UNIVERSITÄT SALZBURG

MASTER THESIS

Portfolio Optimization using Variable Size Genetic Network Programming

Author:
Fabian Köhnke

Supervisor:
Univ.-Prof. Dipl.-Inform.
Dr.-Ing. Christian Borgelt

A thesis submitted in fulfillment of the requirements for the degree of M.Sc. Data Science

at the

Universität Salzburg
FB Artificial Intelligence and Human Interfaces

June 29, 2023

UNIVERSITÄT SALZBURG

Abstract

Genetic Network Programming (GNP) is an extension of Genetic Programming and has been developed and researched since around 2000. As an optimization technique, problems with a large search space could be successfully solved by GNP. Furthermore, due to the network structure and the possibility of reusing nodes, these networks can develop an implicit memory function. For this reason, GNP targets dynamic environments and is successfully applied in fields such as robotics and financial market analysis.

This thesis presents an extension of the GNP by two novel mutation operators. These novel operators allow the GNP to be variable in the number of nodes and to change the function outputs of nodes. Extending the search space without danger concerning overfitting or the bloat problem is feasible using both operators. With a higher solution capability, it is now possible for the GNP to automatically adapt to the complexity of a given problem and to find suitable features of large data sets. The mutation operators are successfully applied to the Iris data set and a financial data set where there could improve the standard GNP and show better performance concerning portfolio optimization. As shown in simulation studies and the experimental part of this thesis, up to 0.38 more accuracy could be achieved by shrinking networks, and up to 0.52 more accuracy could be achieved by growing networks. Furthermore, experiments showed that the standard GNP could be beaten by 61.84% return on the financial data.

Acknowledgements

I want to express my deep gratitude to Univ.-Prof. Dr.-Ing. Dipl.-Inform. Christian Borgelt. Having a space for open questions in regular meetings was very helpful. I enjoyed the discussions, which greatly contributed to my deep understanding of the subject.

Contents

List of Figures

List of Tables

List of Abbreviations

EA	**E**volutionary **A**lgorithm
GA	**G**enetic **A**lgorithm
GNP	**G**enetic **N**etwork **P**rogramming
GP	**G**enetic **P**rogramming

List of Symbols

ϕ	selection pressure
$U(.)$	uniform distribution
$N(.)$	normal distribution

Chapter 1

Introduction

The American mathematics professor and hedge fund manager Edward O. Thorp mathematically proved that the house advantage in blackjack could be overcome by card counting. Then, to beat roulette, he and the father of information theory, Claude Shannon, invented the first wearable computer. What does this have to do with the portfolio optimization in the title of the thesis? Well, because this was not enough, Edward O. Thorp also pioneered the use of quantitative investment techniques in the financial markets and asked in one of his books: Can you beat the market? Should you try? The answer is as "simple" as it is short:

> *"Find a superior method of analysis." [Thorp, 2017]*

Finding such a method sounds like an excellent computer intelligence task. Motivated by the attempt to beat the market, the research for a suitable method began with the primary goal of the thesis: beat the market by applying an active portfolio optimization [1] method to gain more returns compared to a buy-and-hold strategy. At the moment, this is the main goal. After a theoretical introduction to the topic of this thesis, concrete research questions and objectives are formulated at the end of this chapter.

1.1 Theoretical Foundation

One of the main tasks in intelligence data science is finding explanations for an unknown dependency within the data. Usually, a data scientist is interested in why a specific attribute has a particular value. If the model learns by the desired output Y, this type of model is called supervised learning. In the case of a nominal variable, this is often referred to as a classification problem, whereas in the case of numeric variables, it is referred to as a regression problem. This thesis assumes that, in addition to the object description x, the target attribute Y is also available to fit a model (supervised learning). When the target is to find explanations and understand the dependencies of the target variable to the input vectors, it is reasonable to use models where the output is interpretable [Berthold et al., 2010].

1.1.1 From Decision Tree to Genetic Network Programming

The likely most prominent and heavily used method for interpretable model extraction from large data sets is the Decision Tree. This is because they can learn complex relationships in an easy-to-understand manner, and efficient algorithms exist

[1] Portfolio optimization is mainly a task where assets are selected for gaining profits or/ and reducing risks. Compared to index investing, where the investments are passive and mimic a given index, actively managed portfolios try to outperform benchmarks.

to build such models. Typically in computer science related areas, trees grow from top to bottom. The tree builds a hierarchical decision structure that can be traversed from the root node until a leaf is reached. Each inner node is investigated concerning a corresponding attribute, and the branch matching the attribute's value is followed. In the end, the leaves hold the classification. The most prominent Decision Tree algorithms employ a greedy strategy to fit the data. The algorithms focus on building the tree root-first and then adding subsequent branches and splits along those branches. How to split and which attribute is the best is the result of an evaluation measure. Because a "best split" would be one that helps to create leaves as soon as possible, an information-theoretic evaluation measure called Shannon entropy (named after the same Claude Shannon who beat the casino in roulette with Edward O. Thorp mentioned above) is often used [Berthold et al., 2010]. Because of the greedy aspect of finding the best attribute that provides the best split, it is ignored that further splits are possible deeper down the tree, which could improve the classification. In this sense, the attribute selection is greedy. It follows that the "greedy" aspect does not try to find a globally optimal tree but hopes such local choices will yield a sufficiently good tree.

Let us look at a straightforward example of finding explanations for a dependency within the data using a Decision Tree. Table 1.1 shows two attributes A and B and a target variable representing the logical disjunction $A \vee B$. A trained Decision Tree with the Shannon entropy as an evaluation measure leads to the Decision Tree next to the table shown in Figure 1.1. First, the root node investigates the attribute A. If the attribute value is $a = 1$, the path already ends by the leaf holding the classification 1. If the attribute value is $a = 0$, then the attribute B is investigated similarly to attribute A. Overall, the decision tree represents logical disjunction. It is important to note that the order in which the data (points) are presented does not matter for the decision tree result. Therefore, if the Index column of Table 1.1 had a different order, the decision tree would not change. Moreover, since the data do not contain any error, the classification accuracy is 100%.

Table 1.1: Logical Disjunction
Source: own illustration

Index	A	B	$A \vee B$
1	1	1	1
2	0	1	1
3	1	0	1
4	0	0	0
...	...	...	...
N	0	0	0

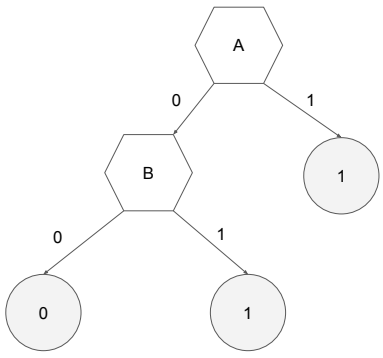

Figure 1.1: Resulting Decision Tree with Accuracy 100%
Source: own illustration

Now let us assume that the two attributes A and B are indicators of financial market analysis. The Table 1.2 shows this with the attributes A and B and the target variable T. If an indicator shows a 1, it indicates a buy signal; if a 0 is given, it indicates a sell signal. The target variable T is the price history of a particular asset, as Figure 1.2 next to the table shows, where a 1 represents an upward movement

and a 0 represents a downward movement. Note that the attributes A and B and the target variable T implicitly assume a time difference between observation and outcome: one would like to buy before a price increase, and one would like to sell before a price drop.

To illustrate the price movement, the index and the target variable T from the Table 1.2 are plotted on the X-axis. To stay with the previous example of disjunction, prices rise when one of the indicators provides a buy signal, whereas prices fall when both indicators provide a sell signal. Thus, prices are expected to rise when one of the two indicators delivers a buy signal.

Table 1.2: Example Data Source: own illustration

Index	A	B	T
1	1	1	1
2	0	0	0
3	1	0	1
4	0	0	0
5	0	1	1
6	0	0	1
7	1	0	1
8	1	1	1
9	0	0	0
10	1	0	1
11	0	0	0
12	0	1	1
13	0	0	1
14	1	0	1
15	1	1	1
16	0	0	0
17	1	0	1
18	0	0	0
19	0	1	1
20	0	0	1
21	1	0	1

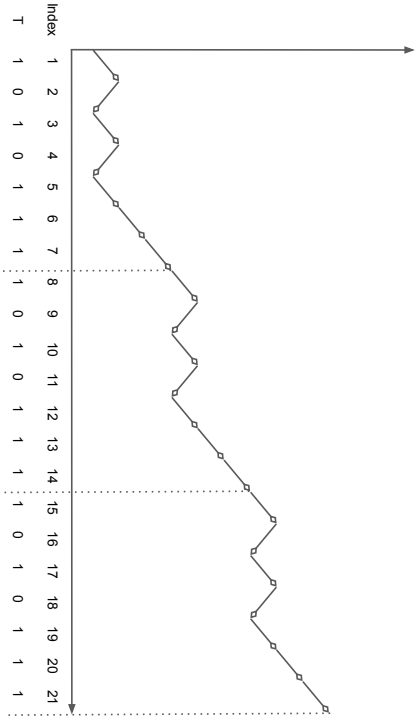

Figure 1.2: Example Time-Series representing a Price Chart Pattern.

A closer look at the data tells us that the attributes A and B with the corresponding target T are almost the same as the data in the previous example in Table 1.1. Therefore, the same decision tree as in the logical disjunction example in Figure 1.1 results when trained on the data. The crucial difference now is that this decision tree only has an accuracy of $\frac{18}{21} \approx 85.71\%$. This is because the indicators make three errors. Both indicators provide a sell signal at the red marked index points 6, 13, and 20, but the target variable *increases* at these data points. Nevertheless, since this error occurs only three times, the trained decision tree ends up in the exact same as Figure 1.1.

Now, is there a way to get an improvement from the data? Unlike the disjunction example above, this data set has a temporal order (it refers to a time series). An

even closer look at the data shows that the error of the indicators always appears when the pattern with the attribute values $a = 0$ and $b = 1$ and the target variable $t = 1$ occurs before. The corresponding rows in Table 1.2 are marked in gray. It would be great if a model could derive decisions from the previous attribute values and the target variable. So, how can a model make predictions depending on previous decisions and evaluations, i.e., have implicit memory? Unfortunately, this is no longer possible with a decision tree because it always goes back to the root node after reaching a leaf node. The learning ability from previous decisions leads us to a *network structure*.

Genetic Network Programming, abbreviated to GNP, is such a network structure. Therefore, its often denoted as a graph-based algorithm. Each traversal no longer ends in a leaf node, and all nodes have at least one outgoing edge connected to another node. In GNP, leaf nodes are called processing nodes, and inner nodes are called judgment nodes. Processing nodes work as action/processing functions and can take actions concerning the target variable. Judgment nodes have conditional branch decision functions. Each judgment node returns a judgment result and determines the next node to be executed. It can be thought of GNP as a decision *graph* [2] rather than a decision tree.

Figure 1.3 shows an example GNP, representing the data in Table 1.2. The network is not much different from the learned decision tree but has a crucially different structure.

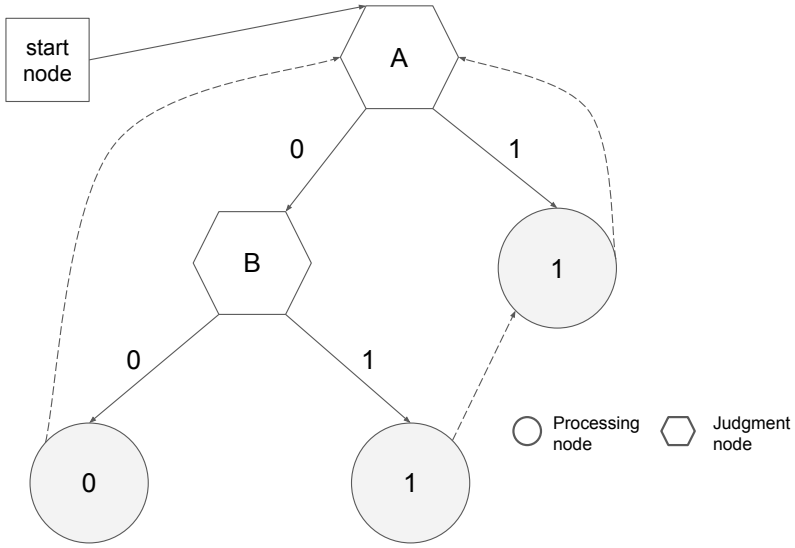

Figure 1.3: GNP Illustration of Table 1.2 with Accuracy 100%
Source: own illustration

Judgment nodes are shown as hexagons, whereas processing nodes are drawn as circles. Since such a network no longer starts at a specific root node, an additional start node initializes the network. The most important difference to a decision tree is that *all* nodes have an outgoing edge. The processing nodes point to another node and determine the further run through the network. If all processing nodes from

[2] Not to be confused with decision graphs as an extension of Bayesian networks, which are not the subject of this thesis.

Figure 1.3 would point back to the Judgment node A, the network would describe exactly the decision tree from Figure 1.1. However, the outgoing edge of the processing node with the 1 points to the other processing node, which also returns a 1. So if the case occurs where the indicator A has a 0, then the judgment node B is the next node. If the indicator B has a 1 at the same time, the processing node 1 is selected, which points to the other processing node 1. So the pattern is: $a = 0, b = 1, t = 1$ which predicts a 1. And this is exactly the pattern that the time series of the asset holds, and the error of the indicators is skipped. This means that the network could learn from its previous decisions, and compared to the decision tree, it has now learned the time series with an accuracy of 100%.

Of course, this is a constructed example, but it makes very nicely clear the difference between the learning ability of a decision tree and a network structure. Notice that this is only possible if the data has a fixed order. A different order of the indices can no longer reach the same results. For time series data, such a fixed order and time-depending structure is given, which is why GNP is perfectly suited for financial time series data. In addition, it has to be noted that a formally correct description of the GNP takes place in Chapter 2, and the Figure 1.3 is used just for illustration.

So, overall, the network was better able to learn the data from the Table 1.2. However, how exactly can such a network be learned? Unlike the decision tree, the judgment and processing nodes cannot use heuristics for a conditional decision. A decision criterion like the Shannon criterion is no longer appropriate. This is caused by the fact that after the initialization of the network, each node's decision depends on the previous state. Therefore, using the whole data set no longer works because it is no longer clear which sample cases/data points will be processed by a node, as this depends on the path through the network taken by preceding sample cases/data points. However, the entire network can be evaluated after the data has been passed through.

Enumerating all combinations of possible networks and evaluating them is impractical. Even the set of directed, acyclic graphs grows superexponentially in the number of nodes. Already for ten nodes, the graph set has a cardinality of $4.18 \cdot 10^{18}$ [Kruse et al., 2015]. For GNP, the search space is even larger since GNP allows directed multigraphs. A multigraph is a graph that is permitted to have multiple edges. Thus two vertices may be connected by more than one edge. A formal definition is given on page 30. Accordingly, performing a complete traversal of the search space is infeasible.

A straightforward solution would be randomly generating a network with a given number of nodes. Then a new network is created and compared with the previous ones. If the new network is better than the previous one, keep it; otherwise, discard it. This process can then be continued up to a termination criterion to obtain a sufficiently good network. It should be noted that the initializations of the networks are independent of each other, and the search space structure is not exploited. The networks are drawn at pure random from the entire solution set. This is equivalent to initializing a certain number of networks and accepting the best network from this set as the best solution. Therefore, such a search through the search space is also called a *blind random search*. Even though this method is very efficient, the quality of the solution depends purely on randomness. An extension that exploits the structure of the search space in addition to the random search leads us to the so-called *metaheuristics* [Kruse et al., 2015].

1.1.2 Metaheuristics

Metaheuristics is a general term for the stepwise solution of numerical and combinatorial optimization problems. Typically, metaheuristics are applied to problems for which no efficient solution algorithm is known, i.e., problems for which all known algorithms have a time complexity that grows exponentially with the problem size. Since such complex problems have unsatisfying computational time and computational power requirements, approximate solutions must be accepted. Many metaheuristics operate on the principle of stepwise improvement of so-called candidate solutions. However, they differ in how candidate solutions are combined, how elements of known candidate solutions are exploited to find new candidate solutions, or how a new set of candidate solutions is selected from those previously generated. In doing so, they have the property of performing a *guided random search*. Thus, the search contains certain random elements and is also guided by a measure of solution quality. Many metaheuristics use methods inspired by nature. This is also the case for *evolutionary algorithms*, which are inspired by the different ways of biological evolution [Kruse et al., 2015].

1.1.3 Evolutionary Algorithms

Evolutionary algorithms are among the oldest metaheuristics. They are based on the theory of biological evolution developed by Charles Darwin. The basic principle of biological evolution can be formulated as follows:

> *Beneficial traits resulting from random variation are favored by natural selection. [Kruse et al., 2015]*

This principle is often referred to as differential reproduction, where individuals with beneficial traits have better chances to reproduce. Two processes, in particular, can produce new or modified characteristics:

1. mutation
2. sexual reproduction

Most changes in the characteristics of an individual are unfavorable or even harmful. Nevertheless, there is a slight chance that some changes will result in characteristics that help the individual to survive. For example, the new characteristic makes it easier to get food. Generally, every individual in their natural habitat is subjected to a test. Thereby it turns out either that it has a high fitness, so that it has a good chance to survive and reproduce or that it has less chances to reproduce and the characteristics of the individual may disappear [Kruse et al., 2015].

The following example shows the difference between a *blind random search* and a *guided random search*, where the goal is to find a global optimum.

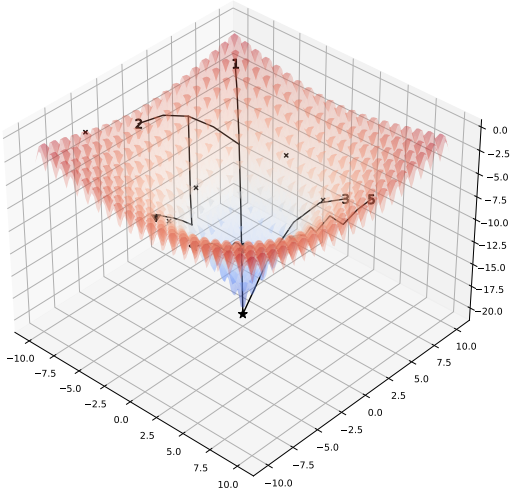

Figure 1.4: Ackley Function 3D Representation
Source: own illustration

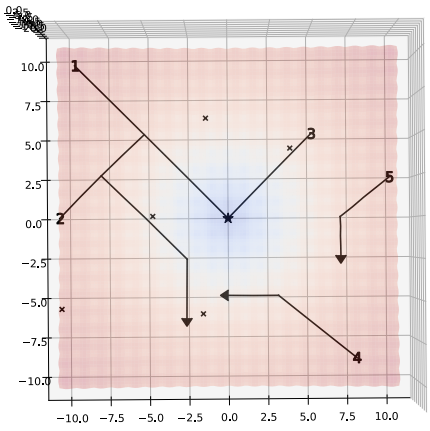

Figure 1.5: Ackley Function 2D Representation
Source: own illustration

Figure 1.4 shows the two-dimensional Ackley function. The definition of this function can be found in the appendix A.1. A figure like 1.4 is often called a fitness landscape because it illustrates how a fitness function varies with independent variables. So, the landscape shows the search space of an optimization problem. In Figure 1.4, it can be seen that the Ackley function has many local minima. The global minimum is shown at the bottom of the funnel-like base of the function at $x = 0$ and $y = 0$. To illustrate the difference between blind and guided random search, five solution candidates each were initialized [3].

The five solution candidates of the blind random search are represented as a cross, and the five solution candidates of the guided random search are illustrated as a number at the beginning of the paths. The difference between the two methods can already be seen from this. The solution candidates of the blind random search are randomly selected and are only points in the search space. The guided random search, on the other hand, can change based on the processes of evolutionary algorithms and tries to exploit the structure of the search space. Therefore, the candidate solutions across generations now represent a path. Since the global optimum is minimal, bad solutions are shown in reddish and better solutions go into bluish. The global optimum is marked as a star. It can be seen that paths of the guided random search can merge and also separate. This can be seen when the three-dimensional figure from above is displayed in two dimensions. Figure 1.5 shows the drawn Ackley function with the solution candidates from the bird's eye view. The five randomly selected candidate solutions can be seen scattered across the search space, with none inside the desired blue region. The solution candidates of the guided random search also initially start far away from the global optimum. However, the candidate solutions can converge to the global optimum over several generations through differential reproduction and modification from the genetic material.

Further, paths can collide and split with the mutation and the crossover of two individuals. This can be seen in paths 1 and 2, which collide, and path 2 additionally

[3] Solution candidate is the equivalent term for an individual of a population when considering an Evolutionary Algorithm

runs away from path 1. At the end of the evolution, path 1 (with the combined genetic material of path 2) and path 3 could reach the global optimum. However, paths 2,4, and 5 could only find a local optimum. Here it also becomes clear that the individuals of a population usually show fitness differences. A detailed description of selection mechanisms and genetic operators is given in Chapter 2.

In conclusion, with a guided random search using differential reproduction, exchanging genetic material and mutation, hopefully, solution candidates reach an optimal solution more efficiently than a blind random search.

Since this is a promising method to solve optimization problems, many studies have been made, and all of them belong to the general term of Evolutionary Algorithms. Genetic Network Programming as an extension of Genetic Programming also belongs to the Evolutionary Algorithms and will be introduced in the next section.

1.1.4 Introduction to GNP

Many studies have been developed on Evolutionary algorithms (EAs) to solve optimization problems. Genetic Algorithm (GA), Genetic Programming (GP), and Evolutionary Programming (EP) are typical evolutionary algorithms [Mabu, Hirasawa, and Hu, 2007].

Genetic algorithms and similar EAs have an intrinsic limitation: They incorporate the assumed solution structure in the representation of their candidate solutions. On the one hand, that gives the possibility to encode problem-specific information into the solution representation. On the other hand, practical problems are often such complex problems that the programmer does not know which parameter needs to be optimized and may not know the structure of those parameters. GP attempts not only to learn the best solution to a problem given a specific structure, but GP can also learn the optimal structure. Thus, GP does not constrain its solution like other EAs, and its nesting is unrestricted in the size, shape, and structure best suited to the given problem [Simon, 2013].

The unlimited size of GP comes from allowing chromosomes to be of different sizes. A formal grammar is used, describing GP's language and leading to symbolic expressions. Such a symbolic expression can be, e.g., the character sequence "$(+(*\ 3\ x)(/\ 8\ 2)))$," which describes the following term:

$$3 \cdot x + \frac{8}{2} \tag{1.1}$$

A symbolic expression is usually represented as a so-called *parse tree*, shown in Figure 1.6. Therefore, the GP makes use of a tree structure for the representation of a candidate solution.

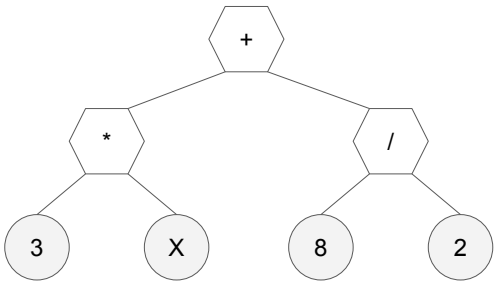

Figure 1.6: Parse Tree Example of Symbolic Expression
Source: own illustration according to [Berthold et al., 2010]

The most crucial problem for GP is the bloat of the parse tree. The increase in depth causes an exponential growth of the search space and calculation time. Constraining the tree's depth is one way to overcome the bloat problem.

Genetic Network Programming, abbreviated to GNP, is a graph-based evolutionary algorithm and can be seen as an extension of classical Genetic Programming. Katagiri, Hirasama, and Hu (2000) first proposed GNP as an application to intelligent agents in a dynamic environment. Compared to GP, which always uses tree structures, each individual is encoded as a network structure which means that the connection rule is different from a GP system. Further, GNP evolves the graph structure with the predefined number of nodes, so it never causes bloat and tackles the main problem of GP. Because of the network structure, GNP does not start from a root node at each individual's sequence and ends up in a leave node, unlike GP.

The building blocks of a GNP are the same as those of a standard evolutionary algorithm, where the algorithm employs evolution principles to generate increasingly better solution candidates. To achieve that, also GNP tries to evolve a population of solution candidates with the help of random variation and fitness-based selection of the next generation. The process of a GNP contains the same ingredients as any evolutionary algorithm :

- an *encoding* the solution candidate
- a method to create an *initial population*
- a *fitness function* to evaluate the individuals
- a *selection method* on the basis of the fitness function
- a set of *genetic operators* to modify chromosomes
- a *termination criterion* for the search, and
- values for various parameter

These ingredients are described in detail with respect to GNP in Chapter 2. But now, this chapter is concluded with concrete goals and the preceding motivation.

1.2 Motivation

Since GNP is specially designed for modeling dynamic environments and an implicit memory function is inherent due to the network structure, it seems reasonable

to use GNP for financial time series data. This was also illustrated by the example described above. Therefore, it is not surprising that many scientific papers on GNP use real economic data from the financial sector.

For example, Chen, Mabu, and Hirasawa (2010) suggested time adapting GNP to distribute capital to a set of stocks. Yan Chen and Xuancheng Wang (2015) extend GNP by statistical models where GNP generates trading rules, and the statistical model distributes the weight of capital. Chen et al. (2009a) proposed a GNP extension with reinforcement learning to obtain a trading model. Some variation to the GNP was made where the graph was altered from a directed graph to an undirected graph in which nodes are coded as particular stocks and edges work to determine the best relation between them [Chen, Mabu, and Hirasawa, 2011]. Also, recent studies combining GNP and MLP to forecast stock returns [Reza Ramezanian, Arsalan Peymanfar, and Seyed Babak Ebrahimi, 2019]. An overview of these papers is part of Chapter 3.

All the previous scientific studies on GNP in the context of financial market analysis have been carried out with GNP fixed number of nodes. As mentioned above, the fixed number of nodes of GNP is often consciously chosen because the fixed size can prevent the bloat problem. However, this advantage can be a disadvantage concerning the complexity of the given problem. When the problem is complex, it is impossible to get enough prior knowledge or to try many different sizes of GNP. On the one hand, if the number of nodes is too small, the network could lack expression ability. On the other hand, if the number of nodes is too large, it needs a large search space, and the individuals will be overfitting easily.

Recently, a new type of GNP has been introduced that allows a variable number of nodes by using a special crossover operator [Bing, 2013]. The newly introduced crossover operator exchanges different numbers of nodes between the selected individuals. A closer look at this operator is part of the Chapter 4. Although this method leads to variable-size GNP, the gene pool is still limited for the following reasons:

1. The number of outgoing edges of a node always stays the same.
2. The number of different node functions is limited.
3. The interval boundaries on the edges of numerical judgment nodes are limited.

To improve the variable GNP concerning these points, we have developed two new mutation operators for the GNP, which simultaneously counter the bloat problem. This now leads us to the following specific goals.

1.3 Thesis Subgoals

In addition to the main objective of successful implementation of GNP in finance, several sub-objectives and research questions emerged through the thesis research.

1. **Problem:** GNP is a fairly new subfield of Artificial Intelligence. Therefore, there are no public packages of the common programming languages. The current scientific results cannot be easily verified.

   Subgoal: Development of a software in the Python programming language to verify and explore the current scientific results.

2. **Problem:** Practical problems are often such complex problems that the programmer does not know which parameter needs to be optimized and may not know the structure of those parameters. GNP provides a good tool for solving such problems. However, the adaptation to the complexity of the problem is limited by a fixed initialization of nodes and gene pool.

   Subgoal: Development of new genetic operators that allow variable network sizes, is not affected by the bloat problem, and yet has an infinite gene pool.

   Subgoal: Improve the efficiency of GNP by initializing the networks with very few nodes and producing suitably large individuals through the evolutionary process.

   Subgoal: Dealing with large data sets and numerous features using new genetic operators to select appropriate judgment node functions during the evolutionary process.

1.4 Thesis Structure

Chapter 2 is the next chapter and gives a further theoretical basis for genetic network programming. For a better understanding of the GNP, examples are given from the author's coded program [4], where a model is trained on the Iris data set.

Chapter 3 briefly introduces financial market analysis and its applications by the GNP.

Chapter 4 contains the research part of this thesis and presents two novel mutation operators. Occurring difficulties will be discussed, as well as advantages and disadvantages. Furthermore, the performance is verified by simulation studies with the Iris data set, and the following project homepage provides additional information about the research of GNP by the author and introduces the novel mutation operation with some videos:

`https://fabiankoehnke.github.io/gnp/hp/#page-top`

Chapter 5 covers a concrete application of the GNP with the new mutation operators and deals with a financial market data set. Detailed results are presented there.

Finally, **Chapter 5** concludes the thesis, and further research opportunities are proposed.

[4] A screenshot of the program with a sample run is shown in Appendix A.2.

Chapter 2

Genetic Network Programming (GNP)

This chapter gives a more detailed look at genetic network programming. First, the explicit encoding of GNP is explained. Then it is described how exactly GNP are initialized and represented in computer memory. Finally, the procedure of selection mechanisms and genetic operators is described. For a better understanding of the GNP, examples are given from the author's coded program, where a model is trained on the Iris data set. The Iris flower data set is a standard data set for any data scientist where Iris flowers of three related species (Iris Setosa, Iris verginica, and Iris versicolor) are tried to classify. Four features were measured for each sample to classify the three species: the length and width of the sepal and petal in centimeters. Overall there are 150 records containing those measurements. The data set is free and is publicly available at the UCI Machine Learning Repository [Dua, Dheeru and Graff, Casey, 2017].

2.1 Encoding of GNP

How candidate solutions to the given optimization problem are encoded can considerably impact how easily an evolutionary algorithm finds a (good) solution. With an unfavorable encoding, it may not even find a helpful solution. Consequently, one should spend a lot of effort and care on designing a suitable encoding and the corresponding genetic operators [Kruse et al., 2015]. As mentioned above, GP is not fixed coded concerning the assumed solution structure, and GNP can represent different solution structures too. Compared to GP, encoded as a parse tree, GNP represents a whole network. The following sections describe the encoding of GNP.

2.1.1 Basic Components

Genetic Network Programming begins to boot at a start node, as shown in Figure 2.1, whereby the start node is executed in the transition path of the network only once. The only rule of the start node is to select the first node to be executed, and it does not have any other function than indicating the first node. Alternatively, one could mark one node in the network as the node where the transition path starts.

The network itself is a directed graph structure with nodes and edges and defines the phenotype of each individual. In addition to the start node, there are two kinds of nodes: the judgement node (J_n) and the processing node (P_n). The functions of these two node types decide all executions in the network. Figure 2.1 shows the processing node as a circle and the judgment node as a hexagon.

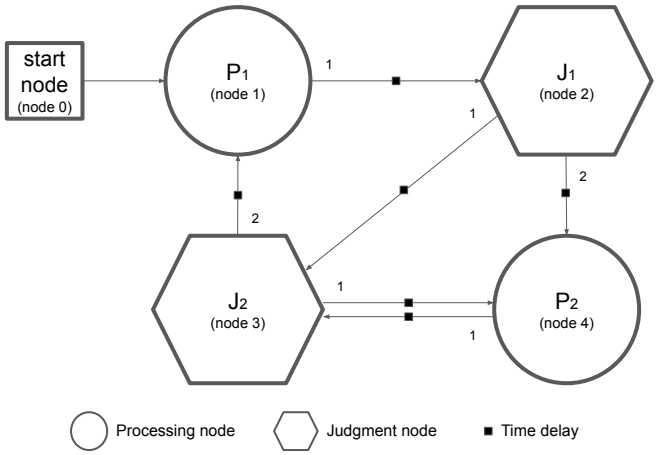

Figure 2.1: GNP graph structure.
Source: own illustration according to [Mabu, Hirasawa, and Hu, 2007]

J_n denotes the execution of the nth judgment and P_n the nth processing with an associated function, which is predefined in the judgment and processing nodes library. A judgment node has a conditional branch decision function and a processing node produces an output. As shown in the introductory example in section 1.1.1, a judgment can refer to a financial indicator and condition the next connection during the run of the network. The result of the processing node function could be a buy or sell decision. The programmer prepares the libraries, and all kinds of judgment and processing functions are set up in that library [Katagiri, Hirasama, and Hu, 2000]. The exact role of the library and its functions will become clear in the next section, with an example concerning the Iris data.

In Figure 2.1, one start node, two judgment nodes, and two processing nodes are shown, and they are connected with edges. The numbers mark the outgoing connections of the nodes. Because judgment nodes have conditional branch decision functions, there are always two or more outgoing connections to other nodes. In contrast, each processing node returns a processing result (e.g., buying or selling a stock or classifying flower species), and there is only one connection outgoing from that node to determine the next node. The GNP can use a large number of combinations of different nodes in a single run. How many judgments and which kinds of judgments are determined by the evolutionary procedure. Concretely speaking, the evolution of the GNP determined all possible transitions between the nodes. Suppose there are six judgment nodes $(J_1, ..., J_6)$ and three processing nodes $(P_1, .., .P_3)$, then a GNP could make an exemplary node transition path, as illustrated in Figure 2.2. In that case, the judgment nodes $1, 5,$ and 2 are needed for processing node P_1. A GNP can repeatedly use particular judgment and processing nodes to achieve a given task. After processing node P_1 is reached, the transition path goes on with the two judgment nodes J_2 and J_4, whereby the judgment node function J_2 is now used twice in that transition path. J_2 does not necessarily have to be the same node, but at least both nodes use the same judgment function from the library. Nevertheless, it is possible that the GNP uses a node several times in one run. The multiple uses of nodes and a fixed node number in GNP avoid the bloat problem [Mabu, Hirasawa, and Hu, 2007].

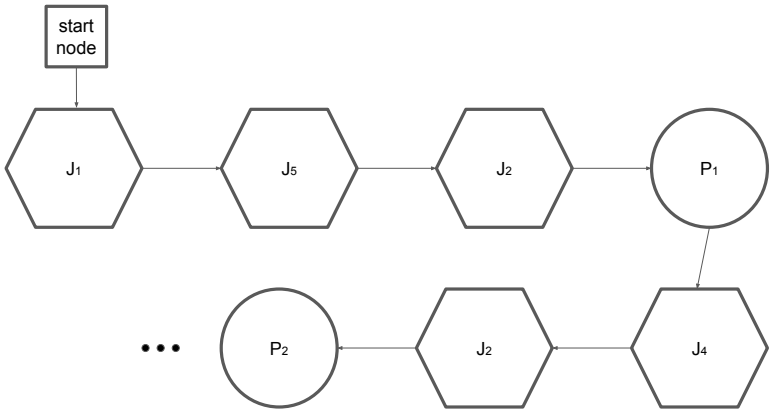

Figure 2.2: Exemplary node transition path.
Source: own illustration according to [Mabu, Hirasawa, and Hu, 2007]

That is an advantage compared to genetic programming, where tree bloat is a typical problem because there tends to produce larger and more complex individuals with every generation. Therefore genetic programming can result in programs that become unreasonably long and require a high level of computational effort unless explicit countermeasures are taken. The extra code that evolves during GP goes under several names, including introns, junk code, fluff, ineffective code, hitchhiker code, and invisible code. Three examples of junk code include the following:

$$\neg(\neg x) \tag{2.1}$$

$$x + 0 \tag{2.2}$$

$$\text{if } 2 > 1 \text{ then } x \text{ else } y \tag{2.3}$$

Consider the notation 2.3 where the if-part of the conditional statement is executed if $2 > 1$. That is always the case, and the else-part can never be executed; why that part is called an intron [1]. More importantly, mutation or a crossover operation that only affects an intron is fitness neutral because the intron is never executed, and the fitness only depends on the active program code. Consequently, introns can grow arbitrarily in size and complexity since this complexity does not carry any fitness penalty [Kruse et al., 2015]. In fact, GNP could also cause such introns shown in equation 2.3. However, because of the fixed number of nodes, GNP are not grown arbitrarily and are fixed in length.

There are several ways to protect against bloat like maximum tree depth, adjusting implementations of crossover and mutation, or using a penalty for extended programs [Simon, 2013]. Because of the fixed number of nodes in GNP, these methods are not discussed in detail here, but the bloat problem will be an issue in variable-sized GNP in Chapter 4.2.3.

[1] In biology, introns are parts of the DNA sequence that do not carry any information in the sense that they do not code for any phenotypical characteristic [Kruse et al., 2015].

2.1.2 Genotype Expression of GNP

The genotype expression of a GNP is illustrated in Figure 2.3. The whole gene structure of GNP is determined by multiple nodes where each node consists of a combination of node genes and connection genes. K_i denotes the node type where $K_i = 0$ means start node, $K_i = 1$ means judgment node and $K_i = 2$ means processing node. ID_i represents the identification number of the node function the programmer prepared in the library. For example, if $K_i = 1$ and $ID_i = 2$, then the node is J_2. If two judgment nodes are assigned to the same judgment function, then they are both J_2 but differ in their node number.

d_i is the time delay spent on judgment or processing described in detail in section 2.1.4. The next genes are continuous connection genes $C_i^1 ... C_i^k$, where k is the number of nodes in the network without the start node. That genes show the node number connected from node i to the next node, and d_i^k specifies the time delay spent on that transition. As mentioned in section 2.1.1, processing nodes always have one outgoing connection. The number of outgoing connections of a judgment node depends on the possible results from the judgment function and the evolutionary process and can be at most $k - 1$. The return of the judgment result determines the upper suffix of the connection genes. For example, if the judgment result is $k = 3$, GNP refers to C_i^3 and d_i^3. As indicated above, the judgment and processing functions are defined in the function library, whereby the exact definition depends on the problem definition. The programmer can define functions that determine the further transition through the network. For example, in a data mining task, the judgment node functions usually consist of the features and the processing nodes function of the target variable. The reason for a separate library is that functions can also be defined to operate in the judging or processing environment. E.g., in robotics, the preprocessing of image data. Note that the gene structure starts at $k = 1$ because the transition path never returns to the start node (node 0). Furthermore, it is not possible to connect a node with itself. The gene structure in Figure 2.3 represents the whole network (phenotype) from Figure 2.1 [Mabu, Hirasawa, and Hu, 2007].

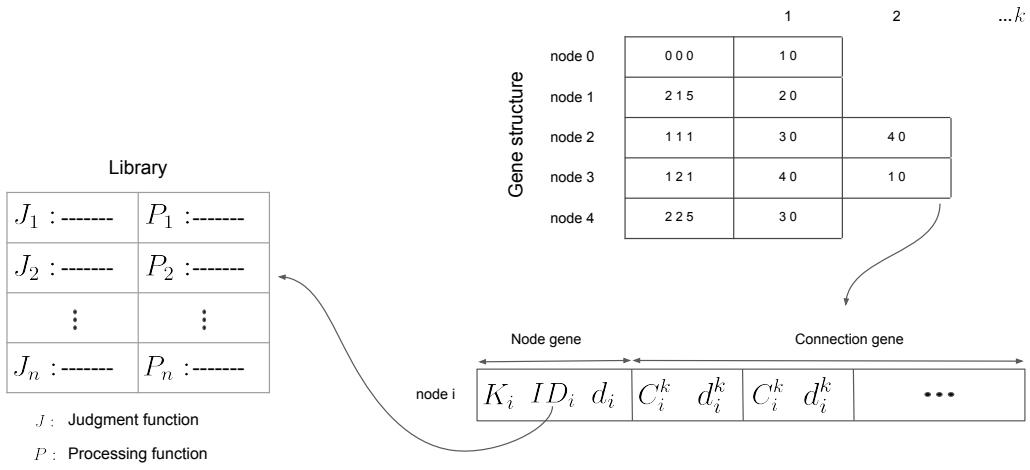

Figure 2.3: GNP gene structure.
Source: own illustration according to [Mabu, Hirasawa, and Hu, 2007]

At this time, the basic components and the genotype were explained. To see how these components work together and to show some example phenotype expression, it is time to look at a run of a GNP concerning the Iris data. Remember, the Iris data comes with four features and 150 records. The Iris data has no intrinsic order, and it does not matter in which order the data is delivered to the GNP. A run of a GNP goes through all the records, and the current node makes a judgment or processing depending on the selected judgment function (features of the data) or the selected processing function (flower species). To make a conditional branch decision on each judgment, the Iris data of the corresponding feature is divided into equal-sized intervals. The number of intervals depends on this node's outgoing connections and can change over the evolutionary process because it is also possible that multiple connections point to one particular node. Therefore the width of the interval can change and adjust to the data. If a GNP runs through the whole Iris data, a whole transition path has a length of at least 150. That is the case if the transition path consists of only processing nodes and is probably not practical because it means that the GNP just memorized the data.

Figure 2.4 shows a whole transition path of an example individual with a transition path length of 300, so in that case, each processing corresponds to precisely one judgment. The x-axis shows the nodes number with the corresponding K of the individual, and the y-axis shows the ID of the nodes function in the gene structure (starting with 0). For example, node $(x, y) = (Num : 3,\ ID : 1)$ shows a judgment node $(K = 1)$ with the first function-ID. Counting all nodes shown in Figure 2.4 shows seven initialized nodes, and four of the seven nodes were used in that node transition path. A used node in the transition path is marked as a green dot, and the not used nodes are shown as a red dot. This is an interesting aspect of a GNP because not all nodes have to be used in the transition path, which means that the GNP can adapt the used features and outputs. For the transition path, just one judgment node $(K = 1)$ and three processing nodes were used $(K = 2)$. Also, the judgment

node $K : 1,\ ID : 1$ is connected with all the other processing nodes, whereby none of the processing nodes are connected with each other. The exploration of the node space during the transition path is an important and exciting aspect that is discussed later in Chapter 4.

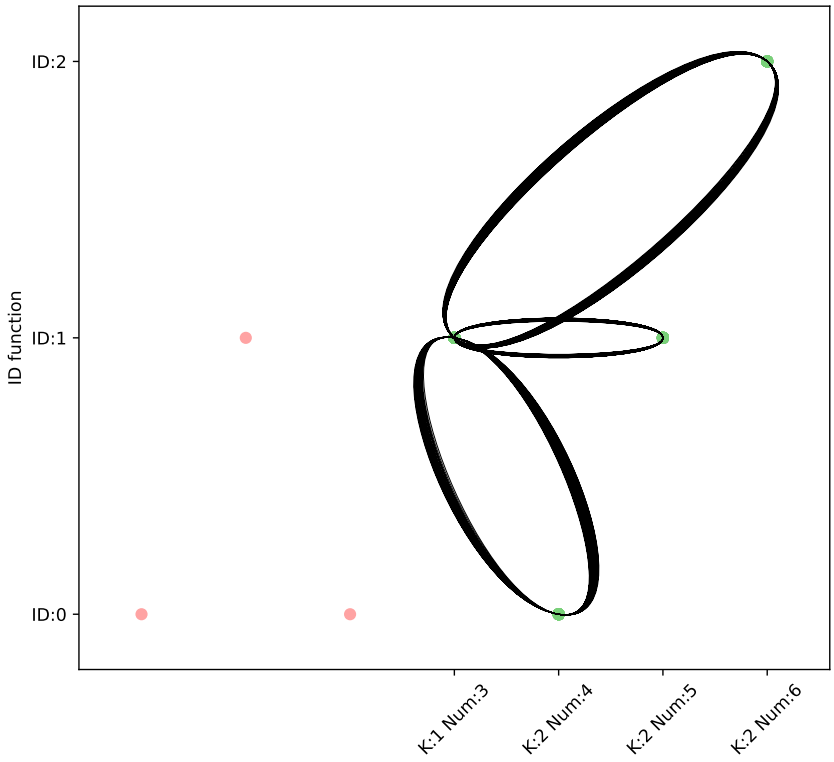

Figure 2.4: Whole transition path of a GNP on the Iris data.
Source: author's personal research results.

One of the advantages of a GNP is the interpretability of the solution candidate because it is possible to traverse and understand the whole transition path. On the other hand, neural networks, e.g., have their strengths in learning ability and forecasting but lack explanatory capability, which leads to so-called black-box models [Chen and Hirasawa, 2010].

To better understand the transition path above, Figure 2.5 shows all used nodes from the transition path more comprehensibly, whereby the whole transition path is summarized to all used nodes with their used connections.

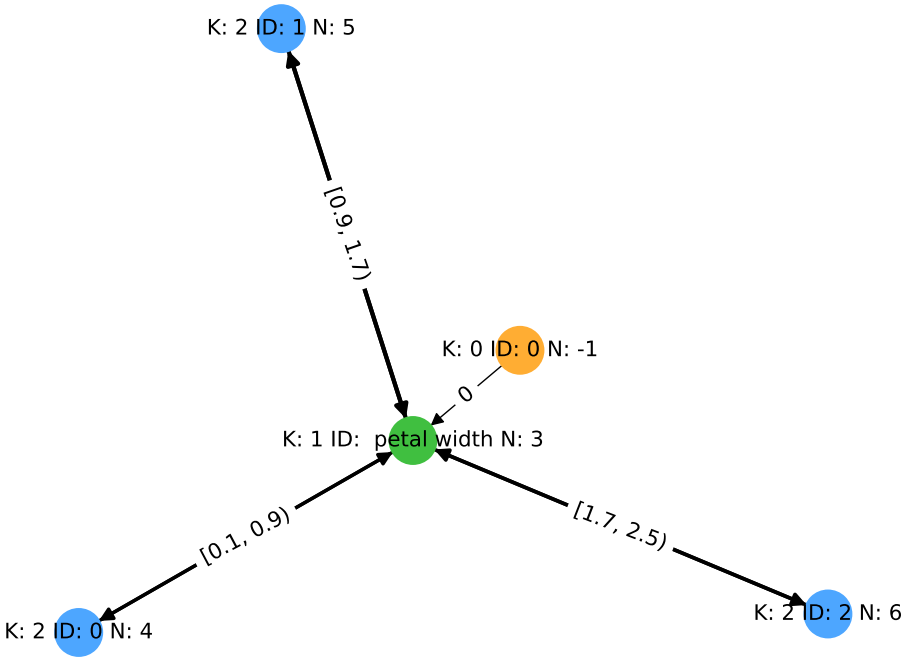

Figure 2.5: Used node of a GNP on the Iris data.
Source: author's personal research results.

The figure shows different colored nodes: the start node (orange), the judgment node (green), and the three processing nodes (blue). The labels on the nodes illustrate the kinds (K), the functions (ID), and the node number (N). The start node points to the node at which the transition path begins and has no further relevance. After the start node selects the judgment node, the network can make decisions by the judgment nodes and the corresponding intervals. Because the intervals are coded for each node connection, they are labeled on the edges of the graph network. Note that the intervals are printed as the lower and upper bound of the judgment whereby values $\geq$ the lower bound are included, and values are restricted $<$ the upper bound. As mentioned, the GNP can adapt to the data where specific features are selected during the transition path. That is exactly what happened in this transition path because just one judgment node (green node) is selected for the transition path, and therefore just one feature is considered. Concretely speaking, if the Iris data comes in, the judgment node $(N = 3)$ selects the judgment function with the ID that correspond to the feature "petal width". After that, the transition path goes on to the following processing node depending on the value of the data. E.g., if the value is 0.2, the next node in the transition path is the processing node with the number $N = 4$. That processing node classifies the flower to the species corresponding to the $ID = 0$. After that, the connection goes back to the judgment node $(N = 3)$ and further to the whole transition path given the Iris data value of the fourth feature (petal width). Note that the thickness of the edges indicates the number of times an

edge was used in the transition path. Not surprisingly, all edges of the nodes in the transition path have the same thickness. That is because the Iris data comes with equal data for each species (50 per flower species).

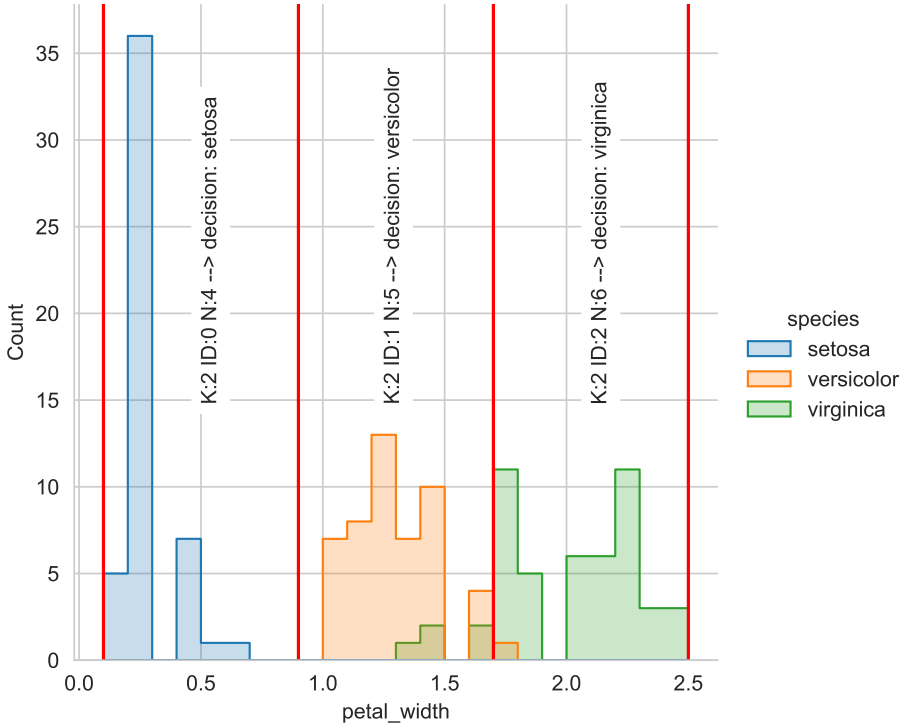

Figure 2.6: Classification of the Iris data by GNP with feature four (petal width).
Source: author's personal research results.

In fact, the particular transition path demonstrates a simple decision tree. The histogram in Figure 2.6 shows the Iris data separated by the intervals from Figure 2.5 to understand better how that solution candidate classifies the whole data and further shows that the GNP was able to learn to classify the Iris flower species (with some errors on overlapping class distributions). Of course, we see that the three red boundaries should not lead to overfitting, but typically one does not train with the whole data, and a test validation is recommended. Later in the experimental part, some kinds of cross-validations are specified.

2.1.3 Memory Function

As mentioned in Chapter 2.1.1, the node transition path begins from the start node. However, a GNP has no terminal nodes compared to a standard Genetic Program that terminates at leaves. Further, the current node is selected according to the last node's connections and judgment results. That means the node transition path of the past influences the selection of the current node. Therefore, the graph structure has an implicit memory function of the past decisions of the network. Although a judgment node is a conditional branch decision function, the GNP program is not only the aggregate of if-then rules because it includes past judgment and processing information. That means, different judgments/decisions can be made for the same attribute values, based on the history of the data, because different judgment nodes

are reached. For example, Figure 2.1 starts the transition path with the processing node P_1 and goes forward to the judgment node J_2. This implies that when the current node is J_2, we know the previous processing was P_1 [Mabu, Hirasawa, and Hu, 2007].

This is an extension compared to GP and, therefore, also of a decision tree that classifies given data. So the GNP can learn from the features of the data to, e.g., classify the Iris data as shown in Figure 2.5 where one judgment node decides the next processing node (classification) for the whole transition path. That is how a GNP can learn from the data horizontally (row by row). The extension now is that the GNP can find patterns in a temporal order (vertical) of the given data. For example, assuming the Iris data comes in a way into the GNP where each flower species repeatedly comes by row. That means when the GNP runs through the data of the target variable (species) it comes in the following order: $\{0, 1, 2, 0, 1, 2, 0, 1, 2...0, 1, 2\}$ where $0 = \textit{Iris Setosa}$, $1 = \textit{Iris Versicolor}$ and $2 = \textit{Iris Virginica}$. Suppose the GNP is initialized with only three processing nodes $\{P_1, P_2, P_3\}$ (the GNP learns from the target variable without any feature). In that case, the GNP can learn the temporal pattern $\{0, 1, 2, 0, 1, 2, 0, 1, 2...0, 1, 2\}$ when the three processing nodes represent a cycle. The resulting nodes of a learned GNP from the author's program are shown in Figure 2.7.

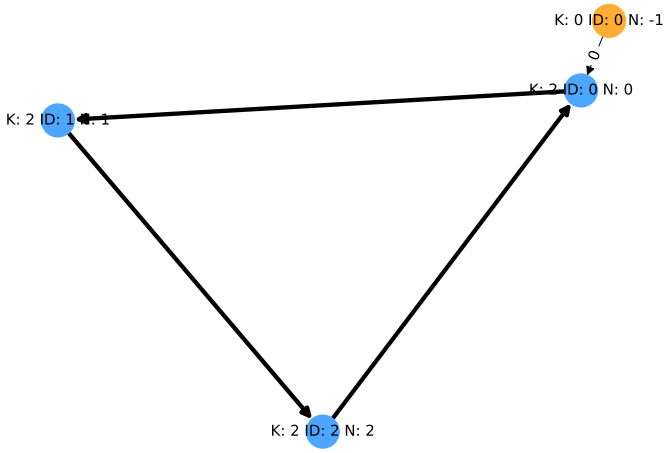

Figure 2.7: Transition path of the best individual after learning the pattern [0,1,2].
Source: author's personal research results.

Admittedly, learning the Iris data in that way does not make sense because the GNP classifies the data just from the target variable and would be bad in accuracy when the data comes in another order. This is especially important since the network consists only of processing nodes, and thus the starting point plays a crucial role. If the pattern starts differently in validation data, for example, {1,2,0...}, then the learned network makes an error in each prediction because it predicts {0,1,2...}. This can just happen when the time series shifts by one data point. Therefore, the temporal learning of a target variable should instead depend on the features (judgment nodes), like in the initial example from section 1.1.1. In Chapter 5, which deals

experimentally with time series, this topic is treated in more detail. However, that simple example clarifies that the GNP can learn from the data horizontally *and* temporal. It should be mentioned that the example with three processing nodes was just an example for the sake of simplicity. The GNP can also learn temporal concerning judgment nodes (features) and is not limited to the processing nodes. That was already shown in the introductory example in Chapter 1. That is why GNP is well suited for time-series data which will be considered in more detail in Chapter 3. An interesting aspect should be mentioned for learning temporal patterns: the successful learning of the GNP depends strongly on the number of processing nodes in the network or the structure of the features. Consider the three initialized processing nodes in the example above. The GNP finds the pattern very fast because the initialization of a correct node transition path $\{0 \rightarrow 1 \rightarrow 2 \rightarrow 0...\}$ has a probability of 4.17%:

$$\frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \sim 0.0417 = 4.17\% \tag{2.4}$$

Since the nodes cannot be connected to themselves, the probability of a correct connection is $\frac{1}{2}$ for each connection in the network. For a population size of 100, the probability of finding a network representing the correct pattern just after the initialization is, therefore

$$1 - 0.9583^{100} \sim 0.9859 = 98.59\%. \tag{2.5}$$

A simulation study was made to analyze more complex patterns and different sizes of initial processing nodes whereby 1000 different initializations of a GNP population with 100 individuals were generated.

Table 2.1: Simulation Study of 1000 different GNP initializations. The first table shows the patterns [0,1,2] and [0,0,1,1,2,2]. The second table shows the pattern [0,0,0,0,0,1,1,1,1,1,2,2,2,2,2].
Source: author's personal research results.

Processing Nodes	3	6	9	18	36	150
Proportion in %	98.2	71.6 0.5	54.1 1.8	29.0 0.6	13.9 0.5	3.7 0

Processing Nodes	15	30	60	150
Proportion in %	0	0	0	0

Table 2.1 shows the resulting proportions for finding the different patterns after 1000 initializations. The first row of probabilities shows the mentioned pattern $[0, 1, 2]$ and with a proportion of 98.2% nearly the same result for three initial processing nodes after the above calculation of the probability. The difference comes from the randomness of the initialization. If more networks were initialized, the probability should converge to 98.59%. Further, that row shows a dramatic decrease when more than three nodes are initialized. For example, an initialization of six nodes results in a proportion of 71.6% of finding the pattern $[0, 1, 2]$, and a simulation with 150 processing nodes can only find a pattern in 3.7% of the 1000 initializations. That is an indicator to initialize a suitable number of nodes for a given data mining task or an assumed pattern. For example, finding a pattern by initializing a number of processing nodes using the pattern's length. However, real-world problems are hard to solve in such a way because there is often not much prior knowledge concerning the

features and target variable. Additionally, looking at the second line of probabilities indicates another problem: even a simple pattern can be better solved by initializing more nodes than the length of the given pattern. Concretely speaking, the second line shows the proportions with respect to the pattern $[0, 0, 1, 1, 2, 2]$. Three processing nodes cannot find the given pattern. Therefore the simulation starts with six processing nodes. With six initial nodes, just 0.5% of the 1000 simulations can find the given pattern. That shows that a longer pattern is much more difficult to learn (71.6% vs. 0.5%) and is also plausible by imagining that more network connections have to be in the correct order.

Moreover, an initialization of nine nodes makes it more likely to find the given pattern (1.8%), which means that more initial nodes are sometimes better for finding a given pattern. Thus, it becomes clear that a proper number of initialized nodes is not trivial and depends on the given problem. Therefore, variable GNPs are suitable for such problems.

Another example with a pattern of length 15 ([0,0,0,0,0,1,1,1,1,1,2,2,2,2,2]) is shown in the second part of the table 2.1. After the 1000 initializations with a population size of 100, not even one individuum of different processing node numbers was able to find the correct pattern. Nevertheless, does that mean a GNP can not find the pattern? The clear answer is no! Figure 2.9 shows a learned transition path of nodes where the pattern of the second part was learned by the GNP during the evolutionary process. That is a perfect example of how an evolutionary algorithm can learn an improbable composition of a candidate solution and also highlights the advantage of a guided random search over a blind random search in a practical simulation. The initialization of the population is just the same as the second part of table 2.1 with 60 initial processing nodes where after 1000 initialization, no individual could represent the given pattern!

It can be seen in Figure 2.8 that the fitness of the best individual (max) and also the mean fitness tend to increase during the generations. Each individual's fitness is the accuracy of predicting the target variable (y-aches of the figure). After 98 generations of the evolutionary process, an individual was found that can represent the pattern [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2]. The used evolutionary operations are explained in the later sections.

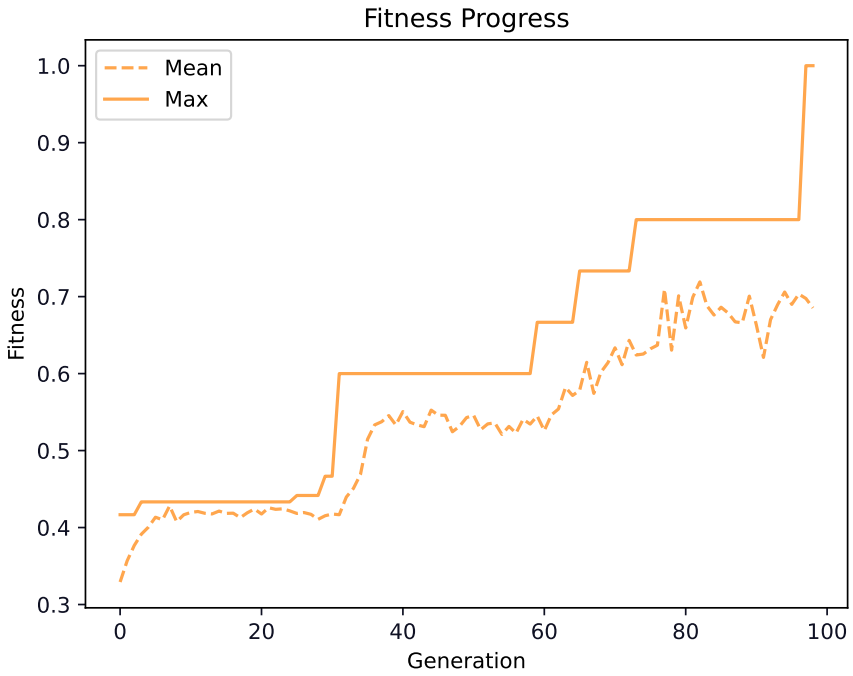

Figure 2.8: Progress of the GNP learning the pattern [0,0,0,0,0,1,1,1,1,1,2,2,2,2,2].
Source: author's personal research results.

The fundamental concept for reaching such an improbable individual is the mentioned *guided random search* in the space of the solution candidates, where algorithms carry out a search that contains certain random elements to explore the search space. However, they are also *guided* by some measure of the solution quality, which controls the parts of solution candidates are focused on or at least kept for further exploration and which are discarded because they are not promising. The used mechanism to reach such behavior in GNP is discussed later in section 2.3, but it should be mentioned that this refers to the core principle of the formulated theory of biological evolution by Darwin: Beneficial traits resulting from random variation are favored by natural selection [Kruse et al., 2015]. So the results in Figure 2.8 are possible by that guided random search where each slight variation is immediately put to a test w.r.t. an environment, and only the beneficial variations are kept and extended and *not* purely random initialization.

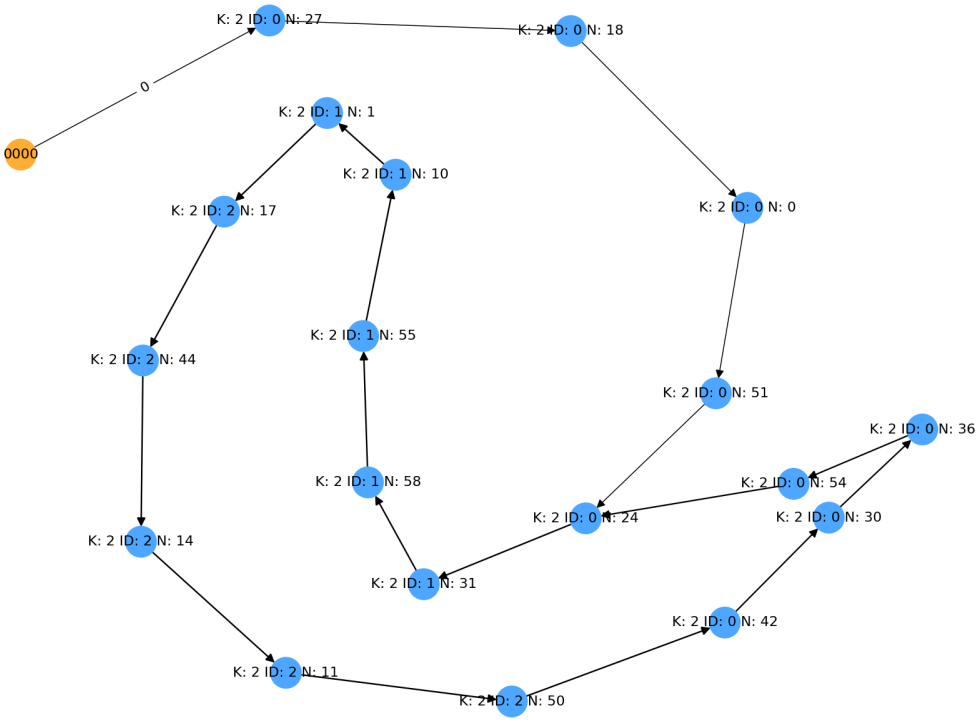

Figure 2.9: Transition path of the best individual after learning the pattern [0,0,0,0,0,1,1,1,1,1,2,2,2,2,2]
Source: author's personal research results.

Further, the shown transition path of the best individual in Figure 2.9 is very interesting concerning the composition of the node connection. The loop to represent the pattern [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2] starts with the node number 24 and is not the first node in the transition path. The transition path always starts with the start node (orange-colored node). Between the start node and node number 24, four processing nodes with $ID = 0$ represent the first part of the pattern. After node 24, the transition path goes on with the nodes representing the $ID's = 1$, $ID's = 2$ and then $ID's = 0$ and so on. The whole transition path is represented by processing nodes (blue-colored nodes) because no judgment nodes were initialized. In total, 19 processing nodes represent the whole transition path, which means that the number of different processing nodes is greater than the length of the pattern, which is 15. As mentioned above, this is why the second row of probabilities in the first part of the table 2.1 shows a higher probability for nine processing nodes than for six processing nodes, although the length of the pattern, in that case, is six ([0,0,1,1,2,2]). That leads to the difficulty/ impossibility of finding a suitable size of initial nodes and further leads to the assumption that using variable-sized networks (individuals) can reach better solutions. That is one of the main parts of the thesis research and is discussed in detail in Chapter 4.

2.1.4 Time Delays

GNP can also have time delays included as a hyperparameter. There are two locations where time delays are used. The first is the time delay GNP spends on judgment or processing, and the second is the time delay spend on the node transition,

as shown in Figure 2.1. Time delays are used for real-world problems where some time is spent on judgment, processing, and transition to the next node. For example, time delays are used when a walking robot needs some time to judge an obstacle (judgment node), to put the judgment into action (delay on transition), and avoid the obstacle (processing node) [Mabu et al., 2002]. Alternatively, in the economic context, time delays can be used to determine the maximum number of technical or fundamental indices to be considered in the transition path [Chen and Hirasawa, 2010].

Another essential role of time delays in GNP is preventing the program from falling into deadlocks. This can happen when the program never reaches a processing node because of a loop of judgment nodes. During this time, the judgment nodes examine the same row of data, and the loop is never interrupted and runs infinitely. Such programs are stopped by a maximum time delay and typically removed from the population in the evolutionary process [Mabu, Hirasawa, and Hu, 2007].

2.2 Initialization of a GNP Population

Figure 2.10 shows a GNP's whole flowchart, which begins with the initialization of the population.

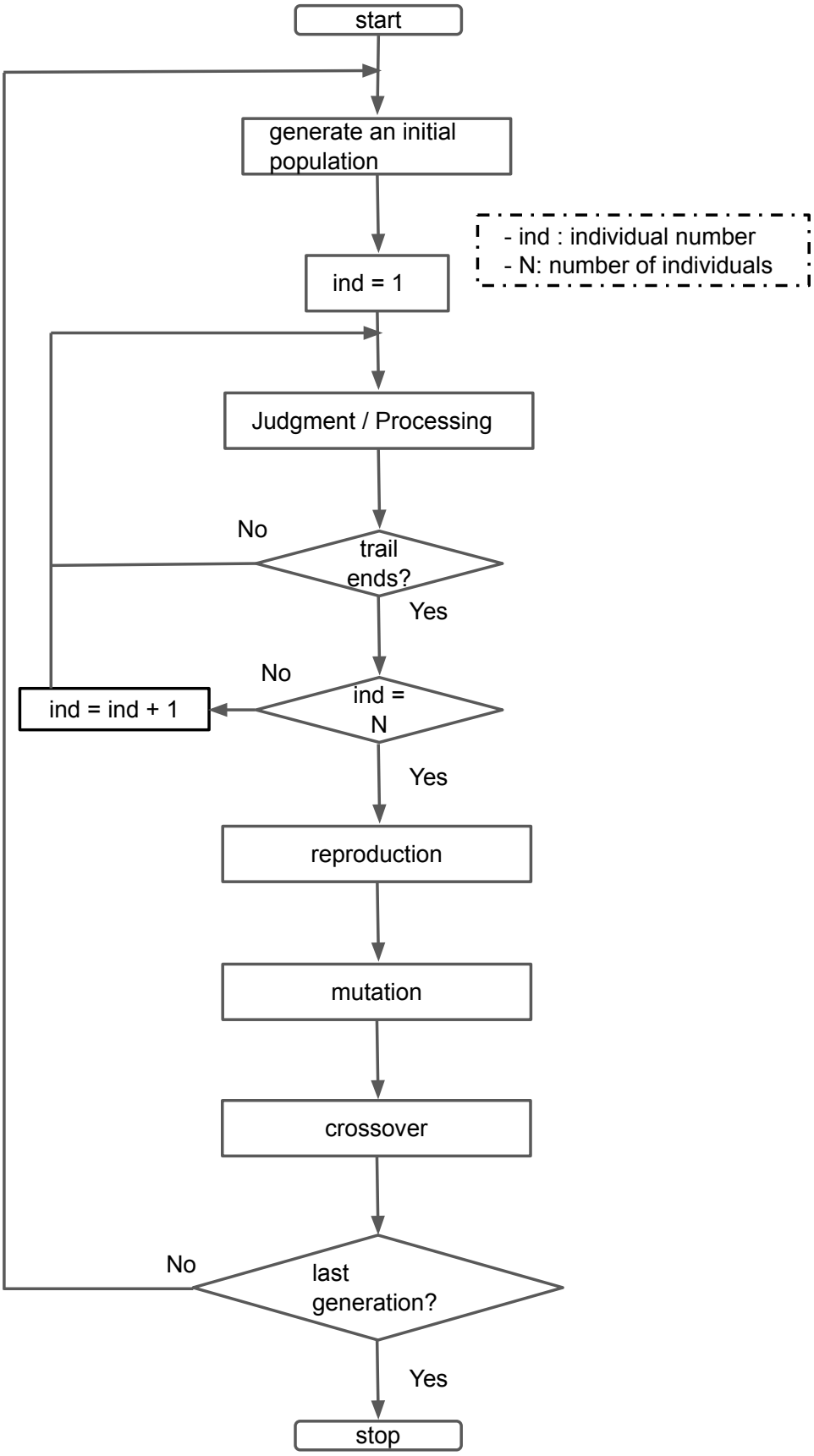

Figure 2.10: Flowchart of GNP system.
Source: own illustration according to [Chen et al., 2009b]

The run of the GNP needs the following hyperparameters for the initialization:

- population size
- number of judgment nodes
- number of processing nodes
- time delays and maximal time units

First, an initial population is produced according to population size, the number of judgment nodes, and the number of processing nodes. For example, a population of 1.000 individuals could be produced where each individual consists of five judgment nodes and five processing nodes where each kind of a node is different. Therefore all individuals in a population have the same number of nodes where the nodes with the same ID have the same function defined in the library, as shown in Figure 2.1. To prevent initializing each node with its corresponding kind via a separate hyperparameter, the inherent functions of the nodes are tried to be equally distributed over the processing and judgment nodes in the author's program. Suppose the modulo operation (%) of the number of nodes of a particular kind divided by the number of functions is not zero. In that case, the returned remainder is partitioned over the nodes by the sequence of functions. For example, if there are five judgment functions and five judgment nodes, then one function of each kind is assigned to one of these nodes. If there are five processing nodes but two processing functions, the following nodes are initialized: $P_1, P_2, P_1, P_2,$ and P_1. Therefore all programs in a population have the same number of nodes, and the nodes with the same node number have the same function [Mabu, Hirasawa, and Hu, 2007]. Remember that different judgments/decisions can be made for the same attribute values based on the history of the data because different judgment nodes are reached (see section 2.1.3). Therefore it is reasonable to initialize multiple nodes with the same node function.

It should be clear that for a problem like the classification of the Iris flowers, at least three processing nodes have to be initialized. If there are less than three processing nodes in the gene structure of the individuals, not all classes can be classified from the network. For example, two processing nodes can only classify two flowers, and therefore the accuracy can be at most $\frac{2}{3}$. The same applies to the judgment nodes. If fewer judgment nodes than features are initialized, not all features can be used by the network.

The connection genes $C_i^1, C_i^2...$ to the other nodes are selected randomly from $1, ..., k-1$ (except i to avoid a loop), with three aspects concerning the number of connections:

1. A processing node has just one random connection to another node.
2. If the node is a judgment node and the ID selects a judgment function of a categorical kind, then one connection will be initialized for each category.
3. If the node is a judgment node and the ID selects a judgment function of a numerical kind, then a random number of connections to other nodes will be initialized. After that, the numerical feature is divided into intervals of equal length, whereby each connection corresponds to one of the intervals.

Note that the connections are just the initial connections. The number of outgoing connections from one judgment node to other nodes can change through the

genetic operations described later.

To initialize and represent a graph structure with vertices and edges in computer memory, two standard ways are common:

1. Representing the graph (network) using an adjacent matrix or
2. using an adjacent list.

An adjacent matrix $A = (a_{ij})$ that represents a directed graph (e.g., a GNP Network) with n ordered vertices where each vertex is called $v_1, v_2, ..., v_m$ is a $n \times n$ matrix defined as:

$$a_{ij} = \{1 \text{ if } v_i \text{ is adjacent to } v_j, \text{ i.e.; if there is an edge } (v_i, v_j). \ 0 \text{ otherwise}\}$$

Such a matrix, in which entries can only be ones or zeros, is called a bit matrix or boolean matrix [Singh and Sharma, 2012].

The other common way to represent a graph is a so-called adjacency list. In that case, all edges of the graph are represented in a list. An adjacency list typically has n linked lists, with each corresponding linked list containing the elements adjacent to a particular vertex [Gaur, Shastri, and Biswas, 2008]. The pros and cons of representing a graph as an adjacency matrix or as an adjacency list are essential to decide which type should be chosen for GNP.

First, an adjacency matrix is very fast in adding and removing connections. The time complexity is $O(1)$ because just the specific indices of j and i have to be selected. The same time is required to check if there is an edge between two vertices. On the other hand, by adding a vertex to the new matrix, the storage must increase to $(|V| + 1)^2$. The whole matrix must be copied to obtain a new matrix; therefore, the complexity is $O(|V|^2)$—the same procedure for removing a vertex. Further, adjacency matrices consume an extensive amount of memory. Because each vertex's connections in the graph are defined as 1 and otherwise as 0, the space complexity is $O(|V|^2)$.

Adjacency lists can store graphs more compactly because only the existing connections are stored. Thus, the overall space complexity is reduced to $O(|V| + |E|)$. This is a particular advantage for sparse graphs with few edges ($|E| \ll |V|^2$). On the other hand, checking if there is an edge between two vertices can be done in the worst case in $O(|V|)$ because the algorithm has to traverse the whole adjacency list. Adding and removing an edge is also more expensive for adjacency lists. This is because to remove an edge, traversing all the edges is required in the worst case. Thus, the time complexity for removing an edge is O(|E|). Adding a vertex is much faster for adjacency lists than matrices because the insertion can be done directly in $O(1)$ time. Also, removing a vertex is faster, but in the worst case, it requires $O(|V|)$ time to search the vertex and $O(|E|)$ to traverse the edges. The search for the vertex is just needed for a list of adjacency lists. Using an array of adjacency lists direct access to the node could be obtained. But, this array needs to be copied if a vertex is added or removed, making $O(1)$ impossible in this case. So in total, $O(|V| + |E|)$ is needed to remove a vertex. [Singh and Sharma, 2012].

In conclusion, an adjacency list will be the better choice for the GNP. The reasons are:

1. The space complexity for an adjacency list is much lower, particularly for sparse graphs. The connection density of the GNP path is not predictable and can change during the evolutionary process. However, because of the initialization, it should not be fully connected unless in sporadic cases. Therefore the memory costs will be lower using an adjacency list.

2. The disadvantage of the lower time complexity in adding and removing edges using an adjacency list is not essential for GNP. As described later, the mutation, e.g., changes the edges randomly but does not change the number of edges.

3. The time advantages of adjacency lists when adding and removing vertices is important for the novel mutation of GNP described in Chapter 4 where the number of vertices is not fixed and can change over the evolutionary process.

4. Each run of a GNP transition path requires looping over the edges of the current vertex. For an adjacency matrix, the algorithm has to scan over the corresponding row of the matrix in the worst case $|V|$ times to find the edges that are present, whereas an adjacency list simply loops over the edges ($|E|$).

The algorithm in the author's program was implemented as an object-oriented incidence structure that works similarly to the linked list. Each node (vertex) is an object owning a list with the nodes to which the edges are directed. This has the advantage of storing additional data in the vertex object, which is, e.g., an excellent way to handle the different node types of the GNP. An independent list of connections for each node is further a simple way to represent parallel edges of the network. This is necessary because the connections can point to the same node during the evolutionary process and can lead to the changing interval width mentioned in section 2.1.2 and described in more detail in the section 2.5.1.

A simple way to represent a graph is a decision tree, as shown in the introduction. A tree structure consists of nodes and edges just like a network and is, in fact, also a graph. However, nodes in a tree are not connected to themselves, and there is always just one edge between the nodes as a connection. These constraints prevent the occurrence of circuits that are not allowed in a tree. If, in contrast, multiple edges are allowed between two nodes in a graph, these are called *parallel edges*. A graph without self-loops nor parallel edges is named a *simple graph* [Deo, 2017]. In the application of GNP, more complex networks are needed, and parallel edges are allowed. Concretely speaking, the represented graph of a GNP is a directed multigraph with parallel edges. A directed *multigraph* with parallel edges can be defined as:

Definition 1 *A **multigraph** M consists of a finite nonempty set V of vertices and a set E of edges, where every two vertices of M are joined by a finite number of edges (possibly zero). If two or more edges join the same pair of (distinct) vertices, then these edges are called parallel edges [Chartrand and Zhang, 2013].*

The object-oriented incidence structure can handle such a graph and, as mentioned, can also store different data, e.g., errors and corresponding groups of the input data, which is important for later topics.

2.3 Selection Method

Before the GNP can combine the individuals to create children, a method has to select which individuals to use as parents. The principle of selection is that better individuals have better chances to reproduce. This principle is the biological counterpart of differential reproduction, where individuals differ in the expected reproduction depending on their fitness. There are many selection algorithms, but GNP's most commonly used method is named tournament selection. All selection methods are generally biased toward fit individuals in the population. So no matter which selection method is used, a more fit individual will have a greater chance of being selected than a less fit individual. How strongly individuals with higher fitness are preferred for having offspring is called selection pressure.

Suppose a selection method is too strongly biased toward selecting fit individuals. In that case, the population may converge too quickly to a uniform solution while not exploring the search space widely enough. On the other hand, if a selection method is not biased strongly enough toward fit individuals, then the Evolutionary Algorithm may not be able to properly exploit the information that is present in the fittest individuals. In the worst case, with no bias toward the fit individuals, the EA is similar to a random search. The strength of that bias can be controlled by the mentioned selection pressure, which is defined as

$$\phi = \frac{P_r(\text{selection of most fit individual})}{P_r(\text{selection of average individual})} \tag{2.6}$$

where $P_r(\text{selection of } x)$ is the probability that individual x is selected for recombination. Therefore selection pressure quantifies the relative probability that a highly-fit individual will participate in recombination [Simon, 2013].

Usually, the best strategy is time-depending selection pressure. In early generations, the selection pressure is kept low to explore the search space well and hopefully obtain subpopulations in all promising regions of the search space. In later generations, the selection pressure should be increased to find the best modifications for the promising regions and, finally, the best solution candidate. The selection pressure can be controlled by either adapting the fitness function or by choosing a selection method and/or adapting its parameters.

In tournament selection, a certain number of k individuals are drawn randomly from the current population, where $k \geq 2$ is the user-defined tournament size. These individuals carry out a tournament in which the individual wins with the highest fitness. The winning individual receives a descendant in the next generation. After the tournament, all participants are returned to the current population (including the winner). Since each tournament selects one individual, N tournaments have to be carried out to fill the next generation, where N is the size of the population. The selection pressure of the mentioned "tournament selection" can be controlled by the parameter k [Kruse et al., 2015].

If the fittest individual of the population is picked to play a tournament, it will be selected for recombination with a probability of 100%. If the average individual x is picked for a tournament, then it must be more fit than $k-1$ other individuals in the tournament to be selected for recombination. Assuming that $k \ll N$ there is an approximate probability of 50% that x is more fit than each individual with which is compared. Therefore there is a $(1/2)^{k-1}$ probability that x will be selected for recombination. The above equation 2.6 then becomes

$$\phi = 2^{k-1}. \tag{2.7}$$

Equation 2.7 shows that as k increases, selection pressure also increases in tournament selection. So that is how the selection pressure can be controlled by the parameter k [Simon, 2013]. It should be clear that individuals with high fitness are more likely to win the tournaments in which they participate. However, individuals with low fitness may still produce offspring if they participate in a tournament in which all other individuals have lower fitness. Only $k - 1$ worst individuals do not have any chance of reproducing. That is an advantage of tournament selection, compared to methods like roulette-wheel selection, because the fitness does not influence how likely an individual will participate in a tournament, but only how likely it is that they will win a tournament.

Therefore tournament selection is an ideal method to tackle the dominance problem, which is a problem of fitness proportionate selection, e.g., roulette-wheel selection. The dominance problem can occur if an individual with very high fitness may dominate the selection. Consequently, the fittest individual receives many descendants, and the population becomes very dense in one or only a few regions of the search space (also called crowding). Crowding usually results in very fast convergence to a (local) optimum in that crowded region. This may be desirable in later generations to locate the best solution, but in earlier generations, it should be avoided in favor of exploring the search space. Because the fitness values do not directly influence the selection probability in tournament selection, the dominance problem can be handled by choosing a smaller tournament size k. For example, even for the best individual, the expected number of offspring is just the number of tournaments in which this individual participates.

The dominance problem illustrates the disadvantage of significant fitness differences between individuals, but small differences are also undesirable, called vanishing selection pressure. As mentioned above, some bias toward fit individuals is needed to push solution candidates toward better fitness values. That can also be handled by specifying a (greater) tournament size k. However, since an evolutionary algorithm tends to increase the average individual fitness from one generation to the next, it may even create the problem of vanishing selection pressure itself.

Another benefit of tournament selection over other types of selection is that it reduces the overall computational time associated with selection because operations are parallelizable. Selection techniques, like roulette-wheel selection, require a central agent that computes the relative fitness values, which can be computationally expensive for large populations. In contrast, tournament selection picks only k random individuals and compares them independently, whereby each tournament can be computed on a different agent [Kruse et al., 2015].

2.4 Elitism

Elitism is a way to ensure that the best individuals in an evolutionary algorithm (EA) are retained in the population from one generation to the next. There are two mean reasons to apply elitism in an EA. First, since the process of many selection methods is random, there is no guarantee that good and even best individuals will enter the new generation. Consider the tournament selection of $k = 2$ and a population size of $N = 100$. Then, to fill the next generation with enough individuals, 100 tournaments must be played. The probability of the best individual *not* being drawn for the tournament is $98/100 = 0.98$ and, further, not being selected for the next generation $0.98^{100} \approx 0.1326 = 13.26\%$. Note that this is the probability for drawing with replacement. However, that does not make much of a difference because the typical

population sizes in an evolutionary algorithm are in the thousands, and therefore $k \ll N$. Nevertheless, there is a chance of about $1/8$ that the best individual will not be selected for recombination [2]. This may be an unacceptably high probability of losing the information of the best individual in the next generation.

The second reason for applying elitism is to prevent the best individuals from being destroyed by one of the genetic operations. As described later, most of these operations are harmful, and there is also no guarantee that the best individuals at the $(i+1)$-st generation will be an improvement over that individuals. Therefore, the E best individuals, where E is a user-defined number of elite individuals, are transferred without modification to the next generations, ensuring that the best solution(s) discovered up to now (the elite) never get(s) lost or destroyed. Elitism can be implemented in at least a couple of different ways [Simon, 2013]:

1. Producing only $(N - E)$ children each generation. In that case, $(N - E)$ individuals would be used for selection and the genetic operations. Afterward, the best E individuals would merge the children to obtain the next generation of N individuals.

2. Producing N children and replacing the worst children with the best E individuals.

3. There are other elitism options also. For example, producing N children and using an inverse-roulette-wheel selection algorithm to select E of them, where the worst children have the most significant probability of selection—then replacing those children (worst fitness) with the best E individuals of the previous generation.

Not that the elite individuals (copies) still enter the standard selection and modification process, hoping that genetic operators may improve them. EAs employing elitism usually show better convergence characteristics since local optima are approached more consistently. However, elitism causes a particular danger of converging prematurely and getting stuck in local optima because no degradations are possible [Kruse et al., 2015].

2.5 Genetic Operators

Genetic operators are applied to a part of individuals in the population after being chosen by a specific selection method. These operators create modifications and recombinations where most of these modifications can be expected to be harmful. However, the hope of using genetic operators is that a few of the modifications result in better individuals.

The main difference between mutation and crossover is the operator's number of parents to treat. Mutation works only on one parent, whereas crossover works on two or more parents. An important characteristic for those operators is whether the search space is closed with respect to the application [Kruse et al., 2015].

2.5.1 Mutation

The mutation is executed on one individual and creates a new one, so the procedure in GNPs is as follows:

[2] Note that the probability for $k = 2$ but a higher population size of, for instance, 10.000 also holds at about $1/8$.

1. Get an individual from the selection.
2. Each node's connection (C_i) is selected with the probability of p_m. The selected C_i is changed to another node number randomly. The current node number has to be excluded from the possible connections to avoid self-loops.
3. The generated new individual is subject to crossover or part of the new generation.

Figure 2.11 shows an example of the mutation by changing the outgoing connection of node 3 [Mabu, Hirasawa, and Hu, 2007]. Only the connections are changeable to other nodes in that kind of mutation. Thus, for the mentioned example above, it cannot happen that the mutation causes an invalid permutation of stocks.

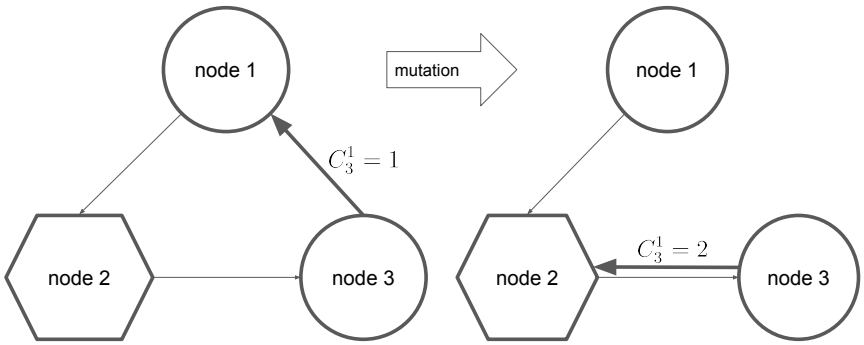

Figure 2.11: Mutation example of GNP.
Source: own illustration according to [Mabu, Hirasawa, and Hu, 2007]

This kind of mutation is equivalent to the so-called standard mutation, where another randomly chosen allele replaces a current allele of a randomly chosen gene. However, because the number of genes is selected randomly, the mutated genes may vary for each individual. Concretely speaking, the number of selected genes follows a binomial distribution with the parameter p_m [Kruse et al., 2015].

The new allele is chosen from other possible alleles (connections) using a uniform distribution (equal probability). Therefore, multiple alleles may point to the same node after the mutation. This can be seen in Figure 2.12 where an exemplary transition path is shown.

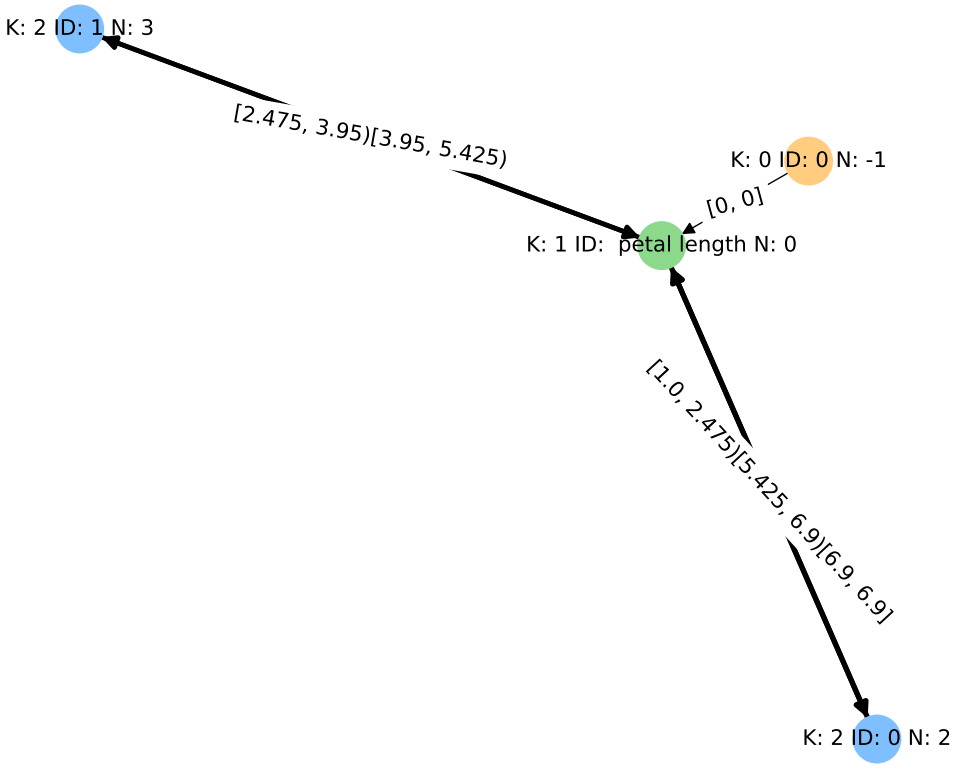

Figure 2.12: Exemplarily transition path with multiple node connections of a judgement node.
Source: author's personal research results.

The transition path is another example of a trained individual on the Iris data. The green judgment node with node number zero ($N = 0$) makes decisions concerning the feature petal length. The minimum measurement of petal length in the data is 1.0 cm, and the maximum is 6.9 cm. Thus the data are divided into equal-sized intervals according to the number of connections (see section 2.2) on page 26. In contrast to the initialization, where each connection points to a different node, all four connections of node zero are related to only two other nodes. This results from several mutations on that node because connections and their corresponding intervals can be changed to a node where a connection already exists. The intervals $[1.0, 2.475)$ and $[5.425, 6.9]$ are connected with node two and the intervals $[2.475, 3.95)$ and $[3.95, 5.425)$ are connected to node three. Note that the pseudo interval $[6.9, 6.9]$ is just added to the figure because the maximum feature value was treated by the transition path and is the only value included = the upper bound. As a consequence of the mutation, different intervals can be merged to represent wider intervals and partition the data coarse-grained. As an example, it should be clear that the two intervals $[2.475, 3.95)$, $[3.95, 5.425)$ can be merged into the interval $[2.475, 5.425)$. Furthermore, the intervals are not restricted to be separated by a given point. This leads to some intervals connecting to another node in between an interval, which can also be seen in the transition path: the interval $[1.0, 2.475)$ points to node 2, then the merged interval $[2.475, 5.425)$ points to node three, and the interval $[5.425, 6.9]$ points again to node two.

The multiple connections (alleles) to the same nodes and, as a consequence thereof, the different widths of the intervals can be an advantage because of a larger search space. However, on the other hand, this kind of mutation leads to the property that the chromosome is no longer a permutation of a set of numbers.

The following algorithm formalizes the mutation operation considering the notation of the genotype expression from section 2.1.2 on page 16:

Algorithm 1: GNP mutation

Function `mutation_connections`(*connection_genes*, i = *node number*, p_m)**:**
  $N \leftarrow length(connection\_gene)$;
  for $k \in \{1, ..., N\}$ **do**
    $u \leftarrow$ random number according to $U([0, 1])$;
    if $u \leq p_m$ **then**
      $n \leftarrow$ random element of $\{1, ..., N\} \setminus k$;
      $C_i^k = n$;
    end
  end

2.5.2 Crossover

Crossover is executed on two parents and generates two offsprings. The procedure of crossover in GNPs is as follows:

1. Get two individuals using a selection method.
2. Each node i from the gene structure is selected with probability p_c.
3. Two parents exchange the genes of the corresponding crossover nodes, i.e., the node with the same node number.
4. Generated individuals become part of the following generation

Figure 2.13 shows a crossover example where node three is selected for both parents simultaneously (ID_3). After that, the crossover operation exchanges the node ID_3 from parent 1 and the node ID_3^* from parent 2 and corresponding connections [Mabu, Hirasawa, and Hu, 2007].

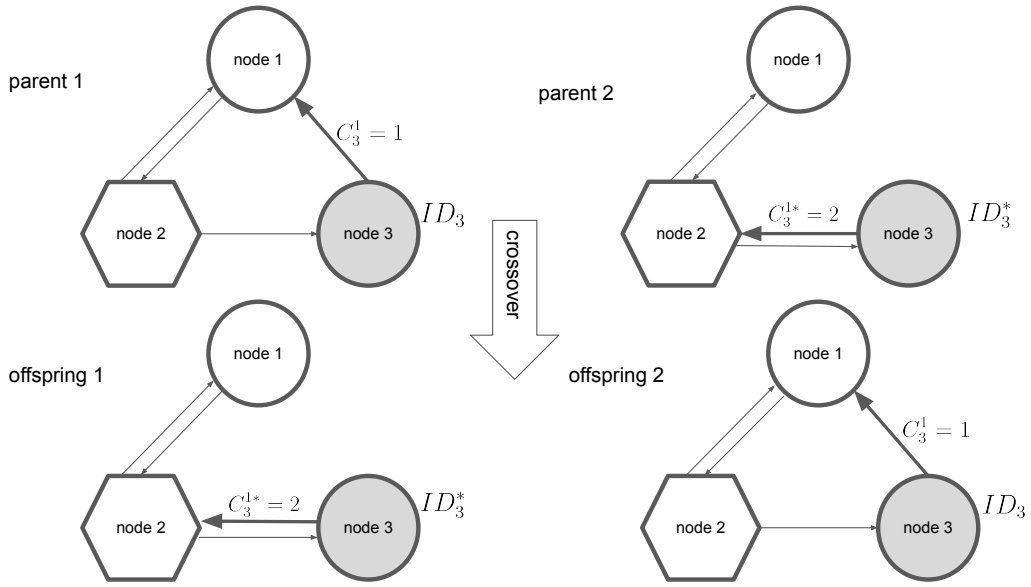

Figure 2.13: Crossover example of GNP system
Source: own illustration according to [Mabu, Hirasawa, and Hu, 2007]

That kind of crossover method is also called uniform crossover because each gene is selected independently, whether it is exchanged or not, based on the given probability p_c. Note that a different number of crossover operations are not equally likely for all numbers of nodes. This is because the number of selected nodes follows a binomial distribution like the mutation operator. Therefore numbers of

$$p_c \cdot \text{number of nodes} \tag{2.8}$$

are more likely than smaller or larger numbers. This characteristic is also known as distribution bias. Distribution bias is often undesirable because it causes partial solutions of different sizes to have different chances of progressing in the evolution. Avoiding distribution bias can be reached with an operator called shuffle crossover [Kruse et al., 2015]. The reason for still using the uniform crossover is to control the size of exchanged nodes with the parameter p_c (considering binomial distribution). However, in standard GNP is no normalization with respect to the size of the networks, whereby the number of changing nodes depends on the number of nodes in the networks. That means more extensive networks lead to more exchanged nodes and vice versa. The crossover holds one crucial characteristic: Because of the initialization (see section 2.2), each individual's node number i references the same node function n from the library. As a consequence, the crossover operation works on the same node function (replacing just the same node number) for each individual, and it can not happen that two nodes with different functions are exchanged. Therefore the number of nodes with different functions in the genotype of the individuals is always the same. The following algorithm formalizes the crossover operation considering the notation of the genotype expression from section 2.1.2 on page 16:

Algorithm 2: GNP crossover

```
Function crossover_standard_gnp(selection, p_c, node_numbers):
    N ← length(selection);
    while length(selection) ! = 0 do
        individual1 ← get individual from selection;
        selection ← selection \ individual1;
        individual2 ← get individual from selection;
        selection ← selection \ individual2;
        for k ∈ {1, ..., node_numbers} do
            u ← random number according to U([0, 1]);
            if r ≤ p_c then
                node1 ← individual1.gene_structure[k];
                node2 ← individual2.gene_structure[k];
                individual1.gene_structure[k] = node2;
                individual2.gene_structure[k] = node1;
            end
        end
    end
```

2.5.3 Epistasis

The critical point is that mutation and crossover can change the complete transition path of an individual if the first connection is selected to change. Therefore the mutation can introduce an entirely new individual, which leads to a crucial point called epistasis.

In evolutionary algorithms, epistasis represents the interaction between the genes of a chromosome. That is, how much the fitness of a solution candidate changes if a gene is modified. Since the mutation of the GNP can change the whole transition path, the resulting fitness can be totally different too. This is undesirable because an optimization problem is then often challenging to solve for an EA. One reason is that epistasis destroys the assumption that small changes to the chromosomes affect only minor changes in the candidate solutions. More generally, if small changes in the genotype can lead to significant changes in fitness, a fundamental assumption underlying evolutionary algorithms, namely gradual improvement, is no longer viable [Kruse et al., 2015].

Epistasis has to be accepted to some degree when applying a GNP because every connection change can lead to an entirely or nearly new transition path. One idea to tackle the problem could be to change just the connection of a node when the connection is used later in the transition path. This could be a topic for future work and is not covered here. More important is to use elitism (see section 2.4) to prevent the best individuals from getting destroyed or extensive fitness declining.

2.5.4 Simulation Study Genetic Operators

Nevertheless, mutation and crossover are necessary, as shown in the following simulation concerning the Iris data. For that, a simulation study was made to compare ten different mutation and crossover probability inputs. First, the different inputs of the mutation and crossover probabilities were simulated 100 times. Then, the mean

of maximal and mean fitness was calculated, which can be seen in table 2.2. The evolution of the GNP was simulated with the following parameter:

- Fitness: Accuracy
- Generations: 100
- Population size: 100
- Judgment Nodes: 4
- Processing Nodes: 3
- Mutation Probability: see table
- Crossover Probability: see table
- Elitism: 2
- Time Delay on Nodes: 1
- Time Delay on Edges: 0
- Maximal time Units: 10

Table 2.2: Simulation Study of 100 different GNP runs for each combination of parameters on the Iris data. The Best Fitness and Mean Fitness are the mean values of the 100 runs.
Source: author's personal research results.

Prob. Mutation	0	0.01	0.05	0.2	0	0	0	0.01	0.05	0.2
Prob. Crossover	0	0.01	0.05	0.2	0.01	0.05	0.2	0	0	0
Best Fitness	0.59	0.92	0.95	0.95	0.63	0.74	0.82	0.91	0.95	0.94
Mean Fitness	0.59	0.88	0.79	0.35	0.63	0.74	0.82	0.88	0.8	0.36

First, it can be seen from the first column an initialization of $p_m = 0$ and $p_c = 0$ which means neither mutation nor crossover was used. For these parameters, the worst results were achieved with the best fitness (accuracy) of 0.59 and mean fitness of also 0.59. Not surprisingly, the mean and the best fitness are the same in the end because without genetic operations, no individuals will change, and the selection process shifts the individuals to the best fitness. As a result, all individuals have the best fitness, and any genetic operation could not extend the initial gene pool.

The following three simulations were initialized by the same parameter for both genetic operations (0.01, 0.05, 0.2). As a result, a high fitness (0.92 respectively 0.95) could be reached in all three cases. Further, a small mean fitness of 0.35 could be seen. In comparison to the last column of the table, where the mean fitness is also that small, it seems to be that the mutation operation with a high probability parameter ($p_m = 0.2$) "destroys" many individuals. Remember that most genetic operations are harmful.

Worst fitness results were reached after applying no mutations and the three different probabilities of the crossover operation: 0.01, 0.05, 0.2. For all three simulations, the best and mean fitness are the same at the end, while a greater probability of the crossover operation yields better results. A fixed gene pool again causes this. After the initial connections of all nodes are set, just the exchanges of the nodes can

cause a different number of connections. Because the initial population of 100 individuals is relatively small, it seems that not all the necessary connections of the nodes are in the gene pool, and just an exchange of nodes cannot find better solutions than the given results.

Conversely, if no crossover is used but just the three different parameters of p_m, shown in the last three columns, very good results can be reached. There is reason to believe that the mutation operation is more important than the crossover operation in tackling the Iris data because the fitness values are also high without the crossover operation and the best and mean fitness is worse if no mutation is applied. However, that could not be a universal conclusion because maybe it is more important to exchange information between individuals via the crossover operation for a more complex problem or a problem of just a different shape.

In conclusion, mutation and crossover are necessary to change connections/ nodes and reach better fitness results. Because mutation changes the connection of each node, it could lead to a higher gene pool of all individuals. It should be mentioned that the gene pool is higher but also fixed. Consider a numeric feature (e.g., a feature from the Iris data). Then the feature would be divided by each node's connections in equal-sized intervals as described in section 2.2 but is limited in the number of connections. For example, a node in a network of five nodes can have at most four connections ($k - 1$) and, consequently, four intervals. Therefore, smaller intervals to fit the given data in greater detail are impossible.

The two novel mutation operations in this thesis tackle this limitation of the gene pool:

1. Applying mutation to change the boundaries of the interval.
2. Applying mutation to change the network size.

These approaches are the topics of Chapter 4, but before covering and applying these topics, the next chapter is first about GNP in the economic context.

Chapter 3

Financial Applications of GNP

Applying GNP for an economic task is reasonable because GNP properties are well suited to time-series data. Remember, time series data are common in applying an economic problem, have a fixed order, and the order of the history matters. The reusing of nodes, where GNP implies a memory to continue a transition path, and the learning ability of the data horizontally and temporally are reasons why GNP is often used for economic data. In addition, GNP often seeks results in a dynamic environment which is why stock data are well suited for them. Many papers regarding GNP are focused on that dynamic environment. To tackle a dynamic task, the so-called Tileworld Problem is often tried to solve. In this problem, an agent interacts with a grid of cells (the environment) and takes suitable actions to push some tiles into holes. The Tileworld Problem is the first problem of a dynamic environment challenged by GNP [Katagiri, Hirasama, and Hu, 2000]. Further, Hirasawa et al. (2001) shows that GNP, compared to GP, is more efficient in dynamic environments.

Due to the successful application of GNP in the dynamic environment, more papers and approaches were proposed regarding the stock market. For example, Chen, Mabu, and Hirasawa (2010) suggested time adapting GNP to distribute capital to a set of stocks. Yan Chen and Xuancheng Wang (2015) extend GNP by statistical models where GNP generates trading rules, and the statistical model distributes the weight of capital. Chen et al. (2009a) proposed a GNP extension with reinforcement learning to obtain a trading model. Some variation to the GNP was made where the graph was altered from a directed graph to an undirected graph whereby nodes are coded as particular stocks. Further, the edges represent the correlation between these stocks and work to determine the best relation between them so that the correlation is minimized [Chen, Mabu, and Hirasawa, 2011]. Also, recent studies combining GNP and MLP to forecast stock returns [Reza Ramezanian, Arsalan Peymanfar, and Seyed Babak Ebrahimi, 2019].

3.1 (Stock) Market Analysis

There are two techniques to analyzing stock data, fundamental and technical. The objective of both methods is to evaluate the future expectations of the stock price, but they are different in assumptions and approaches. The distinction between the two methods is shown in Table 3.1.

Table 3.1: Distinction between Fundamental Analysis and Technical Analysis of Stocks
Source: [Wurm, Ettmann, and Wolff, 2005]

Fundamental	Technical
Assumptions	
• Stock prices oscillate around the intrinsic value of the stock. • The intrinsic value is primarily determined by future earnings prospects.	• Stock prices evolve in trends because of the aligned behavior of the market participant. • Future stock prices can be predicted based on past stock prices.
Approaches	
• Determination of the intrinsic value of the stock, taking into account the macroeconomic, industry-specific, and company-specific factors. • Capitalization of future earning	• Analysis of exchange-related data such as price development, trading volume and technical indicators • Identifying stock market trends and trend reversals

An example of an important ratio of the fundamental analysis is the price-earning ratio. In contrast, the technical analysis mainly focuses on a graphical representation of price and volume data and their derived indicators [Wurm, Ettmann, and Wolff, 2005]. That could be, for example, the standard deviation of the given price data. Clearly, an algorithm can store and analyze all the graphical representations and can also use statistical methods to predict market prices.

Notice the premise of the technical analysis that the price, resulting from supply and demand, discounted everything. The technician believes that anything that can affect the price - fundamentally, politically, psychologically, or otherwise - is reflected in the current market price. Not to be confused with the efficient market hypothesis (EMH), which sounds very close to the technical premise that markets discount everything. The efficient market hypothesis claims there is no way to take advantage of information because markets quickly discount all information. The foundation of technical forecasting is that important market information is discounted in the market price long before it becomes known for everybody and market participants can benefit from skillful analysis of price data to gain an advantage in investing. [Murphy, 1999].

Further, the EMH has its foundation in the random walk theory, which origins the Brownian motion [Fama, 1970]. The Brownian motion is a term for the motion of a molecule in a uniformly warm medium. The mathematician Bachelier suggests that this process can also describe price variation. In conjunction with the EMH, several critical assumptions come together with this idea [Mandelbrot, Hudson, and Grunwald, 2005]:

- *Independence*: Each price change occurs independently from the last, and price changes last week or last year do not influence those today.
- Statistical *Stationarity*: The process of generating price changes stays the same over time.
- *Normal distribution*: Price changes follow the proportions of the bell curve.

A further discussion regarding market efficiency is not the subject of this thesis. However, all three assumptions have been disproved by mathematician Benoit B.

Mandelbrot [1] before the paper about the EMH was published [Mandelbrot, 1997]. Furthermore, the research by psychologist Daniel Kahneman, especially the prospect theory, shows that the assumption that people are rational and aim only to get rich does not hold in the real world [Kahneman, 2012].

As there is too much evidence against the EMH, the author rejects the EMH. Further, as a consequence of the EMH, no predictions of market prices could be successful, and nobody could be better than a simple buy-and-hold strategy. In that case, the work of a financial GNP application to find "good" stocks would be finished here - but only the section is finished with the following apocryphal story:

> *"Eugene Fama, father of the EMH, was strolling across the University of Chicago campus with a graduate student. Looking down, the student exclaimed, "Look, there is a $100 bill on the ground." Without a glance down or a break in stride, Fama replied, "No, there isn't. If there were, someone would have picked it up already."* [Thorp, 2017]

3.2 Fitness Functions

As mentioned in the previous chapter, all the financial data to analyze stocks can be passed as features to an algorithm such as GNP, similar to the Iris data examples. Thus, the GNP can get, e.g., the traded volume of a particular stock, and the judgment node selects an edge given the current value during the transition. The further question for a financial application is: How can the target variable be used in combination with the fitness function and processing nodes? It was pretty straightforward for the Iris data because the task was to classify the three flowers. Therefore, each processing node represents a particular species, and for measuring the individuals, the fitness function was just the accuracy of solving that classification task. What was not covered in the previous chapter is a detailed view of the encoding of the fitness function, whereby each fitness function depends on the target of the data science task.

Since every data science problem is different, it is helpful to consider different problem categories. Typically the following categories are distinguished [Berthold et al., 2010]:

- Classification (e.g., Iris data)
- Regression
- Clustering
- Association analysis
- Deviation analysis

The GNP can tackle problems of different categories depending on the fitness function. However, GNP is mainly used for classification and regression tasks. The Table 3.2 gives an outline of the used fitness function from selected papers above with different fitness functions, showing the assigned category and the target of the task. Further, the role of the processing node is described.

[1] Benoit B. Mandelbrot is recognized for his contribution to the area of fractal geometry and especially know for the Mandelbrot set.

Table 3.2: Outline of different Fitness Functions

Paper	Target of Task	Category	Fitness Function	Processing Node
An integrated framework of genetic network programming and multi-layer perceptron neural network for prediction of daily stock return: An application in Tehran stock exchange market [Reza Ramezanian, Arsalan Peymanfar, and Seyed Babak Ebrahimi, 2019]	Extracting rules from the GNP to match them to an interval representing the return.	Classification / Regression	$error_t = \sum_{i=1}^{n} e(i)$ Where e is the error as the difference between realized return and assigned bins of returns	Represents the intervals of return.
Generating Trading Rules on the Stock Markets with Robust Genetic Network Programming Using Variance of Fitness Values [Chen and Hirasawa, 2010]	Finding trading rules (buy or sell) of stock market data to maximize the profit of each stock and thus in the whole portfolio	Classification	$F = F_o - w\sqrt{\frac{1}{\lvert G\rvert}\sum_{g\in G}(F_o - F_g)^2}$ F_o = see fitness function below F_g is a previously calculated Fitness function of GRA [2]	Represents buying or selling decision.
Genetic relation algorithm with guided mutation for the large-scale portfolio optimization [Chen, Mabu, and Hirasawa, 2011]	Finding an optimized portfolio of stocks for a large number of stocks	Classification	$F_o = \sum_{b\in S(G)} Profit(b,n)$ $Profit(b,n) = Sell(b,n) - Buy(b,n)$ Where b = brand, $n = n$th generation and $S(G)$ = a set of stocks	Represents buying or selling decision.
Enhancing Global Portfolio Optimization using Genetic Network Programming [Parque, Mabu, and Hirasawa, 2010]	Select and allocate assets concerning return and risk of given stock data.	Classification	$F_S = \frac{\sigma_{R_S}\cdot\beta_S}{(R_S - R_F)\cdot L_S}$ Where R_S is the average return of assets, σ_{R_S} is the standard deviation of the returns, β_S = beta coefficient of the assets, R_F = average risk free rate, L_S = liquidity level of assets	Judgment nodes combine strategic measures and attractiveness to deal with return and risk. The combination of both leads to the selected edge in the transition. Finally, the processing nodes summarize these values and make a buy, hold or sell decision.

[2] GRA = Genetic Relation Algorithm. This is an algorithm that runs before the GNP to select a subset of stocks

3.3 Multiple Start Nodes

In many cases, when treating financial market data, the data comes for different assets or at least for different stocks. That means, e.g., that the data contains the price records of each stock for each day for a particular period. Therefore, such a data representation could also be observed for each particular stock which leads to at least two different approaches to treating the data:

1. Training one GNP for the whole data table where the stocks are not distinguished but pooled (one GNP treats all stocks).
2. Training a GNP for each group in the data (one trained GNP per stock).

Training the whole data would be the same as described in Chapter 2 with the Iris data examples, and no extensions of GNP would be needed. However, if desired to treat each stock by itself, each stock needs its own GNP model, and a vast number of separate models are required. Moreover, certain similarities between stocks cannot be exploited by the network. To obtain a GNP that can handle several groups (e.g., stocks), an extension of GNP, where multiple start nodes are permitted, was proposed [Chen et al., 2009b]. With this extension, each group is assigned to its own start node and runs a separate transition path through the same network. That means only one network can handle each data group, and training a separate GNP for each group is not needed. Figure 3.1 shows such a network with multiple start nodes.

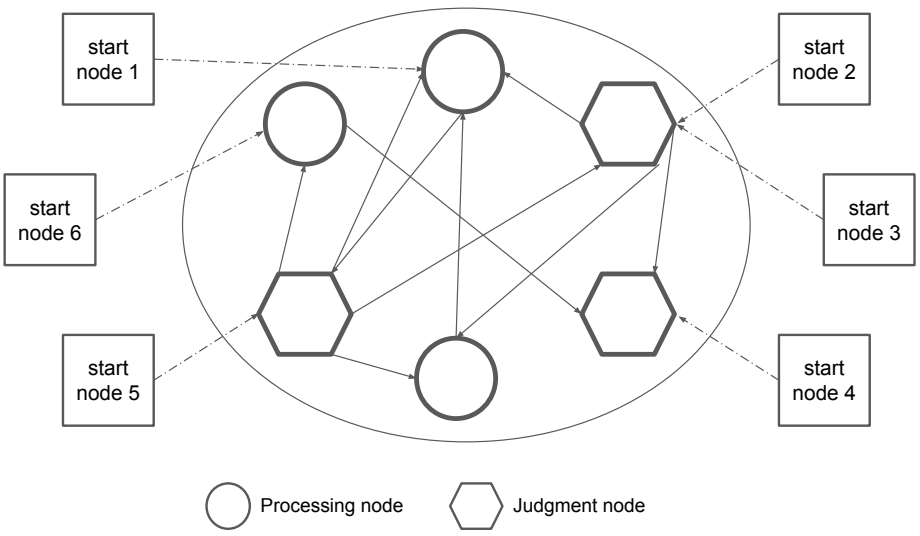

Figure 3.1: GNP with multiple Start Nodes
Source: own illustration according to [Chen et al., 2009b]

Each start node's connection is initialized randomly so that the edges of the start nodes can point to different nodes in the network but also to the same node (compare start node 2 and start node 3). In the financial context, multiple start nodes may be reasonable with the following assumption: if one stock tracks the development of another, but with a time delay, the second stock could have a start node that enters the network path of the first stock a bit later. This is only possible with start nodes

pointing to different nodes in the network and can lead to an interdependent path of multiple groups (stocks). In contrast to the GNP with one start node, an important issue occurs: the significance of nodes and edges concerning the genetic operators is different.

For example, consider the following: If a GNP is initialized with one start node, the mutation can lead to a change of all the edges of the network transition path if the edge of the start node is changed (see Section 2.5.1). It could be reasonable to exclude the edge of the start node from the genetic operation because of the mentioned epistasis problem in Section 2.5.3. Remember that changing an edge can make a massive difference in transition path and fitness. This is for the edge of the start node especially crucial because the start node's edge is necessarily the first edge in the transition path. It is also plausible because the probability of initializing all possible node indices (see the gene structure in Figure 2.3) is high. The reason is that the Network size is usually much smaller than the population size ($|V| \ll |individuals|$). For example, the initialization of 1000 individuals (networks) and a network size of 100 nodes leads to a probability of around 0.00432 in which a start node never points to a particular node index. All node indices are at least once initialized by a start node with a probability of $1 - (\frac{99}{100})^{1000} \approx 99.568\%$. That means the gene pool concerning different initial node indices should be high enough, and by excluding the start node from the mutation operation, the epistasis problem could be counteracted (even if only to a small degree). As mentioned in Section 2.5.3, a mutation operation, which depends on the transition path depth or usage, could be future work.

Nevertheless, for a GNP with multiple start nodes, the significance of each start node is reduced, because the entire fitness is the mean of the fitness values of all groups. Therefore a change of a start node has less impact on the overall fitness compared to a GNP with just one start node. Indeed, if multiple start nodes point to the same node in the network, the edge of that node could be more significant compared to an edge of a start node. For example, if all six start nodes in Figure 3.1 would point to the same processing node, it would probably have more impact on the fitness if the mutation changes the edge of that processing node compared to changing the edge of any start node. In that case, changing the processing node edge would change the transitions of all start nodes. In conclusion, the edges of the multiple start nodes are less significant regarding the change by the mutation operation, and the epistasis problem is less relevant for these edges.

Not only the hope of some dependencies of the different stock and learning abilities with just one network but also fitness functions with dependencies of the different groups are possible with multiple start nodes. For example, consider a fitness function where the profit of each stock is calculated. The total fitness could be the mean of all the profits, where each stock always has the same budget. However, because the same network treats all stocks (groups), different budgets for each stock can be evolved during the evolutionary process. In conclusion, multiple start nodes can lead to different weights concerning the entire stock portfolio.

3.4 Validation of Time-Series Data

The fitness function is the fitting criterion to find the best or at least a good model. It was deliberately avoided to apply validation for the Iris data example, but typically one does not train with the whole data, and a test validation is recommended.

The reason is that complex models can satisfy a simple fitting criterion better than simple models. However, the problem of overfitting grows with the complexity of

the model. If the model is too adapted to the training data, it will likely be worse for predicting new data. The most common approach to estimate a realistic performance of the model for unknown or future data is splitting the data set for training and testing. The model is naturally trained on the training data, and the test data will be used for the evaluation. To prevent too good results on the test data just because the test set contains easy examples or worse results if the test data contains more difficult examples by accident, several splits in test and training data are often used. This method is called cross-validation, where several model error estimates are calculated [Berthold et al., 2010].

Cross-validation can be done in different ways, but it should be mentioned that no random samples can be drawn from the data for time series. This is because time series data comes in a fixed order, and random drawing could destroy the time order. Therefore a version of cross-validation that ensures the fixed order of the data is mandatory [Hirschle, 2020].

Chen et al. (2009b) proposed the "time adapting genetic network programming (TA-GNP)" with real-time updating, which contributes to an improvement of the static GNP. In the traditional GNP method, individuals are involved using the initial data of stock prices in a single period. Then the best GNP individual in the last generation of the training period is used for all future tradings in the validation period. However, since the data in the stock market are time-varying, they are highly influenced by intermediate changes. Because of these dynamic features, they found that the time-adapting GNP method considers the time-related fluctuation and trend of stock prices well. Time-adapting GNP uses historical data for training and validation by shifting a fixed window in each iteration. Note that in time-adapting GNP, the best GNP individual is used for trading immediately after the evolution is finished using the last historical data. As a result, time-adapting GNP could be used for online trading, where GNPs adapt to the changes in stock prices [Chen et al., 2009b].

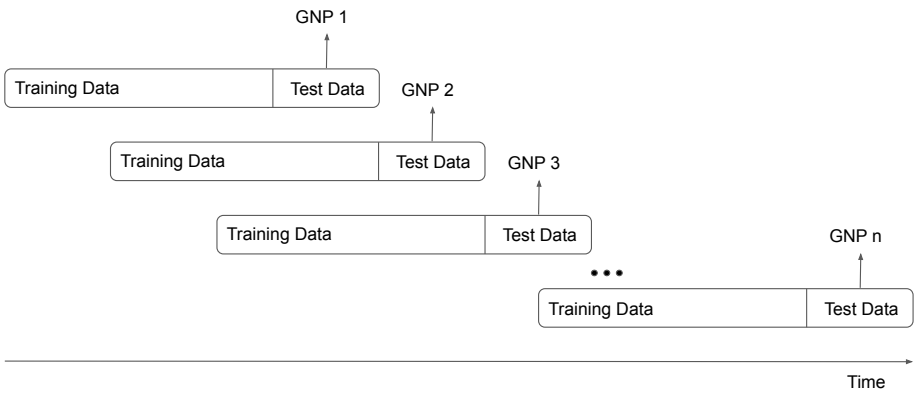

Figure 3.2: Time Adapting Cross-Validation
Source: Author's own illustration

Figure 3.2 shows the time-adapting GNP method. Each population of GNP is applied to the current training data, and the best individual is validated on the test data. The fixed window shift is the size of the test data. Compared to standard

cross-validation, where a final model is built on the entire data set, a separate GNP is used continuously for each training and test data pair. This is illustrated in Figure 3.2 by the different GNPs above the test data. The time-adapting GNP method can be applied to any training and test data split. Overall, the shifting window method can include newly arrived non-stationary information in the optimization process and simultaneously avoid overfitting.

Chapter 4

Variable Size GNP

After the basic GNP and an extension for a financial application were described, variable size GNP and two novel mutation operations for GNP are part of this chapter. The study and evaluation of the two new operators will be done in a separate procedure. Individual components are examined as an ablation study. The performance of the GNP is observed by separating the different components to understand their influence on the entire algorithm. Furthermore, an overall analysis is made and additionally examined in Chapter 5 as the experimental part.

As mentioned in Section 2.5.4, a fixed gene pool of standard GNP could be tackled with the following:

1. Applying mutation to change the network size.
2. Applying mutation to change the boundaries of the interval.

The fixed number of nodes of GNP is often consciously chosen because the fixed size can prevent the bloat problem (see Section 2.1.1), which is an advantage compared to traditional GP. However, this advantage can be a disadvantage concerning the complexity of the given problem. When the problem is complex, it is impossible to get enough prior knowledge or try many different sizes of GNP. On the one hand, if the number of nodes is too small, the individual could lack expression ability. On the other hand, if the number of nodes is too large, it needs a large search space, and the individual will be overfitting easily.

The trade-off between a complex model and a simple model for fitting the data is a typical data science problem. Complex models are more flexible and usually generate a better fit for training data. However, how well a model fits the training data only tells a little about how well the model represents the inherent structure in the data. As a result, too complex models tend to overfit the data. Therefore, selecting a model is a trade-off between simplicity and fitting. Based on the principle of Occam's razor, one should prefer the simplest model that "explains" the data. The problem is finding a model that fits the data well enough and simultaneously fits a simple model [Berthold et al., 2010].

For GNP, more nodes mean much higher expression ability, which is necessary for solving complicated problems, while too many nodes need more generalization ability. For these reasons, Variable Size Genetic Network Programming (GNPvs) was proposed, which changes the individuals' size and tries to obtain their optimal size during evolution. Katagiri et al. (2003) is the first paper on variable-sized Genetic Network Programming that was proposed shortly after the first paper on GNP around 2000. In this paper, the contribution of each kind of node for the fitness value is computed. After that, the algorithm decides whether a node is to be added to all the individuals or is to be deleted from them depending on the contribution of it to

the fitness value. That means the size of the GNP is variable, but for the whole population still the same, which limits the ability to keep the diversity of chromosomes. Furthermore, the exploration ability of the algorithm by containing different sizes of individuals in the population is restricted.

4.1 Variable Size GNP with Crossover Operation

A further improvement of GNPvs was suggested by [Li et al., 2011], where both parents employ a binomial distribution to select a number of their nodes. Concretely speaking, a new type of crossover was introduced in GNPvs to replace the standard uniform crossover in GNP in which each parent has its own crossover rate. Then two children are produced by matching the selected part of one parent with the non-selected part of the other parent.

Although GNPvs is a new type of GNP, the phenotype and genotype of GNPvs are the same as the standard GNP described in Chapter 2. The implemented crossover of GNPvs is an extension of uniform crossover. In the uniform crossover, the corresponding nodes have the probability, i.e., crossover rate of P_c, to swap each other. Since uniform crossover randomly picks up the nodes from the individual with a crossover rate of P_c, the probability of choosing n nodes from the total N nodes follows a binomial distribution. However, in the crossover of GNPvs, each parent has its own crossover rate, and therefore, each network has its own binomial distribution. Nevertheless, when performing crossover, two GNPvs individuals may choose a different number of nodes to move, even when the crossover rate is the same [Bing, 2013].

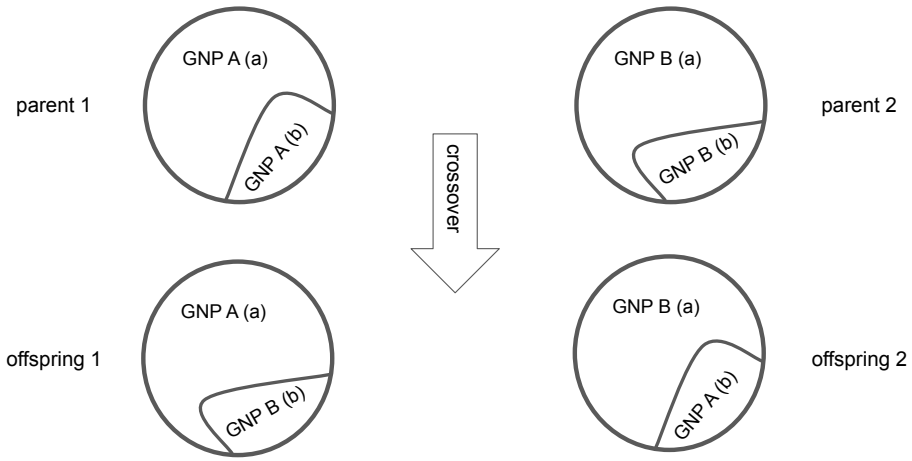

Figure 4.1: Crossover of GNPvs
Source: own illustration according to [Bing, 2013]

Figure 4.1 shows a schema of the introduced GNPvs. Parent 1 is labeled as GNP A, and parent 2 is labeled as GNP B. The crossover operation exchanges the subsets of nodes GNP A (b) and GNP B (b). This creates two new offspring. Offspring 1 now consists of GNP A (a) and the exchanged part GNP B (b). Offspring 2 consists of GNP B (a) and the exchanged part GNP A (b). As mentioned above, the subsets to

be exchanged can have different sizes and follows the binomial distribution, which is indicated by the different shape of the subsets. The number of nodes to be exchanged is the first part of the GNPvs algorithm. To ensure that the set of nodes to be exchanged is contiguous, the nodes to be exchanged are then selected according to the following rules:

- If the current node is a processing node, select the node connected from the current node.
- If the current node is a judgment node, choose one branch from the current node to select the node randomly.
- If the next node is already in the moving set, pick up another node randomly.

After the nodes of two GNPvs are exchanged, the edges of these nodes have to be updated. This is because of two reasons:

1. For the moved nodes, the indices of nodes in the gene structure can change. Therefore the connections of these nodes should be changed to keep the building blocks of these nodes.
2. The connections of all the nodes in the individual, which link to the deleted nodes, must be updated to other nodes.

Figure 4.2 on Page 52 shows an example of the GNPvs crossover. The parents and offspring are shown as networks with the corresponding gene structure. The crossover operation selects node 4 of parent one and nodes 1 and 4 of parent two, which is indicated by the dotted lines in the gene structure and the gray hatching. Then, the selected node 4 of parent one is transferred to the new node 1 of offspring two. Therefore, node 1 of offspring two is written in bold in the gene structure. Since no edge of the replaced node points to a node that no longer exists, none of the edges need to be adjusted. However, since there is no node 4 left in offspring two, the connection of node 5 must be randomly changed to another node. In this example, the edge now points to node 1.

The two transferred nodes of parent two are the new nodes 4 and 5 of offspring one. In that case, the indices of the nodes are changed, and the connections need to be updated. To keep the building blocks of these nodes, the second connection of the new node 4 is changed from 4 to 5. Further, the second connection of the new fifth node of offspring 1 is changed randomly because no self-loops are allowed in GNP.

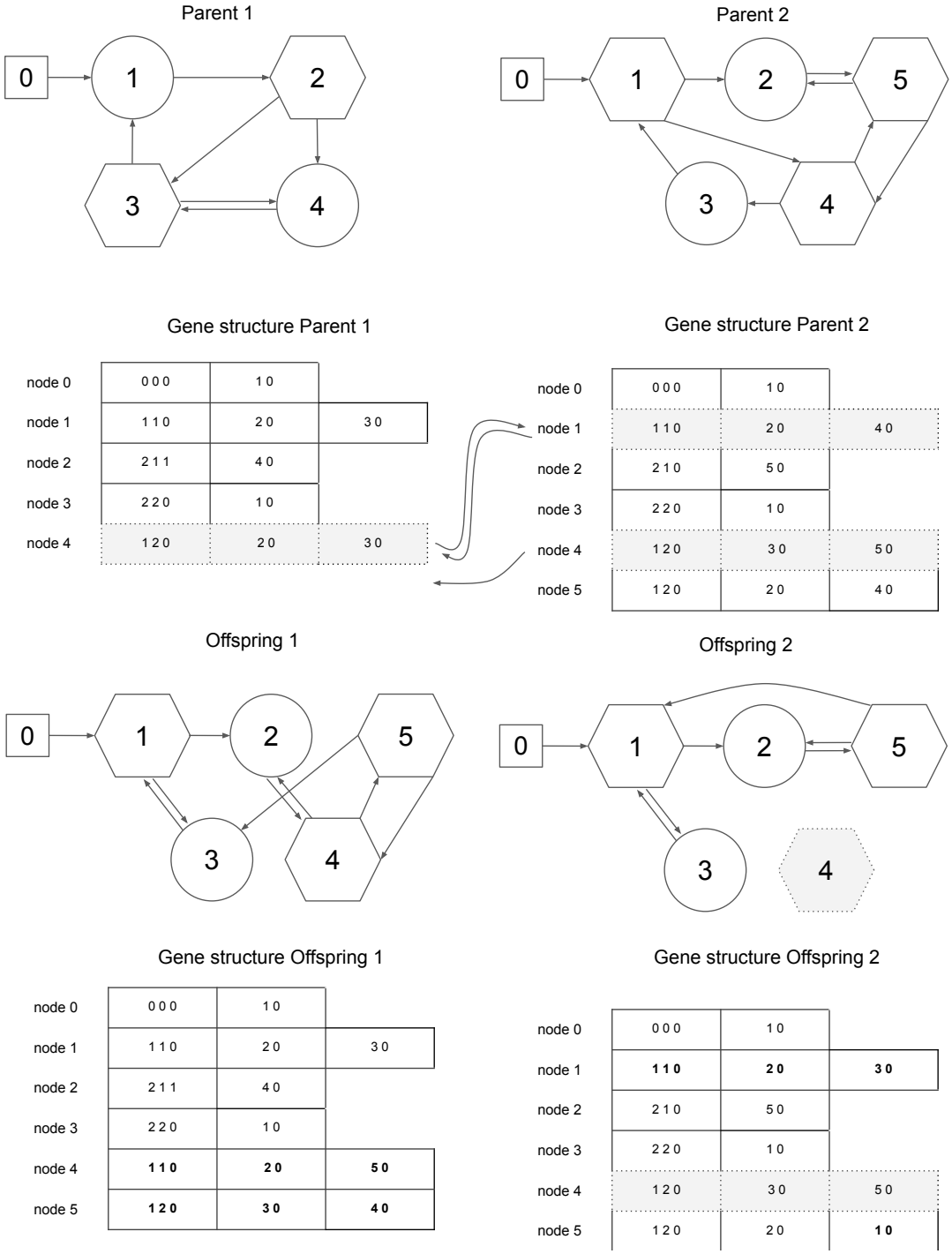

Figure 4.2: Example Crossover of GNPvs
Source: own illustration according to [Bing, 2013]

The limited gene pool mentioned in the motivation of this thesis becomes clear with this example of GNPvs. Although different numbers of nodes are exchanged by the crossover operator and the network could adapt better to the complexity of a problem, the number of outgoing edges always remains the same. There is only an adaptation of the edges when the edges point to another node. The number of outgoing edges of a node in an individual is bounded by $N - 1$ where N is the number of initialized nodes in the network.

Furthermore, the node functions are limited by the size of a GNP. For example, if each network consists of only two processing nodes, then by initializing the network, the processing functions are limited to function 1 and function 2 from the library. The same holds for the judgment nodes. This is especially problematic in a data mining task in which many features are used because the networks must be initialized with a lot of nodes to represent all features.

The restriction by the number of edges and the limited node functions can be a constraint to solving an optimization problem. The following example, again using the Iris data set as an illustration, shows an insufficient gene pool. Assume that the GNP attempts to classify the Iris data set with the initialized network in Figure 4.3.

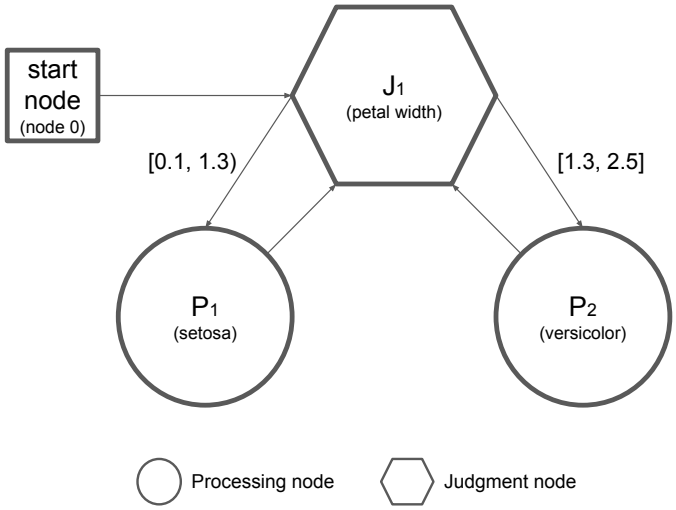

Figure 4.3: Iris Data insufficient Nodes
Source: own illustration

Only two processing nodes have been initialized for this purpose, which makes it impossible for the GNP to classify three different flowers. Moreover, the number of outgoing edges of the judgment node is limited by $N - 1 = 3 - 1 = 2$. This means that data can be divided into, at most, two intervals of equal size.

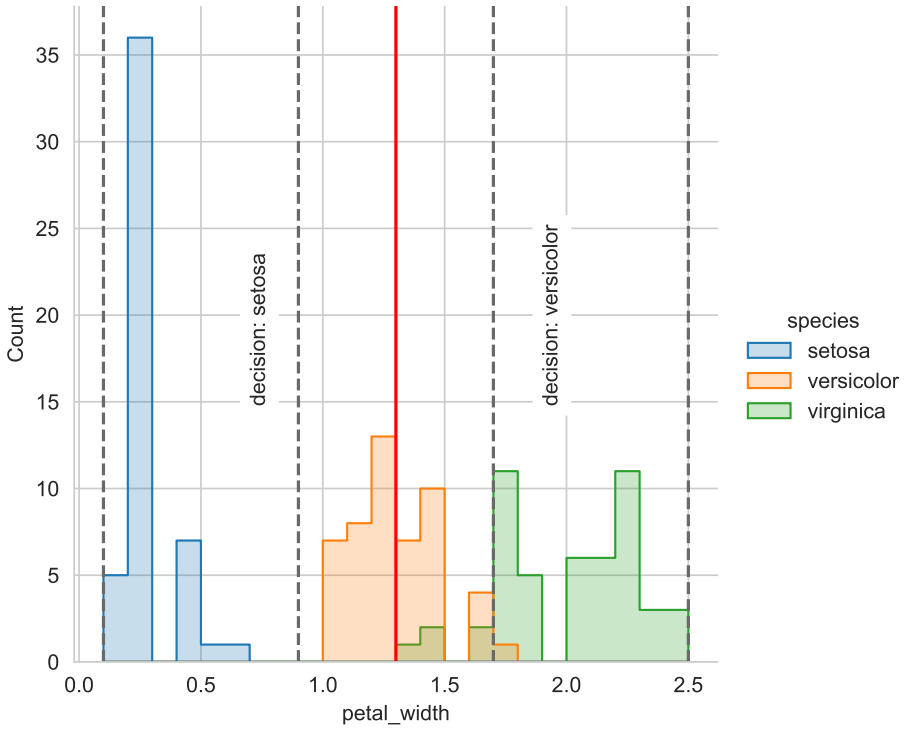

Figure 4.4: Iris Data Classification less Nodes
Source: own illustration

Figure 4.4 shows a similar illustration as Figure 2.6 from Section 2.1.2 where the interval boundaries of a trained GNP were shown. The variable "petal width" was partitioned into three equally sized intervals in the learned GNP used at that time. That classification was only possible because the node had three outgoing edges. Figure 4.4 now shows these edges dashed in gray and the middle interval boundary of the judgment node from the current example with only two outgoing edges in red. Two processing nodes are too few to classify the three flowers, and the maximum accuracy can only be $\frac{2}{3}$.

Moreover, an interesting side effect of the too-small number of edges occurs. Since the petal width data can only be divided into two intervals of the same size, the split of the data at the point of the red-marked boundary is not practical when classifying Setosa and Versicolor. The network leads to even worse results than $\frac{2}{3}$ because the data division is too broad. This can be seen from the fact that a large proportion of Versicolor is misclassified to Setosa. As a result, the GNP can only poorly classify the Setosa and Versicolor species from the data and the results do not improve when the GNPvs crossover operator makes the network larger since the processing functions, and the outgoing edges are fixed.

For the Iris data set, it would be obvious to initialize three processing nodes because of the three species. Furthermore, initializing at least four judgment nodes is reasonable because of the four features. However, in practice, the number of initialized nodes can usually not be estimated easily. Consequently, this example illustrates a too-small gene pool and its problematic consequences. Two new mutation operators have been developed to tackle the problem of a fixed gene pool and will be described next. It is shown that the GNP achieves very good results on the Iris data set regardless of how many nodes are initialized.

4.2 Mutation Operator Add/Delete Nodes

This section introduces a novel mutation operator, which can add and delete nodes in a GNP. The operator allows GNP to grow and shrink, and there is no fixed predefined number of nodes, like in the standard GNP. Since this operator is a mutation operator, it only operates on one solution candidate, unlike the crossover operator presented above. During a generation, the novel mutation operator decides for each individual randomly whether a node can be added or deleted. This means that a node can either be added *or* deleted to an individual during a generation.

From the example, in the previous Section 4.1 it was illustrated that it could be useful to add nodes to a network. This is mainly because of the following:

1. A limited number of outgoing edges and
2. a limited number of nodes and their functions.

By adding nodes, the network can adapt to more complex problems that it could not describe in this way before. Adding nodes is described in detail in Section 4.2.3. However, besides the risk of overfitting, why should nodes also be deleted from a network?

A simulation study was made to show the impact on the fitness of different-sized networks. First, the different numbers of judgment and processing nodes were simulated 20 times. Then, the mean of maximal and mean fitness was calculated, which can be seen in Table 4.1. The evolution of the GNP was simulated with the following parameter:

- Fitness: Accuracy
- Generations: 100
- Population size: 100
- Judgment Nodes: see table
- Processing Nodes: see table
- Mutation Probability: 0.1
- Crossover Probability: 0.1
- Elitism: 2
- Time Delay on Nodes: 1
- Time Delay on Edges: 0
- Maximal time Units: 10

Table 4.1: Simulation Study of 20 different GNP runs for each combination of parameters on the Iris data. The Best Fitness and Mean Fitness are the mean values of the 20 runs.
Source: author's personal research results.

Judgment Nodes	5	10	15	20	30	40
Processing Nodes	5	10	15	20	30	40
Best Fitness	0.95	0.89	0.75	0.65	0.59	0.57
Mean Fitness	0.58	0.37	0.29	0.25	0.25	0.24

Table 4.1 shows that the best and average fitness results become worse with an increasing number of initialized nodes. If five judgment nodes and five processing nodes are initialized, then the GNP can achieve the best fitness (accuracy) of 0.95 on average over the 20 simulations. This accuracy is similar to the accuracy of 0.96 in the example from Section 2.1.2. The decreasing fitness with more initialized nodes ends up with an average best accuracy of just 0.57 when 40 judgment nodes and 40 processing nodes are initialized.

The problem of an inappropriate number of outgoing judgment node edges also becomes evident with *too many* nodes. The network is too interconnected to successfully solve a simple classification problem like the Iris data set. Figure 4.5 shows all used nodes during the simulation run, where 15 judgment nodes and 15 processing nodes were initialized. The represented GNP had reached a fitness of 0.52 after the 100 generations. Compared to the small network in Figure 2.5 on page 19, it becomes visible that the network achieved poor results because it could not generalize the data sufficiently. This is, e.g., shown by node 3 on the left side of Figure 4.5, which is the judgment node that handles the feature Petal width. The node classifies the species Versicolor in an interval of [0.984,1.111). Figure 4.4 shows that the Species Versicolor is correctly classified in this interval. However, the interval is much too small to make a meaningful classification of the distributions due to the too large number of outgoing edges of the node, and such granular data classification also increases the risk of overfitting, as shown below.

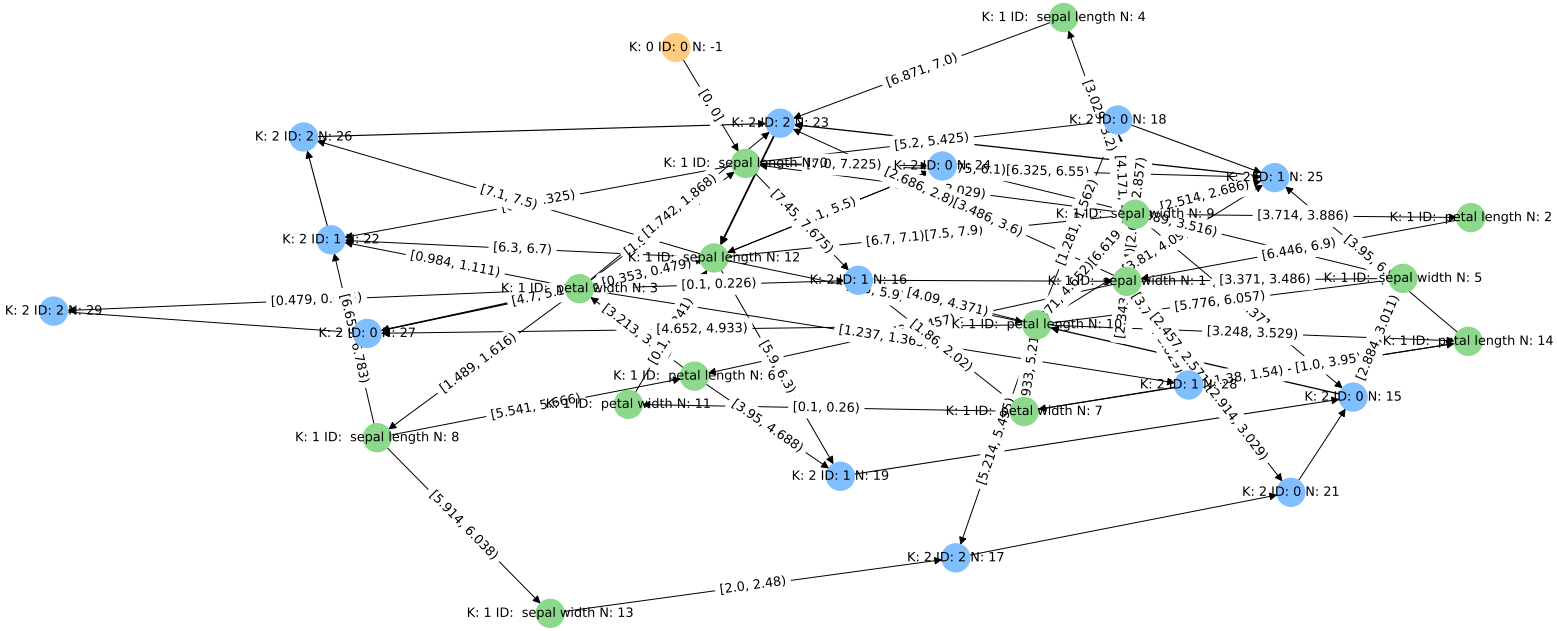

Figure 4.5: Iris Data Classification too much Nodes.
All used nodes of the GNP run are shown. Processing nodes are colored in blue; judgment nodes are in green. The orange node is the start node.
Source: own illustration

Thus, the network shown in 4.5 has too many nodes with too many edges to solve the simple Iris data problem well. However, a GNP can use only a small subset of the nodes for the transition path. As described in Section 2.5.1, intervals of edges could also join by the standard mutation pointing to the same node. This would lead to a generalization using an extensive network. So it might be that the results of the large networks from the Table 4.1 are so worse because the 100 generations were not enough to fit a large network with many edges sufficiently well to the data.

Therefore, the Table 4.2 shows the results of a repeated simulation study with the same parameters but with 1000 generations.

Table 4.2: Simulation study of 20 different GNP runs for each combination of parameters on the Iris data. The best fitness and mean fitness are the mean values of the 20 runs. The simulation was done with the same parameters as above but with 1000 generations. Source: author's personal research results.

Judgment Nodes	5	10	15	20	30	40
Processing Nodes	5	10	15	20	30	40
Best Fitness	0.96	0.93	0.90	0.89	0.80	0.65
Mean Validation	0.83	0.81	0.71	0.63	0.55	0.39

Row three shows that the networks could now achieve better results on the training data. However, the trend remains that larger networks achieve worse results. The average best fitness remains very poor (0.65), with 40 judgment nodes and 40 processing nodes. Additionally, in this simulation, the data set was split into 90% training data and 10% test data. This allowed validation of the best models by the test data after training the GNP. The results of the average validations are shown in row four of the Table 4.2. It can be seen that the validations show significantly worse results compared to the results on the training data. This leads to the conclusion that the large networks did not generalize but adapted much more granularly to the data. The networks are over-fitted to the data.

Altogether, it becomes clear that a GNP can also be too large for a problem to be solved and that deleting nodes is reasonable.

When nodes are deleted from or added to a GNP, it can lead to several problems. These are defined first and then covered in the following two sections to finally end up with the new algorithm:

- The deletion of nodes can lead to huge fitness differences. Thus, similar to the mutation operator that changes edges, epistasis can quickly occur.
- If too many nodes are deleted, the network cannot adapt sufficiently to the complexity of the problem. This is analogous to the problem from Section 4.1 where a too small network cannot describe the Iris data sufficiently. In the worst case, all nodes are deleted. Obviously, this is not desirable.
- Edges that pointed to a deleted node can cause an error when traversing the network.
- The crossover operator (see Section 2.5.2) can cause errors when nodes of different size networks are exchanged.

- Too many inserted nodes can lead to very large networks and overfitting of the problem. Furthermore, unlimited growing networks can lead to the bloat problem.

The different challenges are now described in detail to finally come to a suitable mutation operator.

4.2.1 Deleting Nodes

The deletion of nodes can lead to huge fitness differences. This is evident from the network learned in Section 2.1.2 on page 19, which successfully classified the three species of the Iris data set. Figure 2.5 summarizes the network's entire run to the used nodes. The network used only one judgment node for the transition path, so the three species were classified only by the variable Petal width. Furthermore, all edges of the processing nodes pointed back to that judgment node. Thus, the judgment node was used again after each classification and was the most frequently used node in this network. Assuming that the four nodes are the only nodes in the network, it is clear that if the judgment node were deleted, a crucial node for classification would no longer be in the network. In fact, after deleting the judgment node, only processing nodes would be in the network, and the classification would no longer be possible using any of the features. Furthermore, deleting one of the processing nodes would result in only two species being able to be classified. This would also have a strong negative impact on fitness. Under the assumption that the edges pointing to the deleted nodes are also deleted, Table 4.3 shows the best possible fitness of the individual after the respective node is deleted:

Table 4.3: Fitness Impact of Deleted Nodes
Source: own illustration.

Deleted Node	5 (Versicolor)	4 (Setosa)	6 (Virginica)	3 (Petal Width)
New Fitness	$2/3$	$78/150 = 0.52$	$72/150 = 0.48$	$51/150 = 0.34$
Fitness Impact	≈ -0.293	-0.44	-0.48	-0.62

Note that the original GNP with the four nodes could classify the Iris data set with an accuracy of 0.96. The difference between the 0.96 is calculated in the row "Fitness Impact". It can be seen from the Table 4.3 that the best result is obtained when the processing node that classifies Versicolor is deleted. Nevertheless, even in this case, the influence is strongly negative, with 0.293 on the accuracy. This can also be taken from the Figure 4.4 since the judgment node now has two outgoing edges and separates the data set according to the red line at position 1.3. Since Versicolor can no longer be classified, the accuracy is $0/50$. The other two species (Setosa and Virginia) could be correctly classified in $50/50$ cases, resulting in an overall accuracy of $100/150 = 2/3$.

When one of the other two processing nodes is deleted, the problem occurs that the separation of the data (red line) does not fit well with the species being classified, and the results become worse. So overall, deleting any node significantly impacts fitness, and the problem of epistasis is given for this example.

A reasonable approach to delete nodes from the network could be to rank the nodes according to the number of times they are used during the run. Nodes that are not used very often could be deleted rather than nodes that are used frequently. However, even this approach runs into problems with the network from Figure 2.1.2 on page 19 because deleting *any* node would have a significant negative impact on

the fitness of the individual for the reasons mentioned above, as shown in Table 4.3. The same holds for a procedure concerning fitness contribution as proposed in the first paper about variable size GNP.

As discussed in Section 2.5.1, a mutation operator is usually harmful to an individual, and therefore, fitness disadvantages are also justified to a certain degree. However, as the previous example shows, the problem of a huge fitness impact is given, and epistasis arises again. To avoid epistasis, the newly introduced mutation operator is *fitness neutral* to the individuals. This is achieved by the restriction that a node is deleted only if it is *not* used in the transition path of the network. Nodes that are not used can arise from the properties of a GNP as already described in Section 2.1.2. For example, Figure 2.4 shows that out of seven initialized nodes, only four were used. The remaining red-marked nodes are fitness neutral and can now be deleted by the new mutation operator. In addition, deleting nodes that are not in the transition path prevents invalid networks after the deletion. This can happen, for example, if all processing nodes are deleted and the network is destroyed.

If the unused nodes in the transition path are deleted, then the connection gene is also deleted, and with that, all outgoing edges. Not well-defined graphs can occur if the edges pointing to the deleted nodes are not changed. This is because a graph and its edges are defined as the sets of those nodes that are in the network:

Definition 2 *A graph is a pair $G = (V, E)$, where V is a (finite) set of vertices or nodes and $E \subseteq V \times V$ is a (finite) set of edges. It is understood that there are no loops, that is, there are no edges (A, A) for any $A \in V$ [Borgelt, Steinbrecher, and Kruse, 2009].*

An edge is a subset of the Cartesian product of the set of nodes. Therefore, an edge with only one node is undefined. It is clear that a node with only one edge for the GNP would also lead to errors in the program. For example, a processing node with no successor [1] would cause the transition path to go nowhere. For this reason, the edges that point to a deleted node are randomly changed to another node, and deleting a node can additionally change some other edges in the network. However, these edges were not used for the transition path (the node was never traversed), and their changes are also fitness neutral.

To show the impact of the new mutation operator, the simulation study of the previous section was repeated. The parameters are the same as in Section 4.2, but in addition, the new mutation was applied in this simulation study. Nodes could therefore be deleted during evolution. Table 4.4 shows the average results of the 20 simulations of the best and average fitness.

[1] A successor is a vertex coming after a given vertex in a directed path.

Table 4.4: Simulation study of 20 different GNP runs for each combination of parameters on the Iris data with the novel mutation operator. The best fitness and mean fitness are the mean values of the 20 runs.
Source: author's personal research results.

Judgment Nodes Processing Nodes	5 5	10 10	15 15	20 20	30 30	40 40
Best Fitness (Novel Mutation)	0.94	0.91	0.93	0.93	0.91	0.94
Mean Fitness (Novel Mutation)	0.65	0.61	0.60	0.65	0.58	0.64
Best Fitness (Fixed Size)	0.95	0.89	0.75	0.65	0.59	0.57
Mean Fitness (Fixed Size)	0.58	0.37	0.29	0.25	0.25	0.24
Difference Best Fitness	-0.01	+0.02	+0.18	+0.28	+0.32	+0.38

With the new mutation operator, *all* GNP achieved a maximum fitness of over 0.9 on average. A very good maximum fitness could be achieved no matter how many nodes were initialized in the network. For comparison, the results of the previous simulation study are also presented, and the last line shows the difference between the best fitness values.

The difference is the smallest for GNP with five initialized judgment and processing nodes. With -0.01, the two simulation studies achieve nearly the same result. The difference between the two studies becomes more significant as more nodes are initialized. This is because the standard GNP shows worse results, and the GNP with the new mutation could achieve consistently excellent results.

Figure 4.6 shows how the number of nodes of all individuals decreases on average over the 100 generations. This demonstrates that the mutation deleted unused nodes and smaller individuals with fewer nodes were more likely to get into the next generation. However, this is not due to the new mutation itself, which is fitness neutral. Instead, the standard GNP's operators can better exploit the problem's structure with an appropriate number of nodes and edges.

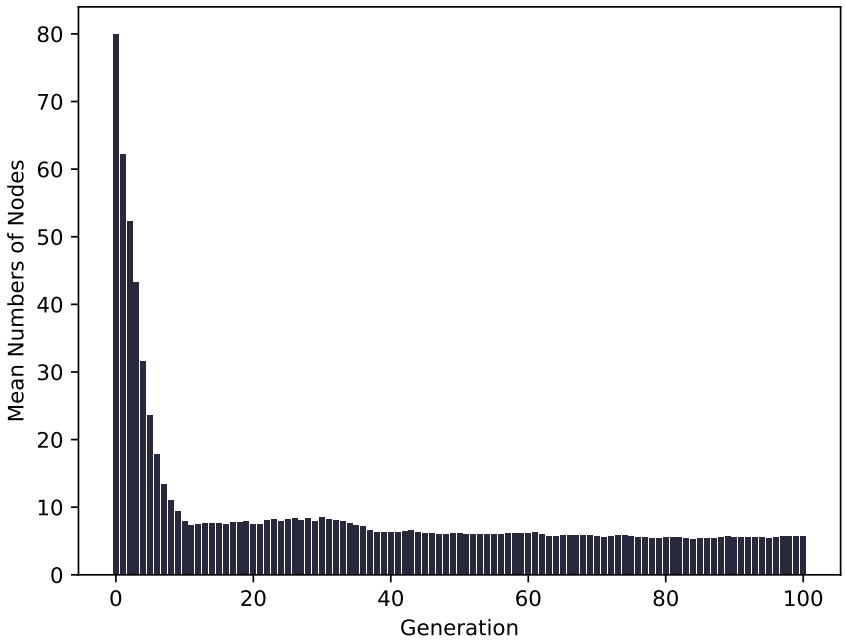

Figure 4.6: Shrinking Mean Node Numbers Iris Data Example
Source: own illustration

The average number of nodes in the GNP decreases rapidly in the first generations. This is because many of the GNPs had unused nodes, which were deleted, especially in the first generations. A steady number of nodes happens approximately after the 10th generation, with an average number of nodes around 7. This is consistent with the simulation study, which already showed the best results with five judgment and five processing nodes in the standard GNP. The network sizes could, therefore, successfully adapt to the appropriate size to classify the Iris data.

A video "GNPdy (Shrinking) - Evolution Best Individual Iris Dataset" of the entire evolution of the best individual can be found on the project homepage in the section *"Research"*:

`https://fabiankoehnke.github.io/gnp/hp/#research`

The better fit to the data with the new mutation operator is also evident when comparing validation results and increasing the generations to 1000. For that the data was split into 90% training data and 10% test data. Table 4.2 illustrated that the standard GNP overfits the training data without the new mutation operator. Table 4.5 now shows these results with those of GNP with the new operator. In addition, rows seven and eight show the average number of nodes of all individuals in the last generation.

Table 4.5: Simulation Study of 20 different GNP runs for each combination of parameters on the Iris data. The Best Fitness and Mean Fitness are the mean values of the 20 runs. The simulation was done with the same parameters as above but with 1000 generations.
Source: author's personal research results.

Judgment Nodes Processing Nodes	5 5	10 10	15 15	20 20	30 30	40 40
Best Fitness without Mutation	0.96	0.93	0.90	0.89	0.80	0.65
Validation without Mutation	0.83	0.81	0.71	0.63	0.55	0.39
Best Fitness with Mutation	0.96	0.97	0.98	0.97	0.97	0.97
Validation with Mutation	0.89	0.87	0.81	0.91	0.93	0.82
Mean Number of Nodes without Mutation	10	20	30	40	60	80
Mean Number of Nodes with Mutation	6.21	6.70	6.72	6.66	6.05	6.86

All validations with the new operator outperform the results without the operator. The differences between the training results and validations in the simulations with the new operator are not significantly different. Thus, there was no overfitting, leading to strongly different results in the validation. Furthermore, it is remarkable that no matter how many nodes were initialized, all individuals contained between 6.05 and 6.86 nodes on average. Thus, the GNP always fits the data with an appropriate number of nodes and can prevent overfitting the data.

The small number of nodes in the network also leads to a significantly faster calculation time. Table 4.6 compares the calculation times.

Table 4.6: Running Time Comparison Iris data of 20 different GNP runs for each combination of parameters on the Iris data. The simulation was done with the same parameters as above but with 1000 generations.
Source: author's personal research results.

Judgment Nodes Processing Nodes	5 5	10 10	15 15	20 20	30 30	40 40
Running Time Faktor without Mutation	1	1.19	1.70	2.75	5.24	8.31
Running Time Faktor with Mutation	1.52	1.53	1.54	1.58	1.59	1.64

The calculation time of the GNP with five judgment and five processing nodes without the new mutation operator was the fastest and is set to 1. All other entries represent the factor of how much longer the calculation was needed. Not surprisingly, the computation time without the operator increases with the number of nodes. The calculation time with the new mutation operator is longer for five judgment and five processing nodes. This is caused by the longer computational effort due to the novel mutation operator. It is interesting to observe that all simulations required almost the same time by deleting the nodes, even if significantly more nodes were initialized. It follows that deleting nodes can also make the algorithm's running time more efficient.

4.2.2 Crossover Adjustment

As mentioned, an individual's fitness is not changed by the new mutation operator but by the genetic operators discussed in Section 2.5. The mutation discussed in

Section 2.5 changed the successor with probability p_m for each edge which still works when the number of nodes in the network changes.

For the crossover operator, exchanging nodes from two individuals of different sizes can result in graphs that do not satisfy the definition 2. This is shown in Figure 4.7. Parent 1 has three nodes, and parent 2 has four nodes. If the crossover operator now exchanges two nodes, two problems can occur:

1. If $i = 4$ has to be exchanged, then parent 1 has no such node.
2. If a node of the larger parent is exchanged, which points to a node that does not exist in the smaller individual, then the edge of the node points to no node. This is shown in Figure 4.7, where node 3 is exchanged. Node 3 of parent 2 points to node 4. If this node now comes to offspring 1, then the edge continues to point to the nonexistent node 4. Such an edge does not satisfy the definition from 2 because node 4 is not part of the graph.

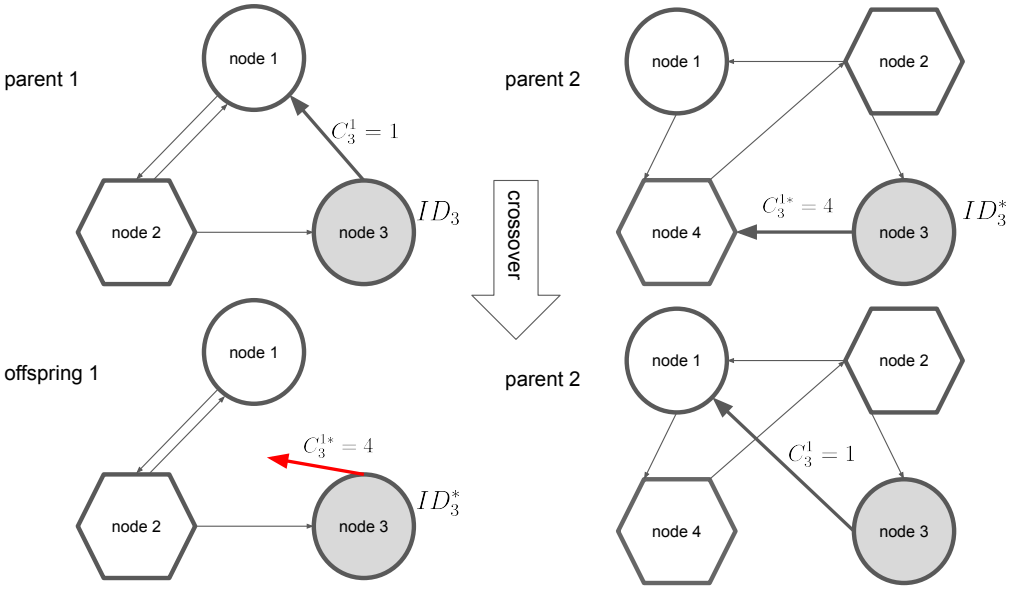

Figure 4.7: Wrong Crossover Example
Source: own illustration

Two adjustments were made to avoid these problems and keep the crossover operator:

1. Only nodes i with a maximum node number of the size of the smaller individual ($|V|$) are exchanged.
2. Edges of the exchanged node with a successor not in the individual are randomly changed to another node.

4.2.3 Adding Nodes

This section describes how to add nodes to a GNP. Adding nodes allows the GNP to adapt to more complex problems. This is based on a limited number of edges, and nodes explained in Section 4.1. Figure 4.3 showed an example network that was initialized with too few nodes and did not solve the Iris data problem well. Also,

the crossover operator from Section 4.1 could not improve results because of a too small gene pool. The maximum accuracy of training the Iris data was limited with $\frac{2}{3}$ by only two available processing nodes. This restriction is prevented by the new mutation operator as will be shown later.

The main problem when adding nodes is the bloat problem. As described in Section 2.1.1 this problem occurs particularly in the GP. In this context, so-called introns can cause an individual to grow into unnecessarily large programs. In addition, an increasingly large individual can lead to overfitting the training data. For this reason, the fixed number of nodes in a standard GNP is a great advantage because only the given network size is used to solve a problem. The bloat problem is consequently prevented. Nevertheless, as mentioned above, this advantage comes with a disadvantage concerning learning ability.

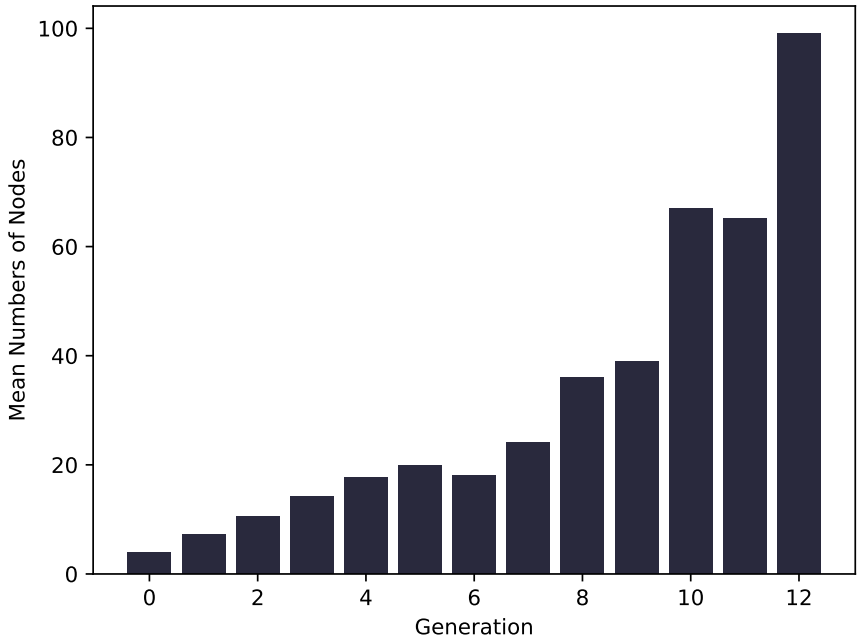

Figure 4.8: Mean Number of Nodes no Constraint
Source: own illustration

If nodes are added to a GNP, it should not be done carelessly because of the bloat problem. This is shown in Figure 4.8 where a GNP was trained with the parameters from Section 4.2 using the Iris data. In this case, nodes were added to the network without any constraints. By adding nodes, the average number of nodes in the individuals grows strongly. In the beginning, each individual was initialized with four nodes. After only 12 generations, each individual consisted on average of about 100 nodes. This can result in individuals whose majority of nodes are not used for the transition path.

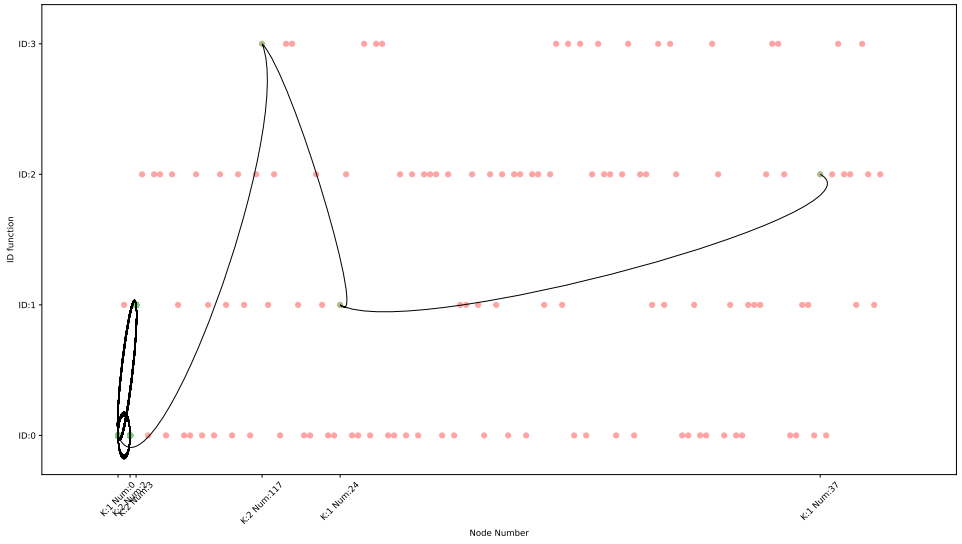

Figure 4.9: Example GNP with many unused nodes
Source: own illustration

Figure 4.9 shows an example individual with all nodes and the corresponding transition path. Used nodes by the transition path are marked in green, and all unused nodes are marked in red. It can be seen that a large proportion of the nodes are not used for the transition path. Similar to the introns, this network bloating is caused by the fact that the unused nodes are fitness neutral. Adding nodes does not affect the individual's fitness and thus does not affect the selection pressure. Therefore, bloated individuals can persist over the generations. This is because a node is inserted according to the initialization from Section 2.2 where outgoing edges of the new node are assigned just to a random successor. When a node is inserted, it is still *not* part of the transition path since no other edge from the network points to the inserted node. The new node can evolve a successor using the standard genetic operators and then become part of the transition path. As a result, an inserted node can initially be in the network without being a successor of any other nodes.

To prevent the number of nodes from increasing excessively, nodes can only be added by the novel mutation operator if all network nodes have already been used in the transition path. This is the only constraint for adding nodes. It follows that only one node can be added to an individual per generation. If nodes are deleted from an individual, multiple nodes can be deleted if they are unused. Therefore, adding nodes is less powerful than deleting nodes. This is motivated by the intention that the algorithm should find the smallest possible suitable networks and, if the complexity of the problem requires it, grow slowly. The imbalance is based on the mentioned principle of Occam's razor and counteracts overfitting [2]. Nevertheless, both parts of the operator are constrained by the uses of the current nodes in the transition path. To prevent a node from being inserted and then deleted again, nodes are only deleted if the difference between the nodes in the network and the nodes used in the transition path is greater than one. This ensures that always one node in the network is protected from deletion of the mutation.

[2]Based on the principle of Occam's razor, one should choose the simplest model that "explain" the data [Berthold et al., 2010].

The following simulation study in Table 4.7 shows the results again with the same parameters as in Section 4.2 on page 55. Twenty independent simulations were run, and the average results of the best and mean fitness were calculated. However, this time only a few nodes were initialized.

Table 4.7: Simulation Study of 20 different GNP runs for each combination of parameters on the Iris data. The Best Fitness and Mean Fitness are the mean values of the 20 runs.
Source: author's personal research results.

Judgment Nodes Processing Nodes	1 2	2 1	2 2	3 3	4 4
Best Fitness without Mutation Mean Fitness without Mutation	0.47 0.45	0.33 0.31	0.62 0.52	0.89 0.67	0.95 0.60
Best Fitness with Mutation Mean Fitness with Mutation	0.94 0.66	0.88 0.58	0.90 0.60	0.94 0.61	0.93 0.63
Difference Best Fitness	+0.47	+0.55	+0.28	+0.05	-0.02

Rows three and four show the results of the standard GNP. As expected, the standard GNP cannot produce good results when the number of nodes is small, especially when less than three processing nodes are initialized.

Rows five and six show the results with the new mutation operator, where nodes can now be added. No matter how many nodes were initialized, the average best fitness is around 0.90. The last row of the Table 4.7 shows the difference between the simulations with and without the mutation. It can be seen that the GNP can sufficiently adapt to the complexity of the Iris data problem with the addition of nodes. Even with a small number of initialized nodes, very good results can be obtained, and the outperformance compared to the standard GNP can be seen.

However, these are the results of the training data. Adding nodes and the ability of GNP to adapt more to the data also increases the risk of overfitting. Therefore, Table 4.8 shows the results of validations using a test data set. The Iris data set has now been split into 90% training data and 10% test data. The results of the Table 4.8 thus show the fitness values on the test data set with and without the new mutation operator. In addition, rows five and six show the average number of nodes of all individuals at the end of the evolution. Finally, row seven again shows the difference between the standard GNP and the new GNP.

Table 4.8: Simulation Study of 20 different GNP runs for each combination of parameters on the Iris data. The validation fitness is the mean value of the 20 independent runs.
Source: author's personal research results.

Judgment Nodes Processing Nodes	1 2	2 1	2 2	3 3	4 4
Validation without Mutation Validation with Mutation	0.47 0.86	0.35 0.87	0.55 0.86	0.79 0.87	0.88 0.85
Mean Number of Nodes without Mutation Mean Number of Nodes with Mutation	3.0 5.83	3.0 5.74	4.0 5.78	6.0 5.67	8.0 6.0
Difference Validation	+0.36	+0.52	+0.31	-0.02	-0.03

First, it can be seen that the results of the validations are similar to the best fitness values from Table 4.7. This leads to the conclusion that no overfitting has occurred. In particular, this is positive for the GNP with the new mutation operator. The GNP with the mutation also consistently achieved excellent validation results using the test data set. All validations were just below 0.9 accuracies. This again shows a clear positive difference between the two methods (last row). It is also remarkable that in all runs, the average number of nodes is between 5.67 and 6.0. This shows that there was no unrestricted growth in the number of nodes and that the number of nodes stayed within this range.

By constraining only nodes to be added when all nodes in the network are used, the bloat problem and overfitting of data could be prevented. Additionally, a GNP can now adapt to the complexity of a problem. This is also the case when the number of generations is increased. For example, after 1000 generations but keeping the other parameters the same, the following results were obtained:

Table 4.9: Simulation Study of 20 different GNP runs for each combination of parameters on the Iris data with 1000 generations. The validation fitness is the mean value of the 20 independent runs. Source: author's personal research results.

Judgment Nodes Processing Nodes	1 2	2 1	2 2	3 3	4 4
Best Fitness with Mutation	0.98	0.98	0.97	0.97	0.98
Validation with Mutation	0.90	0.82	0.88	0.91	0.89
Mean Number of Nodes with Mutation	6.52	6.60	6.93	6.08	6.96

The results are similar to those in the Table 4.8 after 100 generations.

Figure 4.10 shows the mean number of nodes of all individuals from one of the simulations with two initial judgment nodes and one processing node after 100 generations and no overfitting occurs. One reason that no overfitting occurs when adding a node is that the added node integrates into the network structure and connects to the network with an edge. An added node is, therefore, part of the transition path and affects the other nodes. In a tree structure, this would be different because added nodes could extend the depth of a tree, and by going back to the root node, only partial paths could be affected.

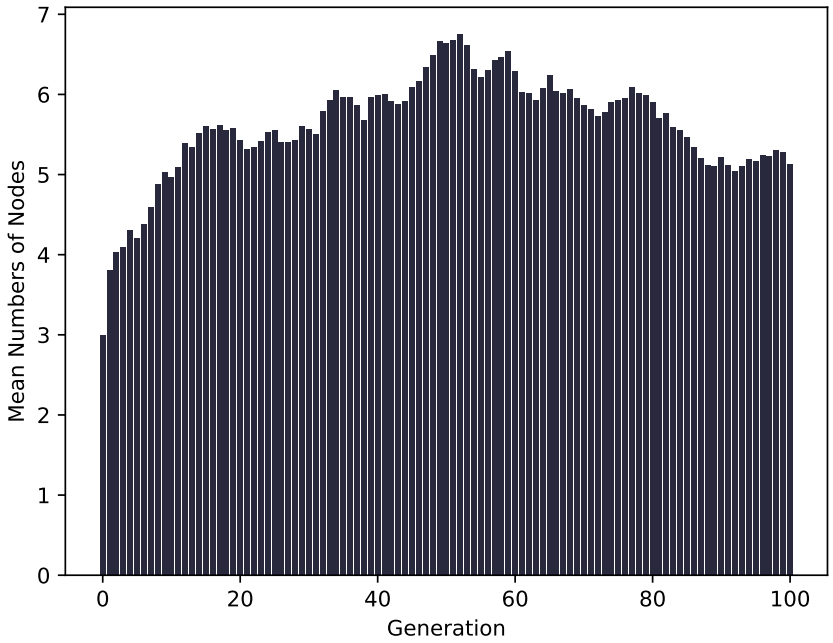

Figure 4.10: Growing Mean Node Numbers Iris Data Example
Source: own illustration

First, the average number of nodes increases from the three initialized nodes in generation zero to about 5.6 nodes around generation 100. Then, the average number of nodes decreases as more nodes are deleted in the following generations. After that, the maximum average number of nodes is reached at almost seven. Finally, the average number of nodes settles at around five nodes, which is a similar number of nodes observed in Table 4.8 after 100 generations.

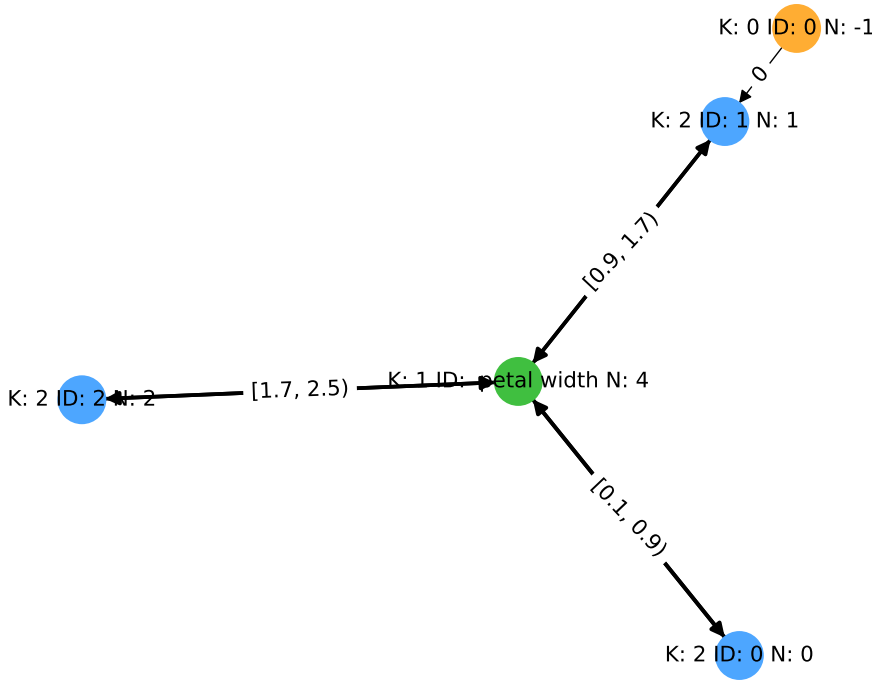

Figure 4.11: Used node of a GNP on the Iris data.
Source: own illustration

At the end of evolution, the best individual ends with the transition path in Figure 4.11. This is almost the same representation of the transition path as in Figure 2.5 on page 19. The only difference is in the first edge since the start node points to a processing node. The resulting transition path in Figure 4.11 is of interest because the feature Petal width again classifies the iris data. Petal width is the fourth feature in the iris data set and thus cannot be selected in a standard GNP if only two judgment nodes are initialized. However, by randomly adding a node, other judgment node functions and different features can now be included in the GNP.

Consequently, the GNP can select suitable features through the evolutionary process. This inherent form of feature selection is especially useful for large data setss with many features and will be discussed in more detail in the experimental part of this thesis. In addition, the individual ended up with the three processing nodes necessary to classify all species, although only two processing nodes were initialized. The other important aspect is the possibility of increasing the number of edges of nodes. For example, a judgment node could have at most two outgoing edges in the initialized network with three nodes. Adding a judgment node to the network can now have more outgoing edges. So node 4 in Figure 4.11 has three outgoing edges.

A video "GNPdy (Growing) - Evolution Best Individual Iris Dataset" of the entire evolution can be found on the project homepage in the section *"Research"*. The adding of the node with the features Petal width, can be observed in generation 14:

https://fabiankoehnke.github.io/gnp/hp/#research

4.2.4 Conclusion

It has been shown that the new mutation operator improves the adaptability of a GNP. By growing and shrinking, a GNP can now precisely fit the complexity of given tasks. Despite the possibility of generating more complex network structures, overfitting and the bloat problem could be counteracted by imposing a few constraints on the operator. The results of the simulation studies have shown that the GNP with the new mutation operator outperforms the standard GNP. Better results in fitness values and validations could be achieved. A great advantage of the new operator is that no hyperparameter is needed. This is shown by the following algorithm 3, which describes the new mutation operator and summarizes the previous topics.

Algorithm 3: Mutation Operation Add/Delete Nodes

```
Function mutation_add_delete_nodes(population):
    for i ∈ {1, ..., |population|} do
        individual ← population[i];
        nn ← individual.k /*number of nodes in gene structure */;
        un ← individual.used_nodes /* array of used nodes */;
        decision ← random element of {"add", "delete"};
        for n ∈ {1, ..., nn} do
            node ← indidvidual.gene_structure[n];
            if decision == "delete" & nn − un > 1 & node ∉ un then
                individual.gene_structure ← individual.gene_structure \ node;
                nn_new ← individual.k − 1 /* set nn new */;
                /*now adapt the connections */ ;
                /* for all nodes in gene structure */;
                for n2 ∈ {1, ..., nn_new} do
                    node ← indidvidual.gene_structure[n2];
                    /* for all connections in node */;
                    for k ∈ {1, ..., length(node.connection_gene)} do
                        connection ← node.connection_gene[k];
                        if k == n then
                            /*connection points to the deleted node*/;
                            new_con ← random number from U([0, nn_new]);
                            node.connection_gene[k] = new_con;
                        end
                        else if k > n then
                            /*adapting pointers*/;
                            new_con ← k − 1;
                            node.connection_gene[k] = new_con;
                        end
                    end
                end
            end
            else if un >= nn then
                no ← new random node object
                 individual.gene_structure ← individual.gene_structure ∪ no;
            end
        end
    end
```

4.3 Interval Mutation

This section covers an additional novel mutation operator that has been developed. In the previous section, it was shown that adding nodes can lead to a more extensive expression ability of a GNP. This was achieved by adding nodes resulting in a larger spectrum of node functions and a changed interval width of the judgment functions. As a result, the gene pool of the whole population and, thus, the search space could be increased successfully.

Nevertheless, the gene pool is still limited because the interval boundaries are restricted in their granularity. For example, in a network consisting of 10 nodes, a judgment node can divide the data into a maximum of 9 equally sized intervals. Although this number could increase as the network grows, if a network size adjusts to a specific complexity, the gene pool is again limited.

The mutation operator presented in this section can change the interval boundaries determined by the number of edges. This is done by sampling a certain "jitter value" from a normal distribution and shifting the interval boundaries. Consequently, this mutation operator works only on the edges of the judgment nodes and, in particular, on judgment node functions assigned to a numeric variable.

To illustrate this, the example from Section 4.1, illustrated in Figure 4.4, is used again. In this example, only two processing nodes and one judgment node were initialized to handle the Iris data. The insufficient number of nodes caused poor results after training the model. Adding nodes could remedy the limitation by providing new judgment nodes containing more edges with the ability to classify the data more granularly.

Here, the data classification by the interval boundaries is considered in particular. In Figure 4.4, the red line shows that the model divides the data according to the feature Petal width at point 1.3. Since only two processing nodes were initialized, the maximum accuracy of the model can only be $\frac{2}{3}$. As mentioned above, this separation of the data leads to even worse results when the species Setosa and Versicolor are classified. This is because the left part of the red separation line misclassifies a large part of the species Versicolor. Instead of an accuracy of $\frac{2}{3}$ the model achieves an accuracy of $\frac{15}{72} = 0.48$. Only an interval boundary between 0.6 and 1.0 would suitably distinguish the two species from each other. As a reminder, the separation of the Setosa and Versicolor species at a petal width value of 0.9 was the successful interval boundary from the learned model in Figure 4.11.

Figure 4.12 shows a trained GNP that illustrates a resulting transition path of this example. As in the example from Section 4.1, two processing nodes and one judgment node were initialized. The processing nodes refer to the species Setosa and Versicolor, and the judgment node deals with the feature Petal width. The new mutation, which can add nodes, was deliberately disabled to keep the number of nodes the same.

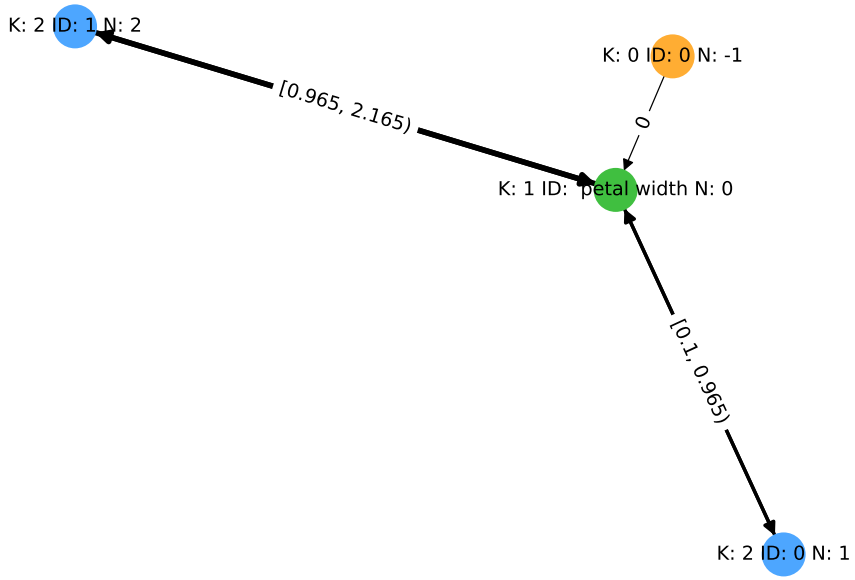

Figure 4.12: Iris Data Classification less Nodes with Interval Mutation
Source: own illustration

It can be seen that the mutation has shifted the boundaries of the intervals (compare Figure 4.4). As a result, the required interval boundary between 0.6 and 1.0 was successfully learned with a value of 0.965. The individual's fitness reaches the maximum possible accuracy of $\frac{2}{3}$.

Notice that all "inner" interval boundaries are shifted. This means all boundaries are shifted at the same distance except the minimum and maximum interval boundaries. If these two interval boundaries were also shifted, not all samples would be within the interval boundaries, and the data could not be fully trained. As a result, the interval boundaries can never reach values below the feature's minimum or exceed the maximum. The mutation of the interval boundaries proceeds as follows:

1. Select each judgment node within a numerical judgment function given probability p_m.
2. Draw a numerical value from a normal distribution where the mean (μ) of the distribution is the mean of the corresponding feature multiplied by a given factor (hyperparameter), and the standard deviation (σ) of the distribution is ~ 1.
3. Decide randomly to add or subtract the drawn value from the interval boundaries.

A hyperparameter is necessary to draw a suitable value from the normal distribution. This parameter determines the size of the mean μ. The calculated mean value of the handled feature, which enters the normal distribution, is multiplied by

this parameter. This allows handling the strength of the interval shift.

The reason that the mutation affects all edges of a judgment node and shifts them equally is because of the expressiveness of a GNP. Another way would be to change each edge individually. In this case, intervals could be changed by such a great distance that they would run past each other, and the order of the intervals would no longer be correct. This should to be handled in the algorithm. With the standard mutation from Section 2.5, however, the GNP can already decrease and increase interval sizes and the distance between the intervals is not fixed. With the additional mutation of adding and deleting nodes, the spectrum of edges increases further. If each edge would be treated individually by the operator described here, the danger of overfitting would increase. Thus, the expressiveness of a GNP is mainly due to the standard mutation and the newly introduced mutation from Section 4.2. It becomes clear that the example given above is only for illustration. The strong shifting of the interval boundaries, which results in values far outside the features, is not necessarily beneficial.

Instead, the mutation operator introduced here should help to expand the still-limited gene pool and perform granular shifts of the interval boundaries. This results from the experience that a slight shift can additionally lead to better results when complex networks are trained. Iris data, however, is not complex enough to reach better final results obtained in Section 4.2. But, during the evolution, an improvement caused by the shifting intervals can be seen. The Figures 4.13 and 4.14 show two individuals, each with one judgment node and four processing nodes. An improvement of the fitness from 0.86 to 0.9 could be reached by shifting the interval boundary of 0.143.

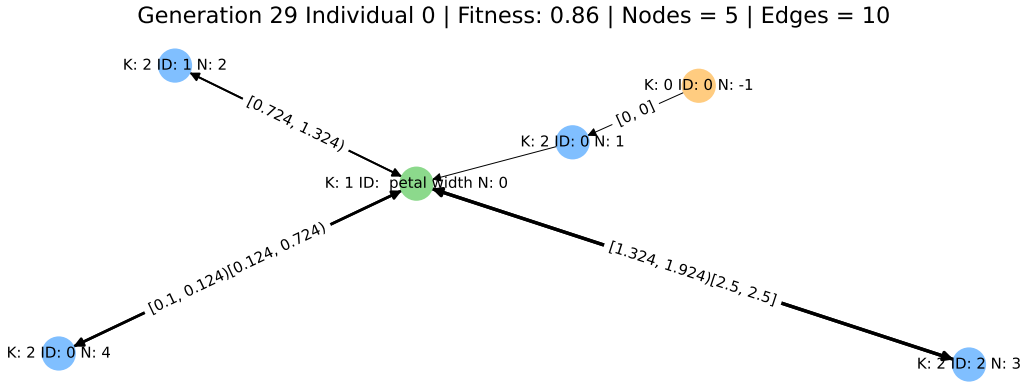

Figure 4.13: Iris Data Interval Shifting Fitness 0.79 in Generation 7.
Source: own illustration

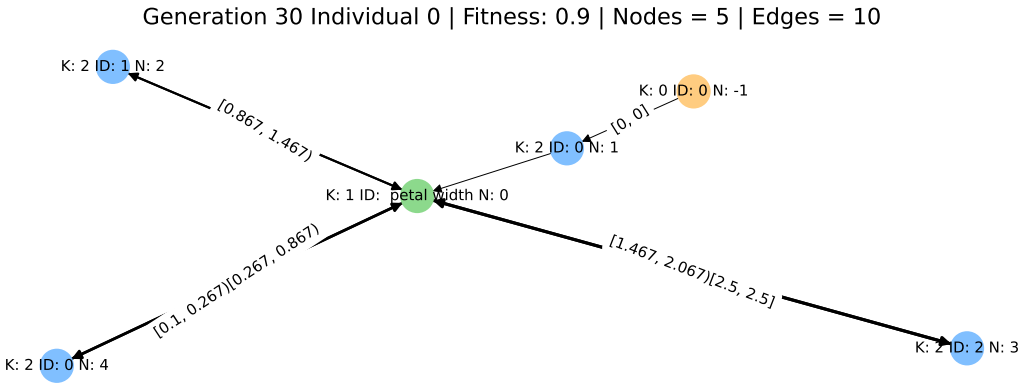

Figure 4.14: Iris Data Interval Shifting Fitness 0.79 in Generation 8.
Source: own illustration

In addition, Figure 4.15 shows how the shift of the intervals with the already known histogram of the Iris data related to the feature petal width. The grey dashed lines are the interval boundaries of the individual from generation 29 (Figure 4.13), and the shifted interval boundaries are shown in red.

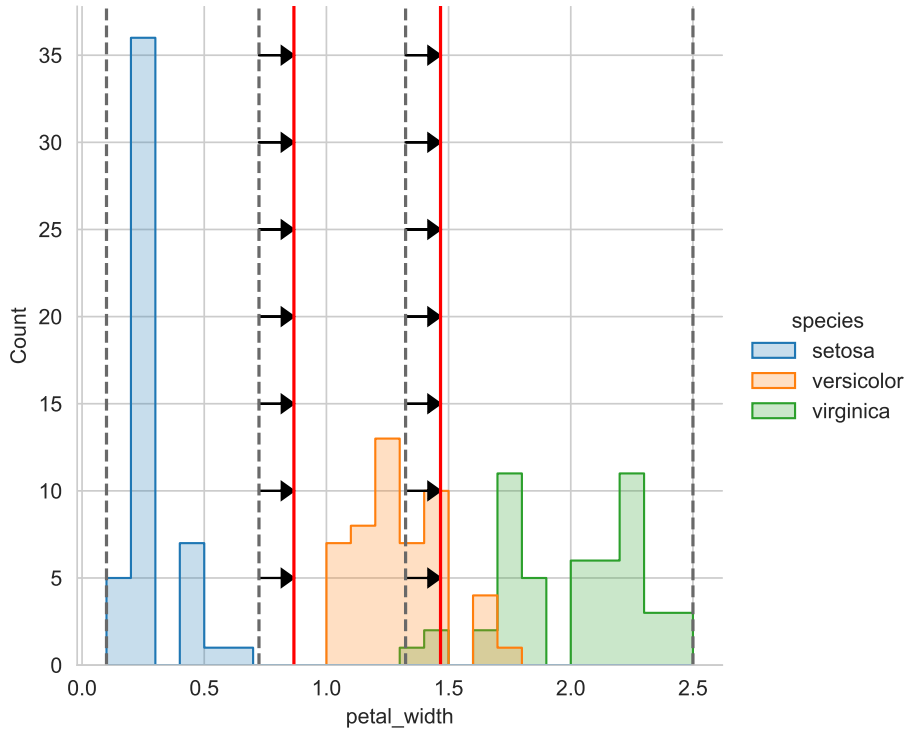

Figure 4.15: Iris Data Classification Interval Shifting
Source: own illustration

To conclude, training a GNP with the standard mutations plus the mutation adding and deleting nodes is recommended. On top, an precise fit to the data can be achieved by shifting the interval boundaries to a small degree. A shift of single edges and not all edges to the same degree could be a topic of future research.

The following algorithm 4 describes the mutation operator in detail. Notice that the error of a node is the value that is added to the interval boundaries and is initialized with zero.

Algorithm 4: Interval Mutation

Function `mutation_intervals`($individual$, p_m, $factor$, $mean\_feature$)**:**
  $N \leftarrow length(individual.gene\_structure)$;
  for $n \in \{1, ..., N\}$ **do**
    $node \leftarrow individual.gene\_structure[n]$;
    $u \leftarrow$ random number according to $U([0, 1])$;
    if $u \leq p_m$ & $node.K == 1$ **then**
      /*node is judgment node*/;
      $value \leftarrow$ random number from $N(mean\_feature \cdot factor, 1)$;
      $sign \leftarrow$ random number from $\{-1, 1\}$;
      $node.error \leftarrow node.error + values \cdot sign$;
    end
  end

Chapter 5

Experiments

This chapter presents the training of a GNP with the new mutation operators using a more complex data set than the Iris data. Corresponding to the topic of the master's thesis, financial market data from the Yahoo Finance website was used for the simulation. First, the objective and the resulting fitness function are described. After that, the data is examined in detail, and then the results are evaluated.

5.1 Fitness/ Objective Function

The main objective of the master thesis is to find a method that can beat the market with active portfolio optimization. Accordingly, a GNP should learn a trading strategy from given data to perform better than the market. In addition, different stocks should be optimally weighted for portfolio optimization. A different weighting of shares can be achieved by initializing several start nodes (see Section 3.1). For this purpose, the fitness function is coded so that different budgets are allocated to the stocks. The fitness function for training the GNP is defined as:

$$Fitness(n) = \sum_{s \in S} Profit(s, n) \tag{5.1}$$

where s is the current stock and n the current generation.

Profit is simply the difference between all sales and purchases during the trading period with T time entries:

$$Profit(s, n) = \sum_{t}^{T} Sell(s, n, t) - \sum_{t}^{T} Buy(s, n, t) \tag{5.2}$$

The profitability of each share can be calculated based on the profit:

$$Profitability(s, n) = \frac{Profit(s, n)}{Initial(s, n)} \tag{5.3}$$

where $Initial(s, n)$ is the initial budget of stock s in generation n.

The initial budget of each stock in the first generation is calculated as:

$$Initial(s, 1) = \frac{Initial(0)}{|S|} \tag{5.4}$$

After that, the initial budget of each stock is weighted differently over the generations and is determined by profitability:

$$Initial(b, n+1) = \frac{exp(Profitability(s,n))}{\sum_{s \in S} exp(Profitability(s,n))} Initial(0) \tag{5.5}$$

This coding of the stocks' fitness and weighting is based on the approach presented in the paper [Chen, Mabu, and Hirasawa, 2010]. In addition, a temperature parameter was used in that paper to control the strength of the different weightings. An extra parameter is not used in this thesis. Nevertheless, each share is weighted differently in the portfolio by the equation 5.5. This equation corresponds to the well-known Softmax function. The Softmax function makes it possible to map the $profitability \in (-\infty, \infty)$ of all stocks into $[0, 1]$ and thus calculate appropriate stock weights.

5.2 Encoding Target Variable

The actual target variable is the return computed from the stock prices. Based on the returns of the stocks, the profit (fitness) of each stock can be calculated. As already indicated in equation 5.2, the profit consists of the purchases and sales of the stock. The processing node function is therefore encoded as:

- 0 = buy
- 1 = sell

The signal of the processing node function is either *buy* or *sell*, and a decision to hold a stock is not explicitly defined. However, this is not needed because if a share was already bought and a buy signal occurs again, the stock is held. Therefore, a signal to hold the share is implicitly possible. Furthermore, an additional purchase is not necessary since the weighting of the shares is done via the objective function. Conversely, a share cannot be bought for the portfolio if a sell signal occurs and the share is not in the portfolio. This implies that shares cannot be held at all.

In summary, the following rules are applied:

- processing function = 0 and stock not in portfolio → buy
- processing function = 0 and stock already in portfolio → nothing (hold)
- processing function = 1 and stock not in portfolio → nothing
- processing function = 1 and stock in portfolio → sell

5.3 Data Understanding

The financial market data set analyzed here consists of 25 stocks between 02.01.2019 and 24.10.2022.

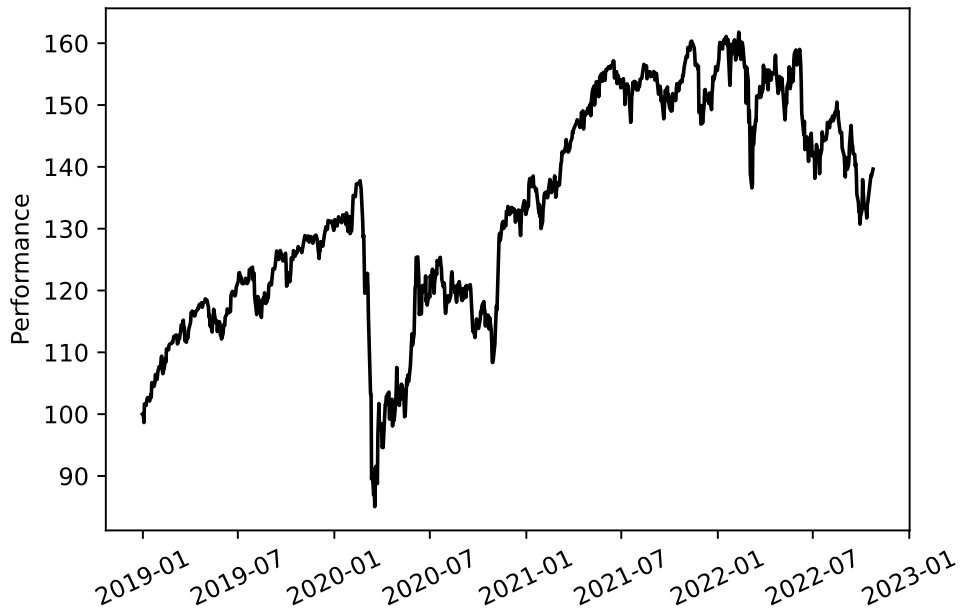

Figure 5.1: Stock Data History
Source: own illustration

Figure 5.1 shows the history as the mean of all stocks. The performance during the period will be the buy-and-hold benchmark. Despite the fact that the analyzed period is up-to-date, the history also shows very different market phases. First, a strongly positive market trend can be seen from the starting point of 100 to around 140 at the beginning of 2020. Then a massive crash follows because of the corona pandemic. From the low, at 85, another positive market trend follows. Finally, the stocks showed a sideways movement with sharp market corrections, e.g., because of the war in Ukraine. Therefore, the training of the GNP contains different market phases and has to deal with positive trends and elsewhere with market crashes.

All stocks are part of the main index of the Eurozone, the so-called EURO STOXX 50. For the selection of the stocks, all possible sectors were included. The following Table 5.1 shows the 25 shares with their sector assignment.

Table 5.1: Shares selected for Analysis
Source: own illustration

Name	Ticker	Sector
Deutsche Telekom AG	DTE.DE	Communication Services
Deutsche Bank Aktiengesellschaft	DBK.DE	Financial Services
Linde plc	LIN.DE	Basic Materials
BASF SE	BAS.DE	Basic Materials
Bayer Aktiengesellschaft	BAYN.DE	Healthcare
Eni S.p.A.	ENI.MI	Energy
Iberdrola, S.A.	IBE.MC	Utilities
Orange S.A.	ORA.PA	Communication Services
Danone S.A.	BN.PA	Consumer Defensive
Safran SA	SAF.PA	Industrials
Fresenius SE & Co. KGaA	FRE.DE	Healthcare
Koninklijke Philips N.V.	PHIA.AS	Healthcare
LVMH Moët Hennessy	MC.PA	Consumer Cyclical
Sanofi	SAN.PA	Healthcare
L'Air Liquide S.A.	AI.PA	Basic Materials
Anheuser-Busch InBev SA/NV	ABI.BR	Consumer Defensive
Airbus SE	AIR.PA	Industrials
Engie SA	ENGI.PA	Utilities
Allianz SE	ALV.DE	Financial Services
ASML Holding N.V.	ASML.AS	Technology
Enel SpA	ENEL.MI	Utilities
Carrefour SA	CA.PA	Consumer Defensive
Schneider Electric S.E.	SU.PA	Industrials
ING Groep N.V.	INGA.AS	Financial Services
Banco Bilbao Vizcaya Argentaria, S.A.	BBVA.MC	Financial Services

For the training of the GNP, the data were grouped according to Section 3.3, where a starting node was initialized for each stock. Thus, each stock has its own transition path. Due to the grouping of the data, each stock has its own minimum and maximum feature values. Therefore, the transition path with the corresponding interval boundaries will be examined in detail for chosen shares below.

The following data was fetched from Yahoo Finance: the share price, outstanding shares of the company, the traded daily volume, the earnings, the turnover, and the gearing of the share. The stock return and various indicators of financial market analysis were calculated from the stock price. All features are listed with the distinction between fundamental and technical analysis in the following Table 5.2.

Table 5.2: Outline of the Features
Source: own illustration

Feature	Description	Category	Calculation
Ticker	Stock ticker to group the data (multiple start nodes)	-	-
VolumeRatio	Traded stock volume on each day	technical	$\frac{volume}{so}$
DiffMA5 DiffMA10 DiffMA20 DiffMA50	Percentage difference of the share price to the n days moving average	technical	$\frac{c-ma_n}{ma_n}$
SD5 SD10 SD20 SD50	Standard deviation calculated from the last N days	technical	$\sqrt{\frac{1}{N}\sum_i^N (x_i-\mu)^2}$
momRel5 momRel10 momRel20	Relative momentum indicator calculated from the last n days	technical	$\frac{c-c_n}{c_n}$
RSI5 RSI14 RSI20	Relative strength indicator calculated from the last n days	technical	$100-\frac{100}{1+RS}$ $where, RS=\frac{acp}{acn}$
KGV	historical Price-earnings ratio	fundamental	$\frac{so \cdot c}{earnings}$
KUV	Historical price-sales ratio	fundamental	$\frac{so \cdot c}{sales}$
DeptRatio	Historical debt-equity ratio	fundamental	$\frac{total\ liability}{assets}$

td = daily turnover
so = outstanding shares
ma = moving average
c = closing price
m = momentum
acp = average c with positive changes
acn = average c with negative changes

The Table 5.3 shows an outline of the statistical key values of the features and target variable *return*. First, it should be noted that the data do not have to be normalized. In contrast to an artificial neural network, numerical values of different sizes do not influence the model's prediction. This is because the decision-making of GNP results from the overall fitness of individuals, and particular nodes and edges do not change according to their judgment function results. Essential characteristics of the table are summarised below:

- The average daily return is 0.05%. On average, the shares move slightly positively, as seen in Figure 5.1. The most significant movements of a stock downwards or upwards are -22.87% and 20.93%.

- On average, 0.29% of the capital of a share is traded on a single day. The maximum was 4.44%.
- The four columns of the differences to the moving average ($DiffMA_n$) increase in their minimum and maximum values as the period for calculating the average is longer. This was to be expected since, with more data, the average is smoothed more.
- The opposite tendency can be observed for the standard deviation. The more data used for the calculation, the smaller the standard deviation because large movements of the return will be averaged out more.
- The RSI indicator is a so-called oscillator and can assume values between 0 and 100. Typically, values above 70 are considered a stock overbought, and values below 30 are considered oversold [Murphy, 1999]. On average, this indicator is usually around 50.
- Concerning the three fundamental indicators, it should be noted that the historical earnings per share ratio can also show negative values if the stock company recorded losses during the period. Furthermore, the earnings-to-turnover ratio assumes higher values than the price-earnings ratio because turnover will usually exceed earnings. Finally, the leverage ratio peaks at 95.35% and is at least 38.86%.

Table 5.3: Statistical Outline of the Features
Source: own illustration

	Return	VolumeRatio	DiffMA5	DiffMA10	DiffMA20	DiffMA50	SD5	SD10	SD20	SD50
count	25377	25377	25377	25377	25377	25377	25377	25377	25377	25377
mean	0.0005	0.0029	0.0010	0.0017	0.0030	0.0067	0.0147	0.0162	0.0172	0.0180
std	0.0204	0.0022	0.0307	0.0407	0.0555	0.0822	0.0107	0.0104	0.0100	0.0095
min	-0.2287	0.0000	-0.3616	-0.4651	-0.5448	-0.6009	0.0010	0.0017	0.0043	0.0054
25%	-0.0085	0.0016	-0.0121	-0.0156	-0.0213	-0.0330	0.0079	0.0098	0.0110	0.0121
50%	0.0004	0.0023	0.0023	0.0038	0.0063	0.0126	0.0121	0.0137	0.0147	0.0156
75%	0.0097	0.0034	0.0155	0.0217	0.0317	0.0532	0.0181	0.0194	0.0202	0.0209
max	0.2093	0.0444	0.3940	0.3205	0.4123	0.5080	0.1750	0.1538	0.1231	0.1320

	momRel5	momRel10	momRel20	RSI5	RSI14	RSI20	KGV	KUV	DeptRatio
count	25377	25377	25377	25377	25377	25377	25377	25377	25377
mean	0.0023	0.0043	0.0081	51.2721	50.5082	50.2481	29.5506	1.9927	0.6867
std	0.0463	0.0651	0.0914	22.9369	12.2153	10.1258	95.5301	2.4378	0.1644
min	-0.4836	-0.5814	-0.6359	0.0000	0.5479	2.8013	-80.4137	0.1229	0.3886
25%	-0.0183	-0.0254	-0.0363	35.9208	42.4336	43.5430	8.6948	0.7104	0.5663
50%	0.0033	0.0063	0.0103	51.4510	50.6558	50.3929	17.3690	1.4156	0.6558
75%	0.0244	0.0363	0.0557	66.7276	58.8178	57.1429	28.4476	2.3610	0.7790
max	0.6562	0.5476	0.7313	100.0000	100.0000	93.6739	1145.2039	22.8426	0.9535

5.4 Results

This section presents the results obtained by training the GNP on the stock data. For a meaningful validation, 30 consecutive test periods were determined. The validation procedure was the same as the cross-validation presented in Section 3.1. The test periods are consecutive samples that were shifted by the size of the test period. The data was divided into 80% training data and 20% test data. The 20% test data size corresponded to 22 trading days and approximately one trading month. After this period, a new GNP was trained. Each training of a GNP was initialized with a starting capital of 100. The profit achieved can thus be easily converted into a percentage result. To keep the results as applicable as possible for practice, a fee of 0.05% was charged for each trade in a stock. The evolution of the GNP was simulated with the following parameter:

- Fitness: Profit (see Section 5.1)
- Generations: 400
- Population size: 100
- Judgment Nodes: 0
- Processing Nodes: 2
- Mutation Probability: 0.15
- Crossover Probability: 0.15
- Tournament Size: 2
- Elitism: 4
- Time Delay on Nodes: 1
- Time Delay on Edges: 0
- Maximal time Units: 10
- Mutation Add/Delete: True
- Mutation Interval: 0.05
- Mutation Interval Mean Factor: 0.05
- Termination after N Generations without Improvement: 50

Figure 5.2 shows the results of the 30 validations compared to a simple buy-and-hold strategy. The buy-and-hold strategy often is used as a benchmark in the financial industry. All shares are bought at the start date and held until the end date. Initially, the main objective of this work was to find a strategy that could outperform this benchmark. The buy-and-hold strategy is shown in orange bars, and the fitness of the best individual is shown in grey with dots. This makes it easy to compare the results of the GNP and buy-and-hold strategies. In addition, the average of both strategies is represented by a horizontal line.

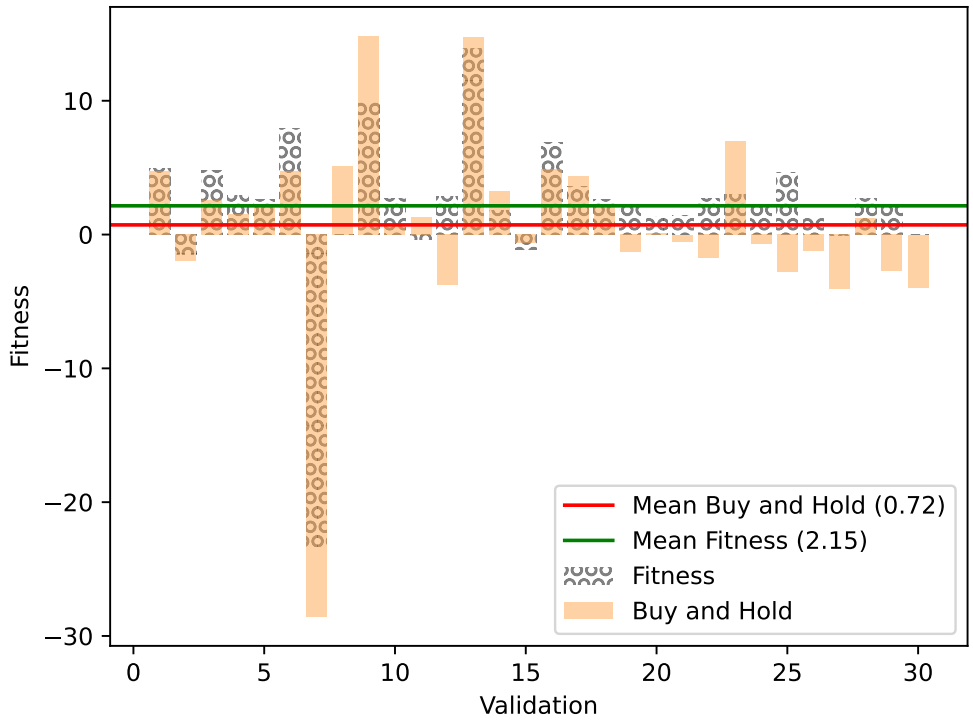

Figure 5.2: Results Cross-Validation
Source: own illustration

The first thing to note is that the average of the achieved validations of the GNP outperforms the buy-and-hold strategy. The GNP achieved better results, with an average of 2.15 compared to 0.72. The model outperformed by an average of 1.43 in each validation (trading month). Furthermore, out of the 30 validations, the model outperformed in $22/30 \approx 0.7333 = 73.33\%$. It is also worth noting that the model only achieved negative results in 5 of the 13 cases in which the benchmark declined. The most significant negative result occurred in validation 7 when the Corona crisis caused stocks to collapse in value. However, the loss of the model was lower than that of the buy-and-hold strategy. In the two largest positive market movements the model was unable to achieve the benchmark's performance. Nevertheless, an impressive outperformance could be achieved, especially in the last validations where the market often showed negative results, whereas the GNP generated positive performance frequently.

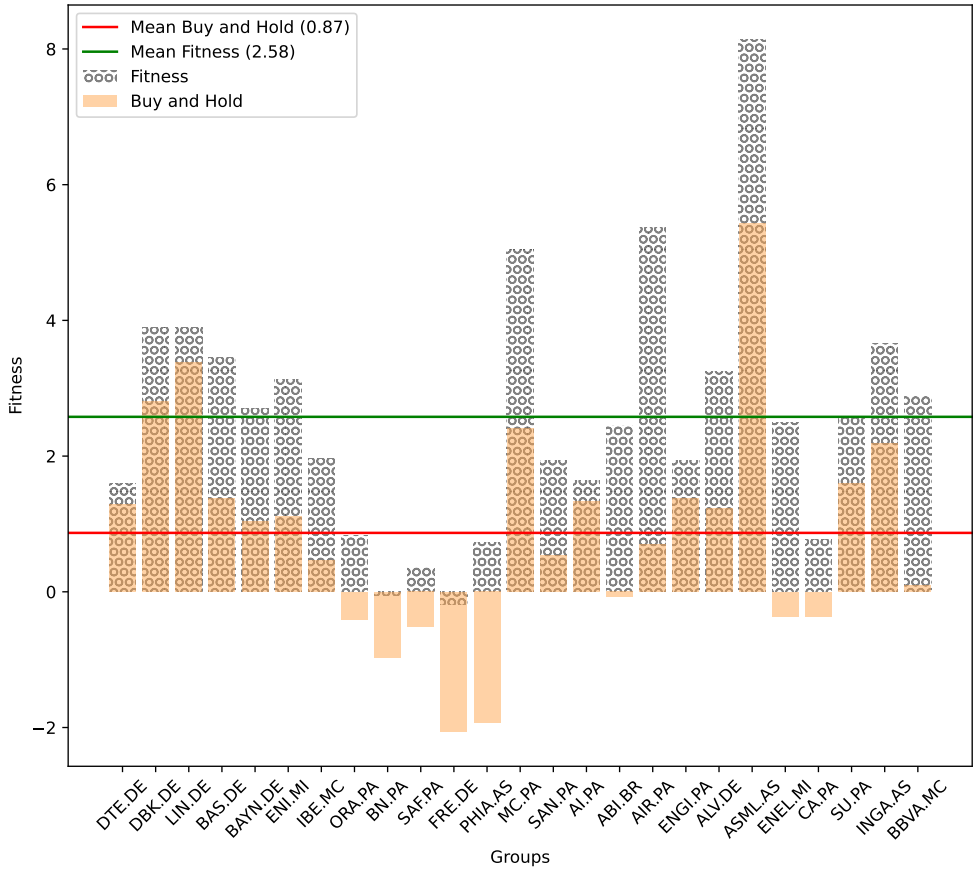

Figure 5.3: Results Each Start Node
Source: own illustration

Figure 5.3 shows each stock's complete validations' results. The results, therefore, reflect the entire validation period. On the X-axis, the Ticker entered from Table 5.1 can be seen and assigned to the corresponding stocks. It is remarkable that after all validations of the GNP, better results could be achieved for *each* stock compared to a buy-and-hold strategy. On average, the models outperformed the buy-and-hold strategy by $2.58 - 0.87 = 1.71$.

The independent validations outperformed the benchmark on average, as can be seen in the two previous Figures 5.2 and 5.3. In practice, however, a conditional validation of the results is especially important. For example, an investor would use his profit in the following period to build a new portfolio. Conversely, an investor cannot necessarily build a portfolio with an initial capital of 100 if a loss has already been accumulated beforehand. Therefore, it is beneficial that the periods of validation are consecutive and cumulative performance can be calculated using the geometric mean. Figure 5.4 shows a graph of the performance of the capital with a dependent (cumulative) and independent (summed) observation.

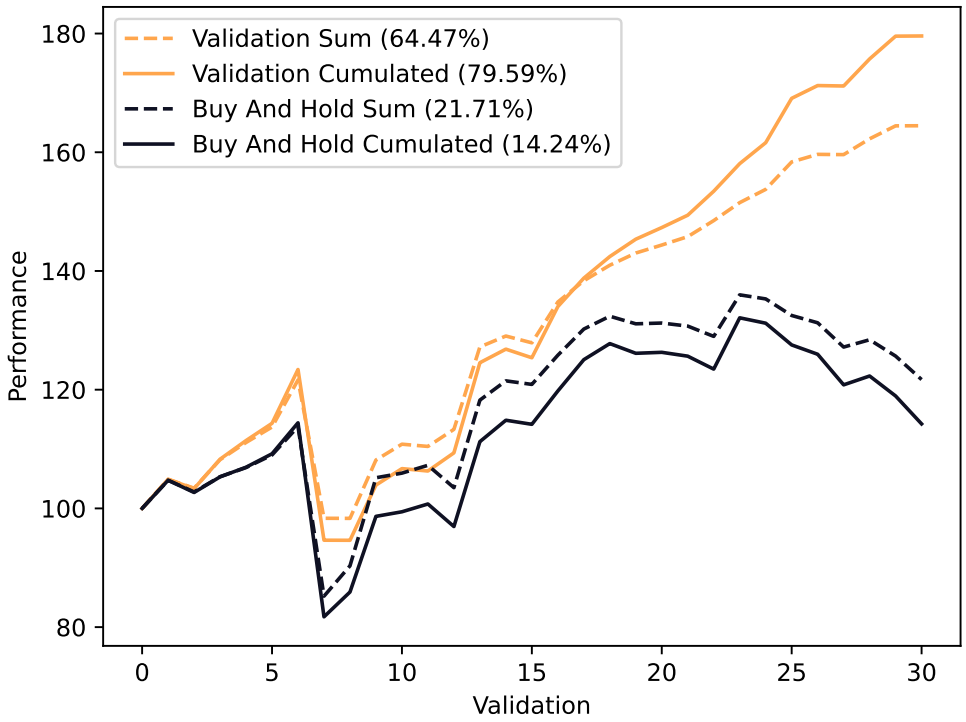

Figure 5.4: Results Cumulated
Source: own illustration

Overall, both curves of the GNP show that a significant outperformance could be achieved compared to the benchmark. Moreover, the cumulative approach is even more outstanding than the simple summation. Over the whole validation period, 65.35%, respectively 42.76% outperformance could be reached compared to the benchmark.

5.4.1 Feature Selection

An essential aspect of the novel mutation operator, which can increase and decrease the size of the network, is the selection of features. From the initial parameters of the simulation, it could be seen that the individuals were initialized with only two processing nodes. In the first generation, there are no judgment nodes in the networks, and the ability to use a feature is impossible. However, by adding nodes, the networks can develop into complex individuals using a wide range of features. The 18 features shown in the Table 5.2 were available for expansion. A significant advantage of the new mutation operator is that a GNP can select suitable features and adjust to the complexity of a given task. This was the subject of discussion in Section 4.2.

Figure 5.5 shows the used features of the best individuals from all segments of training. Each bar shows the number of features used during the transition path. It can be seen that many of the features were used in the network at the end of the training. Furthermore, there is a tendency for variables with a shorter time horizon to be used less frequently. This is particularly evident for the variable RSI. RSI5 was rarely used, whereas RSI20 was used about 2000 times. For DiffMA and momRel, the variables with calculations of 5 days in the past were also used least during the

transition path. This leads to the hypothesis that a five-day period is too short to achieve a reasonable forecast of the stocks. The most frequently used features were the KGV ratio and KUV in the transition paths. However, it cannot be seen that some features were used frequently and some not at all.

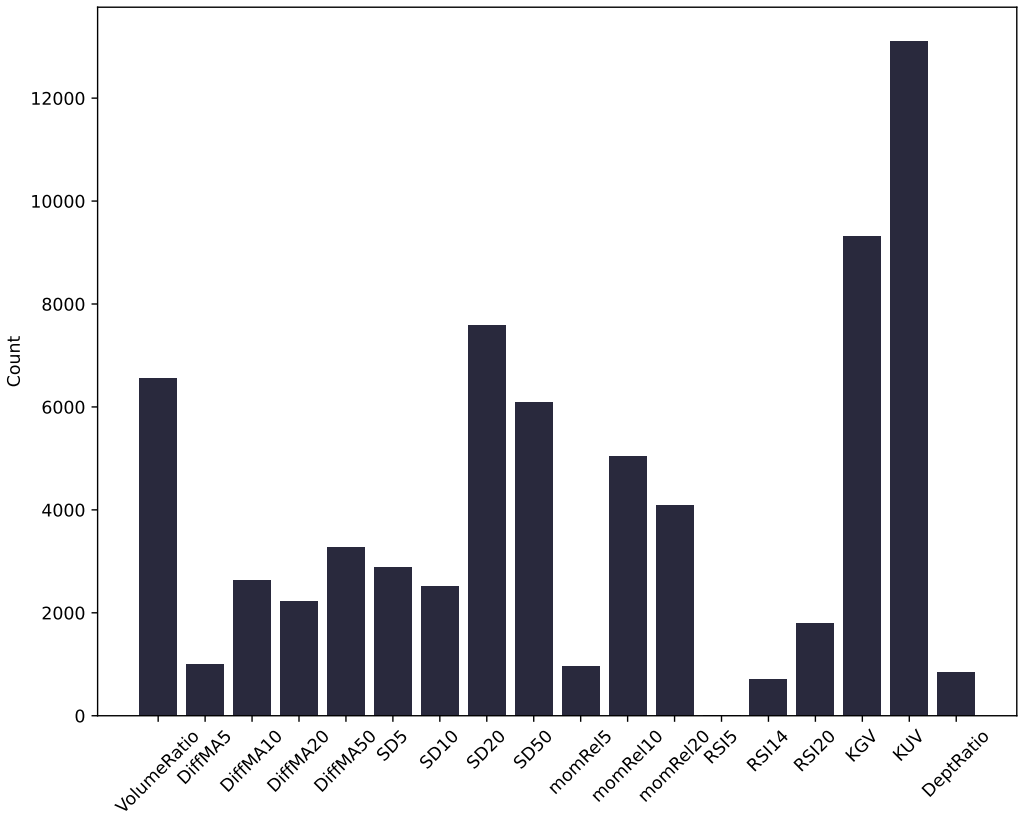

Figure 5.5: Counted Nodes in Transition all CV's
Source: own illustration

A concentration on particular features could be observed in individual training periods. Figure 5.6 shows the number of features used in certain generations of the first training period. In total, 219 generations were completed in this training. After that, the training was stopped because the best individual had no improvement for 50 generations.

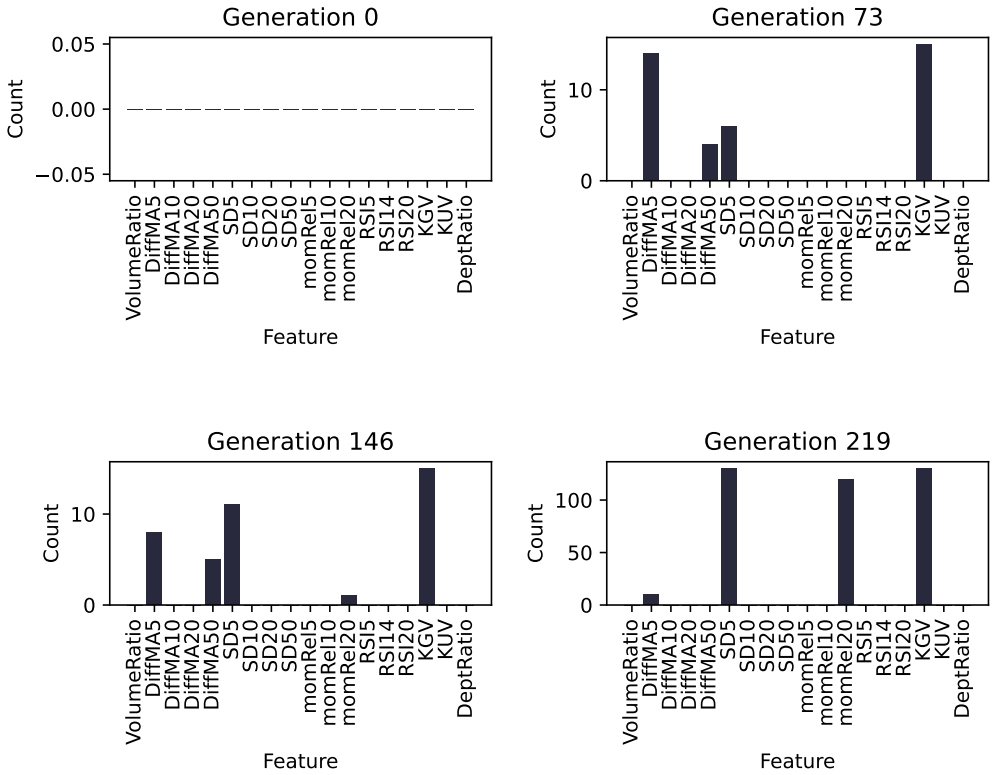

Figure 5.6: Counted Nodes in Transition (1 training period)
Source: own illustration

In generation 0, no features were used because no judgment nodes were initialized. After 73 generations, four features have already been selected by judgment nodes. Over the generations, this number changes, and other features are applied. In the end, the best individual used the four features DiffMA5, SD5, momRel20, and KGV. In parts of the training, the GNP focused on particular features and did not use others. This is an excellent example of the selection of features by a GNP that was initialized with only two processing nodes but could expand the search space through the new mutation operator significantly.

5.4.2 Transition Path Example

To illustrate a strategy learned by the GNP, figure 5.7 shows the best individual of validation 2. As an example, the transition path of the start node of the assigned stock ENEL SpA (ENEL.MI) is examined.

The transition path starts at the judgment node with node number 7, which describes the standard deviation calculated from the last five days (SD5). This node is connected to nodes 8, 0, 3, and 2. All these nodes are processing nodes. A sell signal is generated if the standard deviation is in the interval $[0.011, 0.013)$. If the stock is not currently present in the portfolio, nothing happens. For all other interval boundaries, a buy signal is generated.

A large part of the transition occurred between the two nodes with the numbers 0 and 10. This can be seen by the thickness of the drawn edge. Since node 0 is a processing node with a buy signal, the stock is held if it was bought through node 0,

and afterward, the KUV is in the drawn intervals on the edge of node 10. If the KUV falls within the interval $[0.579, 0.611)$, then a buy signal is also generated by node 3. Node 3 again points to node 0, which can be interpreted as a strong signal because the stock is held for at least two days if the KUV is in the interval [0.579,0.611). This is interesting because the interval expresses the smallest KUV, and in the financial market analysis, a small KUV also indicates favorable stocks.

If the KUV is in the intervals $[0.643, 0.675)$ or $[0.675, 0.707)$, then the indicators DiffMA20 and SD50, respectively, are examined. Note that node 4 has only one outgoing edge in the transition path. However, in the genotype, this node (for a judgment node usually) has several outgoing edges.

Another interesting node is node 9, which owns the DiffMA20 as a judgment node function. This node is connected to nodes 8, 7, and 0. If the stock price is lower or slightly higher than the 20-day average, node 8 is activated, which gives a sell signal. If the stock price distance is between $[0.013, 0.034)$, the standard deviation of the last five days is calculated (node 7). If the stock price is far away from the average, a buy signal is generated (node 0). That means, far above the average, the model remains in buy mode. The standard deviation is additionally consulted in the middle distance, and in the lower range, a sell signal is delivered.

This illustration showed how well the transition path could be interpreted, leading to plausible results.

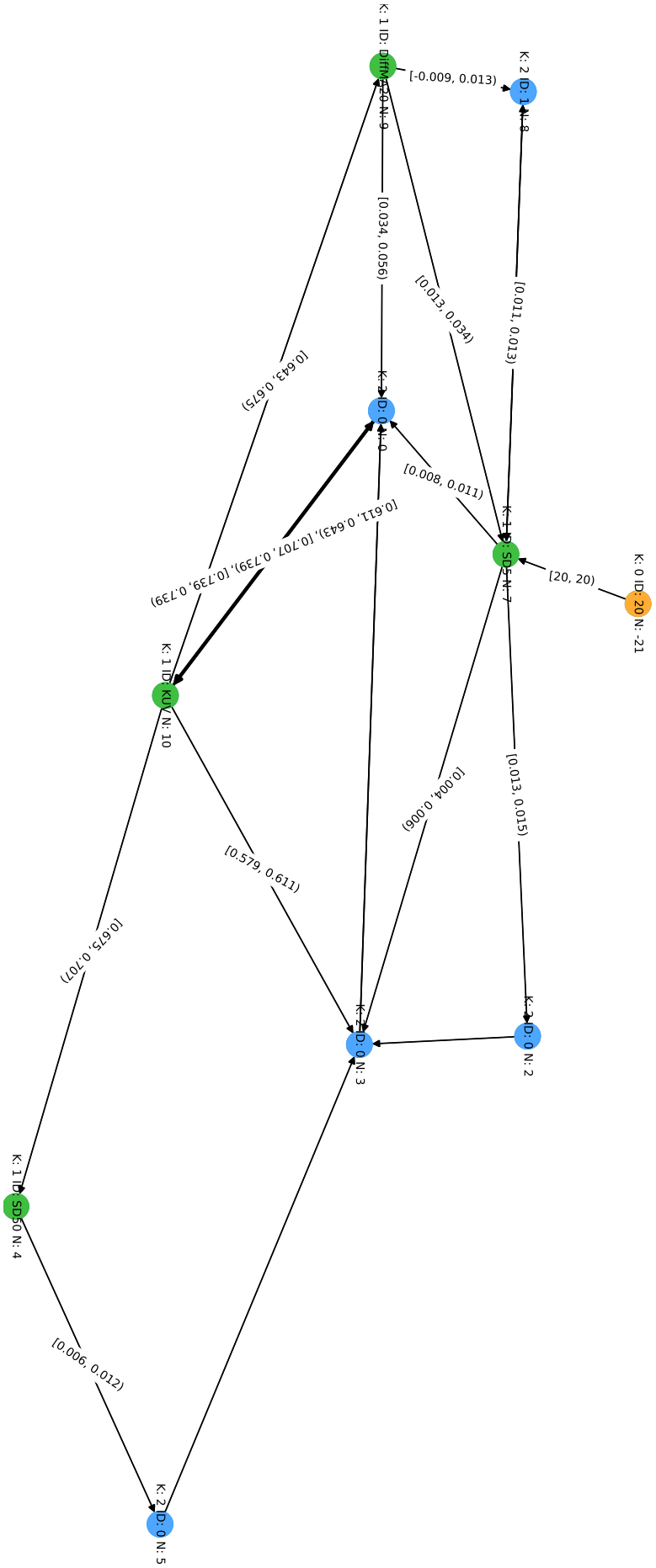

Figure 5.7: Iris Data Classification too much Nodes.
Transition Path Example ENEL SpA
Source: own illustration

5.4.3 Growing GNP

One of the main topics from Chapter 4 was the unconstrained growth of variable GNP and how this could be prevented. Therefore, the growth of the GNP when solving a complex problem is again examined here. Figure 5.8 shows the average number of nodes in the GNP of all 30 training periods. Due to the initialization of only two nodes, this number of nodes is given in the first generation. After that, adding nodes leads to a rapid increase in the network sizes but levels out at around 14 nodes. Therefore, it can be seen that even with the more complex financial data, there is no excessive network growth.

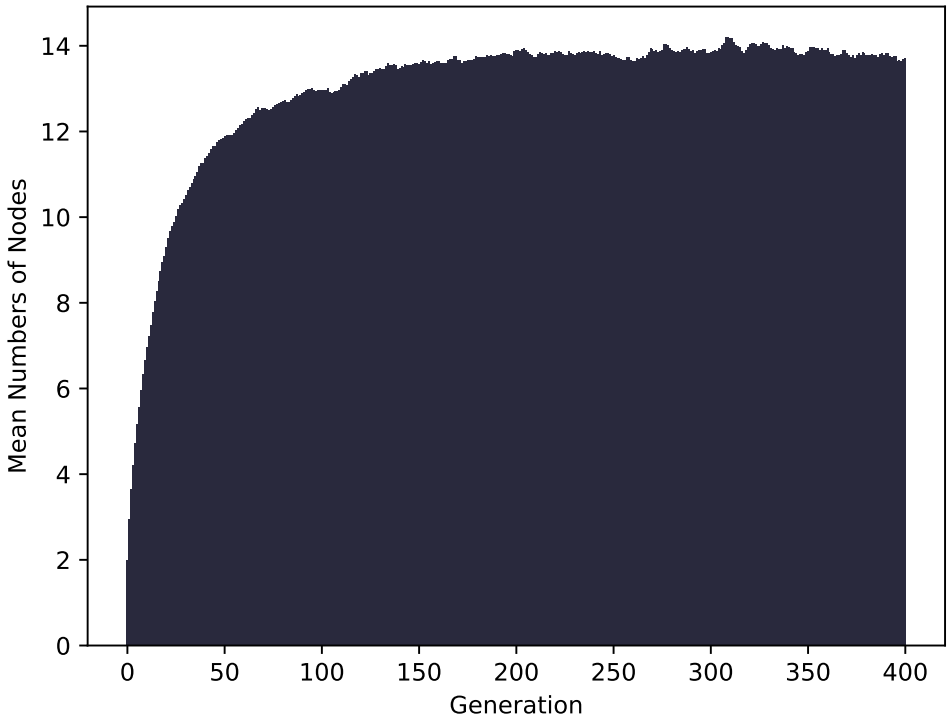

Figure 5.8: Mean Number of Nodes all Runs
Source: own illustration

A similar figure is shown in Figure 5.9 where the mean number of edges and the mean used nodes of the transition path can be seen. The number of used nodes is similar to the number of nodes present in the network, as all unused nodes are deleted except for one protected node. To adjust a GNP appropriately to the complexity of the data, it has already been discussed in Chapter 4 that the number of edges is important. The reason for this is that the number of edges determines the granularity of the data classification. One can see that the number of edges increases with the number of nodes. This is because newly introduced nodes can have more edges than existing ones. The number of edges also highlights that the GNP could increase the search space without growing too much.

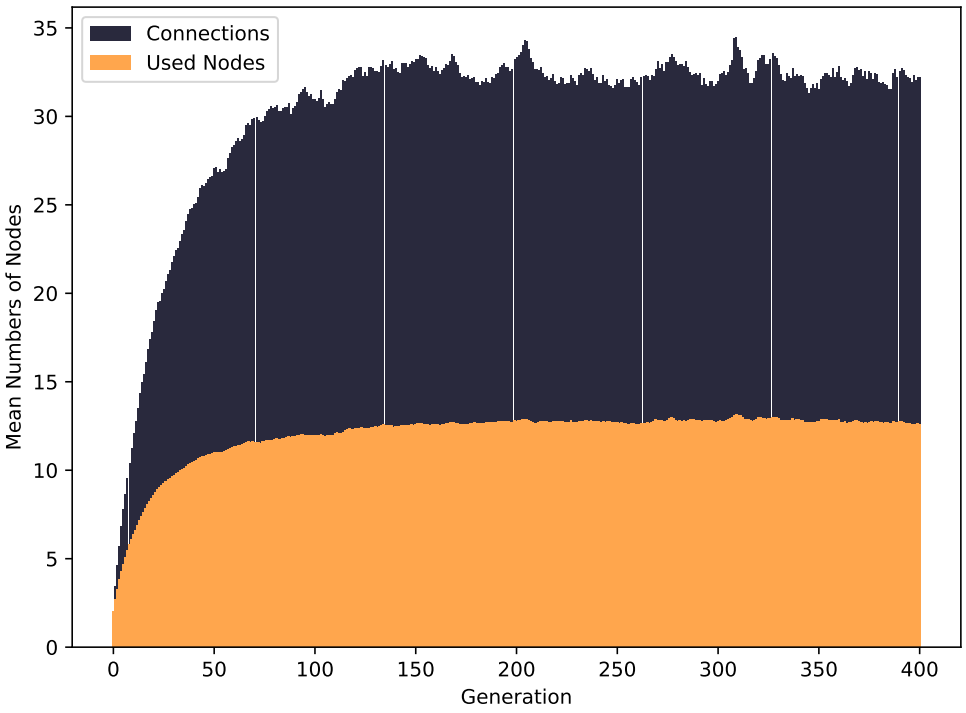

Figure 5.9: Mean Number of Connections all Runs
Source: own illustration

5.4.4 Investment Ratios

A huge advantage of initializing a separate starting node for each stock (see Section 3.3) is the ability to code the objective function in such a way that different initial budgets can be learned for each stock (see 5.1). This is illustrated in Figure 5.10, where the budget progress for each generation is shown. The graphs are the average value of all training periods. During the first generations, the budgets change enormously from one generation to the next. After about 100 generations, the budgets level out and end up in the last generation between 3.39 and 5.33. Thus, portfolio optimization could be accomplished by weighting stocks with the help of the objective function.

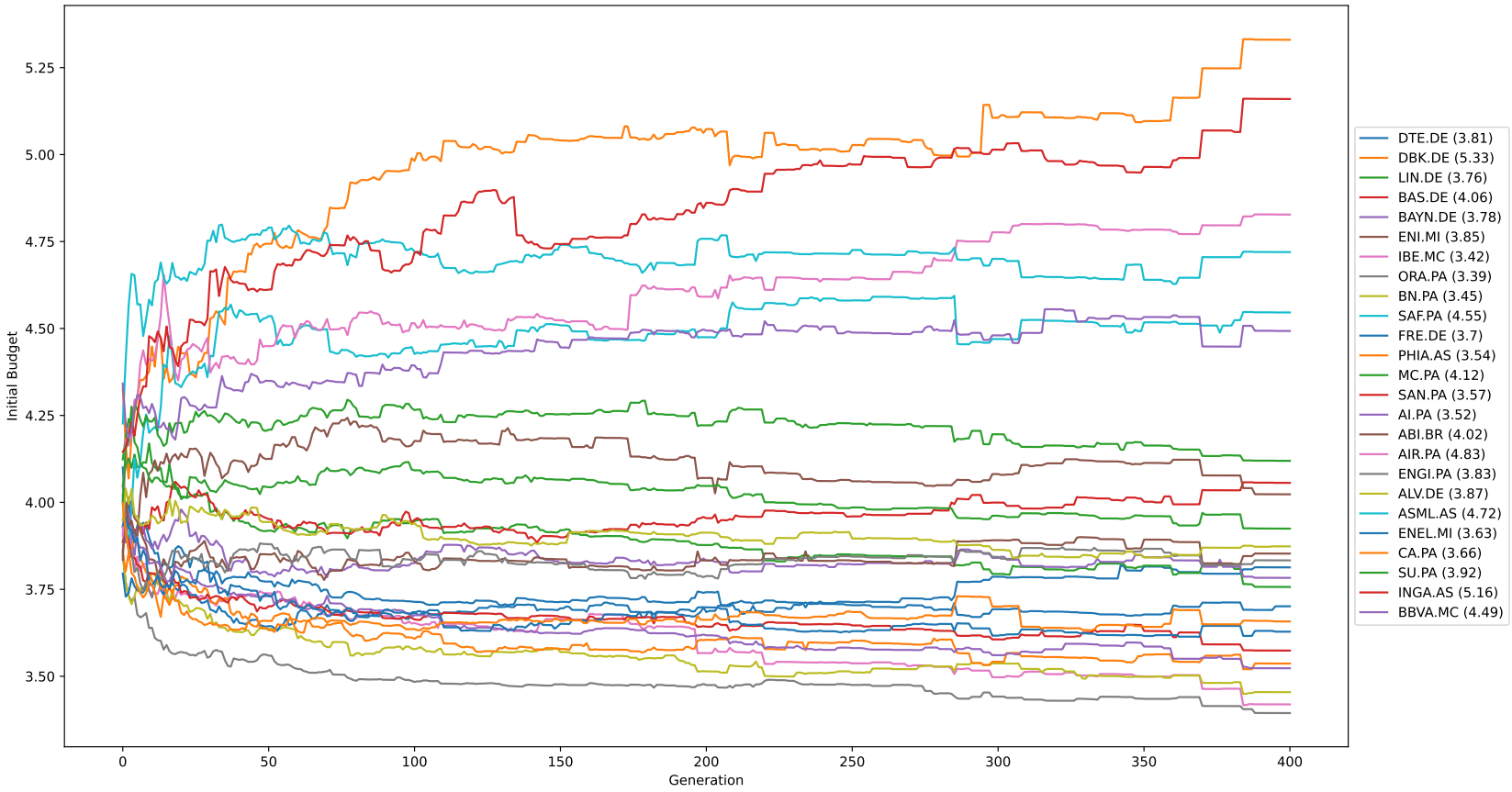

Figure 5.10: Investment Ratios
Source: own illustration

Since the total capital employed was 100 and 25 stocks were handled, each stock starts with an initial budget of 4. Very interesting is the final budget of the eight stocks that achieved negative performance (buy-and-hold strategy) in Figure 5.3. Table 5.4 shows these stocks with the corresponding budget.

Table 5.4: Ratios of Different Stocks
Source: own illustration

Name	Ticker	Sector	Budget
Orange S.A.	ORA.PA	Communication Services	3.39
Danone S.A.	BN.PA	Consumer Defensive	3.45
Safran SA	SAF.PA	Industrials	4.55
Fresenius SE & Co. KGaA	FRE.DE	Healthcare	3.70
Koninklijke Philips N.V.	PHIA.AS	Healthcare	3.54
Anheuser-Busch InBev SA/NV	ABI.BR	Consumer Defensive	4.02
ASML Holding N.V.	ASML.AS	Technology	4.72
Enel SpA	ENEL.MI	Utilities	3.63

Only two of the eight stocks received an increased budget. One stock gets a budget of 4.02, and the rest received a low initial budget. Therefore, optimization of the portfolio by different weightings of the stocks could be successfully implemented.

5.4.5 Comparison without Mutation

This chapter's last section focuses on a simulation comparison with and without the new mutation operators. For a new simulation, 18 judgment nodes and 18 processing nodes were initialized. 18 judgment nodes are necessary to ensure that each feature can be selected because the network size is not variable. The parameters of the two new mutation operators were set to 0. The other parameters were initialized exactly as on the page 86.

With the Iris data set, better results could be achieved due to the growth of the network. The question is if this is also the case for the complex financial data set. Table 5.5 compares the results of the 30 validations. The first column shows the results already presented in Figure 5.2 with the mutation operators. Column two shows the simulation results without the mutation operators, and the third column shows the difference between both simulations.

Table 5.5: Comparison Financial Validation Results with and without Mutation
Source: own illustration

With Mutation	Without Mutation	Difference
4.91	4.76	0.15
-1.50	-1.10	-0.40
4.80	1.63	3.16
2.90	2.86	0.04
2.60	0.81	1.80
7.93	3.11	4.83
-23.31	-21.97	-1.33
-0.01	-0.75	0.74
9.84	7.65	2.19
2.66	3.31	-0.64
-0.38	-2.45	2.07
2.88	-1.47	4.35
13.89	10.65	3.24
1.84	1.46	0.38
-1.15	-0.98	-0.16
6.88	2.32	4.55
3.58	4.18	-0.60
2.62	2.96	-0.34
2.05	-0.43	2.48
1.35	1.20	0.15
1.42	2.89	-1.47
2.70	-0.74	3.44
3.03	2.75	0.28
2.23	-0.39	2.62
4.62	-2.32	6.94
1.27	-0.28	1.55
-0.04	-2.57	2.52
2.66	1.42	1.24
2.18	0.05	2.13
0.02	2.11	-2.09

The new mutation operators achieved better results in 22 of the 30 validations. The sum of the differences reflects an outperformance of 43.81! Furthermore, Figure 5.11 compares the cumulative performance and the benchmark. Cumulated, an even more significant outperformance can be reached compared to the simulation without the mutation operators. The simulation without the operators also beat the benchmark, but only by 3.01%. The newly developed algorithm was again able to adapt precisely to the complexity of the data and achieve an outperformance of 61.84% compared to the standard GNP and 65.35% compared to the benchmark.

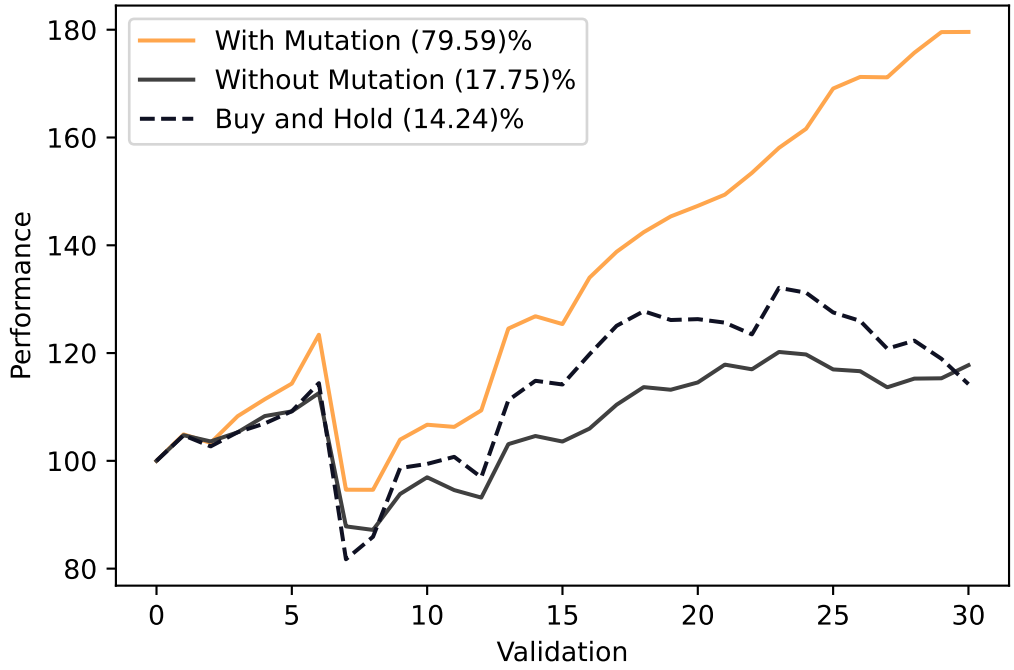

Figure 5.11: Comparison Financial Results Cumulated
Source: own illustration

A video of the entire evolution of an example stock of one validation can be found on the project homepage in the section *"Research"*.

`https://fabiankoehnke.github.io/gnp/hp/#research`

Chapter 6

Closing Words and Future Research

The main objective of this thesis was to find a method that optimizes a portfolio to generate a higher return than the market. The market in this context means the benchmark as a simple buy-and-hold strategy. Finding such an active portfolio strategy was an excellent computer intelligence task.

After studying different topics, Genetic Network Programming seemed to be an appropriate algorithm to achieve the main objective. As a sub-field of evolutionary algorithms, networks (individuals) can be evaluated as an entire unit. A network structure provides the advantage that nodes can be used multiple times, and a GNP can have an implicit memory function. This thesis confirms that GNP is very suitable for dynamic environments and financial market data. Since nodes can be used repeatedly, GNP are usually initialized with a fixed number of nodes to prevent the bloat problem. However, a disadvantage of initialized network structures with a fixed number of nodes is that the GNP can no longer sufficiently adapt to the structure of a problem to be solved. For this reason, operators have recently proposed that can change the number of nodes in the networks. But there was still the problem of a restricted gene pool even with the method presented in Section 4.1.

The first subgoal of this thesis was to code a program that can represent Genetic Network Programming and to extend the research field of GNP with simulation studies. A successful implementation was shown in particular by several illustrations of the results and individuals.

The main effort of this work was to develop operators that prevent a limited gene pool. In addition, the operators should still prevent the bloat problem and not lead to overfitting. After a theoretical introduction of relevant topics, Chapter 4 was focused on these operators to achieve these goals. A comparison of the results between the standard GNP and the GNP with the two novel operators showed that better results could be achieved on the Iris data set.

The first novel mutation operator can increase and decrease the number of nodes, and a GNP can adapt to the complexity of a given problem by itself. This was achieved not only by changing the number of nodes but also by changing the number of edges in the network. The second novel mutation operator changes the outputs of judgment functions, leading to a shift of interval boundaries and a further increase in the gene pool. By implementing these two extensions, the GNP is no longer limited in the solution capability. Furthermore, it was shown that the introduced operators do not lead to overfitting or the bloat problem due to certain constraints.

The variable number of nodes is executed in such a way that results are also very positive if the number of nodes is initialized as small as possible. The network can select necessary nodes and their functions during the growth of networks. This leads to two additional advantages through the use of the novel operators. First, the algorithm's performance could be increased because unnecessary calculations of too large individuals were avoided. Second, through the random selection of judgment nodes, data with many features can now be handled without initializing a judgment node for each feature. It was shown that the network is able to select relevant features automatically.

At the end of this thesis, Chapter 5 contained the application of Genetic Network Programming on a financial data set. The results successfully confirmed the advantages of the novel operators presented in Chapter 4 and also fulfilled the main objective of this thesis. With extended computational capacity, the results could be further increased in significance by simulating the cross-validation of the experiment multiple times. Nevertheless, the model beat the buy-and-hold strategy and generated an outperformance of 65,35%. Even though the standard GNP also led to higher returns than the benchmark, better results could be achieved using GNP with the novel operators.

Based on the thesis results, various additional research areas emerged:

- **Add multiple nodes:** The newly developed mutation operator can add at most one node per generation. Therefore, the growth is limited by the number of generations. Adding multiple nodes in each generation could lead to more suitable networks for complex problems.
- **Add/delete start nodes:** To train data sets with many groups where only a subset is relevant, start nodes could be added and deleted. For example, a portfolio that needs only a few stocks could be created from data containing a huge number of stocks.
- **No equidistant interval shifting:** The equidistant interval shift was intended to prevent overfitting. However, shifts of individual edges could lead to more diversity.
- **Renewed training of GNP:** A new population was trained for each test data shifting during the cross-validation. The results of this work could be challenged by including n best individuals in the new population. In particular, topics such as selection pressure and the dominance problem should be considered.
- **Multi-criteria optimization:** In this thesis, the objective function was expressed only by the return on investment. However, financial management usually considers further factors, especially risk criteria. Therefore, multi-criteria optimization could be applied as a large further field of research. The application is, of course, not limited to the financial sector.
- **Implement Reinforcement Learning:** An exciting extension of the GNP is the implementation of subnodes for Reinforcement Learning, as presented in the paper Mabu, Hirasawa, and Hu (2007). This paper shows how GNP has been improved by Reinforcement Learning by allowing a network to learn also during one generation. An interaction of that method with the new operators from this thesis could be a research topic.

- **Regression task:** The processing node functions in this work are always coded to be assigned to one class (e.g., buy or sell). An extended encoding of the processing node functions for solving regression tasks could be a future research field. Such processing function could represent ephemeral random constants according to Koza (1994).

Appendix A

Appendix

A.1 Ackley Function

The Ackley function is given as

$$f(x) = 20 + \mathrm{e} - 20\,exp\left(-0.2\sqrt{\sum_{i=1}^{n} x_i^2 \cdot \frac{1}{n}}\right) - exp\left(\sum_{i=1}^{n}(cos2\pi x_i) \cdot \frac{1}{n}\right)$$

A.2 Screenshot Example of the Program

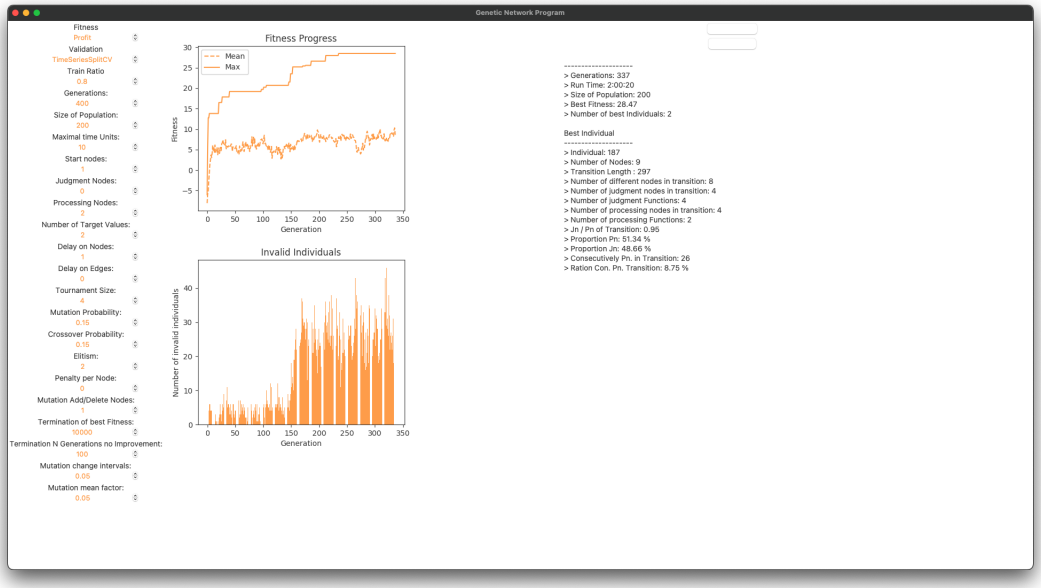

Figure A.1: Program Interface.

The left side shows the parameters to be set. On the main screen, important graphs and statistics can be seen. In addition, various extra graphics and statistics can be generated using the menu (Most shown in the thesis). It is also possible to make further settings, such as the data to be read or simulations to be saved and reloaded.

Source: own illustration

Bibliography

Berthold, Michael R. et al. (2010). *Guide to intelligent data analysis: how to intelligently make sense of real data*.

Bing, LI (2013). "Study on Genetic Network Programming with Variable Size Structure and Genotype/Phenotype Mapping Mechanism". In.

Borgelt, Christian, Matthias Steinbrecher, and Rudolf R. Kruse (2009). *Graphical models: representations for learning, reasoning and data mining*. John Wiley & Sons.

Chartrand, Gary and Ping Zhang (2013). *A first course in graph theory*. Courier Corporation.

Chen, Yan and Kotaro Hirasawa (2010). "Generating trading rules on the stock markets with Robust Genetic Network Programming using variance of fitness values". In: *Proceedings of SICE Annual Conference 2010*, pp. 3095–3102.

Chen, Yan, Shingo Mabu, and Kotaro Hirasawa (2010). "A model of portfolio optimization using time adapting genetic network programming". In: *Computers & Operations Research* 37.10, pp. 1697–1707. ISSN: 0305-0548. DOI: 10.1016/j.cor.2009.12.003. URL: https://www.sciencedirect.com/science/article/pii/S0305054809003281.

— (2011). "Genetic relation algorithm with guided mutation for the large-scale portfolio optimization". In: *Expert Systems with Applications* 38.4, pp. 3353–3363. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2010.08.120. URL: https://www.sciencedirect.com/science/article/pii/S0957417410009280.

Chen, Yan et al. (2009a). "A genetic network programming with learning approach for enhanced stock trading model". In: *Expert Systems with Applications* 36.10, pp. 12537–12546. ISSN: 0957-4174.

Chen, Yan et al. (2009b). "Constructing portfolio investment strategy based on time adapting genetic network programming". In: *2009 IEEE Congress on Evolutionary Computation*, pp. 2379–2386.

Deo, Narsingh (2017). *Graph theory with applications to engineering and computer science*. Courier Dover Publications.

Dua, Dheeru and Graff, Casey (2017). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Fama, Eugene F. (1970). "Efficient capital markets: A review of theory and empirical work". In: *The journal of Finance* 25.2, pp. 383–417.

Gaur, Deepti, Aditya Shastri, and Ranjit Biswas (2008). "Metagraph: a new model of data structure". In: *2008 International Conference on Computer Science and Information Technology*, pp. 729–733.

Hirasawa, Kotaro et al. (2001). "Comparison between genetic network programming (GNP) and genetic programming (GP)". In: *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*. Vol. 2, pp. 1276–1282.

Hirschle, Jochen (2020). *Machine Learning für Zeitreihen: Einstieg in Regressions-, ARIMA- und Deep Learning-Verfahren mit Python. Inkl. E-Book*. Carl Hanser Verlag GmbH Co KG.

Kahneman, Daniel (2012). *Schnelles denken, langsames Denken*. Siedler Verlag.

Katagiri, H., K. Hirasama, and J. Hu (2000). "Genetic network programming - application to intelligent agents". In: *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0*. Vol. 5, 3829–3834 vol.5. DOI: 10.1109/ICSMC.2000.886607.

Katagiri, Hironobu et al. (2003). "Variable size genetic network programming". In: *IEEJ Transactions on Electronics, Information and Systems* 123.1, pp. 57–66.

Koza, John R (1994). "Genetic programming as a means for programming computers by natural selection". In: *Statistics and computing* 4, pp. 87–112.

Kruse, Rudolf et al. (2015). *Computational Intelligence: Eine methodische Einführung in künstliche neuronale Netze, evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. 2., überarbeitete und erweiterte Auflage. Computational Intelligence. Wiesbaden: Springer Vieweg. ISBN: 9783658109035.

Li, Bing et al. (2011). "Variable size genetic network programming with binomial distribution". In: *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE, pp. 973–980.

Mabu, S. et al. (2002). "Online learning of genetic network programming (GNP)". In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*. Vol. 1, 321–326 vol.1. DOI: 10.1109/CEC.2002.1006254.

Mabu, Shingo, Kotaro Hirasawa, and Jinglu Hu (2007). "A Graph-Based Evolutionary Algorithm: Genetic Network Programming (GNP) and Its Extension Using Reinforcement Learning". In: *Evolutionary Computation* 15.3, pp. 369–398. DOI: 10.1162/evco.2007.15.3.369.

Mandelbrot, Beno\^\it, Richard L. Hudson, and Eric Grunwald (2005). "The (mis) behaviour of markets". In: *A Fractal View of Risk, Ruin and Reward. L.: Profile Books*.

Mandelbrot, Benoit B. (1997). "The variation of certain speculative prices". In: *Fractals and scaling in finance*. Springer, pp. 371–418.

Murphy, John J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin.

Parque, Victor, Shingo Mabu, and Kotaro Hirasawa (2010). "Enhancing global portfolio optimization using genetic network programming". In: *Proceedings of SICE Annual Conference 2010*, pp. 3078–3083.

Reza Ramezanian, Arsalan Peymanfar, and Seyed Babak Ebrahimi (2019). "An integrated framework of genetic network programming and multi-layer perceptron neural network for prediction of daily stock return: An application in Tehran stock exchange market". In: *Applied Soft Computing* 82, p. 105551. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2019.105551. URL: https://www.sciencedirect.com/science/article/pii/S156849461930331X.

Simon, Dan (2013). *Evolutionary Optimization Algorithms*. 1., Auflage. New York, NY: Wiley, J. ISBN: 9780470937419.

Singh, Harmanjit and Richa Sharma (2012). "Role of adjacency matrix & adjacency list in graph theory". In: *International Journal of Computers & Technology* 3.1, pp. 179–183.

Thorp, Edward O. (2017). *A man for all markets: From Las Vegas to wall street, how i beat the dealer and the market*. Random House.

Wurm, Gregor, Bernd Ettmann, and Karl Wolff (2005). *Kompaktwissen Bankbetriebslehre*. Bildungsverlag EINS.

Yan Chen and Xuancheng Wang (2015). "A hybrid stock trading system using genetic network programming and mean conditional value-at-risk". In: *European Journal of Operational Research* 240.3, pp. 861–871. ISSN: 0377-2217. DOI: 10.1016/

j . ejor . 2014 . 07 . 034. URL: https : / / www . sciencedirect . com / science / article/pii/S0377221714006006.