a high number of evolution steps, that also leads to exhaustion of the user. The small amount of individuals, but with high quality, can be provided by some heuristic approach. Also, to quicken the evolution that convergates to a desired individual in fewer steps, can be also ensured by generating high quality individuals. These heuristic information are usually provided by user.

Chapter 4

# Grammar evolution of graph interfaces

Usually, spoken languages like Czech have their own grammar. To say a correct sentence in the language means to follow the grammar of that language. It does not mean, that the sentence must have a meaning. It can be a nonsense either. The important is, that the sentence is syntactically corrent. In other words, it means that the sentence is created by applying rules from the grammar of the language. Every programming language is also described by its grammar.

Mathematical abstraction of a grammar is known as **formal grammar**.

## 4.1 Formal grammar

### 4.1.1 Definition

A formal grammar is a set of rules for constructing valid sentences from a language. The formal grammar $G$ is a quadruple $(N, T, P, S)$ [11] where:

- $N$ is a nonempty set of nonterminal symbols.
- $T$ is a nonempty set of terminal symbols.
- $P$ is a set of production rules.
- $S$ is a starting symbol.

### 4.1.2 Vocabulary

**nonterminal**
a symbol that can be replaced/expanded by a sequence of symbols

**terminal**
a symbol that cannot be replaced/expanded by anything else.

**production rule**
a grammar rule that describes an expansion for a symbol. The sequence is replaces by a symbol or sequence of symbols.

**derivation**
a sequence of applied rules that produces finished string of terminals.

**starting symbol**
a grammar has at least one starting nonterminal symbol, from which all strings derive.

**empty symbol**
a symbol that can be replaced by nothing.

**Example of the formal grammar**:

$$
\begin{aligned}
G &= (N, T, P, S) \\
N &= \{A, B\} \\
T &= \{x, y\} \\
P &= \{A \rightarrow Bx|y, \\
&\qquad B \rightarrow x|y, \\
S &= \{A\}
\end{aligned}
$$

This grammar can produce strings **xx**, **yx** and **y**. A derivation tree of the string **xy** is shown in figure 4.1.

### 4.1.3 Chomsky hiearchy

In addition, grammars are distinguished into four categories by the form of their production rules, and by the class of languages they generate. They span

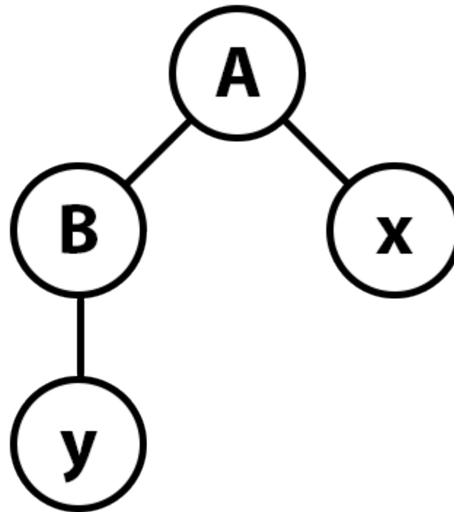



**Figure 4.1:** A derivation tree of a string.

from Type 0 to Type 3. The most general grammars are Type 0 grammars and the most restrictive are Type 3 grammars [11]. Chomsky hiearchy and the constaints on the production rules are described in table 4.1, where:

- $a$ is a terminal symbol.
- $A, B$ are non terminals.
- $\alpha, \beta$ are strings that contain sequences of terminals and nonterminals. They can be replaced by empty symbol.
- $\gamma$ is a string that contains a sequence of terminals and nonterminals. It cannot be replaced by empty symbol.

### 4.1.4 Backus–Naur Form

Backus–Naur Form or Backus Normal Form (BNF) is a notation technique for expression context-free grammars. It consists of **a set of terminal symbols**, **a set of nonterminal symbols**, **a set of start symbols**, and **a set of production rules** it this form:

**Table 4.1:** Chomsky hiearchy

| Grammar | Language | Production rules |
|---|---|---|
| Type 0 | Recursively enumerable | $\alpha A \beta \rightarrow \beta$ |
| Type 1 | Context-sensitive | $\alpha A \beta \rightarrow \alpha \gamma \beta$ |
| Type 2 | Context-free | $A \rightarrow \beta$ |
| Type 3 | Regular | $A \rightarrow A$ or $A \rightarrow aB$ |

```
<Left-Hand-Side> ::= <Right-Hand-Side>
```

The symbol ::= denoted translation symbol from left-hand side to symbol on right-hand side. All the nonterminals are surrounded by inequality brackets <>. A vertical bar | indicates a choise. BNF does not support the empty string $\epsilon$ [14].

**For example, in BNF, the previously shown grammar**:

```
<A> ::= <B>x | y
<B> ::= x | y
```

## 4.2 Evolutionary algorithms for evolving grammars

The concept of the evolution is based on the representation of derivation trees.

### 4.2.1 Genetic Programming

Genetic programming is a simple way for evolving derivation trees. It uses mechanisms that are applied on tree-like genotypes, because it directly works with tree-like structure [12].

### 4.2.2 Grammatical Evolution

Grammatical evolution (GE) is more sophisticated way, that goes even deeper in an attempt to mimic the natural evolutionary process. Genetic Programming uses genotypes, which has the same structure like deviation trees, but GE

encodes a genotype into binary string of variable-length. This string is transcripted integers, which are blocks of 8 binary values. These integers are called codons. Codons are mapped onto grammar rules that with assigned terminals makeup the derivation tree [15]. For better understanding, check the figure 4.2.

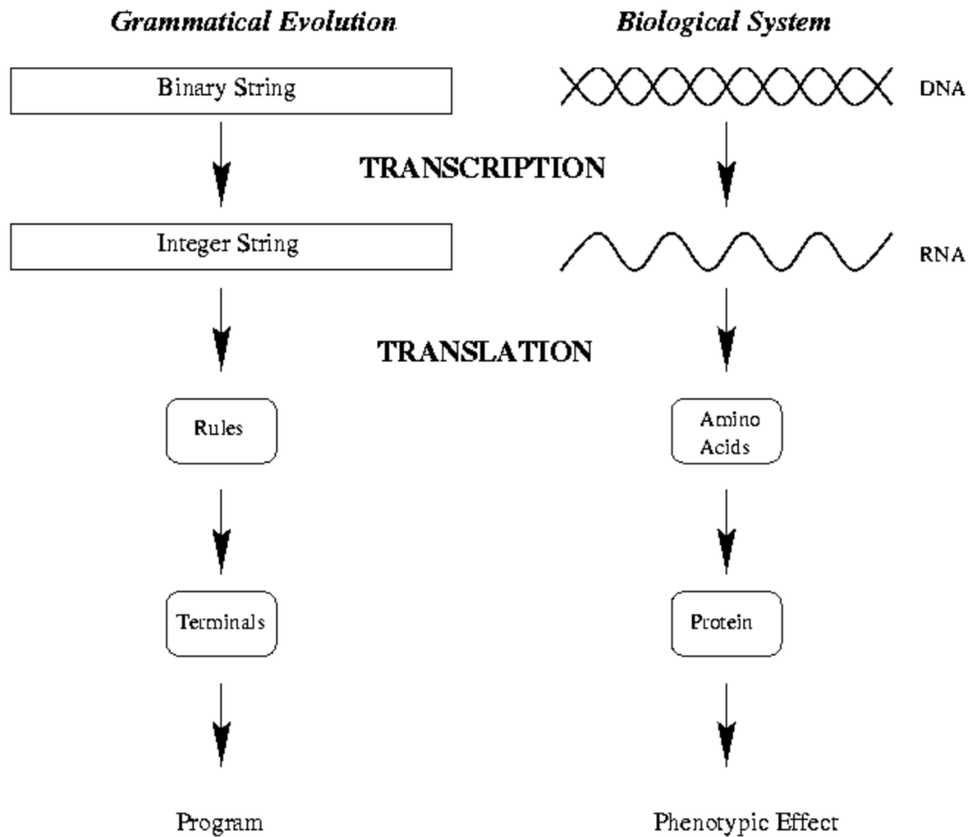



**Figure 4.2:** Comparison between the GE system and a biological genetic system [15].

A grammar for Grammatical Evolution is provided in Backus-Naur Form (BNF). It is not a requirement, to specify the entire language by BNF. Perhaps, it is more useful to specify just a subset of the language geared towards the given problem [15].

The genotype (binary string) is mapped by reading codons (8 bits) to an integer. This integer is then used for selecting an appropriate production rule by using this following mapping function:

$$rule = (codonintegervalue)$$
$$MOD$$
$$(numberofrulesforthecurrentnonterminal).$$

The traversal of the genotype is done by reading codons, and if production rule is selected, another codon has to be read. It is possible that during mapping process, the individual runs out of codons. In that case, EA wraps the individual and reuses the codons. Authors claim that this technique of gene-overlapping phenomen has been observed in many organizms [15].

The GA adopted to the variable-length genotype is used in this case. With GA there comes advantages of GA, which are evolutionary operators and initialization, that are relatively simply to realize for GA, because of the string-based genotype. More evolutionary operators can be applied on the populating, because of recognition of codons in the genotype. They are evolutionary operators, based on mutation and crossover of the codons[15].

## 4.3 Grammar of graph interfaces

For the purpose of applying evolution process on a grammar, Vega [25] framework, as a tool for creating visualizations, was chosen, because it supports specification of a visualization by JSON syntax, that is described by Vega-Lite grammar [26].

### 4.3.1 Vega-Lite

Vega-Lite is a high-level visualization grammar of interactive graphics, a declarative language for creating, saving and sharing visualizations. It is build on top of Vega visualization grammar to provides a concise JSON syntax for rapidly generating visualization to support data analysis [26].

A Vega-Lite specification describes a visualization as mappings from data to properties of graphical marks and this specification is defined in a JSON format[26]. The specification is parsed by Vega-Lite's JavaScript runtime

compiler to automatically generate this specification into a lower-level, more detailed Vega [25] specification that uses its compiler to render visualization components, such as axes, marks, legends and scale of the defined vizualization as a static image or as a web-based view. Thanks to carefully defined rules, Vega-Lite also automatically determines properties of these components to keep specifications succinct and expressive, but still provide user control.

As Vega-Lite is designed for analysis, it supports data transformations such as aggregation, sorting, binning and visualization tranformations stacking and faceting. It also supports composion of visualizations to create layered and multi-view displays with an additional support of interactive selection.

However, for the purpose of this thesis, further descriptions of multi-view composions, also complex transformations and selection of data is not needed. Interactive interfaces of the visualizations are also not needed to be defined, even thought. This thesis primarily focus on application evolutionary mechanism on the mappings of data values to mark properties.

In the following reading, there are described only parts of all features, properties and functions of Vega-Lite, that are minimally required for the understanding of the Vega-Lite, its syntax, structure and which parts of Vega-Lite are used for this thesis.

#### 4.3.1.1 Common Properties of Specifications

All specifications in Vega-Lite can contain properties as **name** for later reference, **description** for comments, **title** of the vizualization, **data** as a data source, that can be loaded inline or from an external source and **transform** array of data transformation of the given data for calculating new fields and for filtering of given data. The prototype in this thesis does not actively work with any of these properties.

#### 4.3.1.2 Top-level Specifications

All specification can contain these following top-level specifications. The prototype also does not actively work with these properties. The properties are **$schema**, which is a source of Vega-Lite specification, **background** as CSS color property, **padding** from the edge of the visualization canvas to the data

rectangle, **autosize**, that sets how the visualization size should be determined and **config**, which is Vega-Lite configuration object.

#### 4.3.1.3 Single View Specification

```
{
  // Properties for top-level specification
  "$schema": "https://vega.github.io/schema/vega-lite/v3.json",
  "background": ...,
  "padding": ...,
  "autosize": ...,
  "config": ...,

  // Properties for any specifications
  "title": ...,
  "name": ...,
  "description": ...,
  "data": ...,
  "transform": ...,

  // Properties for any single view specifications
  "width": ...,
  "height": ...,
  "mark": ...,
  "encoding": {
    "x": {
      "field": ...,
      "type": ...,
      ...
    },
      "y": ...,
      "color": ...,
      ...
    }
}
```

**Listing 41:** A structure of a single view specification in Vega-Lite [27]

A single view specification describes a mark type of the visualization and encoding mapping between data features and properties of the mark. Once the mark type and the encoding is provided, Vega-Lite produces axes, legends and scales of the visualization that follows carefully defined rules. With these rules, Vega-Lite is also able to determine properties of these components and also give a negative feedback, if the specification is not valid and Vega-Lite has

to change the specification [27]. The summarization of available properties for a single view specification are described in the table 4.2. An structural example of a single view specification is show in listing 41.

**Table 4.2:** Description of properties for single view specification[27]

| Property | Description |
|---|---|
| mark | **Required**. A string describing the mark type (one of ”bar”, ”circle”, ”square”, ”tick”, ”line”, ”area”, ”point”, ”rule”, ”geoshape”, and ”text”) or a mark definition object. |
| encoding | A key-value mapping between encoding channels and definition of fields. |
| width | The width of a visualization. |
| height | The height of a visualization. |
| view | An object defining the view background’s fill and stroke. Not needed. |
| selection | A key-value mapping between selection names and definitions. |
| projection | An object defining properties of geographic projection, which will be applied to shape path for ”geoshape” marks and to latitude and ”longitude” channels for other marks. |

#### 4.3.1.4 Encoding

To encode a particular field from the dataset to the mark property, this mapping's definition must be provided by user and described in an **encoding** object. The mapping of the field to the channel (like **x**, **y** or **color**) must contain **field** property and **type** property, which is data type of the field. In addition to top-level **transform** object, the Vega-Lite also supports inline tranforms of fields in a channel's definition.

From all the **encoding channels** shown in listing 42, the prototype allows to map and evolve fields to **x**, **y** channels and all **Mark Properties channels** (e.g., **color**, **fillOpacity**, **strokeOpacity**, **size** and **shape**). The other channels are not needed for purpose of this thesis, because they do not affect appearence of generated visualizations (Text, Tooltip, Hyperlink, Order, Level of Detail and Facet channels) or they would require much more provided information from the user before start of the evolutionary process. For example Geographic Position Channels require an image of a map and transformation of positions in data to project the rows in data on the map.

**Table 4.3:** Supported field properties[8]

| Property | Description |
|---|---|
| field | **Required**. A string defining the field from the dataset. |
| type | **Required** A data type of the encoded field (e.g., **quantitative**, **temporal**, **ordinal** and **nominal**). |
| bin | A flag for binning of a **quantitative** field. It can be **true**, **false** or **BinParams** object. For the purpose of the prototype, only **true** and **false** are considered. |
| timeUnit | Time unit for a **temporal** field. Because of not supporting **temporal** fields, this property is absolutely ignored in the prototype. |
| aggregate | Aggreation function for the field. Available values are: **mean**, **sum**, **median**, **min**, **max** and **count**. |
| title | A title for a field. It is not very interesting for showing functionality of the prototype. Because of that, the default field title from dataset is used instead or the title corresponds with aggregate function over the values of the field. |

```
// Specification of a Single View
{
  "data": ... ,
  "mark": ... ,
  "encoding": {     // Encoding
    // Position Channels
    "x": ...,
    "y": ...,
    "x2": ...,
    "y2": ...,

    // Geographic Position Channels
    "longtitude": ...,
    "latitude": ...,
    ...

    // Mark Properties Channels
    "color": ...,
    "opacity": ...,
    "fillOpacity": ...,
    "strokeOpacity": ...,
    "strokeWidth": ...,
    "size": ...,
    "shape": ...,

    // Text and Tooltip Channels
    "text": ...,
    "tooltip": ...,

    // Hyperlink Channel
    "href": ...,

    // Key Channel
    "key": ...,

    // Order Channel
    "order": ...,

    // Level of Detail Channel
    "detail": ...,

    // Facet Channels
    "facet": ...,
    "row": ...,
    "column": ...
  },
  ...
}
```

**Listing 42:** Structure of the encoding property for a single view specification [28]

# Chapter 5

# Implementation of prototype

## 5.1 Goals

The prototype should allow the user to generate visualization in Vega-Lite for a given data . The generated visualizations should be based on preferences of the user and selections and actions what the user did. The prototype should respect that the user may does not know anything about Vega-Lite grammar and its JSON syntax.

The prototype should has some graphical user interface (GUI) to make the selection of user prefered visualizations easier for the user and make the complete interaction with the prototype easier without any deep knowledge of command line. The GUI is almost required for this task, because the display of visualization and following selection would be much more complicated to realize by user's interaction with the command line.

Also, the prototype should respect that the user may does not know anything about statistical datatypes and it should provide an automatical detection of feature datatypes in the given dataset.

The prototype should use mechasims to prevent user fatique, such as heuristics that restrain size of searched state space of feasible solutions.

Once the visualization is good enough for the user, the prototype should provide a way of retrieving the specifications of visualizations in Vega-Lite JSON syntax, so the user can actually use and share them.

## 5.2 Implemented EA

I have chosen Genetic Programming, respectively Interactive Genetic Programming to evolve the visualization's specification in this prototype, because of its tree representation. Genetic Programming also uses trees to represent individuals from the population. It does not mean, that other types of evolutionary algorithms cannot be used, but they would require an additional encoding of individuals's genotype to its phenotype, which means to translate the genotype to its visualization's specification. Because of the relatively short grammar, I have decided that Grammatical Evolution would be an unnecessary complication. Also, Grammatical Evolution works with translation of the genotype to the phenotype, that would make heuristic approaches on the phenotype more complicated, because of backwards translation from the phenotype back to the genotype.

### 5.2.1 Encoding of grammar

To directly program all the rules, requires a lot of effort and the rules are then hard to read from the final program. This is the reason, why the all the possible specifications of visualizations and desribed by a grammar. The visualization specifications are generated by derivation of production rules included in the grammar. The program derivates these rules to generate the vizualization specifications. Additionaly, the grammar is easier to read and in a case of need, easier to control which specifications are produces by the program.

The JSON definition of the grammar contains properties described in table 5.1 with equivalents from the formal grammar. Used JSON constructs for definition of production rules are described in table 5.2. An example of JSON definition of the grammar and the actual grammar used in this prototype is shown in listing 51.

**Table 5.1:** Properties of the JSON definitios of the grammar with analogy for formal grammars.

| Property | Description | In formal grammars |
|---|---|---|
| StartingSymbol | Evolutionary Algorithms always starts to generate the Vega-Lite specification from the production rule defined in **StartingSymbol** property. | A starting symbol. |
| Expressions | A map of available production rules in the grammar. | A nonempty set of production rules. |
| Terminals | A nonempty set that contains every element in the **Expressions** map. | A nonempty set of terminal symbols. |
| Nonterminals | A nonempty set that contains every key in the **Expressions** map. | A nonempty set of nonterminal symbols. |

**Table 5.2:** Equivalent constructs for the definition of production rules used in the JSON definition of the grammar used in the prototype.

| Construct | Description | In formal grammars |
|---|---|---|
| Key : [ Options[] ] | The key on the left side is a name of the production rule. The right hand side contains an array of Options[]. | The production rule. |
| Options[ Elements[] ] | An array of element arrays (feasible values or keys). | A set strings that can be derived from the production rule. |
| Elements[] | Array of element arrays (feasible values or keys). | A set of strings of terminals and nonterminals. |
| element | A value or the production rule. | Terminal or nonterminal symbol. |
| [] | Empty element. | Null symbol. |

```
{
  "Expressions": {
    "start": [ ["mark", "encoding"] ],
    "encoding": [
      ["fill", "opacity", "shape", "size", "color", "stroke"]
    ],
    "mark": [
      ["bar"],
      ["rect"],
      ["line"],
      ["point"],
      ["area"],
      ["circle"],
      ["square"]
    ],
    "color": [ ["aggregate", "bin", "field", "type"] ],
    "fill": [ ["aggregate", "bin", "field", "type"] ],
    "opacity": [ ["aggregate", "bin", "field", "type"] ],
    "shape": [ ["aggregate", "bin", "field", "type"] ],
    "size": [ ["aggregate", "bin", "field", "type"] ],
    "stroke": [ ["aggregate", "bin", "field", "type"] ],
    "x": [ ["aggregate", "bin", "field", "type"] ],
    "y": [ ["aggregate", "bin", "field", "type"] ],
    "aggregate": [
      ["mean"],
      ["sum"],
      ["median"],
      ["min"],
      ["max"],
      ["count"],
      ["average"],
      []
    ],
    "bin": [
      [true],
      []
    ],
    "field": [/* Names of fields in the dataset */],
    "type": [
      ["quantitative"],
      ["nominal"]
    ]
  },
  "StartingSymbol" : ["start"]
}
```

**Listing 51:** The grammar encoded into JSON syntax.

### 5.2.2 Initialization

At first, the algorithm of Genetic Programming expands **StartingSymbol** from the JSON definition of the grammar. In the case of the implemented prototype and its grammar, it is **start** key. After this expansion, algorithms has only one option, that is expansion of the option **["mark", "encoding"]**. Both of these strings are production rules, because there exists production rules in the **Encodings** map. The important part is, that they are not nonterminals, they create an object accesible by the name of expanded element, then the algorithm continues with the expansion of elements accesible by the key in **Encodings** map. The expansion of a single branch of the tree ends, when the last expanded element is not a terminal. It means, that the element is not one of the keys in the **Encodings** map. In other words, it is not one of the production rules.

Once the generation is done, algorithms removes all the empty strings. Empty strings are in the grammar, to represent, that the rule is only optional and during this process is removed from the individual, which is the final specification of the visualization.

### 5.2.3 Mutation

The individual's tree is traversed by the algorithm of mutation in preorder way. In each node of the tree, the algorithms randomly chooses, if the node's siblings should be replaced by a new subtree or not. If it is true, the new subtree is derivated from the production rule accesible by the node's name in **Encodings** map. This leads, that not every node from the tree is actually traversed by the algorithm. The nodes from the replaced subtree are no longer relevant for the individual, because they are no longer a part of the tree. The probability, that drives the mutation of nodes is set in GUI by the user.

### 5.2.4 Recombination

The recombination in Genetic Programming is defined as the crossover of arbitrary subtrees from two parents. However, I have decided, that the prototype exchanges only subtrees of field mappings. There are two reasons for that. The first reasons is, that properties of each mapping is dependant on data type of the field (e.g., binning of nominal values is not possible, also the

most of the aggregate functions is not defined for nominal fields). The second reason is my decision based on my experience with the previous versions of the prototype. The previous versions of the prototype struggled to combine individuals selected by the user. Previously, the algorithm randomly selected the subtree of the individual and exchanged it with the second parent. For example, the user selected two specification. One of them is containing binning of some field and the second is containing the aggregation applied on another field. The algorithm exchanged them, but the user's preference is mapping of **x** and **y** field, so the binning is not the point of his interests.

### 5.2.5 Selection

All the individuals selected by the user are used for later breeding. The recombination happens only if two individuals were selected by the user. If user selected only one individual, only the mutation is applied on the individual in breeding process.

## 5.3 Heuristic approaches

To quicken the evolution and shrink the number of evolutionary steps, by restricting the seached space, the prototype uses four heuristic approaches.

### 5.3.1 Datatypes

The datatype of the field is marked as **nominal**, if any of the field's values is not numerical. If all the values are numerical, but the number of unique values is less then ten, the datatype of the field is also marked as **nominal**. Howerever, the user can still choose the proper datatypes in the GUI of the prototype. This approach removes the need of defining the **type** property in the mappings.

### 5.3.2 Number of mapping in a graph

The user can select, how many field should be mapped in the visualization's specification. This is based on a principle, that user probably know, which fields he wants to show in the visualization. This approach also removes all the

specifications, that do not have an exact number of mappings to the selected fields.

### 5.3.3 Forbidden transformation of nominal features

As it was said before, the defined grammar does not cover every rule from Vega-Lite specification. On of them is, that **nominal** fields do not transformations, that are supported for **quantitative** fields. To avoid assigning these tranformations to **nominal** fields with the definition of the grammar, the algorithm that expands the rules would have to work with another type of production rules, that are translated differently, but it would complicate the implementation, the grammar and it is not needed for the prototype. Because of that, the prototype checks field mappings and removes the **aggregate** and the **bin** property, if the **type** property is nominal.

### 5.3.4 X and Y axes

The visualization's specification should primarily map fields to **x** and **y** axes instead of mapping them to mark properties. The algorithm prefers mapping to **x** and **y** axes and then, if there is any field left, the algorithm maps the rest of the fields to mark properties. This approach reduces the space search to visualization's specifications, that would not show only as a single mark.

## 5.4 Description of GUI

The GUI of the prototype consists from a left columns, where are most of the controls, and main window, where are visualizations shown.

### 5.4.1 Left Column Window

In the left column, user can control probability of change, projected in a next iteraction of the evolution. Then, there is a button to run the evolutionary algorithm on selected visualization from the main window. The user can also select one of predefined datasets and select columns from the datasets. Also, the user can define, what datatypes are the selected columns. After running

the first iteration, these columns are locked and the evolutionary algorithm can use only the selected columns for the generation visuzalizations.

### 5.4.2 Main Window

The main window of the prototype is used for showing visualizations that can be selected by the user. The main window shows twelve visualizations in total. The user selects them by **Left Mouse Click**, which makes a border around the selected visualization turned green.

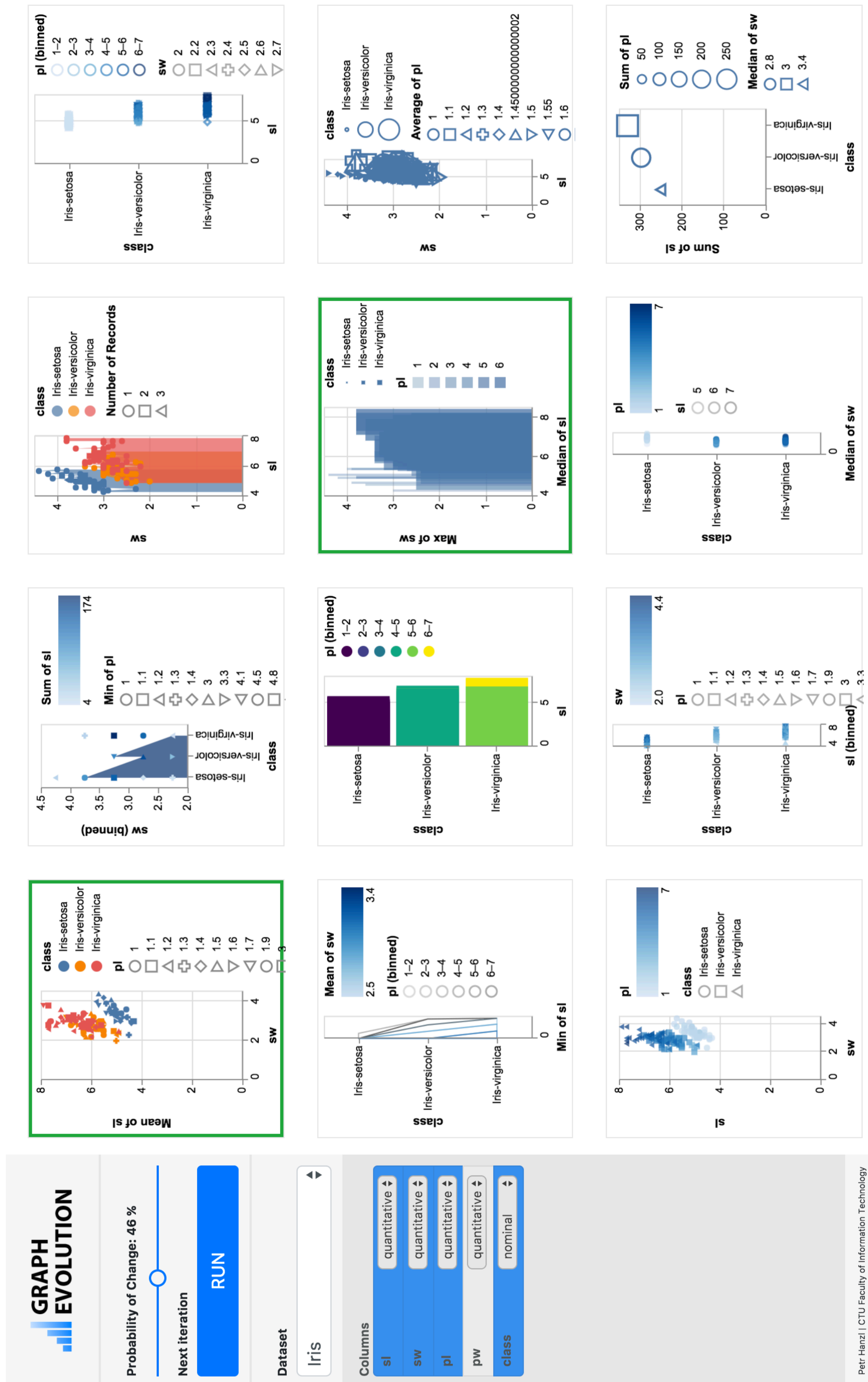



**Figure 5.1:** Graphical User Interface of implemented prototype

Chapter 6

# Experiments

To shown how the prototype works, I have chosen tree experiments. All of them defines visualizations, that the prototype should produce in the end. I also show which visualization were chosen to be bred. All the measurements are strongly dependant on my knowledge, preferences, decisions that I make and randomness of evolutionary approach, which is problematic for the measurements, that are not objective, but strongly subjective and coincidence dependant. The measurements primarily show the actual interactive evolutionary computation applied on a grammar instead of efficiency of evolutionary algorithms.

## 6.1 Experiment 1: Students performance in Exams

The dataset contains information about students, their gender, race/ethnicity, scores in math, reading and writing and other information. The dataset was downloaded [22].

### 6.1.1 Goal

At first, I generated the visualization, that is shown in figure 6.1. I have chosen this visualization, because I would not be able to show crossover on an easier one. The goal of this experiment is to generate at least a similar visualization. It means, that the axes should be mapped in the same way to the fields **math score**

and **reading score**, but mark properties are arbitrary. The final visualization must distinguish categories of nominal field **gender**.

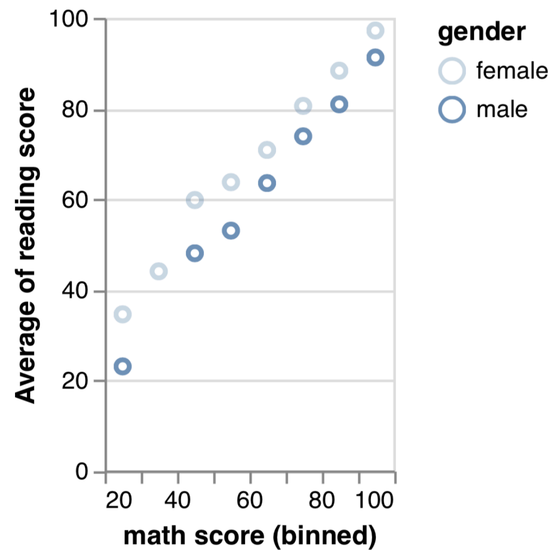



**Figure 6.1:** The desired visualization for the experiment 1.

### 6.1.2 Evolution

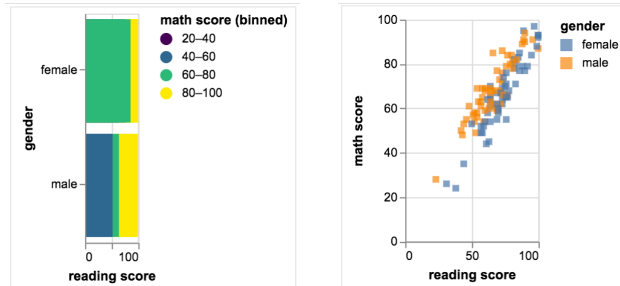
As it is shown in figure 6.2, there are two individuals selected for breeding in the first iteration, which is an initialization. Thwy are chosen, because the first one contains binning of **math score** field and the second is selected, because I prefer its mark encoding, which is **fill** instead of **bar**. Probability of change is set to zero, because I want get in the next generation only siblings, that are simple recombinations of the mappings of these two visualizations.

In the second interation, I get **math score** field mapped to **y** axis and **gender** field to **fill** mark property. The only feature, that is missing in the visualization, is the aggregation of **reading** field. This is fixed in the sixth iteration, after the individual from the second iteration is repeteadly chosen into four iterations with 15 % probability of change. As it is alone in the population, the only genetic operator applied on these on the individual is the mutation.

In the sixth iteration, algorithm returned desired visualization. The axis are swaped and it uses different mark property, but it shows the same relation

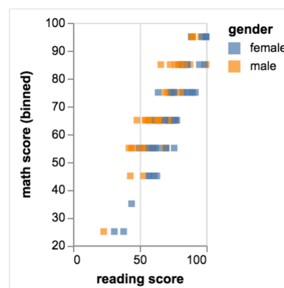
between **average reading score**, **gender** and **math score**, that is also binned. The final visualization specification is shown in listing 61.

**1ST ITERATION (INITIALIZATION)**




CROSSOVER & NO MUTATIONS
Probability of change: 0 %

**2ND ITERATION**




4 ITERATIONS WITH MUTATION AND NO CROSSOVER
probability of change: 15 %

**6TH ITERATION (TERMINATION)**

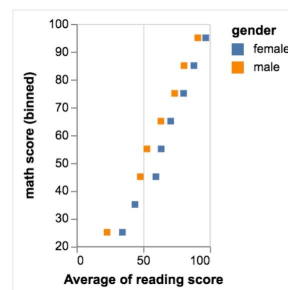



**Figure 6.2:** The diagram of chosen individuals during iterations for the experiment 1.

```
{
  /* Top-level Specifications and common properties were removed,
   because they are not evolved by the prototype. */
  "mark": "square",
  "encoding": {
    "fill": {
      "field": "gender",
      "type": "nominal"
    },
    "x": {
      "field": "reading score",
      "type": "quantitative",
      "aggregate": "average"
    },
    "y": {
      "type": "quantitative",
      "bin": true,
      "field": "math score"
    }
  }
}
```

**Listing 61:** The final visualization's specification from the experiment 1.

### 6.1.3 Results

**Table 6.1:** Measurings of needed interations to reach sufficient visualization for experiment 1.

| Measuring | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| # of interations | 6 | 18 | 25 | 8 | 33 | 17 | 52 | 2 | 43 | 26 | 23 |

## 6.2 Experiment 2: Heart Diseases

The dataset contains information about patients with heart diseases, their gender, type of chest pain that the patient reported, a resting blood pressure, and more vital signs. The dataset was also downloaded from Kaggle [20].

### 6.2.1 Goal

The goal for the second use case is to find such visualization of **Heart Diseases** dataset, that shows different types of chest pain **cp** by **age** and with maximum heart rate achieved **thalach**.

### 6.2.2 Evolution

The process of evolution is shown in figure 6.3. In the first iteration, only one individual was chosen for breeding, because the other individual were subjectively considered as not promising for breeding. From the second iteration to fifth iteration, I was not satisfied with generated individuals and this is the reason, why the same individual was used from the second iteration until I was satisfied enough with two new individuals that I tried to breed. The first parent was chosen, because I thought that a stacked bar graph could be a good approach to reach the goal. The second parent was chosen, because I wanted to save the same mapping of axes. In sixth iteration, I was not satisfied with generated offsprings, so I tried to breed the same parents again in the seventh iteration, which gave me one promising visualization, which I tried to slightly mutate, so I lowered the probability of mutation rate. This gave me two promising visualizations. However, the combination of them took five more iteration to breed such an individual, that I thought is good enough. I needed to add some aggregation to the vizualization, so I set the probability of change relatively low with expections, that the mutation would change only a small part of individual's genotype. By that, I produces stacked bar chart, that shows, which groups of people have problems with heart rate levels accompanied by which chest pains. The final specification is shown in listing 62.

### 6.2.3 Results

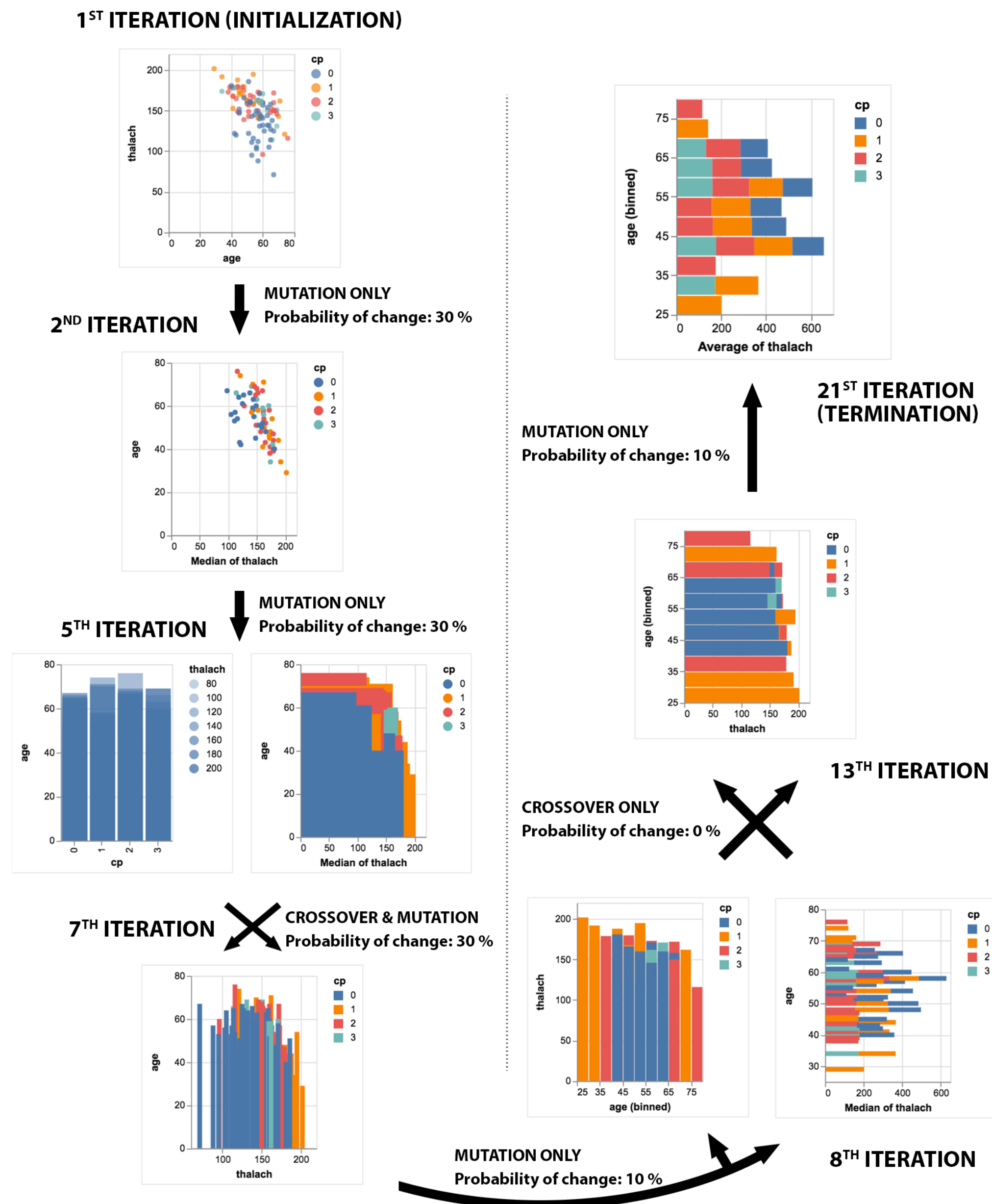



**Figure 6.3:** The diagram of chosen individuals during iterations for the experiment 2.

**Table 6.2:** Measurings of needed interations to reach sufficient visualization for experiment 2.

| Measuring | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| # of interations | 21 | 17 | 19 | 6 | 2 | 5 | 11 | 13 | 16 | 10 | 12 |

```
{
  /* Top-level Specifications and common properties were removed,
   because they are not evolved by the prototype. */
  "mark": "bar",
  "encoding": {
    "color": {
      "field": "cp",
      "type": "nominal"
    },
    "x": {
      "aggregate": "average",
      "field": "thalach",
      "type": "quantitative"
    },
    "y": {
      "field": "age",
      "type": "quantitative",
      "bin": true
    }
  }
}
```

**Listing 62:** The final visualization's specification from the experiment 2.

## 6.3 Experiment 3: Graduate Admission

The dataset contains information are considered to be important for application on master's studies. The information includes TOEFL scores, GRE scores, Change of Admit, university rankings and more. The dataset was also downloaded from Kaggle [1].

### 6.3.1 Goal

Find a bar graph, that shows average change of admit for different university rankings.

### 6.3.2 Evolution

Because of expected simplicity of the final graph, I bet on random intialization every iteration to get two axes with **University Ranking** and **Average of Change of Admit**. After several iterations, both axes were present in some graphs, so I compounded them into one graph by crossover with 0 % probability of change. The same approch is described in figure 6.3. The final specification is shown in listing 63.

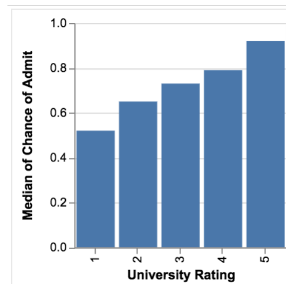
### 6.3.3 Results

**Table 6.3:** Measurings of needed interations to reach sufficient visualization for experiment 2.

| Measuring | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| # of interations | 5 | 1 | 4 | 3 | 3 | 2 | 2 | 2 | 14 | 2 | 3.8 |

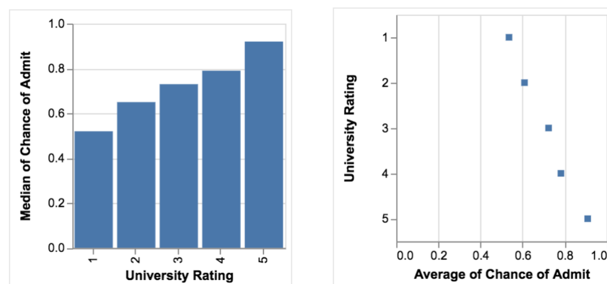
## 6.4 Summary

As it was predicted before, all the experiments took variant number of iterations to reach the desired goal. I have noticed that EA usually reaches an almost same looking visualization in just a few iterations, but struggles to produce from this visualization the exactly same look visualization from the goal. This happens, because the lowest changes in the tree genotype are typically in the lowest level











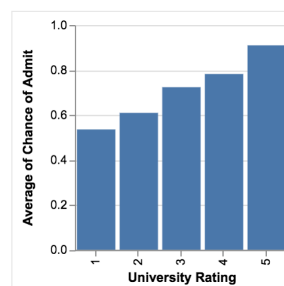




**Figure 6.4:** The diagram of chosen individuals during iterations for the experiment 2.

```
{
  /* Top-level Specifications and common properties were removed,
   because they are not evolved by the prototype. */
   "mark": "rect",
   "encoding": {
     "x": {
       "type": "nominal",
       "field": "University Rating"
     },
     "y": {
       "field": "Chance of Admit",
       "aggregate": "average",
       "type": "quantitative"
     }
   }
 }
```

**Listing 63:** The final visualization's specification from the experiment 2.

of this tree and the mutation process, during the traversal of the tree, more-likely changes a totally different leaf of the tree or it changes the whole subtree.

**Table 6.4:** Summary of needed iterations for all experiments

| Experiment 1 | Experiment 2 | Experiment 3 |
|---|---|---|
| 6 | 10 | 5 |
| 18 | 21 | 1 |
| 25 | 17 | 4 |
| 8 | 19 | 3 |
| 33 | 6 | 3 |
| 17 | 2 | 2 |
| 52 | 5 | 2 |
| 2 | 11 | 2 |
| 43 | 13 | 14 |
| 26 | 16 | 2 |
| 26 | 16 | 2 |
| **Avg.** | **Avg.** | **Avg.** |
| 26 | 16 | 2 |

# Conclusion

In this bachelor thesis, the field of interactive evolutionary computation and evolution of grammars was described. Also, two methods Genetic Programming and Grammatical Evolution for generating graphical interfaces were described.

Based on the information obtained during writing this thesis, genetic programming was adapted to generate graphical visualizations in Vega framework using the methods of interactive evolutionary computation and grammar evolution. Also, custom mapping of the grammar to JSON syntax was described.

Several techniques for shrinking the size of the search space were implement and described in this thesis. Their implementation lead prevent user fatique.

Three different experiments were done and the number of iteration were measured to confirm functionality of the prototype. The reasons, why the measurements do not have any higher meaning, were also investigated and described.

This bachelor thesis, the prototype and the described heuristic approaches can serve as a promising basis of an application from which a website could arise.

# Bibliography


1. ACHARYA, M. *Graduate Admissions | Kaggle* [online] [visited on 2019-05-13]. Available from: https://www.kaggle.com/mohansacharya/graduate-admissions.
2. BÄCK, T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms* [online]. Oxford University Press, 1996 [visited on 2019-04-23]. ISBN 0195099710. Available from: https://books.google.cz/books?id=htJHI1UrL7IC.
3. BAECK, T. *Handbook of Evolutionary Computation* [online]. 1997 [visited on 2019-04-24]. Available from: https://www.taylorfrancis.com/books/9781420050387.
4. DARWIN, C. R. *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life.* London: John Murray, 1859. Available also from: http://darwin-online.org.uk/converted/pdf/1859_Origin_F373.pdf.
5. DRAVES, S. *Electric Sheep - Crowdsourced Evolving Art* [online]. 2019 [visited on 2019-04-20]. Available from: https://electricsheep.org.
6. EIBEN, A.E.; SMITH, James E. *Introduction to Evolutionary Computing*. Springer-Verlag Berlin Heidelberg, 2003. ISBN 9783662050941.
7. *Electric Sheep Archive* [online]. 2019 [visited on 2019-04-20]. Available from: http://electricsheep.com/archives/.

8. *Encoding | Vega-Lite – Channel Definition* [online]. 2019 [visited on 2019-05-11]. Available from: `https://vega.github.io/vega-lite/docs/encoding.html#channel-definition`.
9. FERREIRA, C. *Gene Expression Programming: A New Adaptive Algorithm for Solving Problems* [online]. 2001 [visited on 2019-04-27]. Available from: `https://www.gene-expression-programming.com/webpapers/GEP.pdf`.
10. *GenJam* [online]. 2018 [visited on 2019-04-16]. Available from: `https://igm.rit.edu/~jabics/GenJam.html`.
11. JOHNSON, M.; ZELENSKI, J. *Formal Grammars* [online]. 2012 [visited on 2019-05-13]. Available from: `https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/080%20Formal%20Grammars.pdf`.
12. KOZA, J. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems* [online]. 1990 [visited on 2019-04-23]. Available from: `http://www.genetic-programming.com/jkpdf/tr1314.pdf`.
13. MEHLHORN, K.; SANDERS, P. *Algorithms and Data Structures: The Basic Toolbox* [online]. Springer Science & Business Media, 2008 [visited on 2019-04-26]. ISBN 9783540779780. Available from: `http://people.mpi-inf.mpg.de/~mehlhorn/ftp/Toolbox/Introduction.pdf`.
14. MIGHT, M. *Grammar: The language of languages* [online] [visited on 2019-05-14]. Available from: `http://matt.might.net/articles/grammars-bnf-ebnf/`.
15. O'NEILL, M.; RYAN, C. *Grammatical Evolution* [online]. 2001 [visited on 2019-04-23]. Available from: `https://ieeexplore.ieee.org/document/942529`.
16. *Picbreeder* [online]. 2019 [visited on 2019-04-20]. Available from: `http://picbreeder.org`.
17. POHLHEIM, H. *Evolutionary Algorithms 10 Combination of Operators and Options to Produce Evolutionary Algorithms* [online]. 2006 [visited on 2019-04-26]. Available from: `http://www.geatbx.com/docu/algindex-09.html`.

18. QUIROZ, Juan C. *Interactive Genetic Algorithms for User Interface Design* [online]. 2007 [visited on 2019-04-20]. Available from: https://ieeexplore.ieee.org/document/4424630/.
19. RONG YAN, Jun; MIN, Yong. User Fatigue in Interactive Evolutionary Computation. *Applied Mechanics and Materials*. 2011, vol. 48-49. Available from DOI: 10.4028/www.scientific.net/AMM.48-49.1333.
20. RONIT. *Heart Disease UCI | Kaggle* [online] [visited on 2019-05-13]. Available from: https://www.kaggle.com/ronitf/heart-disease-uci.
21. *Scott Draves - Software Artist* [online]. 2009 [visited on 2019-04-20]. Available from: https://scottdraves.com/sheep.html.
22. SPSCIENTIST. *Students Performance in Exams | Kaggle* [online] [visited on 2019-05-13]. Available from: https://www.kaggle.com/spscientist/students-performance-in-exams.
23. TAGAKI, H. *Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation* [online]. 2001 [visited on 2019-04-16]. Available from: https://ieeexplore.ieee.org/document/949485.
24. UNIVERSITY, Newcastle. *Fitness proportiate selection* [online] [visited on 2019-04-26]. Available from: http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php.
25. *Vega – A Vizualization Grammar* [online]. 2019 [visited on 2019-05-11]. Available from: https://vega.github.io/vega-lite/.
26. *Vega-Lite – A Grammar of Interactive Graphics* [online]. 2019 [visited on 2019-04-16]. Available from: https://vega.github.io/vega-lite/.
27. *Vega-Lite – Single View Specifications* [online]. 2019 [visited on 2019-05-11]. Available from: https://vega.github.io/vega-lite/docs/spec.html#single.
28. *Vega-Lite – Single View Specifications* [online]. 2019 [visited on 2019-05-12]. Available from: https://vega.github.io/vega-lite/docs/encoding.html.
29. YOURDICTIONARY, n.d. Web. *Examples of Genotype & Phenotype.* [online]. 2019 [visited on 2019-04-26]. Available from: https://examples.yourdictionary.com/examples-of-genotype-phenotype.html.

Appendix A

# List of Acronyms

| | |
|---|---|
| **BNF** | Backus–Naur Form |
| **EA** | Evolutionary Algorithm |
| **EC** | Evolutionary Computation |
| **GA** | Genetic Algorithm |
| **GE** | Grammatical Evolution |
| **GP** | Genetic Programming |
| **GUI** | Graphical User Interface |
| **IEA** | Interactive Evolutionary Algorithm |
| **IEC** | Interactive Evolutionary Computation |
| **JSON** | JavaScript Object Notation |

Appendix B

# Supplemental Material

The source code of the thesis and the implementation can be found on the attached medium or online at GitHub.

**Thesis** `https://github.com/Lznah/thesis`

**GraphEvolution** `https://github.com/Lznah/graphevolution`

```
├── README.md ............................ the file with a brief contents description
├── BT_Petr_Hanzl_2019.pdf ............... the thesis text in PDF format
├── GraphEvolution/ ...................... repository for the prototype
│   ├── app/ ............................. source code of the prototype
│   └── release-builds/ .................. release builds for different platforms
└── thesis/ .............................. the directory of LaTeX source codes of the thesis
```

**Directory structure B.1:** Contents of the attached medium