Medizinische Fakultät
Institut für Medizinische
Systembiologie

Coevolution of Boolean Networks using genetic programming

Abschlussarbeit an der Universität Ulm

Vorgelegt von:
Michel Lutz
michel.lutz@uni-ulm.de
1048353

Gutachter:
Prof. Dr. Hans A. Kestler

Betreuer:
Dr. Julian Schwab

2022

Version July 28, 2022

Satz: PDF-LaTeX 2_ε

Abstract

Computer-simulated models of gene regulatory networks have become a powerful tool for understanding the dynamic dependencies between genotypes and phenotypes. Boolean networks are one of the most widely researched models in this area. Although there has been a lot of work on generating Boolean networks by evolutionary algorithms or evolving them according to specific characteristics, little attention has been paid to the coevolution of multiple evolutionary independent populations. In this thesis, a framework was developed and implemented that allows the simulation of the coevolution of any number of populations. It is possible to simulate different forms of coevolutionary interaction between populations by changing many parameters, such as mutation rates, different survivor selection operators or population sizes during the coevolution. Special attention was paid to an exchange of initial states between the populations. In addition, a suitable tracking system was implemented to document all changes that occur during coevolution. Several analysis tools for the evaluation of these data are also available. In multiple extensive experiments, the functionality of the framework could be demonstrated for smaller, randomly generated networks as well as for the larger igf - Wnt crosstalk (igfWnt) and colorectal cancer (CRC) network.

Contents

1 Introduction

To understand the anatomy and physiology of living beings, a deep insight of their genome and how the genotype leads to the phenotype is of fundamental significance. Likewise, many diseases can be explained at this level [1]. The following chapter will show that an understanding of the structure and sequence of the genome of a living organism is not sufficient to explain all its phenotypes. Rather, dynamic descriptions of gene regulation and gene expression are needed [2].

Since all organisms are a product of evolution, methods of evolutionary theory and simulated evolution can help to understand the complex interplay of geno- and phenotype [3]. Moreover single-cells as well as species do not occur alone and in isolation, but influence each other. Thus, the study of this mutual influence, the so-called coevolution, can lead to further insights.

1.1 Gene regulation

The genome represents the entirety of an organism's genetic information. The information itself is stored in physical form in the sequence of deoxyribonucleic acid (DNA), in the case of some viruses also in ribonucleic acid (RNA) [1]. With the exception of mitochondrial and chloroplast DNA, most of the DNA in eukaryotes is found in highly condensed form in the cell nucleus [4]. The following is a highly simplified outline of the organisation of the genome. The actual information is encoded in the sequence of the four bases adenine (A), cytosine (C), guanine (G) and thymine (T). For most eukaryotes the genetic information is organised in a species specific number of chromosomes. The species *homo sapiens* has about 3,200 billion base pairs organised into 46 chromosomes. Of these, 22 are present in duplicate (autosomes) an in addition there are two sex chromosomes (gametosomes). On the chromosomes are localisable sections, the genes, which, to put it simply, form a functional hereditary unit [4]. In addition to regulatory sections such as promoters and enhancers, the genes themselves have coding regions, the exons, and non-coding regions, the introns. The process by which a protein is formed from the information within the gene is called gene expression. In simplified terms, the first step is transcription, in which a copy of the coding regions of a gene is translated as pre-mRNA. Through further steps such as splicing, this copy is processed into the finished messenger RNA. This leaves the cell nucleus and serves as a template for the synthesis of a protein at the ribosomes during translation. In essence, proteins are the functional and structural carriers of all living processes. Therefore the proteins have a decisive influence on the phenotype of a living organism [1] and are responsible for their shape and success [5]. A living being can possesses many different genes, humans about 20.500, many of that can be expressed in different splice variants [1]. Although almost all somatic cells share the same genome and thus theoretically have access to all genes, only specific genes are expressed in specific cells. This allows cells to develop differently. Furthermore, gene expression is always not identical and is modified by external influences. This control of expression is called gene regulation [2]. Although the genomes of many living organisms are published and the human genome has been known since the early 2000s through the Human Genome Project, no decisive breakthrough has yet been achieved through knowledge of the sequences alone. Instead, the question of gene regulation is increasingly coming to the fore.

Gene regulation is a complex process that is not yet fully understood and that occurs at different levels of gene expression. First, the chromosomes of eukaryotic cells are not open in the cytoplasm but are wrapped around histones, which means that not all areas of the DNA are available for transcription [4]. Furthermore, DNA methylation leads to the inactivation of entire genes. Histone modification and methylation are part of epigenetic gene regulation. Cell differentiation and thus the formation of different cell types is mainly based on DNA methylation, which influences gene expression over long time scales [6]. At the gene level, so-called regulatory regions exist to which so-called transcription factors dock. These can activate or suppress the transcription of the corresponding genes. Since transcription factors are also proteins, they are also subject to gene regulation. These results in a complex, mutually influencing regulatory network [2]. Furthermore, the cell can adapt its behaviour through the presence and absence of external signals. In addition to receptors on the cell surface, many other proteins and messenger substances are involved, which often lead to a change in gene expression. This information cascade is called signal transduction [4]. A better understanding of these gene regulatory networks (GRN) allows a deeper understanding of the behaviour of intracellular processes of different physiological and pathological conditions. Since many diseases are caused by defective gene regulation, the analysis of these networks can lead to new therapeutic targets [7]. These networks and their dynamics can be reconstructed based gene or mRNA concentration measurements over time. However, it must be ensured that as many different activation stages of the genes as possible are covered. This can be done by performing the measurements under different environmental conditions or by knocking out or over-expressing certain genes. Since the number of components to be taken into account is enormous and such biological experiments are time and cost intensive, the computer simulation of such regulatory networks is of particular importance [5, 8].

1.1.1 Gene regulatory networks

Since the complex gene-regulatory networks cannot be understood by biological experiments alone, several mathematical models of such networks have been developed which led to new insights through computer simulations. The most common types of models are briefly presented below.

Ordinary differential equations (ODEs)

Ordinary differential equations are certainly one of the most widely used models for simulating dynamic systems in both the natural sciences and engineering [8]. The formalism describes the presence of mRNA, proteins, or other small molecules involved in regulation quantitatively with the help of time-dependent variables. The concentration is represented by rate equations of the form

$$\frac{dx_i}{dt} = f_i(\mathbf{x}), \quad 1 \leq i \leq n \tag{1.1}$$

where $\mathbf{x} = [x_1, ..., x_n] \geq 0$ describes the concentration vector and $f_i : \mathbb{R} \rightarrow \mathbb{R}$ usually a nonlinear function. This usually leads to the fact that an analytical solution of the system is not possible and thus long calculations become necessary. One advantage of this approach is that they lead to Hill curves with a sigmoidal shape that could be confirmed by experimental data [8]. ODEs thus allow a continuous, precise and realistic representation of regulatory networks, whose size, is strongly limited by the high computational effort. Furthermore, the modelling of real regulatory networks depend on exact quantitative measurements of kinetic parameters, which are often not available for larger systems [9, 5]. For this reason, the work presented here does not use ODEs but the Boolean networks presented in detail below.

1.2 Boolean Networks

Boolean networks (BN) as models for gene regulatory networks have a long history and were first proposed by Stuart Kauffmann in 1969 [10]. In BN, n genes are simplified as either active (on, true) or inactive (off, false) and their interaction is done by means of Boolean transition functions. As already mentioned, the concentration levels of components in regulatory networks usually behave like Hill functions [8], but it turns out that the approximation as a dichotomous step function is a good approach to real conditions [11, 12]. Although Boolean networks are among the simplest dynamic gene regulatory models, they are widely used in research [7]. Formally, a Boolean network consists of a set $\mathbf{X}$ of n binary variables

$$\mathbf{X} = \{x_1, x_2, ..., x_n\}, x_i \in \mathbb{B} \tag{1.2}$$

and a Set $\mathbf{F}$ of n Boolean transition functions

$$\mathbf{F} = \{f_1, f_2, ..., f_n\}, f_i : \mathbb{B}^{\mathbf{K}} \rightarrow \mathbb{B} \tag{1.3}$$

where $\mathbf{K} \subseteq \mathbf{N}$ is the connectivity, i.e. the set of nodes to which a node is connected. The RBN-models proposed by Kauffmann use a constant $\mathbf{K}$ for all transition functions. Different types of such functions will be discussed in more detail later. A BN can be seen as a direct graph, with the edges as regulative interactions and the nodes as components. The 10-gene model of the mammalian cell cycle as proposed by Faure et al. [13] serves as an example in figure 1.1.

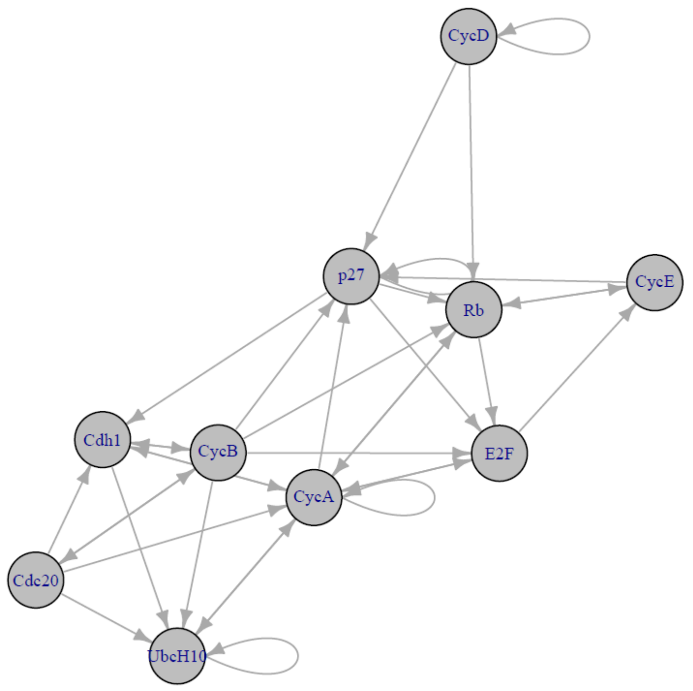


Figure 1.1: Boolen Network of the mammalian cell cycle as proposed by Faure et al. [13]. The nodes of the graph are the genes, and the directed edges show the dependencies of the genes. Figure generated with the boolNet-Package [14].

Boolean networks are dynamic and are updated in discrete time steps. This means that the state of a network at time t is defined by the vector

$$\vec{\mathbf{x}}(t) = \{x_1(t), x_2(t), ..., x_n(t)\} \tag{1.4}$$

and the transition from one point in time to the next is described as [7]

$$\mathbf{x}_i(t+1) = f_i(\vec{\mathbf{x}}(t)), f_i \in \mathbf{F} \tag{1.5}$$

thus a BN with n components can potentially reach 2^n different states [6].

1.2.1 Updating Schemes

There are three general paradigms for updating BNs, which are briefly described below.

In synchronous Boolean networks, all transactions $\mathbf{T_{sync}}$ are applied simultaneously at each time step [12].

$$\mathbf{x}_i(t+1) = \mathbf{T_{sync}}(x(t))$$
$$\mathbf{T_{sync}} = (T_1, T_2, ..., T_n) \text{ and } T_i(x) = f_i(x) \tag{1.6}$$

Consequently, the network has a deterministic dynamic and each state has exactly one successor.

In asynchronous Boolean networks, only one transition function is selected randomly at each time step or according to a predefined ordering system [6]. Thus the update mechanism is not deterministic but stochastic and each potential state can lead to n potential successors [7]. Formally, the asynchronous transition function for component i^* is defined as follows [12].

$$\mathbf{x}_i(t+1) = \mathbf{T^{(i^*)}_{async}}(x(t))$$
$$\mathbf{T^{(i^*)}_{async}} x(t) = \begin{cases} f_i(x) & i = i^* \\ x_i & i \neq i^* \end{cases} \tag{1.7}$$

Since a synchronous update of all components of a GRN is a simplification of real biological systems, an asynchronous update mechanism was considered more representative [7]. However, the single-cell application of transitions leads to unrealistically long duration of biological processes [7]. In addition, the runtime for large asynchronous Boolean networks increases dramatically, limiting their use [7]. Addressing these problems, several different update mechanisms inspired by the asynchronous paradigm, such as random order asynchronous or deterministic asynchronous updating, have been proposed [15]. Regarding to the robustness of GRNs, it could be shown that synchronous updating may be of greater relevance than various asynchronous mechanisms. It can therefore be assumed that synchronous and asynchronous update strategies lead to the same dynamically stable behaviour and thus have comparable biological relevance [15, 7].

Probabilistic Boolean networks allow more than one transitions function for each component. The actual update is performed synchronously, where for each component one of its transitions functions is selected at random. Each transition has a probability of being selected in each step, where the probabilities of all functions for a component add up to one [12, 16]. This class of BNs was introduced to be able to describe also GRN, which are afflicted with an uncertainty in their transitions. This is often the case in the reconstruction of networks from gene expression data [7]. Thus, probabilistic Boolean networks are not deterministic, which requires other tools for their analysis, such as Markov chains [5].

In the context of this work, only synchronous updating was used to model gene regulation, since deterministic behaviour leads to results that can be interpreted more clearly. Furthermore, for the simulation of several large populations, as is the subject of this study, verifiable runtimes for the transitions are necessary.

1.2.2 Transition Functions

In the following section, the different transition functions are classified and their importance for the dynamic behaviour of Boolean networks is discussed. As mentioned earlier, the transition function is a Boolean function $f_i : \mathbb{B}^{\mathbf{K}} \to \mathbb{B}$, which means it evaluates to true, encoded as 1, or false, encoded as 0. In terms of gene regulatory networks, $f_i = 1$ means that a transcription factor or an external signal is present or that a gene is being expressed. $f_i = 0$ signals the absence of a signal or that a gene is not being expressed. The function maps from a K-dimensional space. K describes the connectivity of a BN, where $\mathbf{K} \subseteq \mathbf{N}$ and for biologically relevant networks typically $K \ll N$, e.g. $K \in [1 - 6]$. In total there are therefore

$$2^{(2^K)} = \begin{cases} 2 & K = 0 \\ 4 & K = 1 \\ 16 & K = 2 \\ 256 & K = 3 \\ \dots \end{cases} \tag{1.8}$$

many different functions. The transitions can be classified according to their connectivity [6].

$K = 0$ For K = 0, there are only two constant functions $f = 0$ and $f = 1$. Such transition functions correspond to continuous expressed or never expressed genes in GRN models. $f = 1$ thus corresponds to an "over-expressed" gene and $f = 0$ to an "knock-out" gene in biological experiments [17].

$K = 1$ In addition to the constant functions, for $K = 1$ there are the identity $(f(x) = x)$ and the negation $(f(x) = \neg x)$. Thus, the following possibilities result for $f(x)$ are shown as truth table.

x	0	1	x	$\neg x$
0	0	1	0	1
1	0	1	1	0

Table 1.1: Possible transition functions for $K = 1$

$K = 2$ The possible transitions for $K = 2$ represent all binary Boolean functions. Even if 16 different such functions can be defined, already the operators AND ($\wedge$), OR ($\vee$) and NEG ($\neg$) form a complete basis with which all further operators, e.g. XOR ($\oplus$), can be represented [5]. The two operators AND and OR follow the rules in table 1.2.

x_1	x_2	$\wedge$	$\vee$
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

Table 1.2: Truth table for $\wedge$ and $\vee$

$K \geq 3$ For $K \geq 3$ the number of functions increases strongly. However, all further transitions can be expressed as compositions of the already presented Boolean operators. Such compositions are also called the symbolic representation of a Boolean function. Their evaluation is then done by applying the rules of Boolean algebra. More information about this topic can be found among others in Schöning's "Logic for computer scientists" [18].

A special class of Boolean functions are the so-called canalizing functions. Such a function has the property that an input variable alone dominates the value of the entire function to either true or false [19]. A distinction can be made between fully canalizing functions and normal canalizing functions. For the first one an argument determines the function value completely. That means for an argument $x_i = 1$ and $x_i = 0$ the function always evaluates to $f(x_i) = 1$ and $f(x_i) = 0$ independently of the assignment of all other variables. Exemplary are functions of the form $f(x_1, x_2, ...x_n) = (x_1 \wedge (...x_2...x_n))$. For normal canalizing functions, this is only true for $x_i = 1$ or $x_i = 0$ [6]. See therefore also table 1.3. Thereby, canalizing functions can also occur nested [19]. It is assumed that such functions have high biological relevance [17].

			$f(x_1, x_2, x_3)$			
x_1	x_2	x_3	$\mathcal{A}$	$\mathcal{B}_1$	$\mathcal{B}_2$	$\mathcal{C}$
0	0	0	0	1	1	0
0	0	1	0	1	1	1
0	1	0	0	1	1	0
0	1	1	0	1	1	1
1	0	0	0	0	0	0
1	0	1	0	0	0	1
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Table 1.3: Examples of different canalizing and non-canalizing functions. $\mathcal{A}$ = constant function $f = 0$. $\mathcal{B}_1$ = fully canalizing function of the first argument x_1. When $x_1 = 0 \Rightarrow f = 1$ and $x_1 = 0 \Rightarrow f = 1)$ $\mathcal{B}_2$ = normal canalyzing function of x_1. When $x_1 = 0 \Rightarrow f = 1$ and for $x_1 = 1$ the value is depending on all arguments. $\mathcal{C}$ = generalized XOR $(\oplus)$ as example for a non canalizing function. [6]

Canalizing functions are of special importance because gene regulatory networks based on these transitions are particularly stable against perturbations [19]. In addition, networks with canalizing functions are still in the ordered phase even at higher connectivity. This is due to the fact that, regardless of the canalizing type, the non-canalizing inputs in these functions are practically irrelevant [20].

1.2.3 Dynamic of Boolean Networks

By simulating Boolean networks, insights into the dynamic behaviour of Boolean networks can be gained. Since this work is limited to a synchronous update of Boolean networks, all the following considerations refer to synchronous BNs. Starting from an initial start state, the BN changes its state with each update. The state transitions are deterministic. Since the state space is finite with 2^n different states, the network must move from each initial state either into a single state or into a circle of several states that the network will not leave [6]. These states are called an attractor. All states from which a network moves into an attractor are called the basin of the attractor [7]. In synchronous BNs, the network does not leave an attractor alone, but only when an external disturbance occurs. The attractors of a BN are of outstanding interest, as it is assumed that they can be associated with certain phenotypes [2]. Therefore, as an example among others [12], the attractors of the cell cycle Boolean network could be identified with the phases of the cell cycle [13, 21, 22]. Several different types of attractors can be distinguished [12].

steady state attractors consist of only one state. Such an attractor is shown in figure 1.2.

simple cycle attractors represent the generalisation of steady state attractors. Each state of such an attractor is reached again after a fixed set of state transitions. Thus, the attractor represents a circle with a fixed number of states, the length of the attractor. Therefore, steady state attractors have a length of 1. An example of a simple cycle attractor is shown in figure 1.2.

complex or loose attractors complex or loose attractors can only occur in asynchronous networks. Since in such networks each state can have several different successors, an attractor can also be abandoned, and thus be part of a circle sharing several states. See therefore figure 1.2.

1.3 Genetic Programming

Genetic programming (GP) is a programming paradigm that has given rise to various so-called Evolutionary Algorithms (EA) [23]. There are several major schools of evolutionary algorithms, which differ in the choice and importance of operators

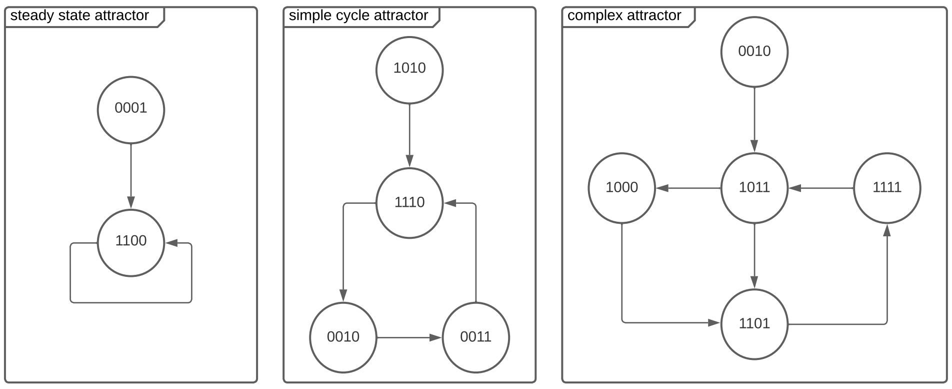


Figure 1.2: Examples of different types of attractors. Left: A steady state attractor consisting of only one single state. Middle: A simple cycle attractor can consist of any number of states. Thus, the steady state attractor is a special case of the simple cycle attractor. Right: Complex attractors can only occur in asynchronous networks and have several possible successors.

and the representation of solution vectors, called genes. What they all have in common is that they are inspired by Darwin's theory of evolution [24]. GP applies the principles of biological evolution to a population of new computer programs in which mutation and recombination generate new programs from an initial parent generation. Inspired by the limited resources of natural environments, only the best, in evolutionary terms the fittest, of these new programs are passed on to the next generation. This selection is known as "survival of the fittest". Therefore better programs evolve over time [25]. EAs are in essence heuristic algorithms for solving optimization problems. They are used for technical optimization tasks as well as for the investigation of biological problems [26].

In EA, solution vectors are considered as genes that move through the problem space, called adaptive fitness landscape. The genes are initialised randomly and there quality is determined by the problem-specific fitness function. Exploitation of the environment occurs through random changes in the genes, so-called mutations. Mutations can be partial or gene-wide and either realised by a bit-flip, for integer representation, or by a stochastic operation, for real-valued genes. In most approaches, two or more solutions could also be recombined by mating, depending on their fitness. This means that partial solutions are combined and a new solution vector, a child, is generated from the parent solutions. Thereby the problem space is usually explored in larger leaps than would be possible through mutation alone. After the creation of the children, through mutation and recombination, only a subset of the parents and children are selected to be carried over into the next generation, so that the number of individuals remains constant in each iteration [27]. Especially recombination and selection exert evolutionary pressure that allows the algorithm to reach optimal solutions fast and to leave local optima [26]. In the following, the general evolutionary loop is shown as pseudo-code, as proposed by DeJong as a "Unified Approach" [24].

```
Begin
   Initialise population with random candidate solutions
   Evaluate each candidate
   While (true)
      1. Select parents
      2. Recombine pairs of parents (mating)
      3. Mutate the offspring randomly
      4. Evaluate by fitness-function
      5. Select individuals for next generation
End
```

In EA, each iteration can be seen as a new generation in which new individuals, i.e. solutions, are born and old ones die. Evolution does not optimise the individual but the species over long periods of time.

In the context of this work Boolean networks, represented symbolic as trees, are evolved by a genetic algorithm towards one, or more, attractors specified by the user. Alternatively, a desired sequence of states, a path, can be chosen as the objective.

1.4 Coevolution

The concept of coevolution, like the theory of evolution itself, goes back to Charles Darwin [28]. He introduced the term coadaptation to describe the adaptation of a distinct organic being to another being in interspecific interaction [29]. Thus, coadaptation describes a reciprocal change. Since the 1970s, starting with the classic work of Ehrlich and Raven on the interactions in the evolution of butterflies and plants [30], the term coevolution has been widely used and has become the central subject of numerous scientific papers [28]. Since coevolution occurs in a variety of forms, and not least because of the lack of a uniformly accepted definition of the term [31], it has become a catch-all term for a wide range of different mechanisms and outcomes of reciprocal evolutionary change [28]. In the context of this work, coevolution according to Murmann is to be understood as follows. Two systems coevolve when they have a causal impact on each other's evolution [32]. It is important to note that two coevolutionary units do not exchange genetic information with each other, so that the species boundaries remain intact. However, one unit exerts more than just a selection pressure on a second, rather a first population reacts to an evolutionary adaptation of a second population on its part with an evolutionary adaptation. This response in turn, triggers a reaction in the evolutionary adaptation of the first group [31]. According to Tompson, five different models can be distinguished, which can be summarised under coevolution [28]. The most important of these will be briefly described.

Gene-for-gene coevolution emerges from the parasite and host model. It assumes that a parasite, or a pathogen, produces a specific gene product to successfully attack a host, and so the host in turn produces a specific gene product to defend itself. This leads, in the sense of a competitive coevolution, to an "arms race" between host and parasite in which both sides try to adapt the corresponding gene to the hostile counterpart [33].

Specific coevolution is always present when no gene-for-gene relationship can be specified. In addition to an "evolutionary arms race" this can also leads to a divergence of traits in competing species, or to convergence of traits in mutualisms [28].

Diffuse- or guild coevolution is the term used to describe the reciprocal influence of the evolution of more than two species. This means that the evolutionary unit is extended from one pair to a whole community [28].

2 Methods

The following chapter highlights some basic conceptual choices and methods that form the basis of the framework.

2.1 Evolution of Boolen Networks

In this work, the influence of several parallel-coevolving populations on each other was investigated. For this purpose, single individuals were modelled as synchronous updated Boolean networks, as described in Chapter 1.2.

2.1.1 Symbolic Representation as Tree

The individual transitions are symbolically represented as trees. Since symbolic Boolean expressions consist only of literals, unary $(\neg)$ and binary operators $(\wedge, \vee)$, they can naturally be represented as trees instead of terms. In this work, all transitions are represented as such trees. An example can be found in figure 2.1. This allows an intuitive and syntactically correct changing of the transitions, as it occurs in the context of the evolution of these networks.

2.1.2 Attractor Search

Since attractors play such a prominent role in BNs, the question of attractor search arises. Many approaches have already been proposed for this. For synchronous BNs, one of the simplest and most intuitive ways is a brute force approach, where the network is computed for all 2^n possible states until an attractor can be identified. Due to the exponential growth of the number of states to be examined, such

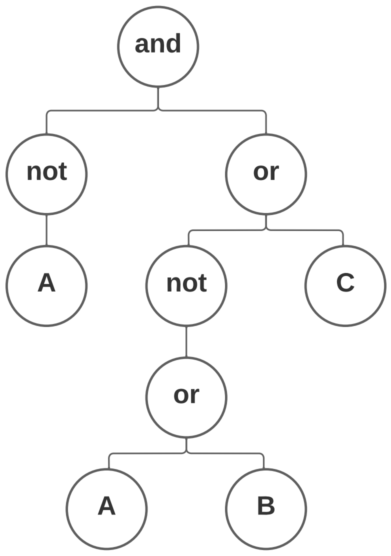


Figure 2.1: Example for a transition function, represented as tree. The example expression represented as term is:$((\neg A) \wedge ((\neg(B \vee A)) \vee C)$

an approach is limited to small networks of at most 30 genes. Such an algorithm was proposed by Hopfensitz et al [12]. Nonetheless, such an exhaustive search has the advantage that all attractors and their corresponding basins can be identified. Since it could be shown for the attractor search that it belongs to the class of NP-hard problems [34], no significantly faster methods for exhaustive attractor searches are to be expected. One possible approach to dealing with larger networks is to transform the attractor search into a satisfiability problem [35]. There are many potent heuristic approaches to solving such problems, but they have the disadvantage that no information about the state transitions can be obtained [36].

Another approach is to search for attractors only from certain start states [7]. In this way, not all attractors of a BN can be found with certainty, but attractors with a small basin of attraction are considered biologically less relevant [37, 38]. Thus, it can be assumed that even with a non-exhaustive search, many important attractors can be found. Furthermore, not all possible states of a BN are actually biologically plausible. For example, it is plausible to assume that a newly formed cell after mitosis does not occupy a random position in the state space but is in one or a few specific states. Furthermore, neighbouring cells of the same and nearby populations influence a cell through their metabolic activities. These influences are simulated in the experiments conducted in this work by the fact that the fitness of individuals, as described in the chapter 2.1.3, is always determined by the attractor that can be reached from a certain starting state. This means that an exhaustive search for an attractor plays only a minor role in this work.

2.1.3 Genetic operations

Using of an evolutionary algorithm, a population is optimised towards improved fitness. For this a new generation, must be generated from the existing individuals by genetic operators. Genetic operators change the characteristics of an individual in a random direction. Various mutation and recombination operators are conceivable on the Boolean trees already presented. The following are used in the context of this work [39].

Point mutations change a randomly chosen node of the tree. Either a negation can be inserted as an additional node or can be removed. Furthermore, a literal can be exchanged for a literal or a binary operator for another binary operator. Point mutations thus lead to new traits [24]. Example mutations can be found in figure 2.2.

Recombination exchanges complete transition functions between two or more parents. Therefore, recombination results in a new arrangement of already evolved traits [24]. The operator is often also referred to as crossover. An example recombination can be seen in figure 2.3.

Applying random mutations to transitions often results in a functionless expansion of the Boolean expression. For example, double negations (e.g. $\neg(\neg A)$) or idempotents (e.g. $A \wedge A$) can occur. This effect is well known for such evolutionary algorithms and has found its way into the literature as “bloat” [24]. By applying rules, to simplify the Boolean expression as outlined below, this can be counteracted. s

Simplification Rules

Symbolically represented Boolean functions can be simplified in many cases by applying suitable rules of the Boolean algebra on the expression without changing the truth value. The most important ones are briefly outlined below [18].

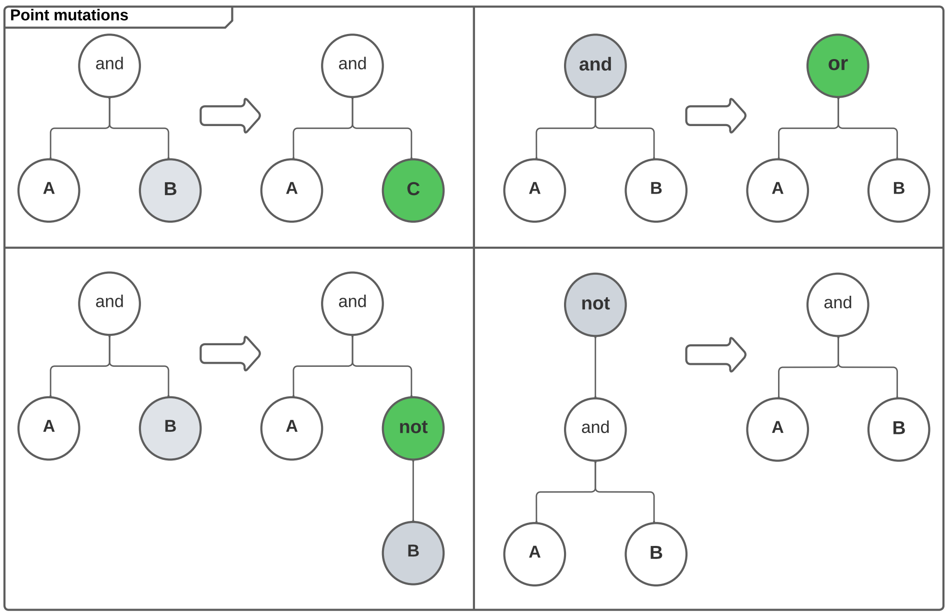


Figure 2.2: Different types of point mutations. Top: Exchange of a literal (left) or an operator (right). Below: Insertion or removal of a negation.

Dominance laws

$$\begin{aligned} A \vee 1 &= 1 \\ A \wedge 0 &= 0 \end{aligned} \tag{2.1}$$

Idempotent laws

$$\begin{aligned} A \vee A &= A \\ A \wedge A &= A \end{aligned} \tag{2.2}$$

$$((A \wedge B) \vee ((A \wedge B) \vee B)) = ((A \wedge B) \vee B))$$

Complement laws

$$\begin{aligned} A \vee \neg A &= 1 \\ A \wedge \neg A &= 0 \end{aligned} \tag{2.3}$$

Double Negation

$$\neg(\neg A) = A \tag{2.4}$$

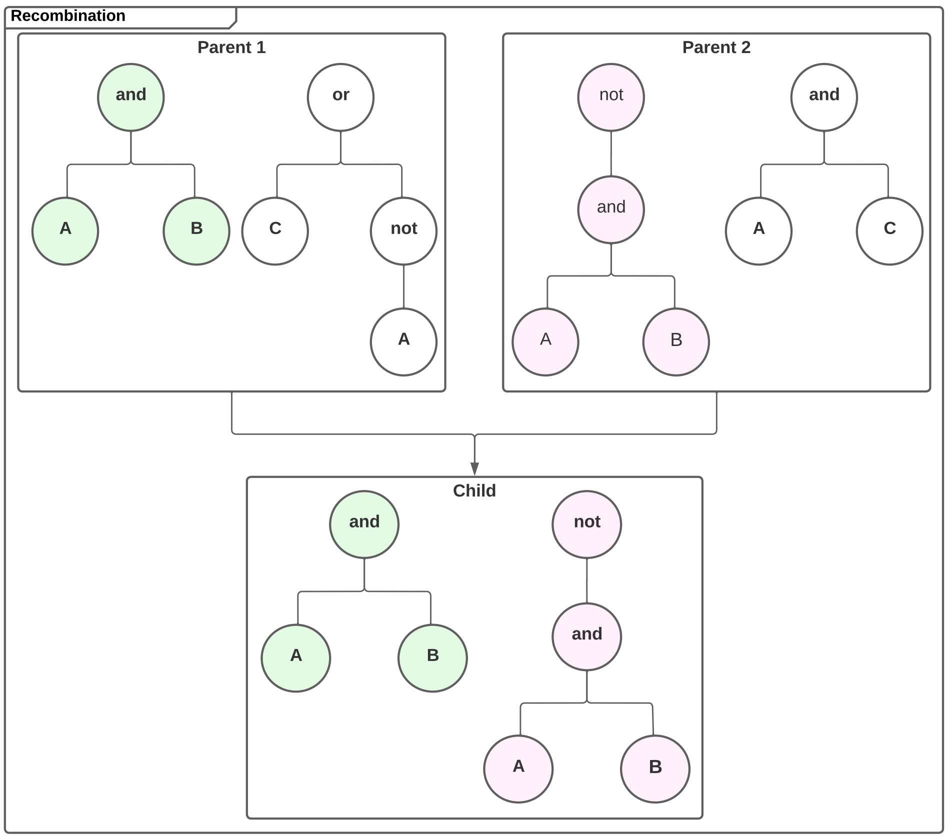


Figure 2.3: Example recombination in which the transition function of a gene from each of the two parents is combined in the child.

De Morgans's law

$$\begin{aligned}\neg A \vee \neg B &= \neg(A \wedge B)\\ \neg A \wedge \neg B &= \neg(A \vee B)\end{aligned} \tag{2.5}$$

A complete simplification of a Boolean expression is generally an NP-complete problem. Karnaugh maps and the Quine-McCluskey algorithm are well-known methods for this task [40]. Since the transitions used in this work are normally not too complex, they are simplified exclusively with the rules presented.

Fitness function

In the context of genetic programming, an appropriate choice of a fitness function is crucial. In the context of this work, different Boolean networks should be optimised towards a previously defined attractor or a specific sequence of states, called a path. The search for an attractor or path is starting from one or more initial states [24]. The fitness calculation used in this work was carried out using the method described by Aly [39], which is briefly outlined below. For experiments with attractor comparison, the similarity of the previously determined attractor with the one reached from the initial state was compared as fitness. The Hamming distance is used to compare two states within the attractors. For attractors with more than one state, the order of the attractors must be considered to achieve an optimal fitness. If the size of the attractors differs, the absence of a state is punished more severely with a lower fitness than the presence of an additional state. This is based on the assumption that a missing state is significantly more problematic than an additional state [39]. For the comparison against a path, the optimal fitness is achieved when all states occurring in the target path also occur in the same order as a transition in the individual. Additional states are not taken into account. The similarity of individual states is determined by their Hamming distance [39].

Survivor Selection

In each run of the evolutionary loop of the algorithm, each individual of the population produces a child by mutation or recombination. From the resulting new population, consisting of parents and children, a fix number of individuals must now be determined for the next generation. For this operation, known as survivor selection, several different procedures have been described in the literature, which have different influences on convergence and the maintenance of diversity [24].

Three different selection modes were used in this work. The simplest mode selects just the n fittest from the old and new networks. Such a selection strategy is called *truncation selection* in literature. This purely fitness-driven approach generally exerts a high selection pressure, but also allows the networks to converge quickly. This can lead to a fast loss of diversity, especially if the initial population already had widely varying fitness values [24]. As a result, optimisation often gets caught in local optima and cannot leave them again. The second selection mode is often referred to in the literature as "roulette-wheel selection". This involves creating a discrete probability distribution for all individuals based on their fitness. This distribution can be viewed as a ”wheel of fortune” on which each individual takes up an area, which size is depending on their relative fitness value to the population. This wheel is now spined n times and in this way the survivors are determined. A general problem with this approach is the high variance in repeated sampling and therefore low selection pressure [24]. The last operator is called elitism selection. Such an approach keeps a certain number of the fittest individuals and selects the remaining ones by any other method. Such an approach is normally characterised by greater selection pressure with increased diversity than the ”roulette wheel” method [24]. By maintaining diversity, optimisations with this selection strategy could often leave local optima.

2.2 BoolNet

The R package BoolNet[1] provides a wide range of different functionalities for dealing with Boolean networks. Among other things, synchronous, asynchronous and probabilistic Boolean networks can be randomly generated, analysed and visualised [14]. In the context of this work, various randomly generated networks were created with this tool, exported in SBML[2] format and used for various coevolution experiments. Furthermore, the visualisation capabilities were used several times.

2.3 Program: Evolution of Boolean Networks

The whole project can be seen as a comprehensive extension module of the program implemented by Aly in her master's thesis [39]. There are several good reasons for this choice.

Firstly, the program is implemented in the programming language Clojure[3], a Lisp[4]

[1] https://cran.r-project.org/web/packages/BoolNet/index.html
[2] Systems Biology Markup Language, https://sbml.org/
[3] https://clojure.org
[4] http://lisp-lang.org/

dialect. Clojure offers through its functional programming paradigm, without side effects, the possibility to implement simple and safe concurrent programming. Since the coevolution of multiple populations, each consisting of many individual Boolean networks, requires massive computing time, a consistent parallelisation of the project is mandatory. Another advantage of Clojure is that it runs on the Java Virtual Machine, which is available for practically every relevant operating system and computer architecture.

Secondly, a Common Business Module has already been implemented in the program, which provides some functionalities also required in this project. For example, methods to read and write back existing Boolean networks in SBML format. This allows to use already generated networks from the literature, such as the igfWnt / CRC networks, as well as randomly generated networks as a basis for experiments. Furthermore, coevolved networks can be exported again to be visualised or further analysed with other tools such as BoolNet [14]. In addition, many of the core functionalities, such as the attractor search and the implementation of the genetic algorithms, which are needed in this work, are already implemented within this module. However, many functionalities were modified for this purpose, their possible applications were generalised and new features were added. Nevertheless, this work maintains complete backward compatibility with the experiments described by Aly [39]. Finally, this approach has the advantage that by extending the existing code and adapting it to current Java and Clojure versions, a new powerful tool for working with Boolean networks is created.

In a sense, the coevolution module developed in this work can be seen as a generalisation of Aly's timeseriesevolution module, which allowed the evolution of a population towards a particular attractor or state sequence. This approach was taken up and extended by the possibility of evolving several genetically separated populations in parallel, in the sense of coevolution. As will be described in the following, a variety of interactions between the populations can be determined and investigated.

3 Implementation

3.1 Terminology

The research fields of Boolean networks and evolutionary algorithms use a language borrowed from biology, which leads to a ambiguity of certain terms and their meaning. This is due to the fact that the biological terms contain implications that the implementations designated by them do not or only partially adhere to. In order to achieve verbal clarity and an easy delimitation of terms, the following are used.

Individual An Individual represents an independent regulative unit. It is implemented as a Boolean network. In Biology it corresponds for example to a single cell.

Population A population is a set of individuals at a certain point in time. In the sense of genetic programming, it usually represents the entirety of an isolated evolutionary unit.

Generation A generation assigns an age to the individuals of a population. In the sense of genetic programming, the current population can be regarded as a new generation each time it passes through the evolutionary loop. However, according to the biological model, the term often refers to parent and child individuals for certain genetic operators.

Tribe The term tribe was used in the context of this work to distinguish individual units of coevolution from each other. It was chosen carefully because it is free of biological implications. Coevolution often describes reciprocal effects of different species on each other. However, in the context of this work, different non-sexually reproducing cell types or tissues of the same species are mostly to be studied, which makes the term species inappropriate. At the same time, there is a certain risk of confusion in referring to all coevolutionary units as populations. Since a coevolutionary unit also contains meta-information about its evolutionary history, the term tribe, which implies historicity, is an appropriate choice.

World The world contains all the information involved in an experiment, which are meta-information as well as the tribes, which are involved in coevolution.

3.2 Architecture

For the coevolution of multiple tribes, it is important to organise all individual networks, all the necessary meta-parameters and the results of the experiments in a compact data structure. Since the entire program must be concurrent for runtime reasons, there should be as few resources as possible shared by several individual networks. Another design goal is to be able to read the Boolean networks used for the experiments into SBML as external files and to write back the evolved networks at the end. For this reason, the implementation enforces the following fixed folder structure with the start data for each experiment.

3.2.1 Folderstructur

A folder must be created for each experiment that contains the subfolders with the individual populations as well as a global specifications file.

Experimentfolder

```
experimentfolder
|- tribe_0
|- ...
|- tribe_n
|- gobal_specificationsfile.txt
```

The experiment folder consists of one or more tribe folders. The tribe folders contain information about the individual tribes. The folders can have any name, but should end with _x, where x should be consecutive numbers beginning with zero. This serves to identify the tribes by their number. Furthermore, the folder must contain a .txt file with the global specifications. All specifications defined here are adopted for all tribes. If the tribes specify a parameter by his own, the global specifications are overwritten for the corresponding tribe. The format of the specifications file is described below.

Tribefolder

Each tribe folder must contain three subfolders.

```
tribe_0
|- input
|- output
|- specs
```

The input folder contains one to any number of .sbml files with Boolean networks. The given nets should end with _x.sbml, where x represent consecutive numbers from zero onwards. If the input folder contains fewer nets than the tribe specific population size provides for, the program will create the necessary missing nets by randomly changing the given ones. At the end of an experiment, the evolved networks can be returned as .sbml files. The output folder is intended for this purpose. The specs folder contains the specifications valid for the tribe. If specifications are already defined globally, the tribe specific ones will overwrite them. The specifications format is described below.

Specificationsfile

A specification file is a simple .txt file. Any number of parameters can be defined in a specification file, but each parameter must appear at least once in the tribe or the global specification. The tribe-specific values override the global values in case the parameters are set twice. The specification file is read in line by line. Lines beginning with a ”#” are ignored and can be used as comment lines. Empty lines are be ignored. In order to separate parameters and values, for the sake of readability, a space characters should be used, but ”:” and ”;” are also accepted.

The format for specifying the parameters is described below. The exact meaning and the definition of the different meta-parameters are described in chapter 3.2.2.

Attractor: or Chain: This parameter specifies the comparison type. Standing alone in a line, only the keywords “Attractor” or “Chain” are accepted. They be completed with a ”:”. However, this is only possible for reasons of coherence with the timeseries-module [39] and is not mandatory.

Initial conditions: The keyword must stand alone in one line. All following lines are then treated as lines in which a new initial condition is defined until a completely empty line or a new keyword follows. Each of the following lines defines a distinct start state. The fitness calculation is then performed separately for each of these states against the associated attractor and is then averaged. Any set of genes can be named (e.g. CycD) and will be interpreted as true. If a gene is preceded by a ”!” (e.g. !CycD), the gene is set as false in the initial state. Not all genes in the network must be defined. Genes that are not named are interpreted as wildcards and assigned with a random value for the fitness calculation.

State specifications: The format is the same as for the initial conditions. Any number of conditions that are part of the target condition can be specified in the following lines. This means, for example, that they should occur in an attractor. The order of the states is taken into account. Non-active genes are marked as false with a ”!” as prefix. If a gene is not specified in a state, this is interpreted as a wildcard. Such wildcards can be included in the fitness calculation in two different ways. The first approach is to assign randomly selected values to the unspecified genes before the calculation. In a second approach, all genes that are not explicitly specified in a target state are masked in the fitness calculation. Both variants are implemented in this framework.

Meta-parameters First the name of the parameter and then, separated by a space, the value of the parameter is given in the same line. All other parameters can now be defined with the same syntax in any order. The possible parameters are presented in detail in chapter 3.2.2.

It is possible to specify more than one pair of initial conditions and state specifications in one specification file. The assignment of initial conditions and the corresponding target states is done in the order of occurrence in the file. All states, after the first *initial-conditions* tag, are calculated in the fitness calculation against the target states after the first *state specifications* tag and so continuously. It does not matter whether all initial-conditions are specified first and then all state-specifications, or whether they are specified in pairs.

The following is a sample specification file, which contains tow sets of initial-condition / attractor pairs.

```
#default specifications for the experiment

#Mode and States
Attractor:
Initial condition:
!Gene1

State specifications:
!Gene1 Gene2 Gene3
!Gene1 !Gene2 Gene3

Initial condition:
Gene1 !Gene2
Gene1 Gene2

State specifications:
!Gene1 !Gene2 !Gene3

#Metaparameters
population-size 5
mutation-per-network 1
parents-num 2
fitness-thresold 1
generations-number 10
initial-state-pool-size 2
```

3.2.2 Datastructures

The project uses several records as data structures to bundle all Boolean networks, together with their metadata and results. These are briefly explained below.

World

```
(defrecord World [folderPath
                  tribe-list
                  world-gene-list])
```

The *World* record contains the path to the folder from which the experiment was started. The folder structure must follow the one presented in chapter 3.2.1. The tribe-list is a list of the individual tribes that are implemented as a *Tribe* record. The last value in the *World* record, the *world-gene-list*, contains the names of all genes that appear in the networks and is relevant for internal functions.

Tribe

```
(defrecord Tribe [folderPath
                  gene-list
                  specifications
                  TribeAnalytics
                  population-list])
```

A *Tribe* record contains all the information about a tribe, which are necessary for the evolution. These are firstly, the path, which specifies the folder where the Boolean networks and specifications of the tribe can be found. The second value is the *gene-list* which contains the names of the genes that must occur in all Boolean networks of the tribe. The *specification* entry and the *TribeAnalytics* contain the meta-parameters and are described in more detail below. Last but not least, all individual Boolean networks can be found in the *population-list*. In addition, to the actual transition functions, the individual network records also contains the path to the .sbml file from which the network was originally imported, the current fitness value and the meta-information of the network. Furthermore, there is a track that makes all mutations carried out on the network traceable.

Specification

```
(defrecord specification [comparison-type
                          inital-condition
                          states
                          population-size
                          mutation-per-network
                          mutation-percentage
                          parents-num
                          fitness-thresold
                          generations-number
                          initial-state-pool-size
                          path-size])
```

All meta-parameters necessary for the evolutionary algorithm are stored in the *specifications* record. Since the choice of these parameters has a considerable influence on the outcome of the coevolutionary experiments, all parameters and their range of definition are explained below.

comparison-type The parameter determines the target against which the networks of the tribe are optimised, or more precisely determines the mode with which the fitness of the Boolean networks is evaluated. A more detailed description of the fitness calculation has already been given in chapter 2.1.3. There are two possible values for the parameter, which are *Attractor* and *Chain*. The first one causes the target to be interpreted as the states of an attractor. The second one that these states should be part of the path, beginning from the initial states.

inital-condition The *inital condition* describes one or more sets of initial states from which the fitness of the networks is determined.

states The *states* parameter contains the set of target states, which are interpreted, depending on the comparison-type, as states which should be part of the path or, that should occur in the attractor, reached from the states defined in the *inital-condition*.

population-size The population size describes the number of Boolean networks a tribe contains. If the number is greater than the number of networks given in the input folder, the missing nets are generated from the existing ones. However, it should be avoided to provide more networks in the input folder than defined here.

mutations-per-network This parameter specifies how many mutations should be performed on each Boolean network per generation.

mutation-percentage This value must be an integer between 0 and 100 and describes the proportion of mutations to recombinations in percent. 100 means that only mutations are performed. Therefore, a value of 50 specifies that 50% mutations and 50% recombinations are applied. It is important to note that mutations can only be tracked in the mutation track if the value is 100 and only mutations are performed.

parents-num The parents number describes the number of networks that are used as parents in a recombination. This value must be at least 2 and must not exceed the population size.

fitness-thresold The fitness-threshold indicates the value above which networks with poorer fitness are not carried over into the next generation. Since this work is a minimization problem, the fitness-threshold can be seen as an upper fitness limit that an individual has to reach in order to be included in the next generation.

generations-number This value describes how many generations the tribe should develop. That is, how many passes through the evolutionary loop should be made.

initial-state-pool-size Describes the number of initial states from which the fitness is determined. This value should be at least the size of the actual maximum number of initial states per start/finish pair. Larger values are only used if not all genes have been specified in the initial conditions. In this case, true or false is randomly selected for the unspecified ones and the number of states specified in this parameter is randomly selected from these resulting states.

path-size The path-size specifies the maximum number of interactions that are performed on a Boolean network for fitness tuning. For most experiments performed, a network should reach an attractor within these steps, which finishes the calculation without the parameter becoming relevant.

TribeAnalytics

The *TribeAnalytic* contains a tracker for each parameter presented in the *specifications*. Each tracker is a map with a new entry for each change of the meta-parameter, using the age of the tribe as key. In addition, the *TribeAnalytic* contains the *TribeFitness* map, which tracks the average fitness of the tribe for each generation. The age of the tribe describes the number of runs through the evolutionary loop. The trackers of this data structure are primarily used to evaluate different experiments.

3.3 Coevolution Workflow

The separation of the tribes as described above, as well as the possibility to define global and tribe specific parameters leads to the possibility to simulate coevolution. A typical experiment follows a fixed workflow, which is called, in reference to the evolutionary loop, coevolutionary loop.

3.3.1 Coevolutionary Loop

The coevolutionary loop, see figure 3.1, extends the classical evolutionary loop by several parallel evolved populations and allows different interactions between them. The loop passes through the following steps.

1. The global specification file is read in and validated for formal errors. In addition, the folder structure of the experiment is checked and the number of tribes is determined.
2. This step and the following are executed in parallel for each tribe individually. First, the specification file of the tribe is read in and the *specification* data

structure is created. Global and individual metadata are merged and, if there is a double declaration, the global metadata is overwritten. Then a formal check of the parameters is carried out. If a meta parameter is not defined, an error message is displayed and the program ends.

3. Once the specification of a tribe has been created, the entire *Tribe* structure is generated. For this purpose, the networks in .sbml format are read in, validated and transferred to the internal network data structure. Based on the *population-size* defined in the specifications, additional networks may be created from the existing ones through slight mutation of the given ones. Next, the *gene-list*, which contains all genes present in the tribe, is generated. Finally, the *TribeAnalytic*, with the start metadata as first entry, is created.
4. After all *Tribes* are finished, the *World* data structure is created and the *shard-genes* list is determined.
5. The fitness of all individuals and in addition the average fitness of each tribe is calculated in parallel. The experiment set-up is then complete.
6. The actual coevolutionary loop begins. This can either run through a certain number of loops or be repeated as often as desired depending on a certain parameter, such as fitness. When the loop ends, the coevolved networks can be written in .sbml format to the output folder of his tribe and the tracker of *TribeAnalytics* can be used for evaluation.
7. After each run of the co-evolutionary loop, an interaction between two or more tribes can take place. For this purpose, update functions are available that can extract and change any metadata in the tribe specifications. These functions are described in more detail in chapter 3.3.5. Each change is noted in the *TribeAnalytic* track of the corresponding tribe. This makes it possible to track which and when changes occurred.
8. After a possible interaction between the tribes, the evolutionary algorithm is called in parallel on each tribe, with the currently valid, tribe specific specifications. Each tribe runs through the loop exactly as often as its *generation-number* parameter provides. In this way, it is possible for tribes to develop at a different rate or, to say it in other words, to age at different velocity. When the evolutionary algorithm ends, a new run of the coevolutionary loop begins at step 6.

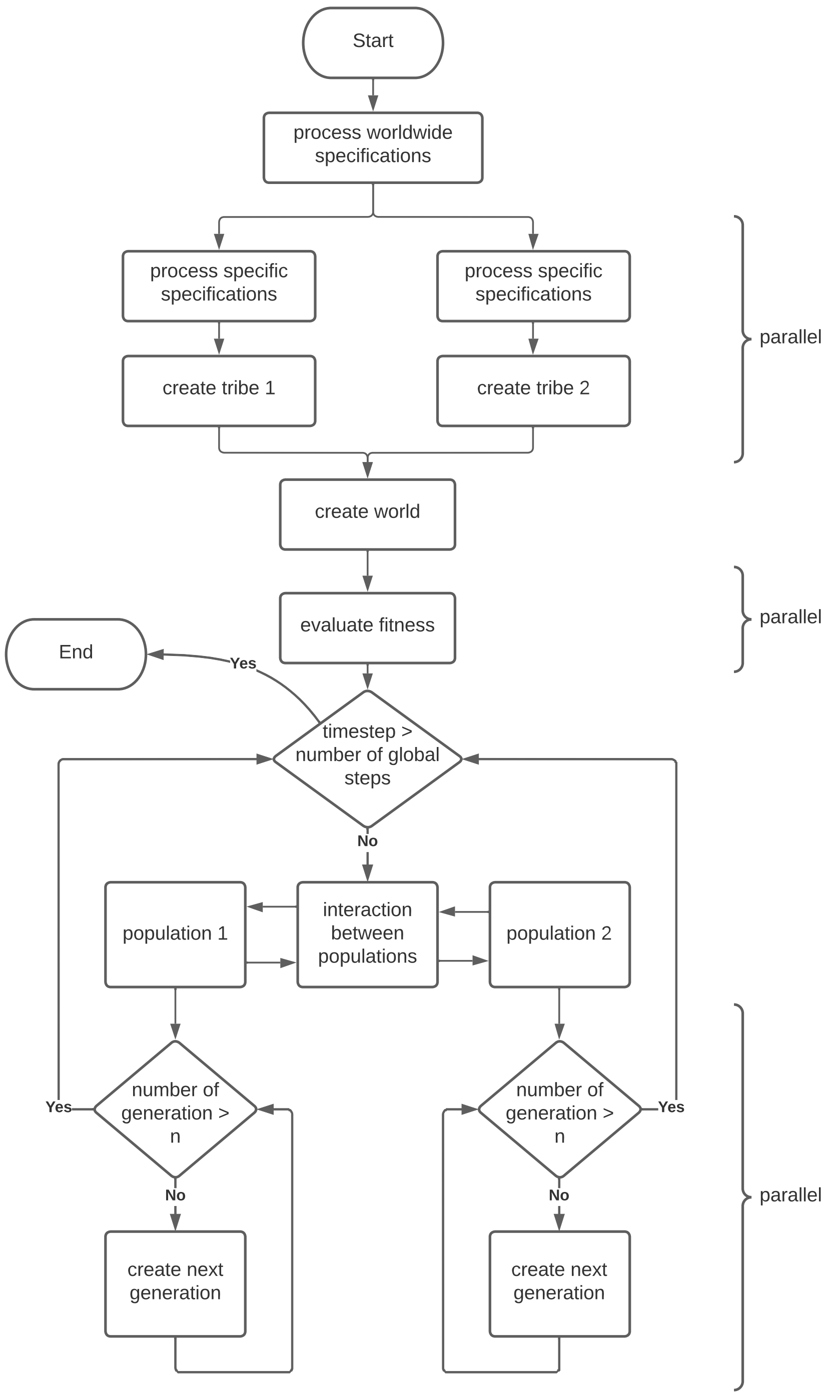


Figure 3.1: Coevolutionary loop for tow tribes.

Thereby the progress of the entire algorithm can be tracked using the timbre logging library[1]. At the level of evolutionary operators, progress can be tracked using the tracker-level *trace*. At the *info* level only the steps of the coevolutionary loop are tracked.

3.3.2 Parallel computing

Since this loop is highly computational, especially for many tribes, phases of interaction between the tribes and phases in which the tribes develop independently of each other in the sense of the evolutionary approach are clearly separated. This allows to call the evolutionary algorithm on the tribes in parallel. Since this algorithm also computes the single individuals simultaneously, the evaluation of the coevolutionary loop can be split up in an efficient way. For thread handling, the claypoole library[2] is used. This offers the use of either the future or pmap function [41]. A future returns an object, which is running in its own thread. A pmap on the other hand offers threading over the map function, which takes a list of objects and a function. The pmap applies the function on each object in a different thread [42]. In the context of this work, the pmap function was mostly used. Through the claypoole library the number of threads can be controlled. Moreover, the threadpool can be shut down in a safe way, after the computation is done [41].

3.3.3 Evolutionary operators and functions

To carry out the evolution of the individual tribes, several different operators, such as mutations, cross-over, survivor-selection and fitness calculation, are necessary. Some of these functions were already implemented in Aly's work and could partially be used [39]. However, some of behaviours of these functions are undesired had to be adapted or changed functionally. All changes were made in such a way that the original modules of the project remained unchanged and the framework presented here included all original experiments and modules. In the following, the most fundamental adaptations of already existing operators and functions will be presented,

[1]https://github.com/ptaoussanis/timbre
[2]https://github.com/TheClimateCorporation/claypoole

as long as they are not only to support the new data structures described in the previous sections but lead to changed functional behaviour.

Versions

The project is implemented in the programming language Clojure, which runs on the Java Virtual Machine as described in section 2.3. However, at the beginning of the project it was not totally clear which Java version the original project was based on. During the first test runs of the framework, several errors and decapitate warnings were immediately reported. Upon further investigation it was discovered that the original xml parser[3] was used in a beta version which was no longer supported and had several serious bugs. Since Clojure is a relatively young programming language and the framework developed by Aly is more than 5 years older than this work, it was found that several libraries had similar problems. After researching the previously reported bugs in Clojure[4], it was decided to update the project from Clojure version 1.8.0 to the latest version 1.11.1. In this step, all the libraries used were updated to the latest versions and the Java version 18.0.1 compatible with them was used.

Fitness Calculations

Since the original project did not assume that meta-parameters or specifications change during evolution, it could be assumed that a network always has a fitness that never changes. For this reason, the original implementation checked just before the actual calculation of the Hamming distances if a fitness value had been calculated for this network in the past. In this case, the calculation was interrupted without comment and the old value was returned. Since fitness values could change in the course of this work, for example due to changing initial states, the fitness calculation had to be re-implemented except for the calculation of the Hamming distances and their comparison.

Furthermore, the original implementation could not handle incompletely defined states in the target specification. In this work, two different variants were imple-

[3]org.clojure/data.xml 0.1.0-beta1

[4]https://clojure.atlassian.net/jira/software/c/projects/CLJ/issues

mented to calculate meaningful fitness values for such specifications. The first variant fills missing values with a random value before calculating the fitness of the single individuals. Therefore the optimisation problem becomes, by decreasing selection pressure, much more difficult. The second variant masks all genes that are not defined in one of the target states for the fitness calculation. This means that they no longer have any influence on the optimisation. The following function can be used to calculate the fitness of every tribe. The new fitness values are updated directly on the individuals and the mean fitness of the entire tribe is recorded in *TribeAnalytics*. The parameter mode can accept the values *"ignore-wildcards"* or *"add-wildcards"* to specify how to handle incompletely defined states.

```
(defn update-world-fitness
  [world wildcard-mode no-threads]
  ...
```

Mutations

To modify the selection pressure, it is essential for evolutionary algorithms to determine the number of mutations performed on an individual per generation. The framework provides the *evolve-generation* function in the *commonbusiness.Network Utility* module for this purpose. The parameter *numOfMutations* can be specified for this function. However, contrary to intuition, the associated comments and the function description, this parameter does not specify the number of mutations per network. Rather, it must be stated that the mutation operator works in such a way that exactly one mutation is always applied to the given network. This process is repeated as often as the numOfMutations parameter describes, i.e. instead of mutating a network *numOfMutations* times, the functionality returns *numOfMutations* many networks with exactly one mutation. The number of individuals is then corrected by the immediately following survivor selection. Furthermore, the Clojure function *repeat* is used to generate these networks. However, since the type of mutations depends on a random function, such a call does not apply different mutations, as the *repeadly* function would have done, but always the same mutation on all networks. Since this behaviour is not desired for the present work, the mutation function was re-implemented.

3.3.4 Survivor Selection

The original framework contained only one form of survivor selection. In the sense of truncation, this selection only took the best individuals into the next generation. To allow different types of selections, the following function was implemented.

```
(defn survivor-selection
  [tribe sortedHammingDistances selection-modus]
  ...
```

This function takes a tribe and a map with the numbers of the individuals as the key and their fitness as the value, sorted with ascending fitness. The selection-mode parameter determines how the selection is to be made. The following three parameters are possible.

”fittest”	In this mode, only the fittest population-size individuals are carried over into the next generation. This mode corresponds to the truncation survivor selection.
”elitism”	In this mode, 80% of the individuals are drawn, as in the fittest selection, according to fitness. The remaining 20% are selected purely at random from the remaining individuals.
”roulette”	As described in Section 2.1.3, this mode is carried out by, metaphorically speaking, spinning a wheel-of-fortune population-size many times. The chance of an individual being selected depends on its relative fitness. This selection procedure was realised as follows. Firstly, the individual fitness values were inverted, that means that fitter individuals have a greater fitness score. Secondly, a hash-map is generated with the relative frequency as key and the number of the individual as value. This data structure is then converted into a TreeMap [5] originating from Java. This data structure, implemented as a red-black-tree, allows the retrieval of a key with effort log(n), even if not the actual key but only the next larger existing key to a random value is searched. Repeatedly retrieving an entry with such a randomly generated key generates the next generation.

[5]https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html

3.3.5 Update functions

The framework provides numerous functions that enable all kinds of interaction between different tribes. Central to this is the update-world function.

```
(defn update-world
  [world tribe-num spec-to-change new-value no-threads]
  ...)
```

This function takes a world, the number of the tribe to be changed, the parameter to be changed and the new value and returns the world with updated specifications. Each change made in this way is recorded in *TribeAnalytics* along with the age of the tribe when the change was made. This means that every change can be reconstructed.

The following code snipped shows the use of update-world. This Example updates the *World* world, by changing the population size of tribe number 1 to 10.

```
(def world (update-world world
                         1
                         :population-size
                         10
                         no-threads))
```

Initial state exchange

For the exchange of initial start states, the *get_new_inital_condition* function is available to determine the states or partial states.

```
(defn get_new_inital_conditions
 [world tribe cond-num shared_genes modifier no-threads]
  ...
  )
```

This function is based on the assumption that tribes, which biologically can be understood as several cell assemblies, interact through the exchange of transcription products, such as messenger substances. The messenger substances produced by a cell association depend on the phenotype of the cells. Since this can be identified with an attractor of the network [2], it is plausible to select a state from one of the attractors of the donor tribe and use this state as new initial state of the acceptor tribe. The function realises this by selecting a random state from the fittest attractor of the donor tribe. Since such a state is evolutionarily preferred, it is also achieved by a large proportion of the individuals of the tribe. The *cond-num* parameter defines from which initial-state/attractor pair the state is to be selected and the *shared_genes* parameter determines which genes are to be selected from this state. The following modes are available.

"all" This parameter causes all common genes of the tribes to be used for the exchange. The modifier has no influence in this mode.

"rand" In this mode, a random subset of the shared genes is used for exchange. If the modifier is between 0 and 1, each gene is selected individually with the probability given by the modifier. If the modifier is an integer >=1, exactly as many genes are randomly selected as specified by the modifier.

(list Gene1 Gene2 ...) The last option is to specify a list of genes that are the basis for the exchange. Here, a modifier between 0 and 1 can be used to choose a gene with the probability given by the modifier. If the modifier is equal to 1 each of the genes in the list will be used.

For updating the initial states, the generic update-world function can be used, but the more expressive *update-initial-states* function should be used. This function offers the update in two different modes, defined through the mandatory mode parameter and the optional max-states argument.

```
(defn update-initial-states
  [world cond-num tribe-num new-value no-threads
   mode & max-states]
  ...
  )
```

The mode argument can have following the values.

"overwrite" The *overwrite* mode causes the last currently existing initial state to be overwritten by the given state. If the last or the new state does not specify all genes, a union is performed. All genes already specified are retained and these are supplemented by the new ones. Old genes that reappear are overwritten. If you call the update-world function to change *:initiale-conditions*, an information is displayed and *overwrite* mode is executed.

"append" In append mode, the new states are appended to the existing states. If the *initial-state-pool-size* is smaller than the number of initial states, the value is adjusted so that all initial states are actually used for the fitness calculation. In this mode, the optional parameter *max-states* can also be specified. This has the effect that only the last *max-states* much are retained after the new states are appended. This works when adding a new initial state several times with *max-states* = 2, that always the newest and the first, i.e. the initial initial-state, are kept.

All updates carried out by the function are tracked in the *TribeAnalytic*.

3.3.6 Analysis Tools

In order to evaluate conducted experiments, the framework provides some functions to prepare certain data. Besides the fitness of the tribes in each generation, every change of the parameters is documented in the *TribeAnalysis*.

Fitness

For the evaluation of the fitness histories, the *TribeFitness* of each repetition of an experiment can be output as a file in JSON format. With the help of some of the R scripts included in the project, these can be easily used to calculate the mean value of all repetitions and the results can be prepared graphically for analysis.

Furthermore, there are some functions that allow certain information to be extracted from the coevolved *world* datastructure after an experiment.

(get-fittest-individual tribe) This function returns the number of the individual of the given tribe that achieves the best fitness.

(get-fitness-of-state tribe individual state) With the help of this function, the achieved fitness of an individual can be determined from any given start state.

(compare-tribe-to-specification tribe) Calculate the fitness of the tribe, based on the current initial-states.

(get-attractor-with-best-fitness-in-tribe tribe) This function returns the attractor with the best fitness, as a list of states, that can be reached from any individual and any starting state.

Attractors

The following functions are available to determine different specific attractors that are relevant for coevolutionary experiments.

(get-attractors-from-state tribe individual-number state) Returns the attractor that a individual reaches from the given start state.

(get-sequence-from-state tribe individual-number state) Outputs all states, which an individual passes through up to the attractor, starting from the given start state.

(get-attractors-from-initial-conditions-track tribe individual-number) This function returns all attractors, as state list, that a individual can reach from each state in the *initial-conditions-track* of his tribe.

Since it is important for some experiments to be able to easily compare and name attractors without having to specify all their states, the possibility was created to represent an attractor as an integer value. Therefore, the states of the attractors are interpreted as a binary string and represented as an integer to the base 10. Each attractor is then named after the smallest integer value of its states. This mapping is bijective for a fixed network. The framework provides functions to switch between representing attractors as a list of states, optionally as a Boolean or integer vector, and representing them as a single integer attractor name. The following exemplary function makes use of this representation.

(attractor-list-boolvec->int attractor-list) This function takes a list of attractors, represented as a list of boolean vectors and returns a list of states represented as an integer for each attractor. A similar function also exists for a representation of the states as a binary string.

(attractor-list-boolvec->attractorname attractor-list) Returns, as described above, a list of attractors represented as their representative integer name.

(attractor-names-from-initial-conditions-track tribe) This function works similar to the *get-attractors-from-initial-conditions-track* function, but it will return just the attractors reached from the fittest individual in the tribe. This makes sense in many cases because the tribes converge very fast and so the reached attractors are very similar for all individuals. Moreover, the returned attractors are named as a integer.

With the help of these functions, it is easy to investigate whether and which initial states lead to the same attractors. This information is interesting for the analysis of the robustness of the coevolved networks and was used among others for the experiments in chapter 4.5.

R scripts

The project also contains several R scripts that serve to evaluate and visualise the data obtained. Firstly, there is a script that averages and visualises the the finesses of all tribe over all runs. This script uses the .json files that are automatically generated by the experiment functions of the framework. Furthermore, the experiments were designed in such a way that the fittest network of each tribe can be saved as an .sbml file at the end of each experimental run. These networks can be read in with the help of another R script and the adjacency matrices of these networks can be generated. These allows an evaluation in two ways. On the one hand, the difference network to the original network can be calculated by subtracting the original from the coevolved network. Secondly, by applying an AND to each edge, only those edges are kept that are found in both the original network and the coevolved network. The average of all runs can be visualised as heatmap.

3.3.7 Example experiment

The following simplified code shows the setup of an exemplary experiment. Here tribe0 shares a subset of a state of its fittest attractor with tribe1 every loop. The meaning of the parameters was described in more detail in the previous chapters.

This example uses only one initial-states / attractor pair. Therefore, the parameters added for this have been omitted for ease of reading. The folder parameter specifies the path to the experiment folder. This must be structured as described in section 3.2.1. The *steps* parameter defines the total number of coevolutionary loops. Furthermore, any random function can be determined. This is used to generate all random values in the course of the experiment. By default, the rand-int function can be used. The number of threads made available to the experiment can be set by the *no-threads* parameter.

```
1  (defn experiment_1
2    [folder steps
3     random-function no-threads
4     selection-mode wildcard-mode
5     shared_genes modifier share_mode max-states]
6    (let [world (create-world folder
7                              random-function
8                              no-threads]
9     (loop [timepoint 0
10           world (update-world-fitness world
11                                       wildcard-mode
12                                       no-threads)]
13      (if (> timepoint steps)
14        world
15        (let [world (evolve-world world
16                                  selection-mode
17                                  wildcard-mode
18                                  random-function
19                                  no-threads)]
20          (let [new_initial_cond
21                 (get_new_inital_conditions world tribe0
22                                            shared_genes
23                                            modifier)]
24            (let [world (update-initial-states
25                                        world
26                                        1
27                                        shared_genes
28                                        modifier
29                                        no-threads)]
30              (timbre/info timepoint (get_tribe_fitness 1)
31              (recur (inc timepoint) world))))))))))
```

First, the world is created from the Boolean networks and specifications in the experiment folder (line 4). The coevolutionary loop begins in line 6. To initialise the *world* for the first time, the fitness of all Boolean networks must be calculated one time (line 10). As long as the loop is running, the evolutionary algorithm is called on each tribe every pass (line 15). Therefore, one of the three modes for survivor selection must be selected. Subsequently, a state of tribe0 is determined (line 20) as described above and the *inital-conditions* of tribe1 are updated by overwriting the old initial-conditions (line 24). In order to follow the progress of the algorithm, the time and the fitness are output each loop pass (line 30). After the loop is completed, the coevolved *world* is returned (line 14).

4 Results

In the context of this work, a framework was developed that allows to perform different experiments to study the coevolution of several populations of Boolean networks under different conditions. This framework was used to conduct the following experiments.

4.1 Initial state exchange

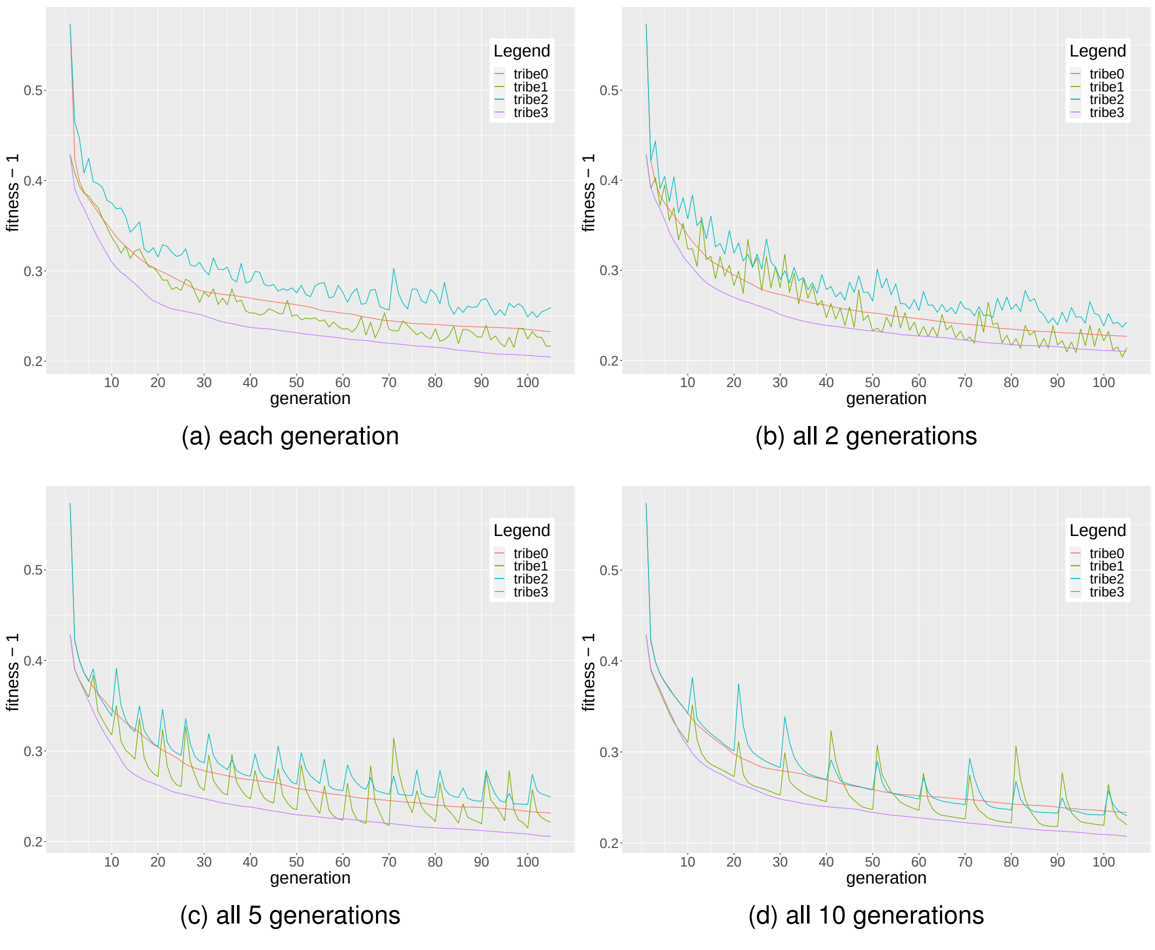
For this series of experiments, two tribes with ten randomly generated NK networks $(N = 10, K = 5)$ were created using the tool BoolNet. For each of these tribes, a copy was created as control group. Thus, *tribe0* and *tribe2* and likewise *tribe1* and *tribe3* were identical. In each run of the coevolutionary loop, tribe0 has shared a state of its fittest attractor with tribe1 as its new initial state. In the process, the old states were overwritten by the new ones. The same applies to tribe3 and tribe2. Therefore, see also sketch 4.5.

$$\begin{aligned} tribe0 &= tribe2 \\ tribe1 &= tribe3 \\ \\ tribe0 &\xrightarrow{init\_state} tribe1 \\ tribe3 &\xrightarrow{init\_state} tribe2 \end{aligned} \tag{4.1}$$

Here, all four tribes start in the same randomly chosen starting state and optimise against the same randomly chosen attractor of size six. Likewise, all parameters concerning coevolution are identical (*population-size* 10, *mutation-per-network* 1, *mutation-percentage* 100, *fitness-thresold* 1). The experiments differ in the number of generations after which the initial states are exchanged and in the subset of genes which are therefore used. Each experiment was repeated 50 times. The mean values of the fitness over all runs are shown in the figures below.

4.1.1 Sharing all genes

In this series, each run of the coevolutionary loop exchanged all the genes of the state. The exchange was carried out after 1, 2, 5 and every 10 generations.



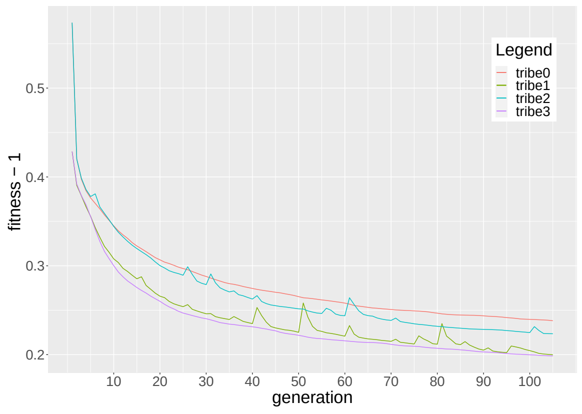
(a) each generation (b) all 2 generations

(c) all 5 generations (d) all 10 generations

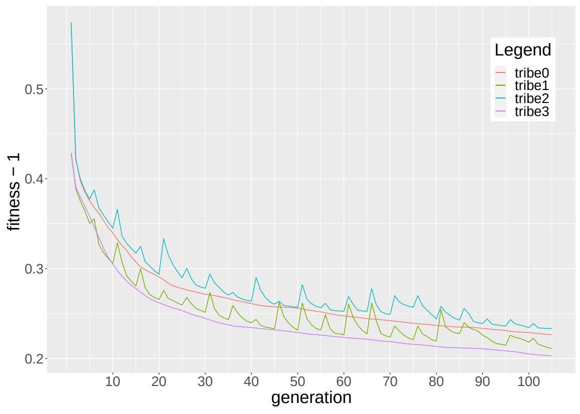
Figure 4.1: Initial state exchange - all genes shared

4.1.2 Share genes partially

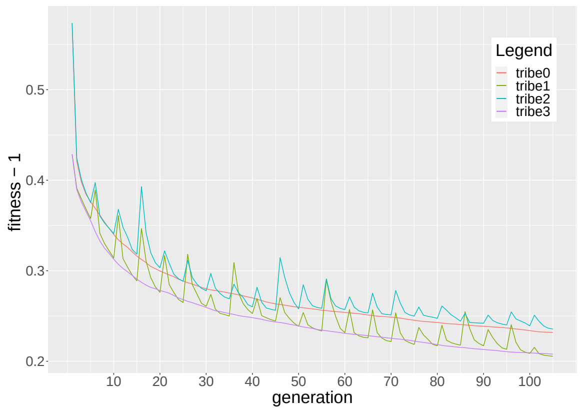
In this series, an exchange was carried out every five generations in each experiment. In part a), each gene was exchanged with a probability of 1, 10 and 25% in each exchange. In part b), only a fixed subset of genes was exchanged. For this, 3, 5 and 7 of the ten genes were fixed and exchanged every five generations.



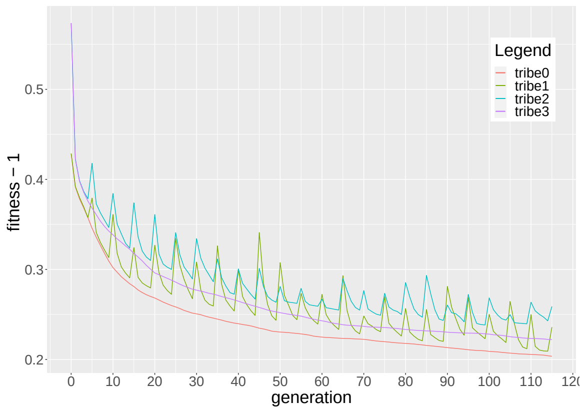
(a) probability 1%



(b) probability 10%

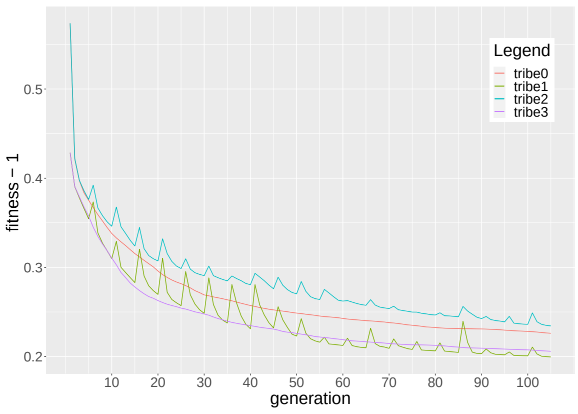


(c) probability 25%

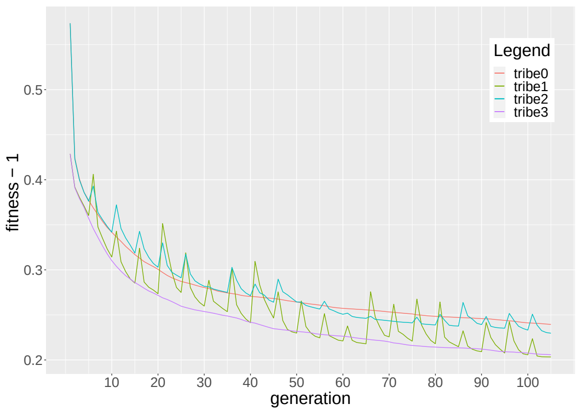


(d) probability 50%

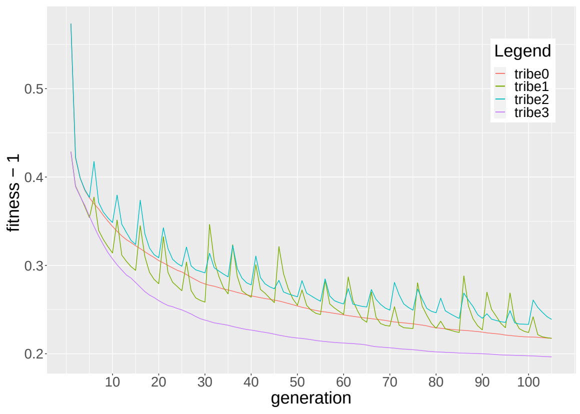
Figure 4.2: Initial state exchange - genes shared with a fix probability to include a gene.



(a) gene1 - gene3



(b) gene1 - gene5



(c) gene1 - gene7

Figure 4.3: Initial state exchange - include a fix subset of genes.

4.2 Mutual state exchange

In the experiments of series two, the set-up was the same as in the initial state exchange experiment 4.1. That means, 10 randomly generated NK(10,5) nets in each tribe with the same fixed parameters (*population-size* 10, *mutation-per-network* 2, *mutation-percentage* 100, *fitness-thresold* 1) and the same size six attractor as target was used. The difference to the previous experiment is that for the exchange of the initial states, one tribe was not specified as a fixed donor and one tribe as a fixed acceptor, but for each exchange, the fitter one of the two tribes served as the donor and the less fitter one as the acceptor. The exchange of initial states took place every five generations.

$$
\begin{aligned}
&tribe0 = tribe2\\
&tribe1 = tribe3\\
\\
if \quad &fitness(tribe0) < fitness(tribe1)\\
&tribe0 \xrightarrow{init\_state} tribe1\\
else&\\
&tribe0 \xleftarrow{init\_state} tribe1
\end{aligned}
\tag{4.2}
$$

This experiment was repeated 50 times. The mean values are shown in figure 4.4.

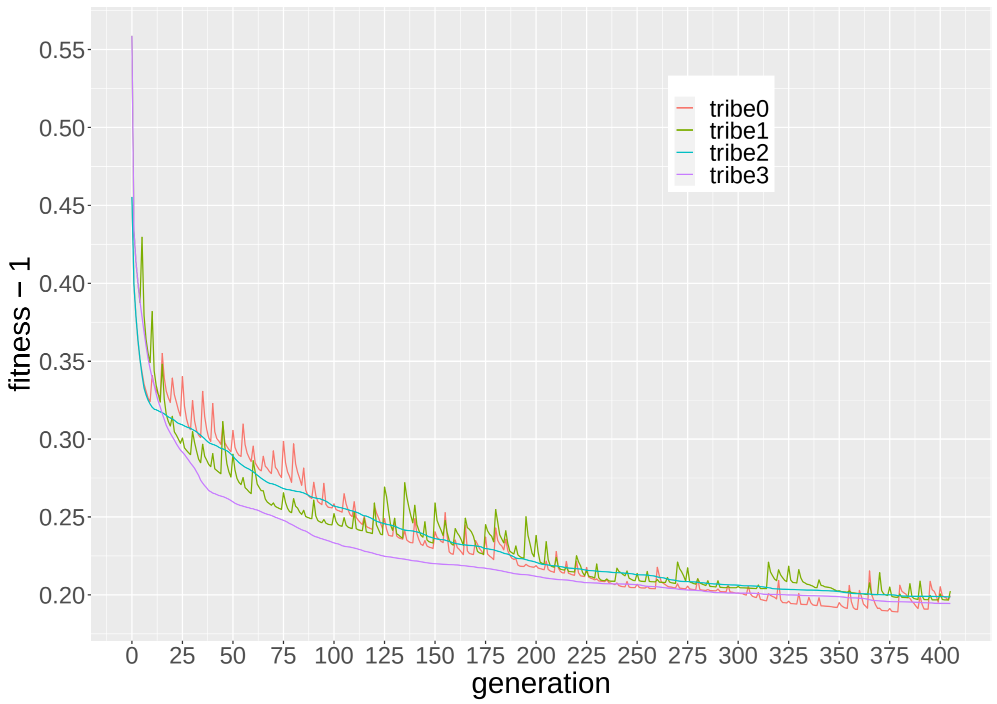


Figure 4.4: Averaged fitness curve over 20 repeats. Tribe0 and tribe2 and likewise tribe1 and tribe3 are identical. The tribes 0 and 1 share initial states dependent on there fitness.

4.3 Evaluation of selection modes

In this series of experiments, the influence of the 3 selections modes, truncation, elitism and roulette, which are implemented in the framework (3.3.4) was investigated. For this experiment, two tribes with 10 NK(10,5) networks each were created and one copy of each of these tribes was made. Here, tribe2 is a copy of tribe0 and tribe3 is a copy of tribe1. The meta-parameters are identical for all four tribes (*population-size* 10, *mutation-per-network* 2, *mutation-percentage* 100, *fitness-thresold* 1). The exchange of initial states is the same as in the previous experiments. That means from tribe0 to tribe1 and from tribe3 to tribe2. Each experiment was followed for 400 generations and the fitness values averaged over 50 runs.

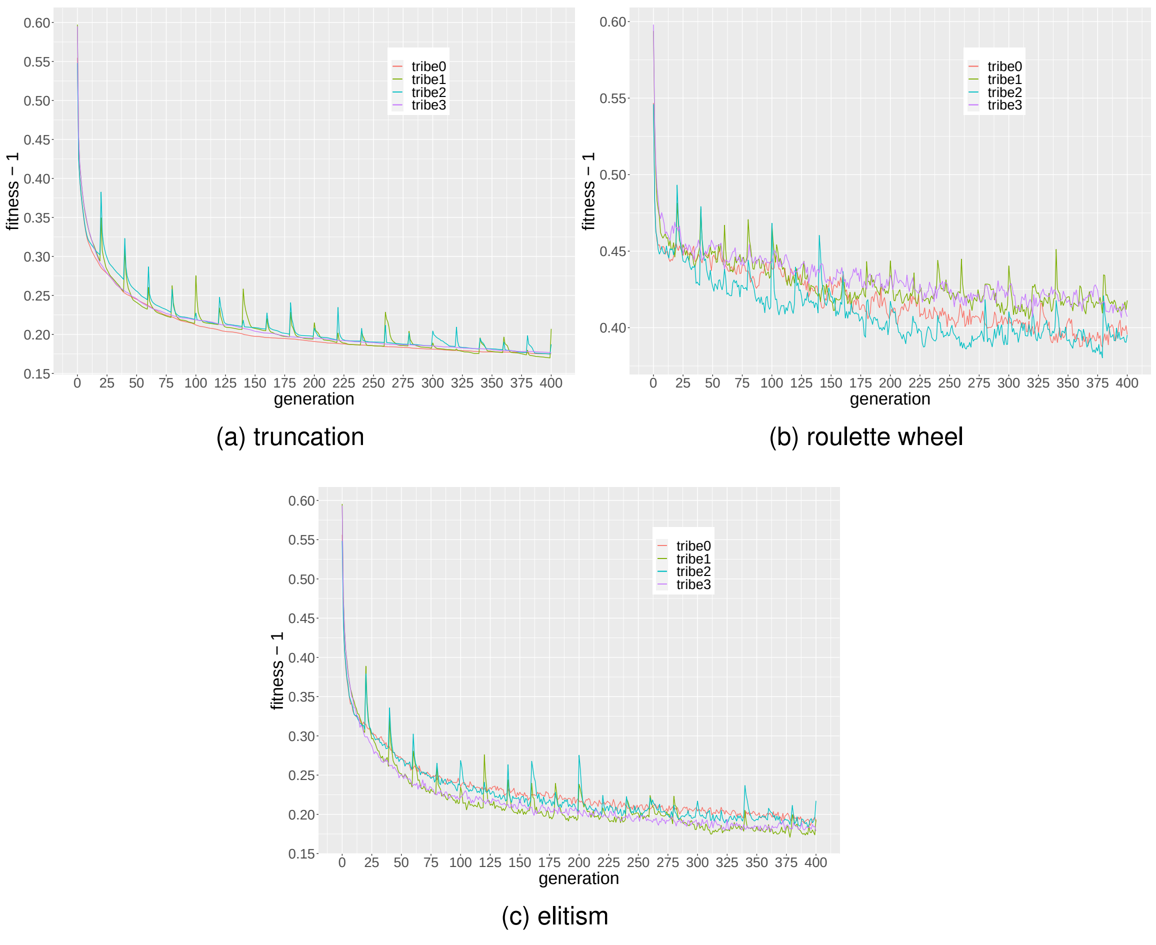


Figure 4.5: Different survivor selection modes, mean fitness curve over 50 runs.

4.4 Evaluation of the runtime of the program

In the following experiment, the runtime of the program was investigated as a function of the available number of computing cores. For this purpose, four tribes with ten randomly generated NK(5,2) networks each, were coevolved for 100 generations. Thereby two of the tribes exchanging initial states with each other every five generations. Such a run was repeated ten times and the duration averaged over the runs. The experiment itself was performed on an ARM architecture with 256 cores (ThunderX2 99xx, ~1.5TB memory).

In a first version of the program, the parallelisation was done nested. In the way that first a separate thread was called for the coevolution of each tribe. Each of these threads then called a separate thread for each individual network for parallel mutation and fitness evaluation. In order to investigate the influence of a nested parallelisation on the runtime, the threads was no longer called nested but only applied at the one hand at the level of the tribes and on the other hand at the level of the single individuals. The results are shown in figure 4.6.

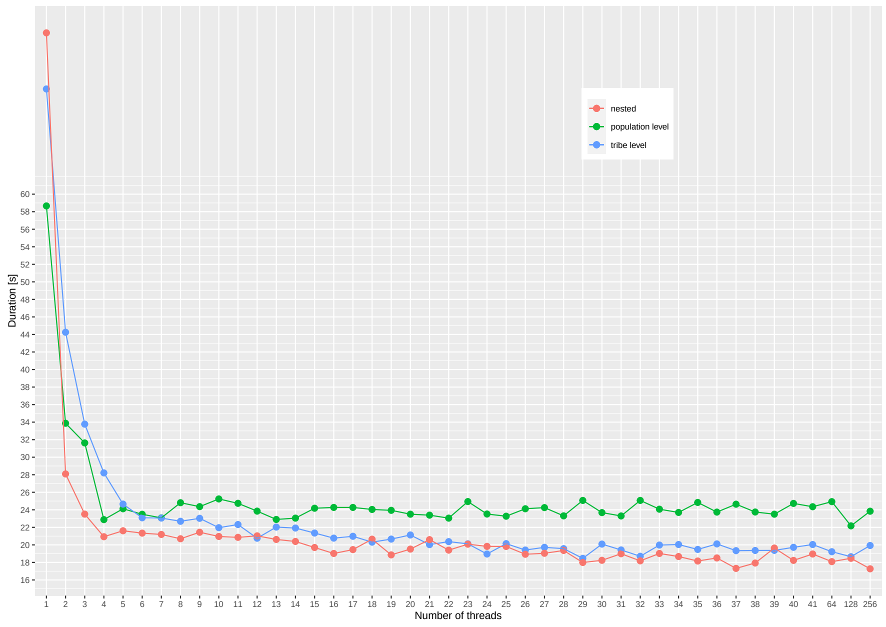


Figure 4.6: Influence of the available cores on the runtime of the program. Parallelisation was done: 1. exclusively at the level of individuals (population level), 2. at the level of the tribes and 3. nested at both the tribe and individual level (nested).

4.5 Influence of different coevolutionary sharing strategies

For this series of experiments four identical tribes was created, each consisting of the same ten random generated NK(10,5) networks. All tribes had the same fully defined initial state and the same six-state randomly generated attractors as the optimisation target. The evolution parameters for all tribes were identical (*population-size* 10, *mutation-per-network* 1, *mutation-percentage* 100, *fitness-thresold* 1). Tribe0 was the control group, which was not influenced by any other tribe. Tribe0 shared a randomly selected state of its fittest attractor with the other three tribes every five generations. Therefore, tribe1 overwrote its initial-condition every time the state was exchanged (*overwrite*). Tribe2 extended its initial-conditions with the new condition and kept all previous ones (*append*) and tribe3 appended each new state to the existing ones, but kept only the most recent four states (*append 4*). Therefore, a state is maintained as the initial state for 20 generations, a period in which a significant effect can be assumed.

$$tribe0 = tribe1 = tribe2 = tribe3$$

$$\begin{aligned} tribe0 &\xrightarrow{overwrite} tribe1 \\ tribe0 &\xrightarrow{append} tribe2 \\ tribe0 &\xrightarrow{append\ 4} tribe3 \end{aligned} \tag{4.3}$$

The experiment was carried out over 240 generations. Next to the fitness curve (2.1.3), it was investigated for tribes 1, 2 and 3 at the end of coevolution what attractors the fully evolved tribes reach starting from all the initial states that occurred in the course of the coevolution. This serves to determine if networks that have optimised themselves starting from one initial state destroy the old trajectory to the desired target attractor after a change in the initial states occurred. This would lead to different attractors being reached from the two states. This experiment was repeated 25 times and shows how many different attractors the tribes reach on average from all the initial states in their track. The results for the different tribes and sharing modes are shown in figure 4.8.

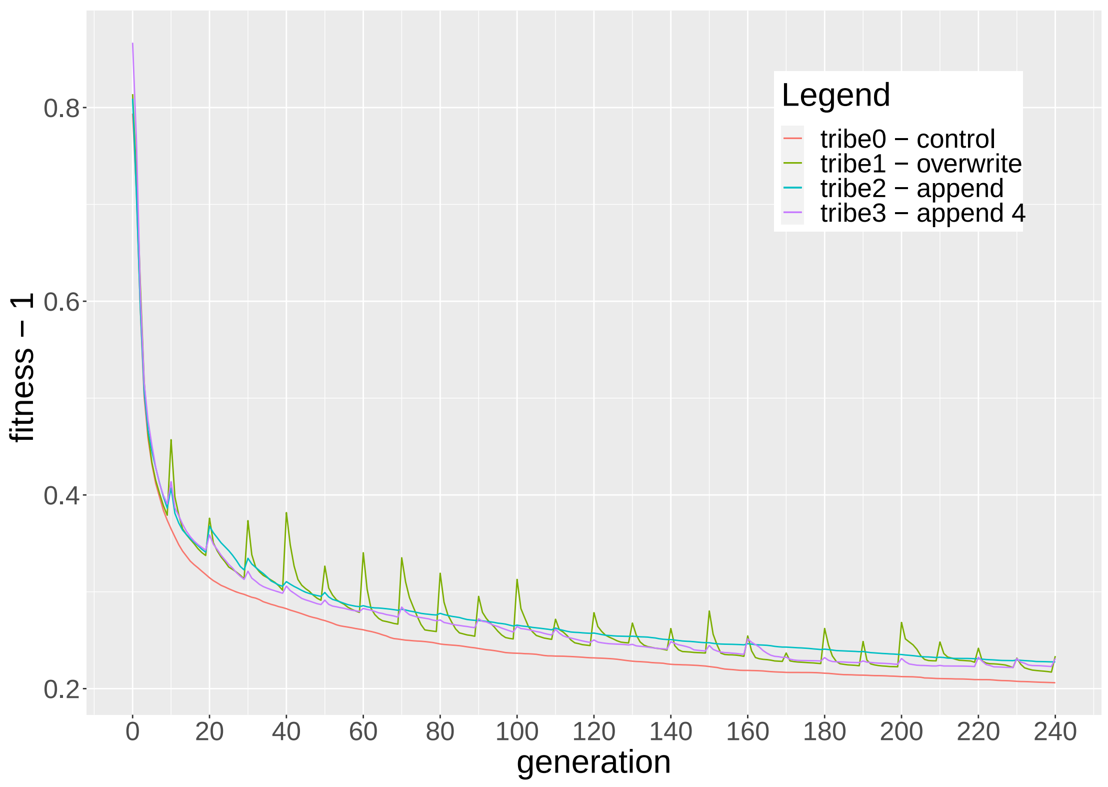
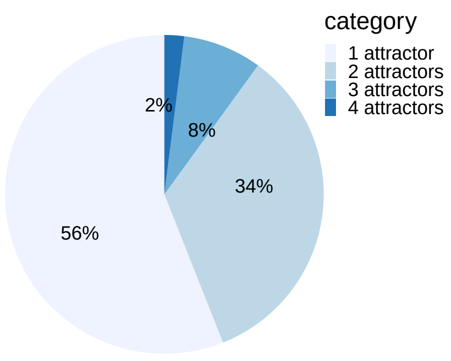
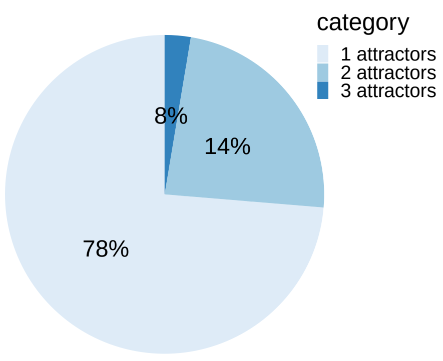


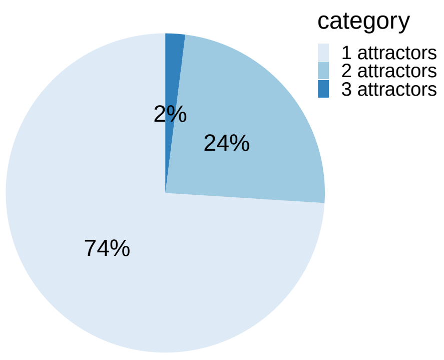
Figure 4.7: Fitness curve for initial conditions with control group (tribe0) and the exchange strategies *overwrite* (tribe1), *append* (tribe2) and *append 4* (tribe3).



(a) overwrite



(b) append



(c) append 4

Figure 4.8: Number of attractors - with exchange strategies *overwrite*, *append* and *append 4*.

4.6 Biologically Plausible Experiment

To test the framework with biologically plausible Boolean networks, two networks were selected that share a subset of genes. The first network is based on a modified variant of the igf - Wnt crosstalk (igfWnt) pathway. The exact network specification can be found in appendix A.1. The second network is based on the colorectal cancer (CRC) signal path, which specifications can be found under A.2. The experiments also investigated how the static relationships of the genes change during the coevolution. For this purpose, differences and commonalities in the adjacency matrices of the original networks compared with the coevolved ones were investigated. The Adjacency matrices of the original networks are shown in annex fig. B.1 and B.2.

For the igfWnt network, four fixed initial state / attractor pairs have been defined (A.1.1). Each of these pairs consists of a state that represents both the initial state and the target attractor. In a first series of experiments, these four states are defined as the exact inverses of the four single state attractors of the network. Later, exactly the four attractors were selected for these states, so that the network had an optimal fitness from the beginning and thus did not change. For the CRC network the fixed initial state, which can be found in the appendix A.2.1, was defined for all experiments. However, several differently defined target attractors were used. All this one state attractors have in common that out of a total number of 45 genes, only a few have been assigned to fixed values. The first attractor has 6 out of 45 genes defined. In the second the S6K and RSK markers of EGFR/ERK activity and in the third WNT inactivation is added. For the fourth attractor, the markers of EFGR/ERK and WNT are combined. All the definitions can be found below.

```
Attractor1:
mTORC1 ERK !BCAT !TSC2_1 !cMYC !AXIN2

Attractor2:
mTORC1 ERK !BCAT !TSC2_1 !cMYC !AXIN2 S6K RSK

Attractor 3:
mTORC1 ERK !BCAT !TSC2_1 !c_MYC !AXIN2 FZD DVL GSK3B_deg
GSK3B_DC APC AXIN !TCF_LEF

Attractor 4:
mTORC1 ERK !BCAT !TSC2_1 !c_MYC !AXIN2 FZD DVL GSK3B_deg
GSK3B_DC APC AXIN !TCF_LEF S6K RSK
```

Because of the non defined genes within the attractor definitions, the fitness calculation can either be done in such a way that the missing values for the fitness calculation are either randomly added or masked. Both variants were investigated in this work. In the first sequence of experiments (4.6.1), random addition of the undefined genes was performed and in the second series (4.6.2), the missing genes were masked for the fitness calculation.

In all of the experiments, the igfWnt population serves as donor tribe, which shares a state of its fittest attractor with the CRC networks all 100 generations.

The two networks have nine genes in common, which are listed below.

```
AXIN BCAT ERK GSK3B_DC JNK PI3K RAC1 TSC2_1 mTORC1
```

Consequently, for the exchange of the initial state, only the condition of these nine genes was used, taking into account all genes in each exchange and overwriting the initial state of the acceptor tribes.

4.6.1 Addition of missing gene values

For the following experiments, all missing assignments within the attractor definitions were completed by random values before each fitness calculation. As already mentioned, four different attractor definitions were investigated for the CRC network.

For every attractor definition the fitness of the tribes was tracked over 2500 generations, with an exchange of initial states every 100. The CRC tribe consisted of 30 individuals and in each generation up to five random mutations occurred per network. For calculation reasons, only ten individuals per tribe with up to one mutation per generation were used for the igfWnt network. The experiment was repeated 20 times and the fitness values, averaged over all runs are shown in figure 4.9.

In addition to the fitness values, the fittest network of each tribe was saved in *sbml* format after each run of the experiment. Using an RScript the adjacency matrix was determined for each of these networks and compared with the unmodified original networks. The adjacency matrix of the original igfWnt and the CRC network are shown in figure B.1 and B.2. First, the differences to the initial networks were determined by subtracting the adjacency matrix of the original from the matrix of the coevolved networks. In this way, the changes that have occurred in the course of coevolution become apparent. The results of attractor two are shown in figure 4.10 and 4.12. In addition, an AND operation was applied to the adjacency matrix of the experimental and the original networks in order to detect any common edges that

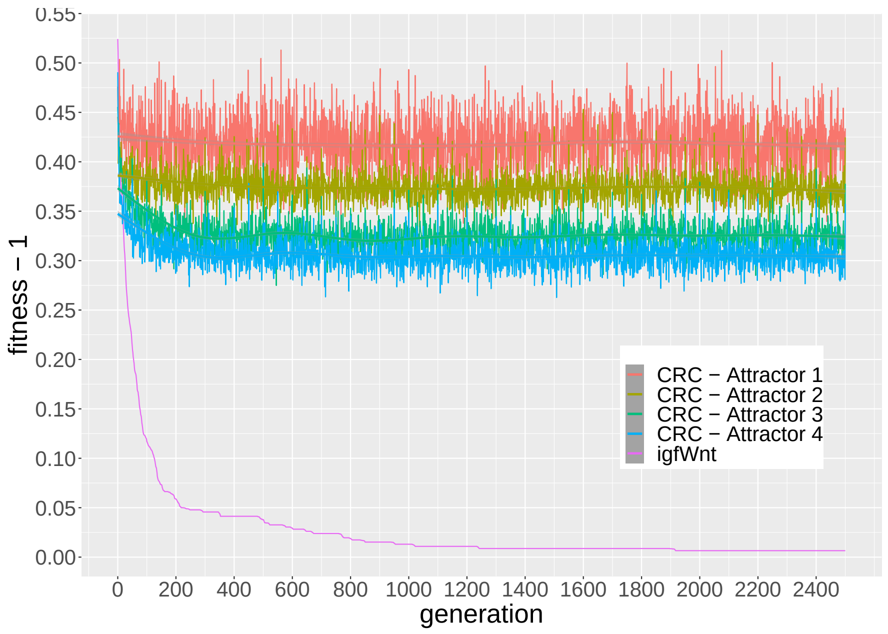


Figure 4.9: Fitness curve for the igfWnt/CRC network and the different attractor definitions for the CRC networks.

may still exist. As example the results of attractor two are displayed in figure 4.11 and 4.13.

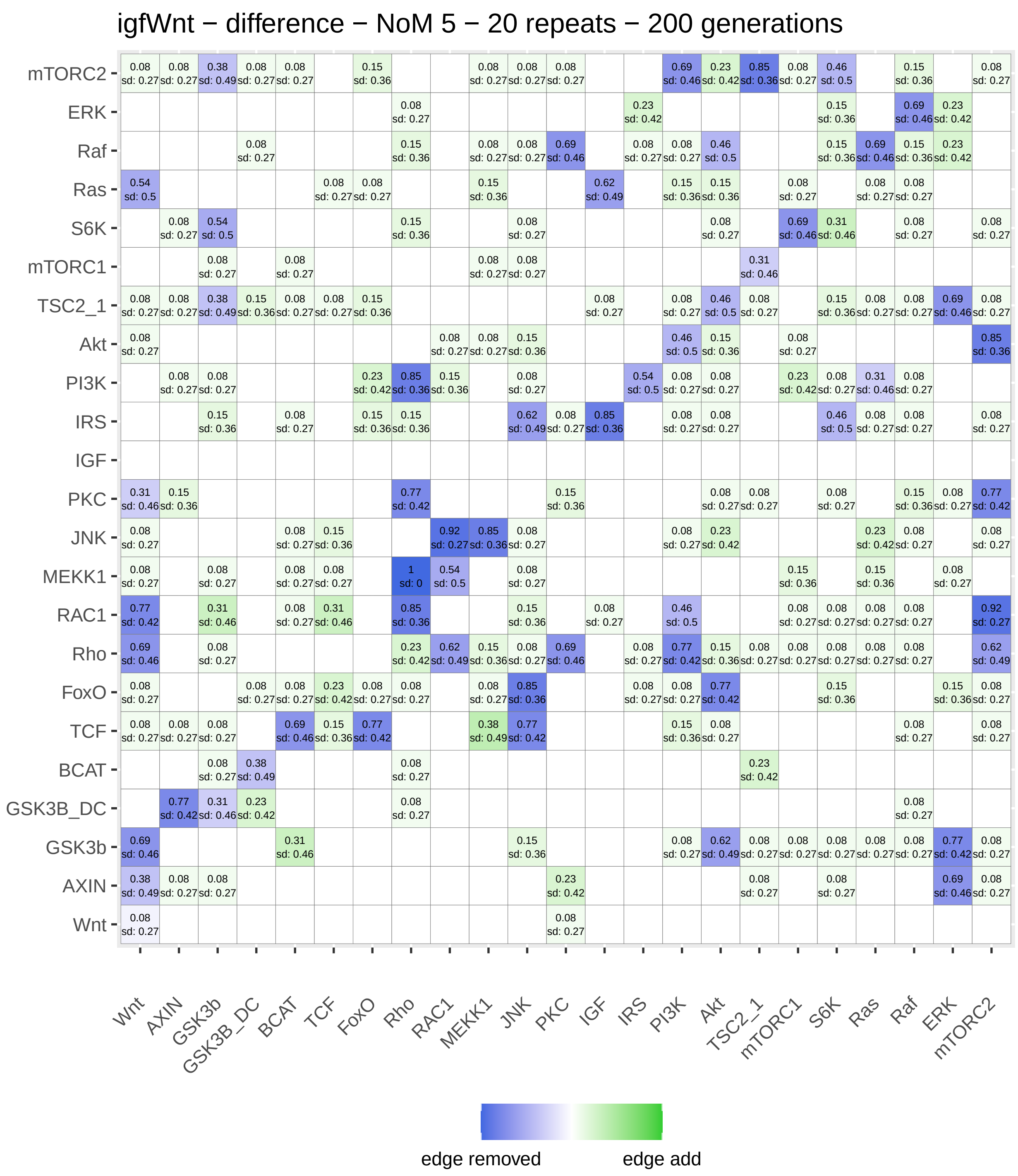


Figure 4.10: Differences adjacency matrix of the coevolved igfWnt networks compared with the original ifgWnt network. Deleted edges are in blue, in green added edges. In white are edges with no differences between the networks. The labelling shows mean and standard derivation.

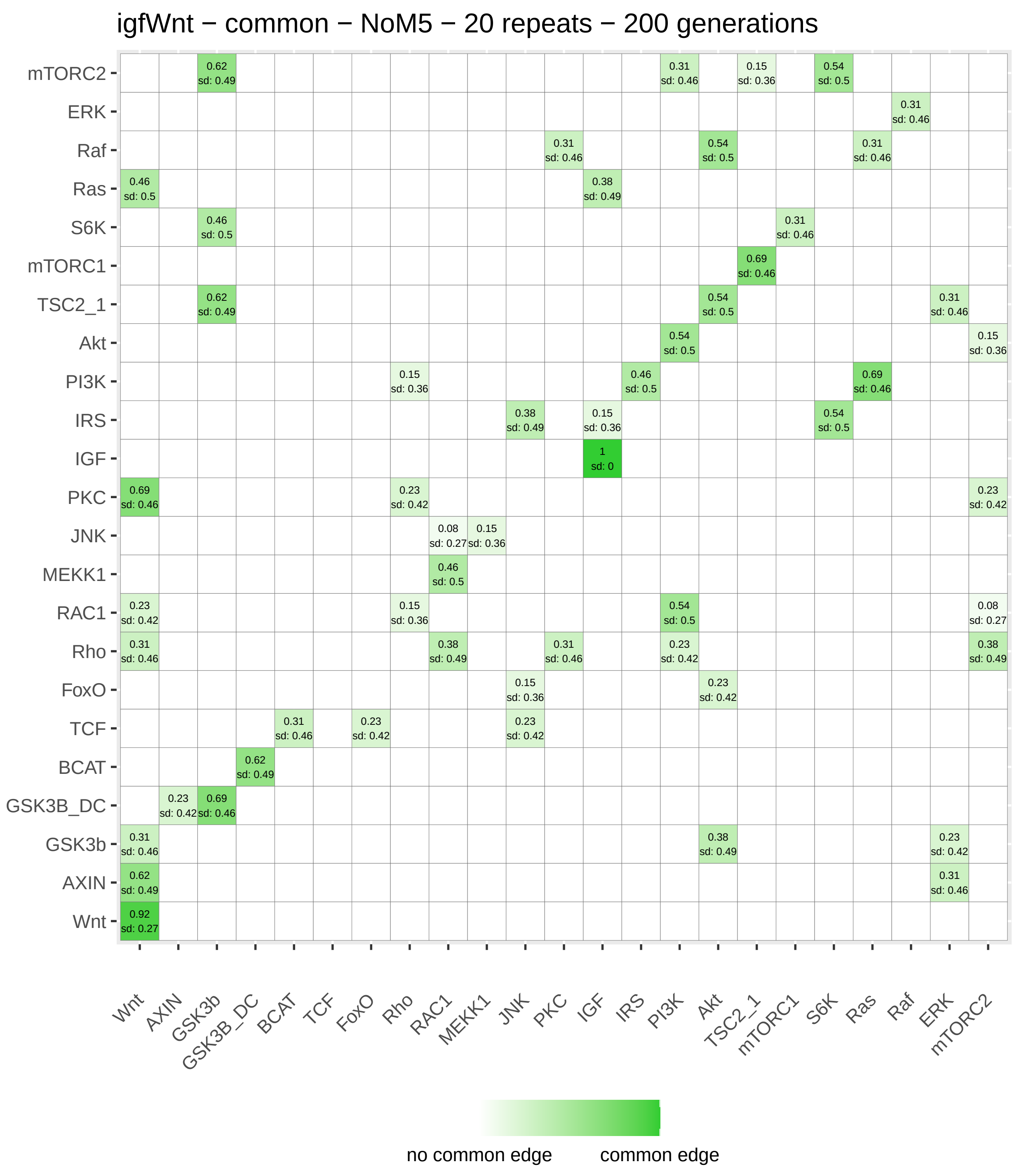


Figure 4.11: Common edges adjacency matrix of the coevolved igfWnt networks compared with the original ifgWnt network. Green indicates common edges. The labelling shows mean and standard derivation.

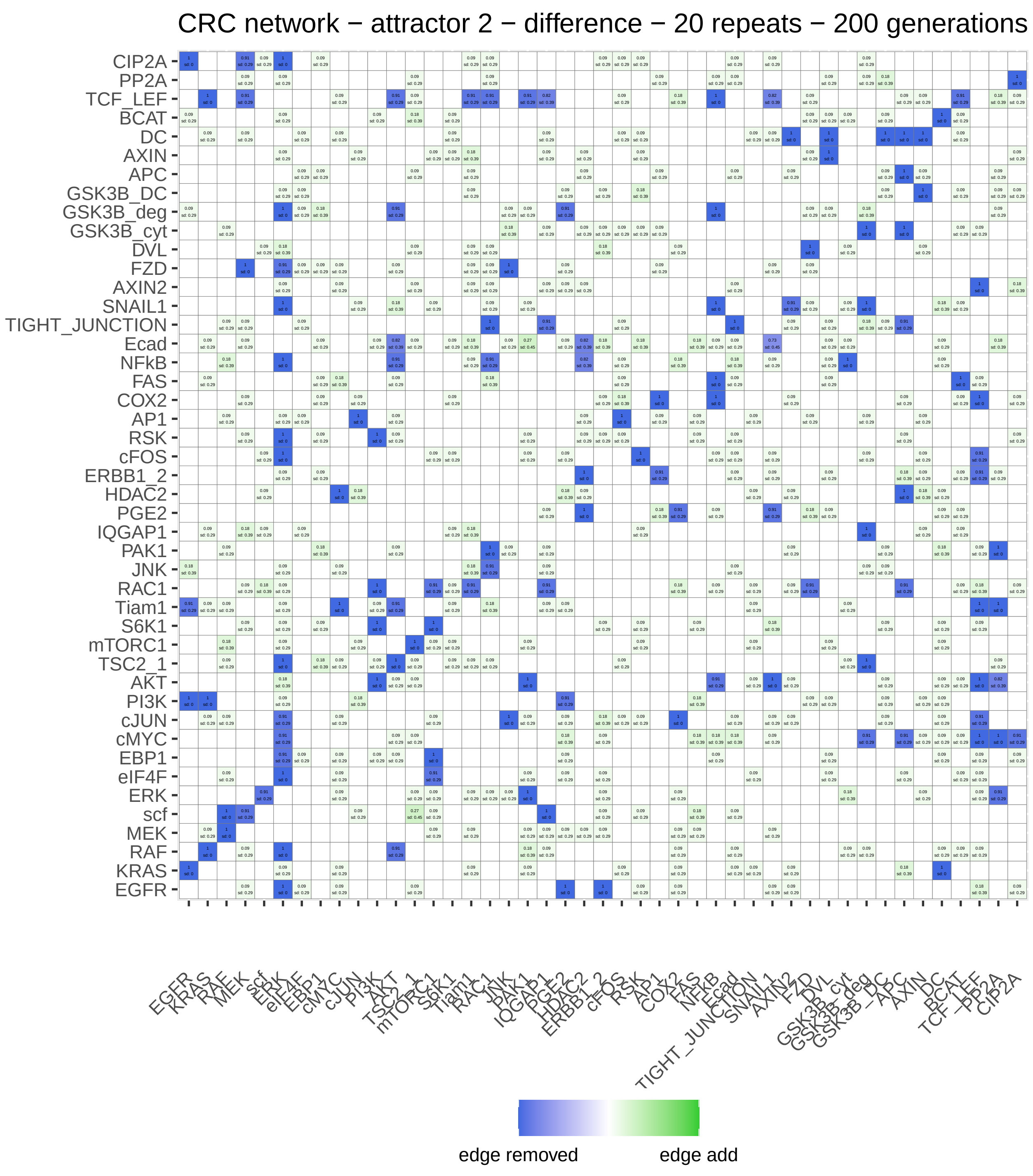


Figure 4.12: Differences adjacency matrix of the coevolved CRC networks compared with the original CRC network. Using the attractor definition with S6K and RSK markers of EGFR/ERK activity added (attractor 2). The labelling shows mean and standard derivation.

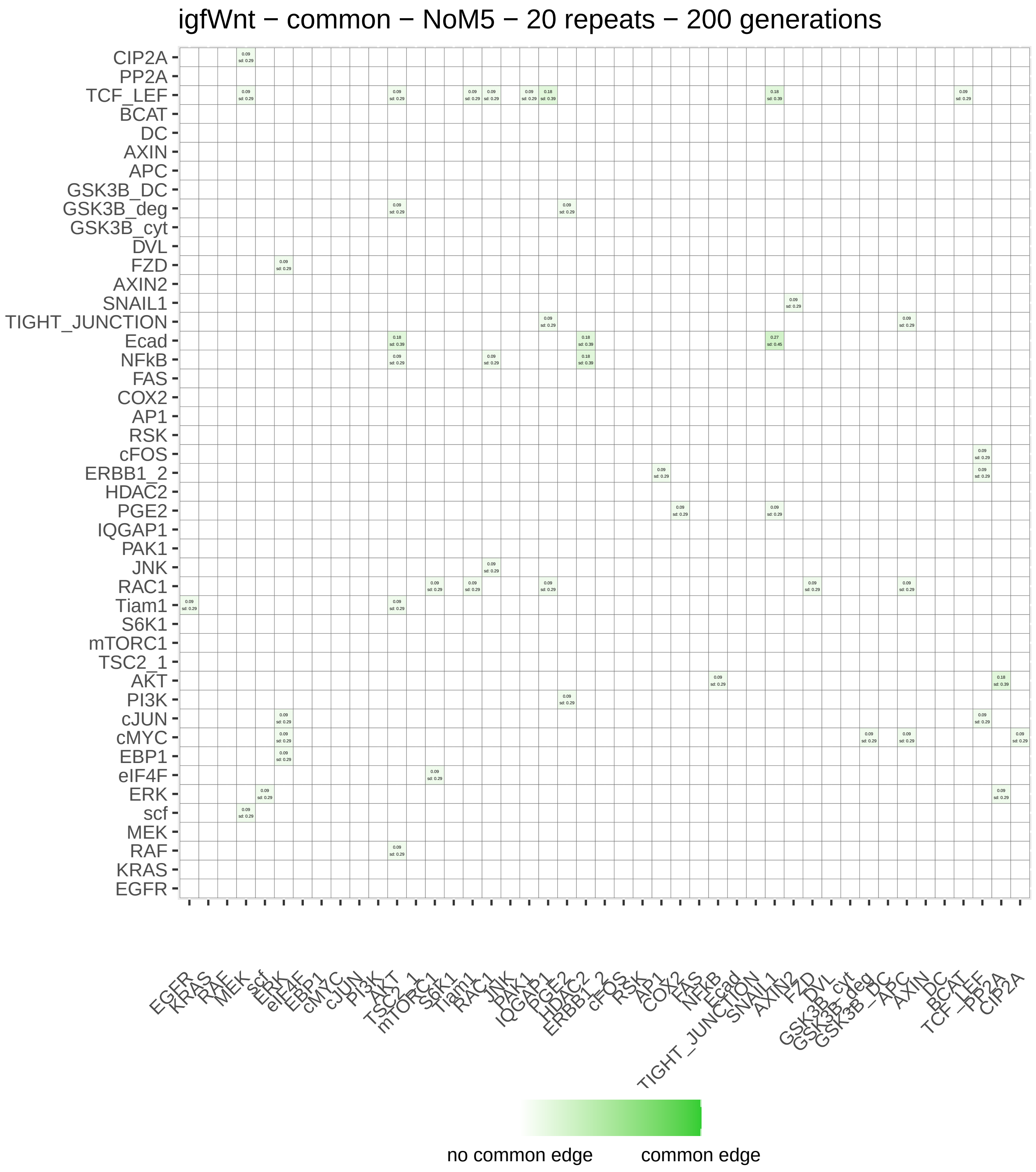


Figure 4.13: Common edges adjacency matrix of the coevolved CRC networks compared with the original network. Using the attractor definition with S6K and RSK markers of EGFR/ERK activity added (attractor 2). The labelling shows mean and standard derivation.

As for attractor two, the comparative adjacency matrices for attractors three and four were investigated. For the sake of clarity and because the matrices do not differ fundamentally to those of attractor two, they are listed in the appendix B.2.1 and B.2.2.

Since the specification for the igfWnt network has remained unchanged for all these experiments, the comparative adjacency matrices are also, except for the smallest stochastically caused deviations, identical to those shown in figure 4.11 and 4.10 and are not listed again.

Attractor 4 - Exchanging no initial states

As a reference to the previous experiments, the influence of the exchange of initial states between the tribes were investigated for the experimental setup with the combined EFGR/ERK and WNT markers (attractor 4). The experiment was followed for 1000 generations and the fitness values were averaged over 20 runs (4.14).

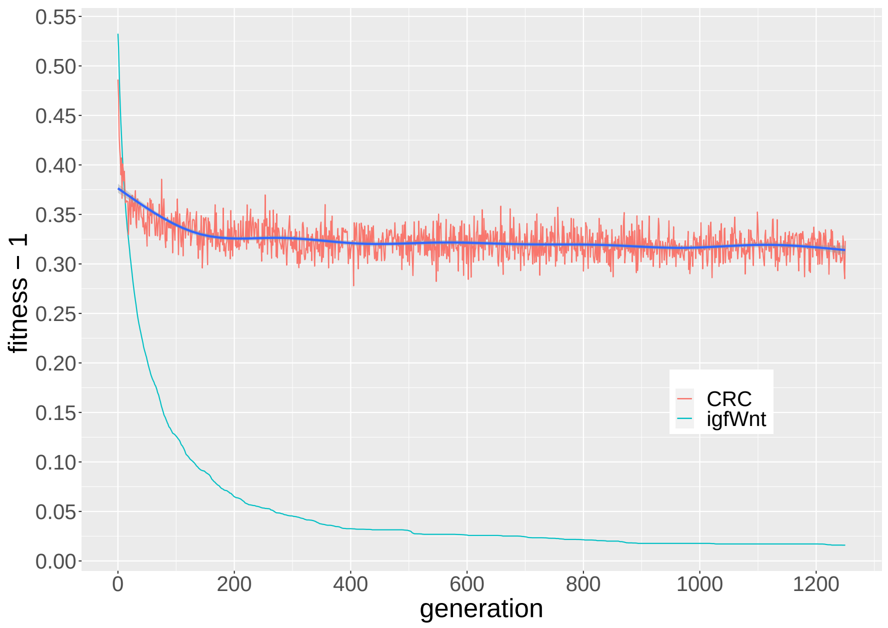


Figure 4.14: Fitness curve for the igfWnt/CRC networks with no initial state exchange. Using the combined EFGR/ERK and WNT markers attractor definition (attractor 4) for the CRC networks.

Influence of crossover

After testing different attractors for the CRC network, the 4th attractor was fixed and the influence of other variables was investigated. First, for the four initial state / attractor pairs of the igfWnt network, the inverses of the attractors were exchanged for the actual attractors. The exact definition can be found in appendix A.1.1. Furthermore, the number of individuals for the CRC tribe was increased to 150. The mutation rate of 5 mutations per individual and generation was maintained, but the mutation-percentage parameter was lowered from 100 to 50, which means that about 50% of all mutations are replaced by the cross-over operation. As in the previous experiments, 2500 generations were examined with the exchange of initial states every 50 generations. The results of 20 runs were averaged and the fitness curve is presented in figure (4.15).

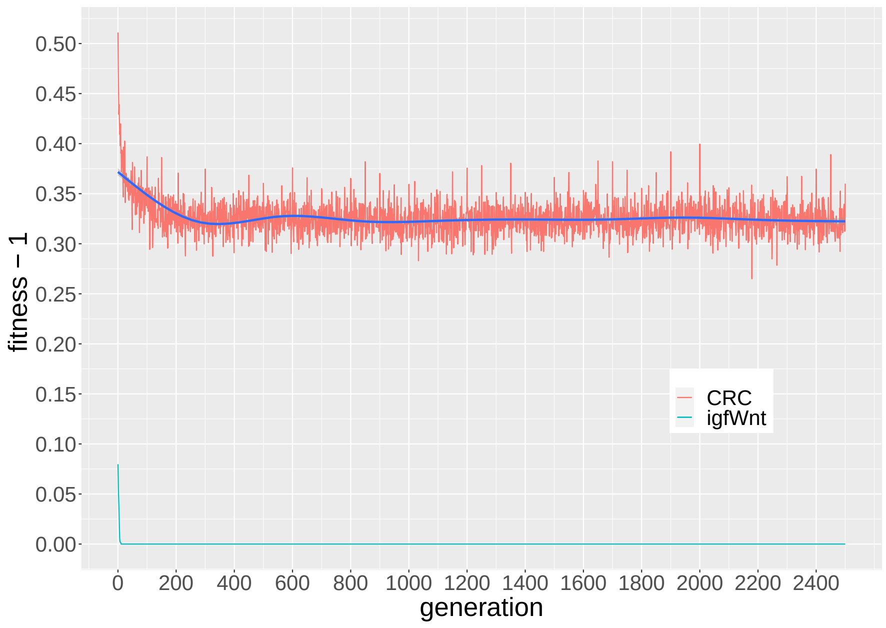


Figure 4.15: Fitness curve of second attempt for the igfWnt/CRC network. Using for the CRC networks the attractor definition with combined EFGR/ERK and WNT markers (attractor 4).

The differences matrix for the CRC network as well as the commonality matrix for the igfWnt network can be found in appendix B.2.4.

Attractor 4 - Simulation with a large population size

For the following experiment, the second igfWnt attractor definition (”initial conditions equals attractors”) and the attractor definition 4 with combined EFGR/ERK and WNT markers for the CRC network were used. For the CRC population, a population size of 400 with a mutation rate of 15 mutations per individual and generation, without crossover, was chosen. The experiment was studied over 2500 generations and repeated 20 times. Every 100 generations, initial states were exchanged with a state of the fittest igfWnt attractor as donor. The fitness curve averaged over the runs are shown in figure 4.16.

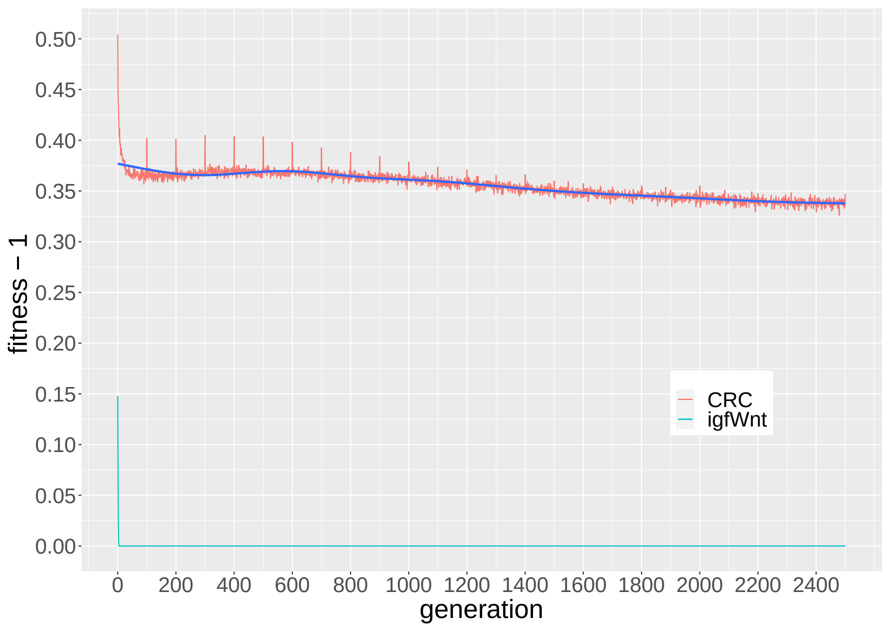


Figure 4.16: Fitness curve igfWnt and CRC network. With a population size of 400 and 15 mutations per individual and generation in the CRC tribe. Exchange of initial states from igfWnt to CRC each 100 generations. Average over 20 runs.

The comparative adjacency matrices associated with this experiment are given in the appendix B.7 and B.8.

4.6.2 Masking missing gene values

For the following experiments, the missing genes in the attractor definition for the CRC network were not randomly added as before but ignored in the fitness calculation. This approach was investigated for all four attractor definitions (A.2.1) of the CRC network. The specification of the igfWnt network was done with the ”initial states equal attractors” definition (A.1.1). A population size of 20 was used as parameter for the CRC network for all experiments. The exchange of initial states from the igfWnt to the CRC network took place every 10 generations. As mutationrate a number of mutations (NoM) of two or rather five was used. A crossover was applied for each attractor definition in a separate run. All experiments were repeated 20 times and the results was averaged.

Attractor 1

For the following results, attractor definition 1 with a mutation rate of 2 was used.

```
Attractor 1:
mTORC1 ERK !BCAT !TSC2_1 !cMYC !AXIN2
```

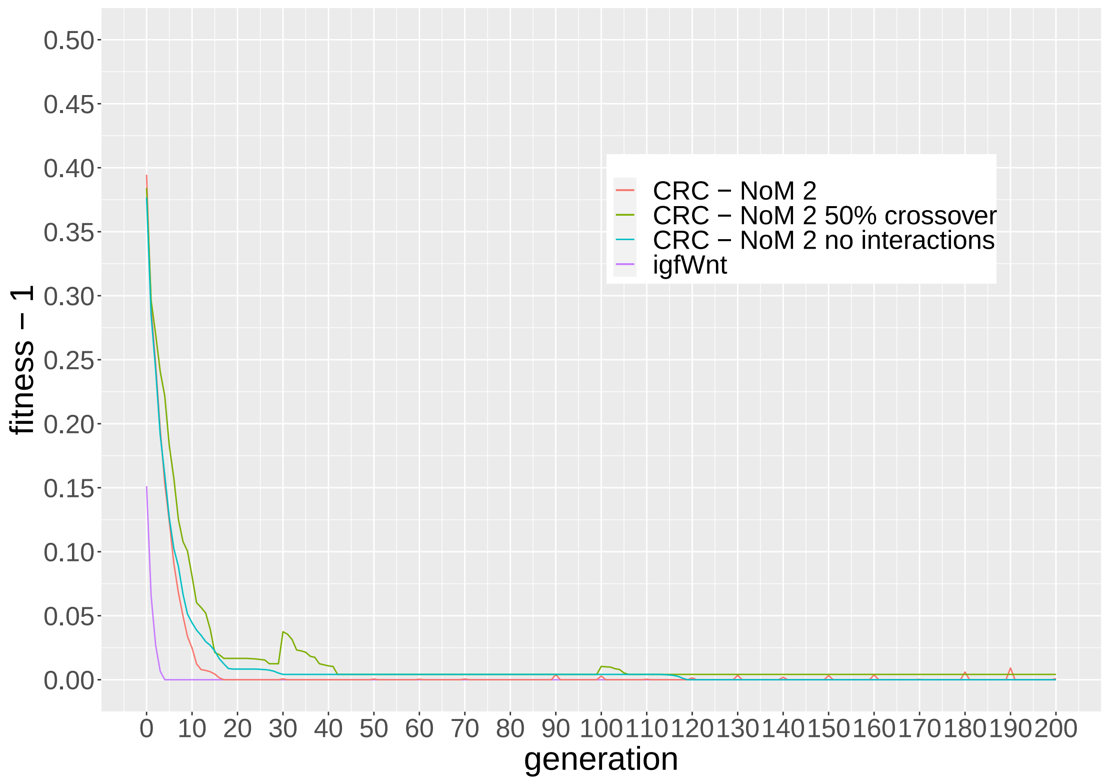


Figure 4.17: Fitness curve for CRC attractor definition 1 and with a mutationrate (NoM) of 2 and initial state exchange all 10 generations. In addition, one run was performed with 50% crossover instead of mutations only. Moreover one run was without coevolutionary interaction between the tribes.

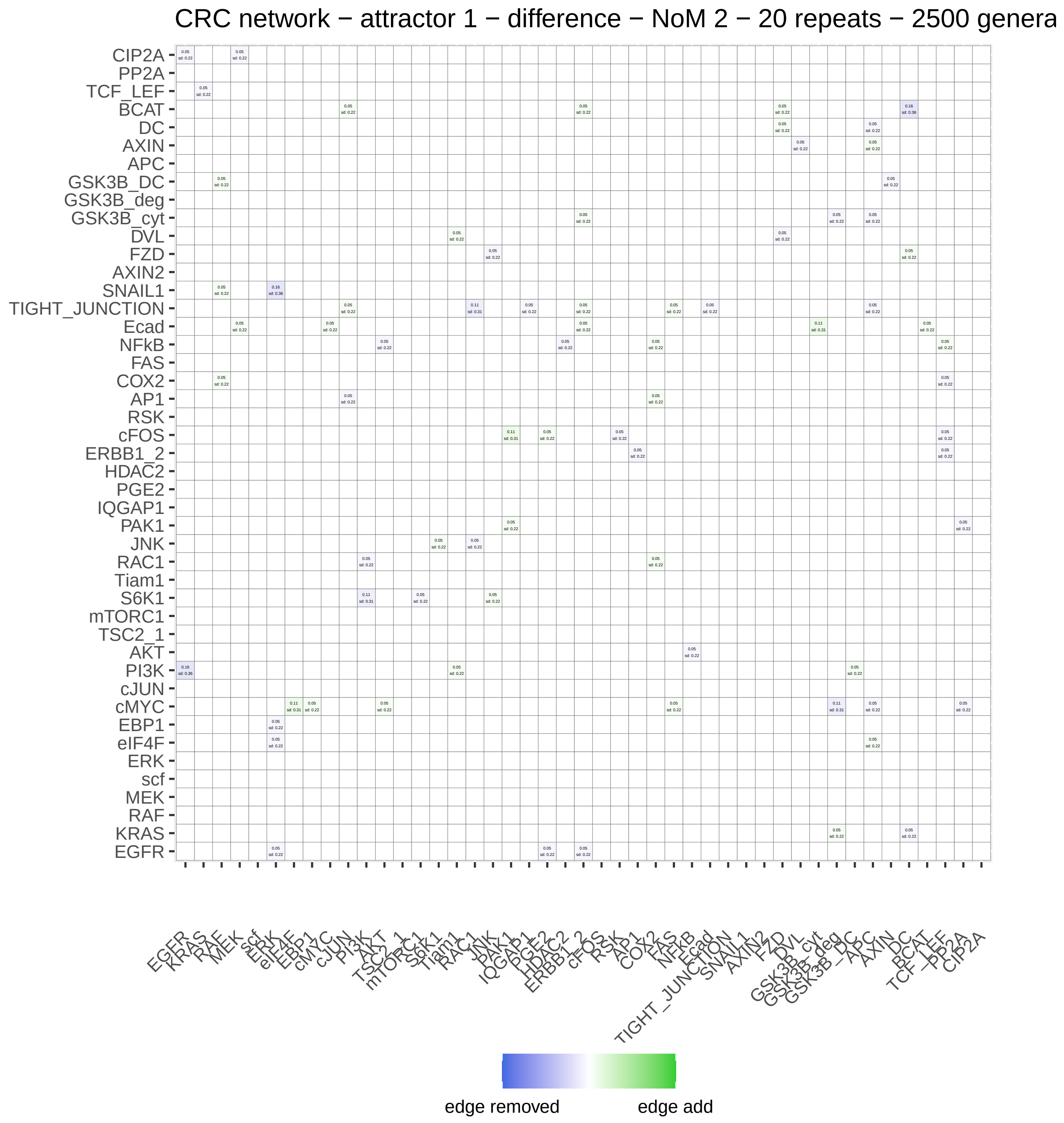


Figure 4.18: Differences adjacent matrix of the coevolved CRC networks with attractor definition 1 compared with the original CRC network. The mutation rate (NoM) is 2 without crossover. Blue fields dedicate deleted edges and green ones added edges. In white are edges with no differences between the networks. The labelling shows mean and standard derivation.

The results of the commonality adjacent matrix are listed in the appendix for completeness (B.3).

Attractor 2

For the following results, attractor definition 2, with added S6K and RSK markers of EGFR/ERK activity, and a mutation rate of 2 was used.

```
Attractor 2:
mTORC1 ERK !BCAT !TSC2_1 !cMYC !AXIN2 S6K RSK
```

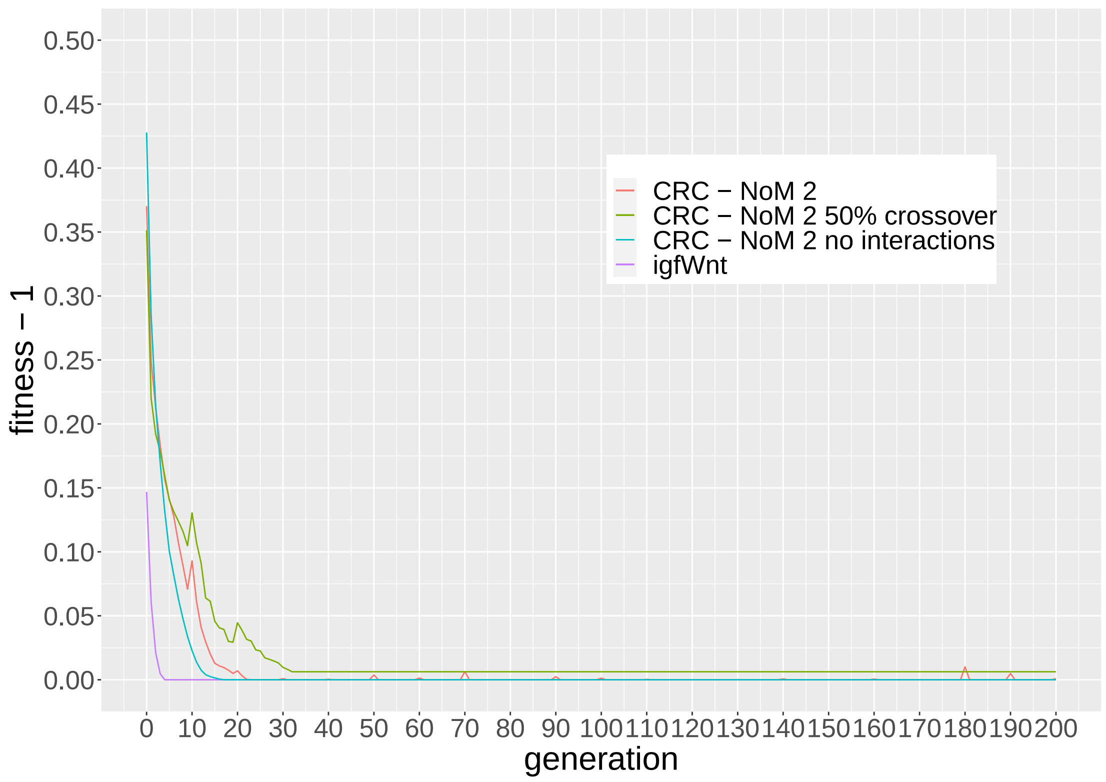


Figure 4.19: Fitness curve for CRC attractor definition with added S6K and RSK markers (attractor 2) and with a mutationrate (NoM) of 2 and initial state exchange all 10 generations. In addition, one run was performed with 50% crossover instead of mutations only. Moreover one run was without coevolutionary interaction between the tribes.

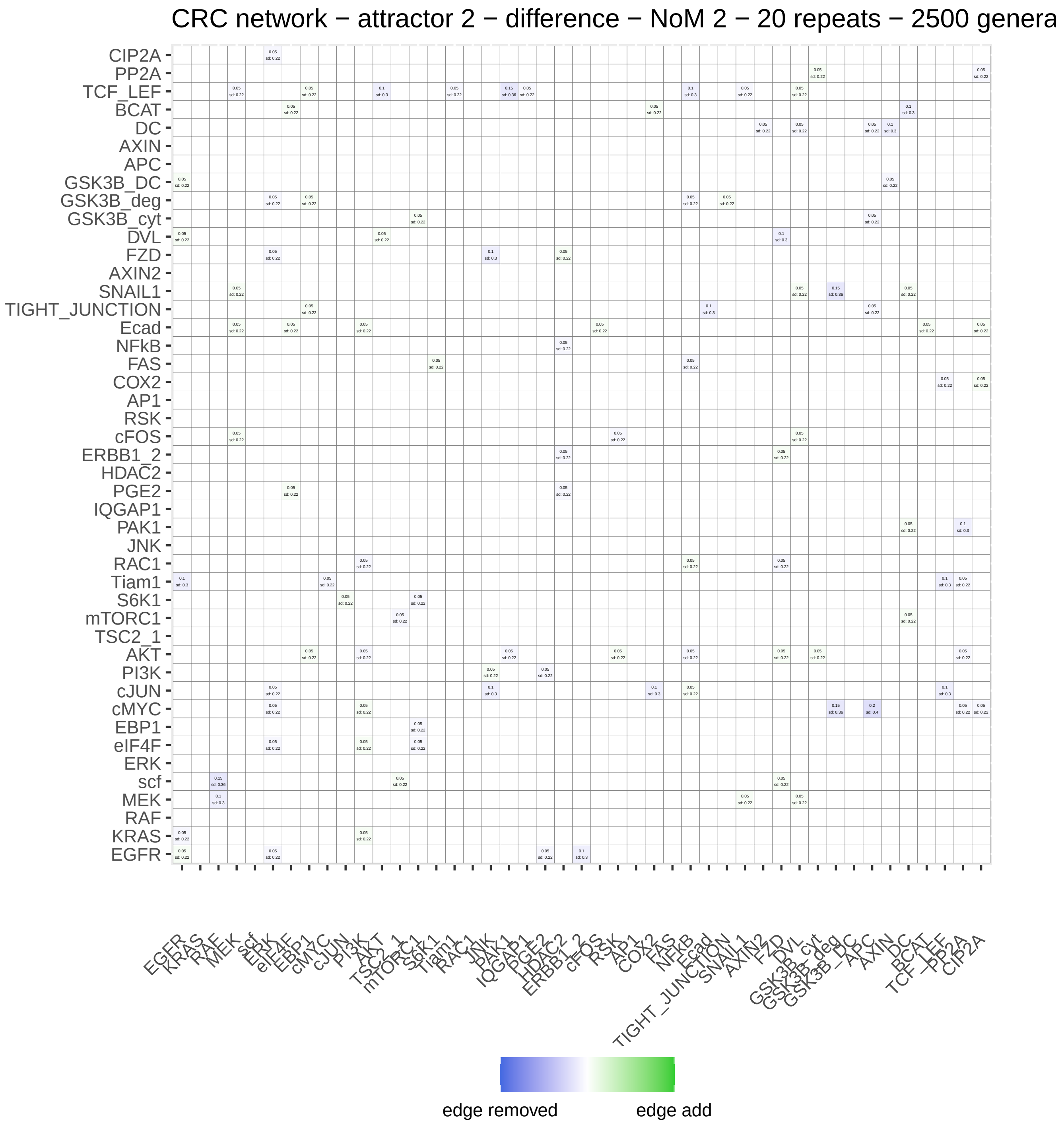


Figure 4.20: Differences adjacent matrix of the coevolved CRC networks with attractor definition 2 (added S6K and RSK markers) compared with the original CRC network. The mutation rate (NoM) is 2 without crossover. Blue fields dedicate deleted edges and green ones added edges. In white are edges with no differences between the networks. The labelling shows mean and standard derivation.

The results of the commonality adjacent matrix are listed in the appendix for completeness (B.3).

Attractor 3

For the following results, attractor definition 3, with WNT inactivation was used.

```
Attractor 3:
mTORC1 ERK !BCAT !TSC2_1 !c_MYC !AXIN2 FZD DVL GSK3B_deg
GSK3B_DC APC AXIN !TCF_LEF
```

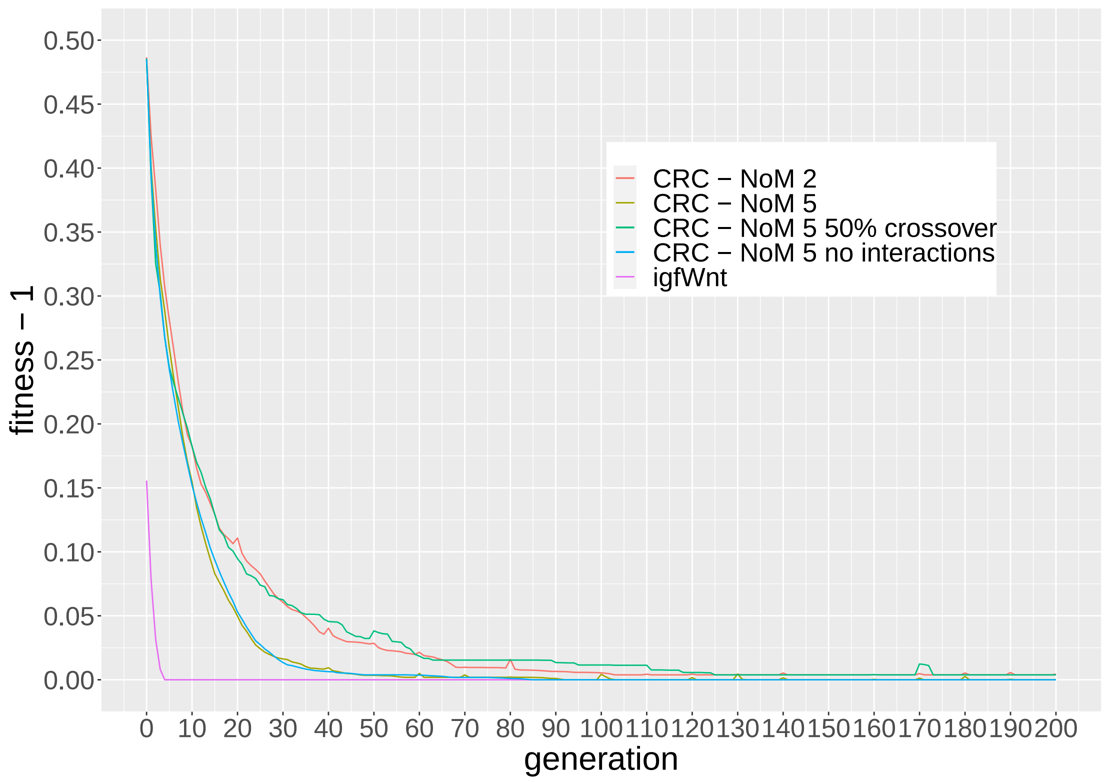


Figure 4.21: Fitness curve for CRC attractor definition 3 (WNT inactivation) and with a mutationrate (NoM) of 2 and 5. The initial state exchange was performed all 10 generations. In addition, one run was performed with 50% crossover instead of mutations only. Morover one run was without coevolutionary interaction between the tribes.

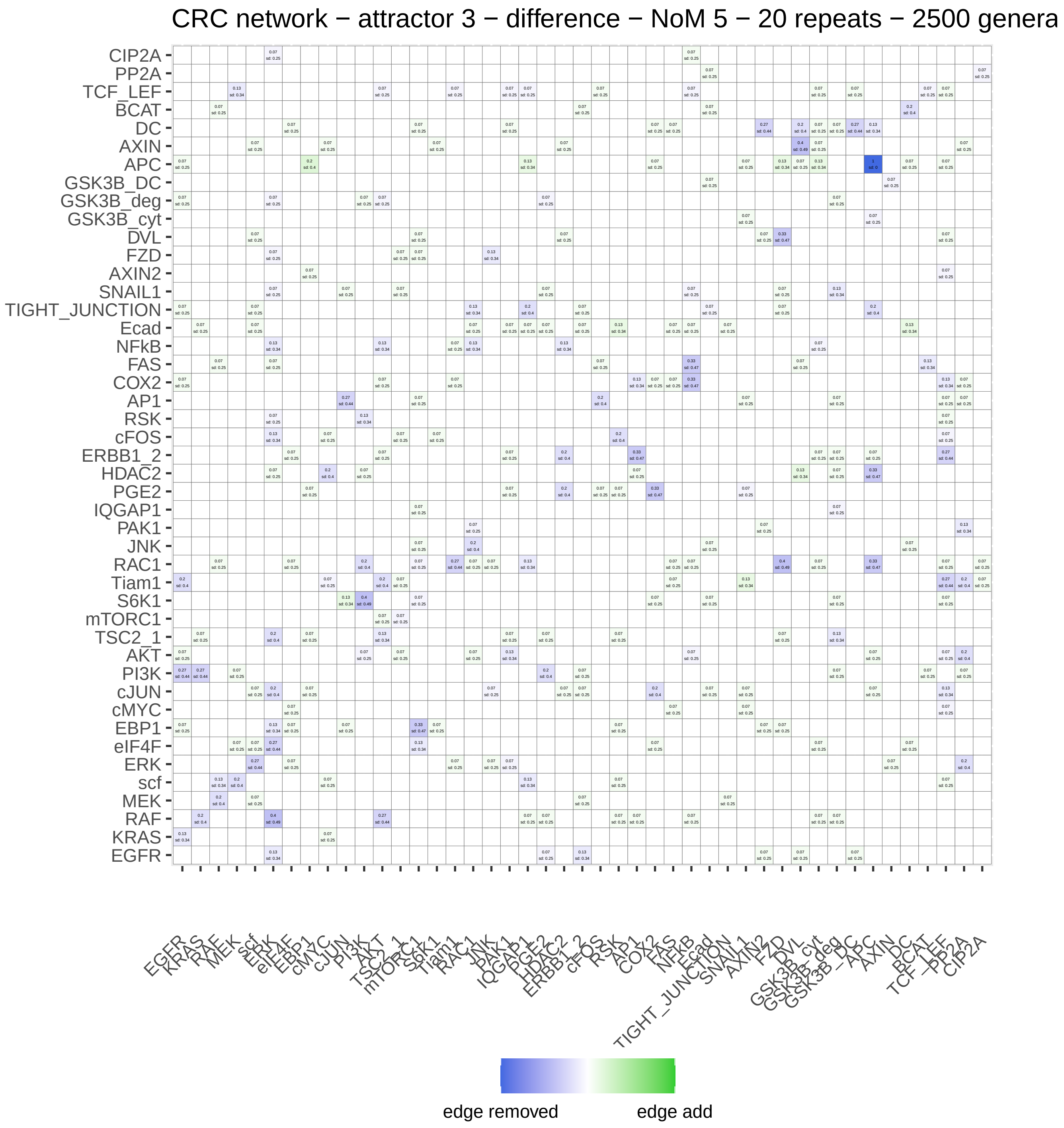


Figure 4.22: Differences adjacent matrix of the coevolved CRC networks with attractor definition 3 (WNT inactivation) compared with the original CRC network. The mutation rate (NoM) is 5 without crossover. Blue fields dedicate deleted edges and green ones added edges. In white are edges with no differences between the networks. The labelling shows mean and standard derivation.

The results of the commonality adjacent matrix are listed in the appendix for completeness (B.3).

Attractor 4

For the following results, attractor definition 4, with combined EFGR/ERK and WNT markers, was used.

```
Attractor 4:
mTORC1 ERK !BCAT !TSC2_1 !c_MYC !AXIN2 FZD DVL GSK3B_deg
GSK3B_DC APC AXIN !TCF_LEF
```

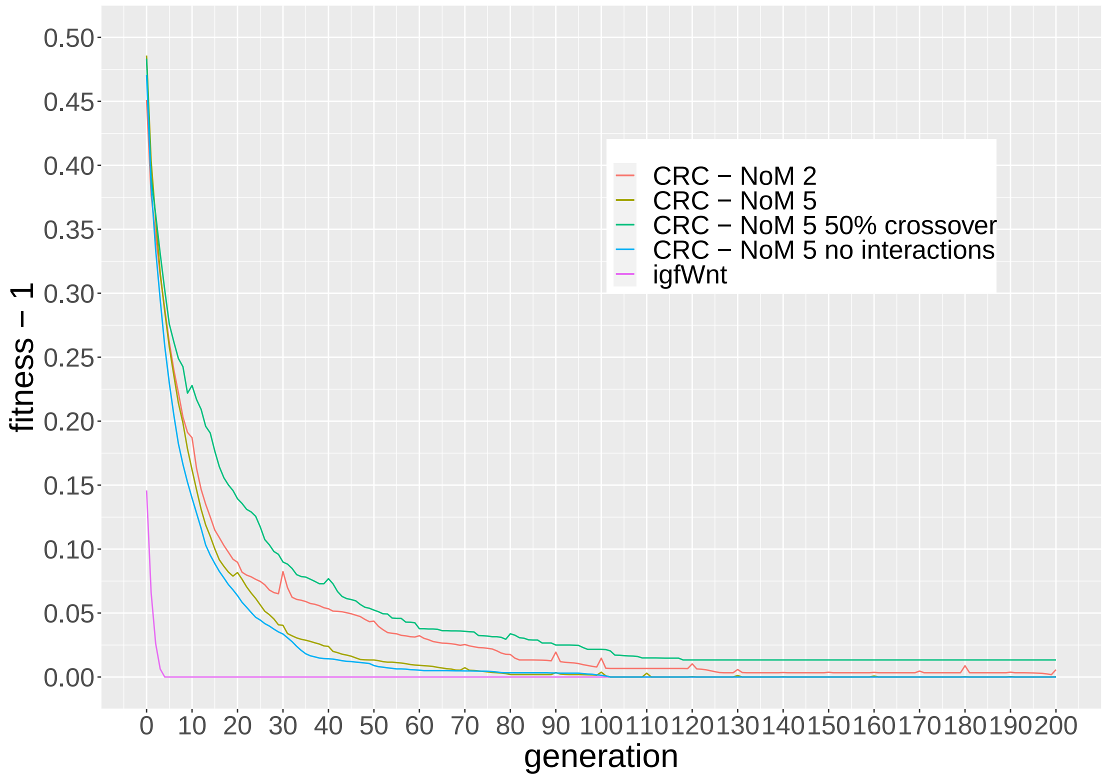


Figure 4.23: Fitness curve for CRC attractor definition 4 (combined EFGR/ERK and WNT markers) and with a mutationrate (NoM) of 2 and 5. The initial state exchange was performed all 10 generations. In addition, one run was performed with 50% change for applying a crossover instead of a mutation. Morover one run was without coevolutionary interaction between the tribes.

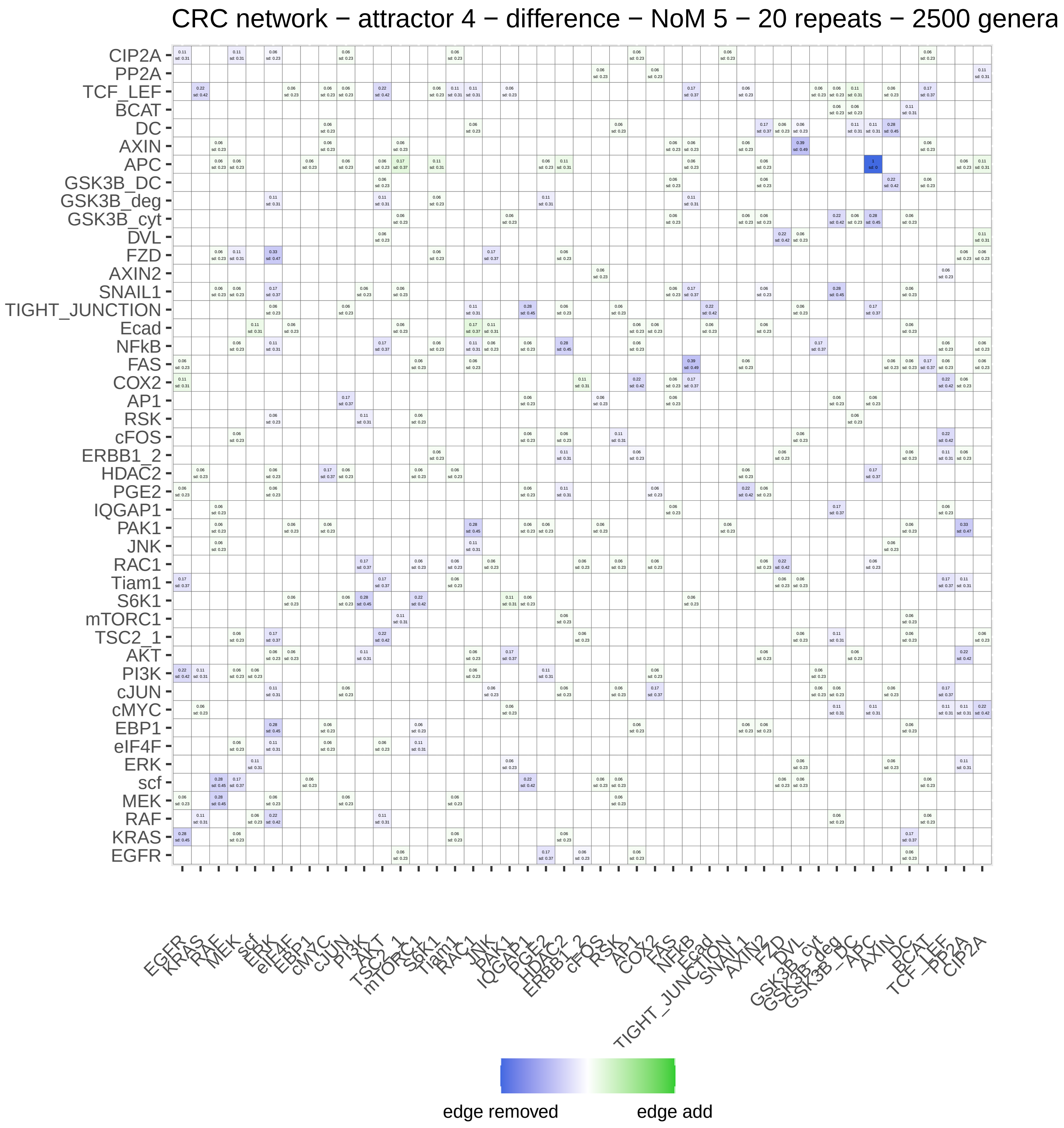


Figure 4.24: Differences adjacent matrix of the coevolved CRC networks with attractor definition 4 (combined EFGR/ERK and WNT markers) compared with the original CRC network. The mutation rate (NoM) is 5 without crossover. Blue fields dedicate deleted edges and green ones added edges. In white are edges with no differences between the networks. The labelling shows the mean and standard derivation.

The results of the commonality adjacent matrix are listed in the appendix for completeness (B.3).

Influence of different coevolutionary sharing strategies

In this series of experiments, as in the 4.5 experiments, the influence of the various sharing strategies on coevolution was investigated. For this purpose, the CRC tribe was copied 3 times and the sharing strategies *overwrite, append and append 4* were used to exchange the initial states. As in the previous experiments, a mutation rate of tow was used for CRC attractors 1 and 2 and a mutation rate of five for attractors 3 and 4. The population size was 20 and the exchange of initial states was done every 10 generations. The experiment was repeated 30 times for each attractor definition and the results were averaged. The fitness progression can be seen in the figure 4.23.

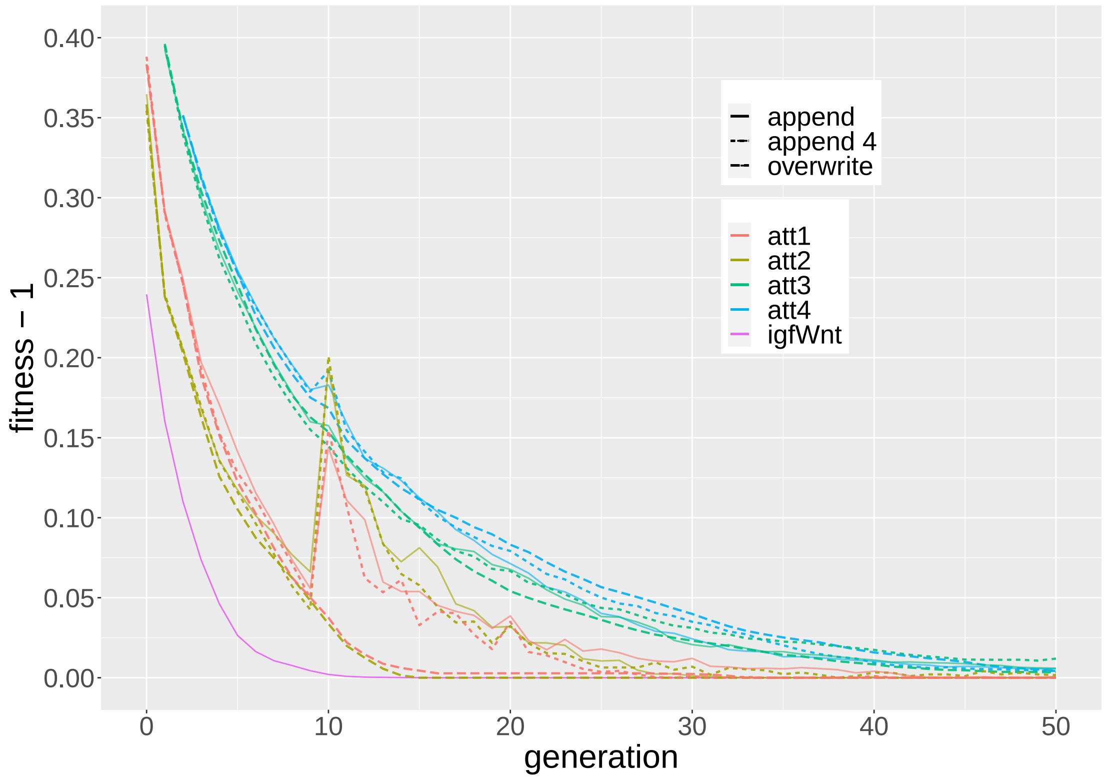
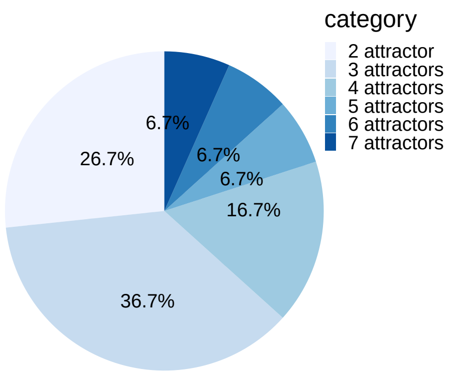
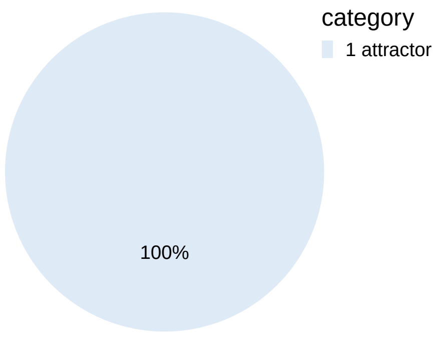


Figure 4.25: Fitness curve for CRC attractor definitions 1-4. The fitness is compared with an initial state exchange every 10 generations with the sharing strategies *overwrite, append and append 4*.

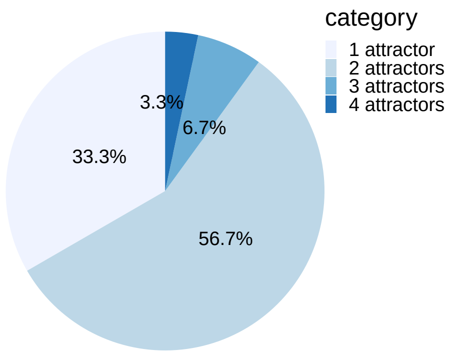
Furthermore, it was investigated how many different attractors the coevolved networks reached from all initial states occurring during coevolution. The results for attractors 1 and 4 are given in figure 4.27. The results of the other two attractors can be found in the appendix (B.15, B.16).



(a) Overwrite

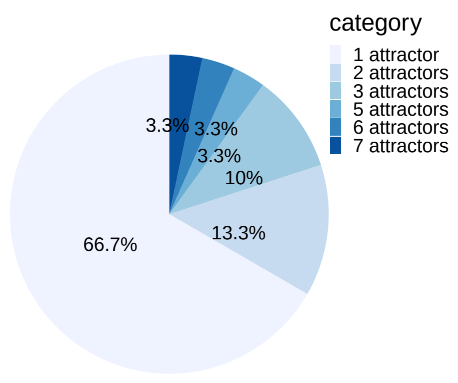


(b) Append

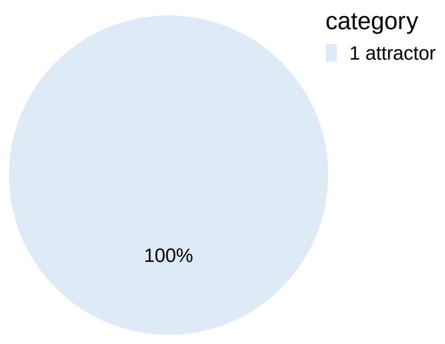


(c) Append 4

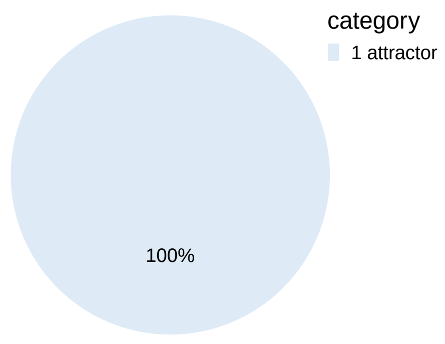
Figure 4.26: Number of attractors - with exchange strategies *overwrite*, *append* and *append 4* for the CRC network for attractor definition 1.



(a) Overwrite



(b) Append



(c) Append 4

Figure 4.27: Number of attractors - with exchange strategies *overwrite*, *append* and *append 4* for the CRC network for attractor definition 4.

The difference adjacency matrices for all attractor definitions are listed in the appendix for completeness.

5 Discussion

The objective of the present work was to design and implement a framework that allows to simulate the coevolution of several populations and to subsequently investigate the course of the coevolution, as well as the resulting network.

5.1 Initial state exchange

To test the framework, several small random generated networks were experimented with to investigate the functionality and the influence of different coevolutionary parameters and operators.

In this series of experiments, the influence of exchanging initial states on the coevolution of different populations was investigated. The randomly generated networks were initialised with a biologically untypically high connectivity of five. This results in networks with complex transitions and many large attractors. This should lead to the fact that different initial start states contains to different basins. Therefore, the exchange of initial states should generally have the effect that the networks reaching different attractors than before and therefore there should fitness decreases spontaneously.

5.1.1 Sharing all overlapping genes

All networks consisted of the same 10 genes and were therefore potential candidates for initial state exchange. In the first series of experiments, all genes were actually exchanged. This should maximise the influence of coevolution. Looking at the experiment 4.1, it becomes obvious that tribes 0 and 3, which were not influenced by coevolution, evolve as expected and their fitness increases during the optimisation process from initially 0.65 to around 0.25 after 100 generations. This proves that the implementation of the evolutionary algorithm works correctly and efficiently, even with a difficult target attractor, consisting of six randomly picked states, a low mutation rate of one and a small population. Nevertheless, it is noticeable that the optimisation problem seems to be harder for the initially identical tribes 0 and 2 than for the also initially identical tribes 1 and 3. This can be explained by their random generation.

As expected, the exchange of initial states worsens the fitness of the acceptor tribes, but if they have enough time, they return to a fitness value comparable to that of their unperturbed twin tribes. The faster the exchange of initial states takes place, the more difficult the optimisation becomes. This can be attributed to the fact that the problem changes with each exchange and mutations, that previously had a positive effect on fitness can now be unfavourable. Through frequent exchanging, the evolutionary algorithm does not get enough time to perform a sufficient number of optimisations before the problem space changes again. Nevertheless, even an exchange after each generation still leads to an improvement in fitness. This is possible due to the fact that the new states are not randomly selected, but are a state of the achieved attractor of the donor tribe. Since this attractor has only a limited size and does not permanently change completely, but develops towards the target attractor, a completely new state is not always exchanged. Furthermore, it must be considered that the target attractor is identical for donor and acceptor tribe. Thus, the optimisation problem should tend to become easier rather than more difficult starting from the new initial states. To confirm this principle, more extensive experiments would have to be carried out in the future.

5.1.2 Share genes partially

Since it is atypical for most biologically inspired experiments that all genes of one network influence another in the sense of coevolution, the following experiments investigated how the tribes of the first experiments behave under the condition that only a subset of genes is exchanged, or that genes are exchanged only with a certain probability. In a first set of experiments (4.2), all 10 genes were potentially eligible for sharing, but only a certain percentage of each gene was actually shared in each exchange. It is expected that the influence of such an operator is comparable to the previous experiments (4.1), but the peaks describing the deterioration of fitness due to the exchange should become flatter inversely proportional to the probability of the exchange. This behaviour could indeed be observed.

It is remarkable that a clear and strong perturbation effect is already achieved with only 10 percent exchange probability. Moreover, even if each gene is only exchanged with a probability of one percent, which should lead, in a 10 gene network, to no exchange at all most of the time, effects can be observed. This can possibly be attributed to the fact that the connectivity of the initial networks is very high and thus even the smallest changes can lead to massive effects. Moreover, it is remarkable that in the one percent run, tribe1 has a better fitness than tribe0. This could not be observed in previous experiments with these networks. This could be due to the fact that the exchange of a small subset of genes allows a population to leave a local optimum that it could not have left by the evolutionary algorithm alone. The fact that this effect does not occur in general could be a result of the low probability of initial state exchange. Therefore after a positive "kick" the network has enough time to adapt to the new favourable conditions.

In further experiments 4.3 in which only a subset of the genes was exchanged, similar effects were observed. If only genes one to three were exchanged, tribe one managed to achieve a better fitness than its undisturbed twin after about 50 generations and was able to maintain this advantage. A similar behaviour is seen for tribe two, when genes one to five are exchanged. However, this effect is not positive in all cases. For example, when genes one to seven are exchanged for tribe one, there is a clear deterioration in fitness compared to the other experiments in the same series, which could not be recovered by the following optimisation. Thus, the exchange of initial states seems to be able both to free a network from an unfavourable local optimum or to move a network into such an optimum. Certain genes seem to be more vulnerable or more potent for causing such positive or negative effects. Since the networks used in these experiments, as well as the initial start states and the target attractor, were generated randomly, the genes have no significance in this context. It is therefore advisable to examine populations whose networks and genes have actual biological meaning, for a better understanding of a possible positive and negative effect of the overcoming of local optima through the exchange of specific subsets of genes.

5.2 Mutual state exchange

The experiment (4.4) investigated the question of how far the framework can be used to simulate an experiment in which the acceptor and donor tribe are not fixed from the start, but can be changed by an additional parameter, in this case the fitness. Since the fitter tribe acted as the donor and the less fit tribe as the acceptor, it was unclear whether the disturbance of the less fit tribe, occurring every five generations, would hinder it in its development or whether the possibly more favourable new initial states would lead to an alignment of the two tribes. The experiment seems to suggest that both effects are active at different times. Thus, fitter and less fit tribes alternate several times during the course of the experiment, and there are also phases in which the tribes involved in coevolution catch up with the unaffected ones in fitness. The influence of the exchange decreases with increasing duration of the experiment. This can probably be attributed to the fact that tribes adapt to the limited number of exchanged states and therefore the two tribes converge. Above all there is the assumption that the basin of the attractor becomes larger for both networks, since the exchange of more and more states shifts the trajectory in the direction of the attractor. In conclusion, it can be said that the framework can also be used to investigate more complex interactions between different populations.

5.3 Evaluation of selection strategies

Within the framework of this work, further survivor-selection operators were implemented. Compared to a pure selection of the fittest individuals, these are characterised by the fact that less fit individuals also reach the next generation with a certain probability. Although this generally reduces the selection pressure, the slower convergence maintains a higher diversity in the population. In certain cases, this should lead to local optima being abandoned in favour of the global optimum. A significant decrease in selection pressure is clearly evident for roulette wheel selection (4.5). Although fitness continues to improve, the speed of optimisation decreases dramatically. However, it must be taken into account that the randomly generated NK(10,5) networks with the complex target attractor also describe a rather difficult optimisation problem. Possibly, this operator would offer an advantage over truncation selection in a different problem with more local optima. Such an example should be investigated in further work. The *elitism* operator has a comparable selection pressure like the *truncation*. However, the expected much less smooth fitness curve indicates that the tribes have a higher diversity. Nevertheless, the experiment shows no fitness advantage over truncation. Whether such an advantage exists would have to be shown by experiments that follow fitness over more generations.

5.4 Evaluation of the runtime

The coevolution of several tribes, each of which can consist of a large number of individuals, requires a high computing effort. Therefore, the software architecture provides to be distributed as widely as possible over several threads. Nevertheless, it can be assumed that unfavourable memory management leads to a deterioration of the runtime under the condition of many threads. Therefore, the thread-number to runtime plot should show a U-shaped curve. Moreover, the fitness calculation itself is based on functions that were largely taken from Aly's work. For these no runtime advantage for more than four threads could be found [39]. In this work for coevolution a separate thread was called for each tribe. This in turn calls another thread for each individual for mutation and fitness calculation. It is unclear whether such a nested threading leads to an improvement in runtime. Therefore, it was tested whether threading only at the individual level or only at the tribe level is faster. Although nested threading seems to offer slight advantages over threading only at the level of individuals or tribes (4.6), all three variants show that after four available threads no more runtime gains were achieved. The plot does not show the expected U-shaped progression. The fact that the optimum is reached with four threads could be due to several reasons. Since the experiment has four tribes and an exchange of initial states already takes place after all five generations, which is quite computationally intensive due to the selection of an appropriate state of the donor tribes, it is plausible to assume that the bottleneck that slows down the entire program is located at this level. As long as an initial exchange takes place, all tribes must inevitably wait for this process to be completed in order to start the evolutionary loop with the correct new parameters. In addition, an exchange every five generations lead to the fact that very often threads have to be created for the single individuals, which are then only needed for a short time before all networks have to wait on each other again through the coevolutionary operator. This leads to a large overhead that consumes a possible runtime gain. It is possible that in this case the system does not benefit sufficiently from the possibility of calculating each individual in a separate thread. A further question that has not yet been answered is the influence of the ARM architecture on which the experiments were carried out. It is possible that Clojure does not sufficiently support this hardware in the area of threading. Thus, this experiment could lead to interesting results on a comparable x86 architecture. Finally, it must be said that although the runtimes of the framework allows larger experiments like the igfWnt/CRC example, the concurrency still needs to be improved.

5.5 Influence of different coevolutionary sharing strategies

In the context of this work, three different variants were implemented with which new initial states can be exchanged. Either the already existing states of the acceptor tribe can be overwritten, or the new states are appended to the previous ones. Therefore, either every old initial state can be kept or only the most recent n states. The experiment 4.7 answers the question how the different variants affect the optimisation. As expected, *overwriting* the initial states slightly worsens the optimisation under all three variants, compared against an identical control tribe which was not involved in coevolution. As in all previous experiments, the overwrite operator shows the typical sawtooth-like pattern in the fitness curve. The same is found at the beginning for the *append* variants, but as expected the spikes flatten out quickly. This will be due to the fact that the set of different shared states is not large and no new states are added after a few exchanges. Furthermore, the fitness for each individual initial state is calculated separately and then averaged. Thus, the importance of a single state decreases in the course of coevolution for this variant, because even unfavourable new initial states influence the fitness only slightly. This distribution of fitness over several trajectories should also lead to a decrease the selection pressure. Furthermore, multiple initial states lead to local optima, which result from the different influence of different states on the fitness, in the way that a mutation improved fitness from one initial state and worsened it from another. These should also have a negative effect on the optimisation. However, these effects could not be clearly observed in the context of this experiment and should be the subject of further investigations. As expected, the result for the *append 4* variant is between the extreme variants *overwrite* and *append*.

When exchanging initial states, the question arises whether the system, when receiving a new initial state, dismantles the trajectories again from the old start state or whether the improvements already made tend to be maintained. If the latter is the case, this would indicate that even if the optimisation is generally slower when initial states are exchanged, more robust networks can be created. Robustness here means that the basin of the target attractor is extended by at least a part of the old initial states and the trajectories that originate from them. To address this question, the fittest networks after the coevolution were selected and examined how many attractors these networks reach, based on all the initial states they had received in the course of their coevolution. Since the networks were studied for 240 generations and a potentially different initial state was exchanged every 10 generations, this results that in the worst case 24 different attractors could be reached. However, for all variants of the exchange it could be observed that in more than 50% of all repetitions of the experiment, see section 4.8, the same attractor is reached from all initial states. More than four different attractors were not observed in any run. This shows impressively that by exchanging initial states, not all previously achieved optimisations of the networks are simply discarded, but that are retained. As expected, this behaviour is weakest for the *overwrite* operator and strongest for the *append* operator. If only the last four states are always retained, which of course has a considerable advantage with regard to runtime and memory space, similarly good results are shown as with the retention of all states. These results pose numerous questions for the future. First, it should be investigated whether this effect can also be transferred to other biologically more plausible networks. Moreover, it would be interesting to investigate to what extent coevolution in general and the three operators presented here in particular, have an influence on other robustness measures of the coevolved networks. Furthermore, the entire attractor landscape of these networks should be compared with that of the original networks in future works.

5.6 Biologically Plausible Experiment

5.6.1 Addition of missing gene values

After the general functionality of the framework was proven in smaller randomly generated networks, it was applied to a large pair of networks borrowed from biology. The igfWnt network served as the donor and the modified CRC network as the acceptor in all subsequent experiments. In each exchange of initial states, all nine shared genes (4.6) of the two networks were exchanged. The CRC network consists of a total of 45 genes. For this network, there are four different potential target attractors 4.6, numbered from 1 to 4, with the number of defined genes increasing from 1 to 4. However, even attractor 4 has values for only 15 of the 45 genes. Nevertheless, in a first series of experiments, it was tested how the optimisation behaves when the missing values are randomly added and not masked.

As expected, in all experiments the igfWnt network, which has as target attractors the inverses of the existing attractors, succeeds in approximating them almost perfectly. Nevertheless, it must be considered that for the igfWnt network four initial state/attractor pairs were defined that had to be optimised at the same time, which is in fact not an easy optimisation problem. The CRC network fitness curves for all four attractors 4.9 initially showed a slight improvement. After about 50 generations the curves reaches a fixed plateau from which no further optimisation could be achieved. The fitness of these plateaus correlates inversely with the number of defined genes in the attractors. This observation is certainly due to the fact that the severity of the optimisation problem in these experiments increases with the number of undefined genes. The fact that the fitness of the CRC populations for all four attractors initially increases rapidly cannot be explained by the fact that the original population is initialised with slight variations of the original network. On the one hand, these variations are only small mutations which correspond to the mutation rate during the experiment. On the other hand, truncation survivor selection leads to a much faster convergence than observed here. This suggests that there are certain optimisations that already occur under low selection pressure and keep the networks trapped in the local optimum described by the plateau. To investigate this question, the adjacency table of the original networks was compared with the coevolved networks to see what changes the networks have undergone in the course of coevolution. If we look at the difference matrices (e.g. 4.12), it must be notice that significantly more edges were removed than gained. Furthermore, these edges were removed in almost all repetitions of the experiment, whereas not newly created edge appeared in more than half of all runs. Comparing the pattern of removed edges with the adjacency matrix of the original CRC network B.2, it is noticeable that almost every edge of the original network became the target of deletion. A look at the comparison of common edges of the evolved networks with the original network 4.13 confirms this observation. It is therefore plausible to assume that the fitness calculation prefers a less interconnected network over the initial network and therefore edges are often removed first. This results in the problem that removed transitions could only rarely add again by random mutation and then only slowly operator by operator. This reduces the possibility of actually optimising the network through individual random mutations and the selection pressure decreases rapidly. These findings are, except for minor statistical variations, consistent for all attractors and no specific differences between their adjacency matrices could be found. Looking at the comparative adjacency matrices for the igfWnt network, a heterogeneous picture, with both specific edges removed and new edges added, emerges as expected. However, it should be considered that a common edge does not mean that it is also the same transition. Rather, it can be assumed that common edges are often the target of transition altering mutations. Further work would have to investigate this suspicion separately.

Unfortunately, the strong variation of fitness for all four attractors in the CRC networks did not allow to identify a possible effect of the coevolutionary initial state exchange. To corroborate the above observations and assumptions against this influence, the same experiment was repeated for attractor 4 without the initial state exchange. As expected, the results 4.14 shows that, the influence of the nine exchanged genes compared to the 30 undefined genes in attractor 4 have no significant effect.

As described, it is suspected that the removal of transitions at the beginning of coevolution yields a stronger fitness gain than the modification and addition of new transitions. For this reason, it seemed appropriate to replace half of all mutations with crossover operations that could compensate the removal of transitions. Furthermore, the number of individuals was increased to 150 in order to slow down the convergence of the population and to increase the change to a positive mutation. Moreover, the target attractors of the igfWnt network were changed so that the target attractors represent the already existing attractors of the network. Looking at the result of the experiment 4.15, there is no improvement for the CRC fitness compared to the previous experiments. As expected, the igfWnt population, which already matched the target attractors, showed rapid convergence at optimal fitness. This can be explained by the rapid elimination of the individuals, that are slight variations of the original network, through the truncation survivor selection. In the further course, the population no longer undergoes any change, which is confirmed by the common adjacency matrix B.10.

Since all previous attempts to optimise the CRC network had shown insufficient success, a final experiment was set up with 400 individuals and a mutation rate of 15. The results 4.16 initially show the same course as the previous experiments. At first, the fitness increases rapidly before it unexpectedly gets slightly worse. Moreover a clear peak is observed every 100 generations. This can be explained by the exchange of initial states. This should also be the explanation for the slight decrease in fitness. During the next 800 generations, the fitness hardens to an almost constant level, which corresponds to the previous smaller experiments. However, it becomes apparent that in this phase the peaks become increasingly smaller. This means that the population slowly adapts to the perturbations triggered by the exchange. This phase is followed by a very small but constant increase in fitness. Nevertheless, even after 2500 generations, the population reaches just a bad fitness of 0.3. Future work could possibly explore how far this optimisation can be pushed forward with considerably more time and resources. Examining the comparative adjacency matrices (B.7, B.8) for these networks, there shows unfortunately no significant differences to the data collected in the smaller experiments. This can certainly be attributed to the fact that the actual fitness is only slightly better than in the previous experiments and to obtain meaningful results, a greater effort would have to be made, which is beyond the scope of this work.

5.6.2 Masking missing gene values

Since the approach of filling all missing genes in the definitions of the attractors for the CRC network with randomly chosen values turned out to be a very hard optimisation problem, it was now investigated how coevolution behaves when the missing genes are masked for the fitness calculation. It is to be expected that the optimisation problem becomes therefore significantly easier. Moreover, in contrast to the first series of experiments with the igfWnt/CRC networks the optimisation should become harder instead of easier with the number of defined genes. This approach is also more in line with the biologically inspired questioning that is primarily interested in the influence of the genes defined in the attractor.

The results (4.17, 4.19, 4.21, 4.23) shows that perfect optimisation is possible within 200 generations for all four attractor definitions. Therefore, a population rate of 30 and a mutation rate between 2 and 5 is already sufficient. Thus, as expected, the optimisation problem is much easier when the missing genes are masked than when they are added. It is also confirmed that the severity of the problem increases with the number of defined genes in the attractors.

Interestingly, an optimal fitness value could only be achieved for attractors 3 and 4 with a mutation rate of 5, whereas a rate of 2 is sufficient for the other attractors. This may indicate that for these two optimisations there is a difficult to leave local optimum, which does not exist for attractors 1 and 2. However, further investigations are necessary to identify such an optimum and to fathom its biological significance. Furthermore, the influence of the crossover operator on the optimisation was investigated. It was found that a crossover rate of 50% reduces the selection pressure and the networks are optimised more slowly and less well. This may be due to the fact that the exchange of entire partial functions between the transitions of the individual genes is rarely useful, as the genes do not have any functional dependence at this level and each individual gene performs an idiomatic biological function within the network. The exchange of initial states has only a minor influence on the CRC networks. Although there are smaller peaks in the fitness curve of all four attractors, which indicates the exchange, the networks adapt very quickly. However, it must also be considered that of 45 genes in the CRC network, only 9 are shared with the igfWnt network. Of these, only four (mTORC1, BCAT, ERK, TSC2_1) occur in attractors 1 and 2 and only 6 (mTORC1, BCAT, ERK, TSC2_1, AXIN, GSK3B_DC) in attractors 3 and 4. This means that according to the findings from 4.1, the influence of these four or six genes can only be slight.

Comparing the comparative adjacency matrices of the experiments in which the missing genes were ignored with those in which they were randomly added, it is immediately noticeable that the tendency to remove edges in particular does not exist here. On the contrary, almost all the old edges were retained. Looking at the difference matrix for attractor 1 and 2 (4.18, 4.20), it is noticeable that several edges were added and removed, but none more frequently than in 15% of all examples. There are several possible explanations for this. The first thing to consider is that the comparative adjacency matrix only indicates the presence or absence of edges. A pure change in a transition function remains hidden from this analysis. A more detailed investigation of the coevolved networks is necessary to prove effects at the level of transition functions. Furthermore, it should be noted that the set of potential optimal solutions is very large. Since only 6 (8) genes are defined, the value of the remaining 39 (37) genes is irrelevant for the fitness calculation. Thus, at least theoretically, there are 2^{39} or rather 2^{37} states with optimal fitness. Even if not all these states are of practical relevance, it is unlikely that a randomly acting optimisation like the coevolutionary algorithm will often find the same solution.

The comparative adjacency matrices for attractors 3 and 4 (4.22, 4.24) show a superficially similar picture as the results of attractors 1 and 2. However, it is particularly noticeable that for both attractors the self-referential edge $APC \rightarrow APC$ was removed in each trial. This gene is found in attractor definitions 3 and 4 but not in those of 1 and 2. Thus, this gene seems to be of particular importance for these attractors. In addition, other edges can be identified that were removed several times. For attractor 3, the edges $RAF \rightarrow ERK$, $AXIN \rightarrow DVL$, $S6K1 \rightarrow AKT$, which were each removed in 40% of all trials, should be highlighted. For attractor 4, the edges $RAF \rightarrow ERK(39\%)$, $S4K1 \rightarrow PI3K(35\%)$, $FAS \rightarrow NFkB(39\%)$ should be mentioned. However, the experiment was only repeated 20 times, therefore larger samples are needed in future work to clarify the significance of these genes for the network.

Influence of different sharing strategies

The experiment 4.25, which investigates the influence of different sharing strategies on the coevolution of the igfWnt / CRC coevolution, first shows, coherent with the findings already obtained, that the optimisation speed increases with the amount of defined genes within the attractors. All in all, the different sharing strategies have only a minor influence on the overall fitness curve. This can be explained by the generally low influence of the exchanged initial states on the coevolution of the igfWnt/CRC system. However, it is surprising that, in contrast to the expected results of the comparison of the different strategies in randomly generated networks (4.7), the *overwrite* operator causes a smaller peak than the *append* strategy during the first exchange of initial states. This effect is already almost lost in the second exchange. The fact that the *overwrite* strategy performs better after the first exchange can possibly be attributed to the large solution space of the optimisation problem. Through the exchange, some of the networks that had been created in the same generation certainly achieve a similarly good result under the new conditions, and thus they can stabilise the overall fitness of the population through the truncation selection. This is certainly facilitated by the fact that the few shared genes only exert a marginal influence on the overall fitness anyway and the shared conditions seem to tend to be more favourable for the CRC. On the other hand, for the *append* strategies, the addition of a second initial condition tends to make the optimisation problem more difficult at first. This can be attributed to the fact that two trajectories with possibly opposing influences must be optimised at the same time. However, this disadvantage is quickly compensated for by the fact that the populations are already accustomed to multiple shared initial states and are therefore become quickly more robust to the disruptive influence of coevolution.

An analysis of the number of different attractors, see sections 4.26, B.15, B.16, 4.27, that could be reached from all exchanged initial states of the coevolved networks shows, as expected, a similar result as the same investigation (4.8) for the randomly generated networks. For the *append* strategy it is possible to maintain the trajectory to the target attractor from all shared initial states. Interestingly, the *append 4* strategy for attractors 3, see section B.16) and 4 (4.27, also uniformly delivers exactly one attractor. Whereas for the less extensively defined attractors 1, see section 4.26) and 2 (B.15, a more heterogeneous picture emerges. Consequently, more different attractors are found for these two also in the *overwrite* than in the experiments with the CRC attractor definitions 3 and 4. This effect can possibly be attributed to the fact that the solution space is larger due to the less precise attractor definition. This makes it more likely that different solutions have a similarly good fitness and thus trajectories that have already been built can be discarded more easily. Nevertheless, even the *overwrite* strategy clearly shows that out of 20 possible different initial states, almost all states still belongs to the same basin after the population has converged. These results suggest that the networks may gain greater robustness through coevolution. A more detailed analysis of the attractor landscape of the coevolved networks in future work could provide new insights into this effect.

5.7 Comparison with other evolutionary and coevolutionary approaches

A large number of research have already been presented in the literature that use genetic algorithms in the context of Boolean networks. Most work in this area addresses the construction of Boolean networks based on biological experimental data [2] [43] [44] [45] [46] [47]. Moreover, some work focused on the creation of artificial Boolean networks with specific statistical properties, such as scale-free typologies [48]. Among these approaches, there are various genetic algorithms that, like this work, operate on Boolean networks represented as a trees. Moreover, some also use the Hamming distance between the target and the current attractor for the fitness calculation. However, these works differed greatly from this work due to their fundamentally different objectives.

Other work as presented by Fretter et. al. uses genetic algorithms to investigate the robustness of Boolean networks [49]. The work of Szeijka, Mihaljev and Drossel explore the fitness landscapes of BNs, with regard to the robustness of dynamic attractors against small perturbations through an adaptive walk, realised by an evolutionary approach [50] [51].

Research more comparable to the work presented here comes from Quayle and Bullock. They used simulated evolution to design specific phenotype behaviour. There work differs in, apart from the lack of a coevolutionary approach, that they use a so-called artificial genome to represent the Boolean networks. These genomes model a number of specific genes and their interactions as a binary string, which is strongly inspired by the natural gene structure [52].

Larry Bull introduced the so-called RBNK model, which extends Kauffman's RBN model [2]. As with RBN, randomly generated networks, consisting of R genes and B regular connections, each with a 2^B long binary string, describing the Boolean transfer function, are used. However, out of the R genes N ($N \subset R$) genes are fixed as so-called trait nodes. The N trait nodes of the network are linked to each other by K trait connections. By means of a genetic algorithm, the networks are now optimised against specific trait behaviour. However, the N trait genes are fixed with the K trait compounds and thus cannot be changed by evolution. Bull has investigated various properties of such systems, especially the network size [53]. In a further step these connections could be drawn not only within a network but also between the networks of different populations, extending the model to a coevolutionary approach.

A coevolution model similar to the RBNK model comes from Gorski et. all [54]. They divided a Boolean network into several sub-networks, which are connected to each other by permanent links, i.e. transition functions, which could not be influenced by evolution. These connections can have a similar influence on the evolution of the sub-networks as the exchange of initial states does in the work presented here. Nevertheless, the differences are also obvious. Such links must be defined in advance, which means that while known coevolutionary interactions can be simulated, no interesting new links can be discovered. Furthermore, this model does not allow interaction between populations beyond these linkages. Finally, this approach does not allow different populations to evolve independently and be optimised against separate goals before interacting with each other.

Thus, although the use of genetic algorithms is widespread in the field of Boolean network research, little attention has been paid to a coevolutionary approach and the work presented here has no equivalent in the literature.

6 Conclusion and future work

The intention of the work presented here was to develop and implement a framework that allows the simulation of the coevolution of any number of populations. It should be possible to represent interactions between populations by modifying many parameters and boundary conditions during the ongoing coevolution. Furthermore, a suitable tracking system should document all changes occurring during a coevolution for subsequent analysis. In conclusion, the framework presented here supports these functions and extends the core evolutionary algorithm with several features, such as different survivor selection. Contrary to the original intention, the framework does not adequately support multiple available computational kernels. The concurrency of the coevolutionary algorithm should be improved in further work. Nevertheless, the runtime is already sufficient to investigate even large networks such as the 45-gene CRC network in a reasonable amount of time. An exemplary investigation of coevolution on the igfWnt / CRC network system has so far yielded only limited specific insights into the influences of coevolution on the behaviour of this Boolean network. Nevertheless, in several extensive experiments, the functionality of the framework could be shown for smaller randomly generated networks as well as for the larger igfWnt / CRC network. Coevolutionary influence was simulated by exchanging initial states of the populations among each other, for which a large number of different operators and variants as well as analysis tools are available within the framework. Future work should also investigate other interactions, for example at the level of mutation rate, crossover and selection operators or population size. The necessary functions therefore are already implemented. Therefore, further systematic work on different biologically inspired networks could investigate the significance of this approach for a better understanding of gene regulation in the future.

A Network rules and specifications

A.1 IGFWnt network rules

```
targets, factors
Wnt,     Wnt
AXIN,    ERK | !Wnt
GSK3b,   !(Wnt | ERK | Akt)
GSK3B_DC, AXIN & GSK3b
BCAT,    !GSK3B_DC
TCF,     BCAT & !(JNK & FoxO)
FoxO,    !Akt & JNK
Rho,     (Wnt | PI3K | mTORC2) & !(RAC1 | PKC)
RAC1,    (Wnt | PI3K | mTORC2) & !Rho
MEKK1,   RAC1 | Rho
JNK,     MEKK1 | RAC1
PKC,     Rho | Wnt | mTORC2
IGF,     IGF
IRS,     IGF & !(S6K & JNK)
PI3K,    (IRS | Ras ) & !Rho
Akt,     PI3K | mTORC2
TSC2_1,  !(Akt | ERK) | GSK3b
mTORC1,  !TSC2_1
S6K,     mTORC1 | GSK3b
Ras,     IGF | Wnt
Raf,     (Ras | PKC) & !Akt
ERK,     Raf
mTORC2,  !(S6K | GSK3b) & (PI3K | TSC2_1)
```

A.1.1 igfWnt specifications

The following initial conditions / attractor states are the inverse states of the four single state attractors of the network.

```
Initial condition:
Wnt  !AXIN !GSK3b !GSK3B_DC BCAT TCF FoxO Rho
RAC1 MEKK1 JNK PKC IGF IRS PI3K Akt !TSC2_1
mTORC1 !S6K Ras Raf ERK mTORC2

Attractor:
Wnt !AXIN !GSK3b !GSK3B_DC BCAT TCF FoxO Rho
RAC1 MEKK1 JNK PKC IGF IRS PI3K Akt !TSC2_1
mTORC1 !S6K Ras Raf ERK mTORC2


Initial condition:
!Wnt AXIN GSK3b GSK3B_DC !BCAT !TCF FoxO Rho
!RAC1 !MEKK1 !JNK !PKC IGF IRS !PI3K !Akt
TSC2_1 !mTORC1 !S6K !Ras Raf ERK mTORC2

Attractor:
!Wnt AXIN GSK3b GSK3B_DC !BCAT !TCF FoxO Rho
!RAC1 !MEKK1 !JNK !PKC IGF IRS !PI3K !Akt
TSC2_1 !mTORC1 !S6K !Ras Raf ERK mTORC2


Initial condition:
!Wnt AXIN GSK3b GSK3B_DC !BCAT !TCF FoxO Rho
!RAC1 !MEKK1 !JNK !PKC !IGF IRS !PI3K !Akt
TSC2_1 !mTORC1 !S6K !Ras Raf ERK mTORC2

Attractor:
!Wnt AXIN GSK3b GSK3B_DC !BCAT !TCF FoxO Rho
!RAC1 !MEKK1 !JNK !PKC !IGF IRS !PI3K !Akt
TSC2_1 !mTORC1 !S6K !Ras Raf ERK mTORC2
```

```


Initial condition:
Wnt !AXIN GSK3b GSK3B_DC !BCAT !TCF FoxO Rho
!RAC1 !MEKK1 !JNK PKC !IGF IRS !PI3K !Akt
TSC2_1 !mTORC1 !S6K !Ras Raf ERK mTORC2

Attractor:
Wnt !AXIN GSK3b GSK3B_DC !BCAT !TCF FoxO Rho
!RAC1 !MEKK1 !JNK PKC !IGF IRS !PI3K !Akt
TSC2_1 !mTORC1 !S6K !Ras Raf ERK mTORC2
```

The following specifications are the ones used for the second approach of the igfWnt/CRC experiment. The states are the four single state attractors of the ifgWnt network.

```
state1:
!Wnt AXIN GSK3b GSK3B_DC !BCAT !TCF !FoxO !Rho
!RAC1 !MEKK1 !JNK !PKC !IGF !IRS !PI3K !Akt
TSC2_1 !mTORC1 S6K !Ras !Raf !ERK !mTORC2

state2:
Wnt !AXIN !GSK3b !GSK3B_DC BCAT TCF !FoxO !Rho
RAC1 MEKK1 JNK PKC !IGF !IRS PI3K Akt
!TSC2_1 mTORC1 S6K Ras !Raf !ERK !mTORC2

state3:
Wnt !AXIN !GSK3b !GSK3B_DC BCAT TCF !FoxO !Rho
RAC1 MEKK1 JNK PKC IGF !IRS PI3K Akt
!TSC2_1 mTORC1 S6K Ras !Raf !ERK !mTORC2

state4:
!Wnt AXIN !GSK3b !GSK3B_DC BCAT TCF !FoxO !Rho
RAC1 MEKK1 JNK !PKC IGF !IRS PI3K Akt
!TSC2_1 mTORC1 S6K Ras !Raf !ERK !mTORC2
```

A.2 CRC network rules

```
targets, factors
EGFR,    ERBB1_2 & PGE2 & !ERK
KRAS,    EGFR & !DC
RAF,     KRAS & !AKT & !ERK
MEK,     RAF
scf,     IQGAP1 & RAF & MEK
ERK,     (scf | PAK1) & !PP2A
eIF4F,   ERK | mTORC1
EBP1,    !ERK & !mTORC1
cMYC,    (ERK | TCF_LEF ) & !APC & (!PP2A|CIP2A) &
         !GSK3B_deg & ERK
cJUN,    (ERK | TCF_LEF | COX2) & JNK
PI3K,    PGE2 | EGFR | KRAS
AKT,     (PI3K | PAK1 | SNAIL1) & !PP2A &
         (NFkB | TCF_LEF | SNAIL1)
TSC2_1,  GSK3B_deg & !ERK & !AKT
mTORC1,  !TSC2_1
S6K1,    mTORC1 & PI3K
Tiam1,   (EGFR | AKT ) & !PP2A  & (cMYC | TCF_LEF)
RAC1,    (Tiam1 | IQGAP1 | mTORC1 | PI3K | FZD) & !APC
JNK,     RAC1
PAK1,    RAC1 & !PP2A
IQGAP1,  !GSK3B_deg
PGE2,    COX2 | (SNAIL1 & HDAC2)
HDAC2,   !APC & cMYC
ERBB1_2, HDAC2 | AP1 | TCF_LEF
cFOS,    (TCF_LEF | ERK) & (RSK | ERK)
RSK,     PI3K & ERK
AP1,     cFOS & cJUN
COX2,    AP1 | NFkB | TCF_LEF
FAS,     NFkB & !BCAT
NFkB,    (RAC1 | ERK | AKT) & HDAC2 & GSK3B_cyt
Ecad,    (!SNAIL1 & !HDAC2 & !AKT)|
         (!SNAIL1 & HDAC2 & AKT)  |
         (!SNAIL1 & !HDAC2 & AKT) |
         (!SNAIL1 & HDAC2 & !AKT) |
         (SNAIL1 & !HDAC2 & !AKT) |
         (SNAIL1 & HDAC2 &!AKT)   |
         (SNAIL1 & !HDAC2 & AKT)
TIGHT_JUNCTION, Ecad & (!IQGAP1 | APC | (RAC1 & IQGAP1))
SNAIL1,  ((AXIN2 | ERK | NFkB) &  !GSK3B_deg) |
         (AXIN2 & GSK3B_deg)
AXIN2,   TCF_LEF
FZD,     MEK | ERK | JNK
DVL,     FZD
GSK3B_cyt, !APC | GSK3B_deg
GSK3B_deg, !PGE2 & !AKT & !ERK & !NFkB
GSK3B_DC,  AXIN
APC,     APC
AXIN,    !DVL
DC,      !DVL & GSK3B_DC & APC & (AXIN | AXIN2)
BCAT,    !DC
TCF_LEF, BCAT & KRAS & RAC1 &
         (PAK1 | AKT | MEK | IQGAP1 | Tiam1 |
         NFkB | SNAIL1)
PP2A,    !CIP2A
CIP2A,   EGFR | MEK | ERK
```

A.2.1 CRC specifications

```
Initial condition:
EGFR KRAS RAF MEK scf ERK eIF4F !EBP1 cMYC cJUN PI3K
AKT !TSC2_1 mTORC1 S6K1 Tiam1 RAC1 JNK PAK1 IQGAP1
PGE2 HDAC2 ERBB1_2 cFOS RSK AP1 COX2 FAS NFkB !Ecad
!TIGHT_JUNCTION SNAIL1 AXIN2 FZD DVL !GSK3B_cyt
!GSK3B_deg !GSK3B_DC !APC !AXIN !DC BCAT TCF_LEF
!PP2A CIP2A
```

```

Attractor 1:
mTORC1 ERK !BCAT !TSC2_1 !cMYC !AXIN2

Attractor 2:
mTORC1 ERK !BCAT !TSC2_1 !c_MYC !AXIN2 S6K RSK

Attractor 3:
mTORC1 ERK !BCAT !TSC2_1 !c_MYC !AXIN2 FZD DVL GSK3B_deg
GSK3B_DC APC AXIN !TCF_LEF

Attractor 4:
mTORC1 ERK !BCAT !TSC2_1 !c_MYC !AXIN2 FZD DVL GSK3B_deg
GSK3B_DC APC AXIN !TCF_LEF S6K RSK
```

B Annex Results

B.1 Adjacency matrix of the orginal igfWnt and CRC network

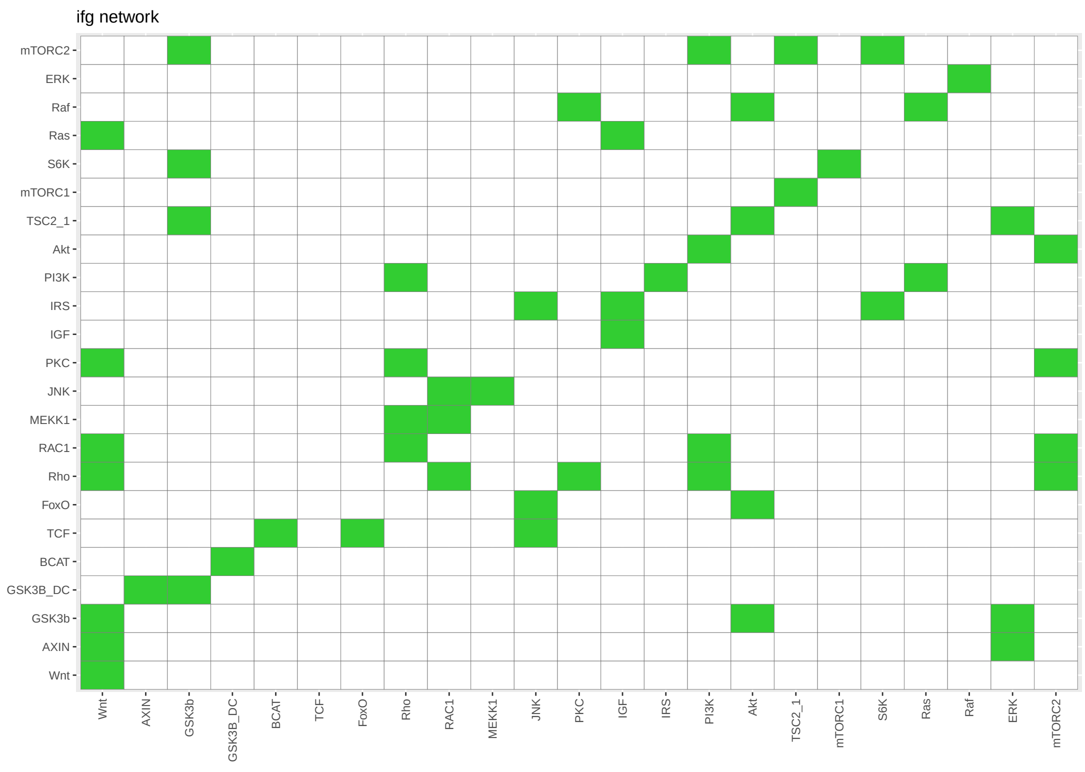


Figure B.1: Adjacency matrix of the original igfWnt network.

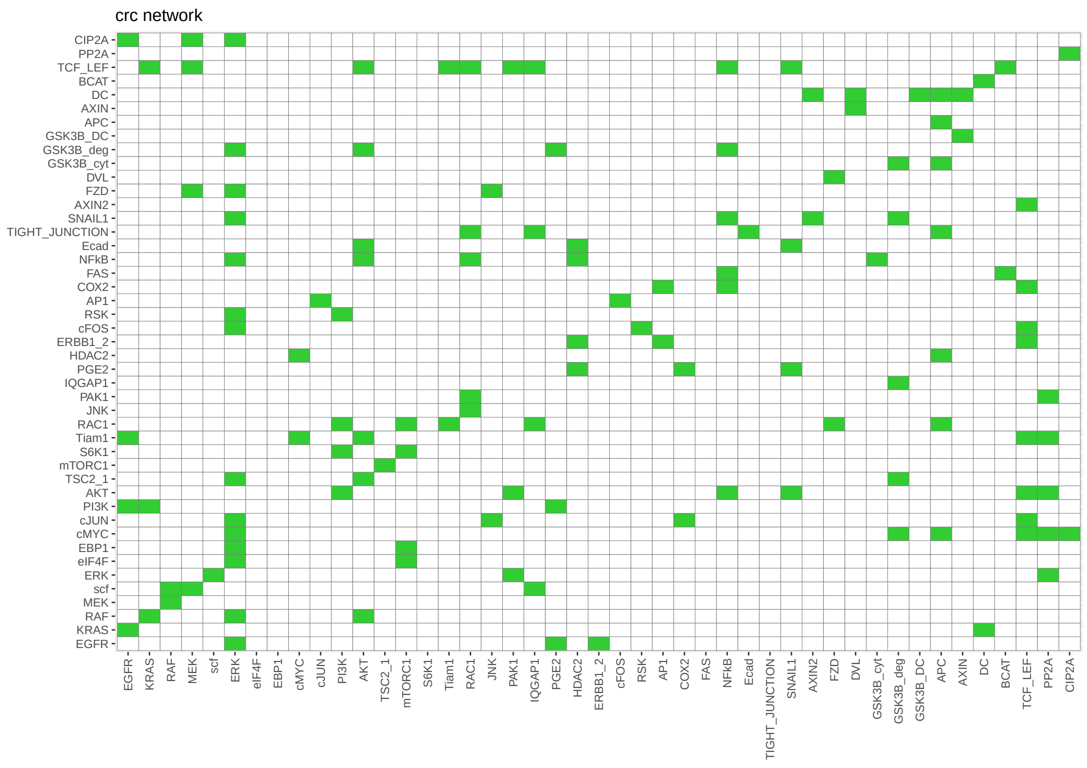


Figure B.2: Adjacency matrix of the original CRC network.

B.2 Annex: Biologically plausible experiment - Addition of missing gene values

B.2.1 Attractor 3

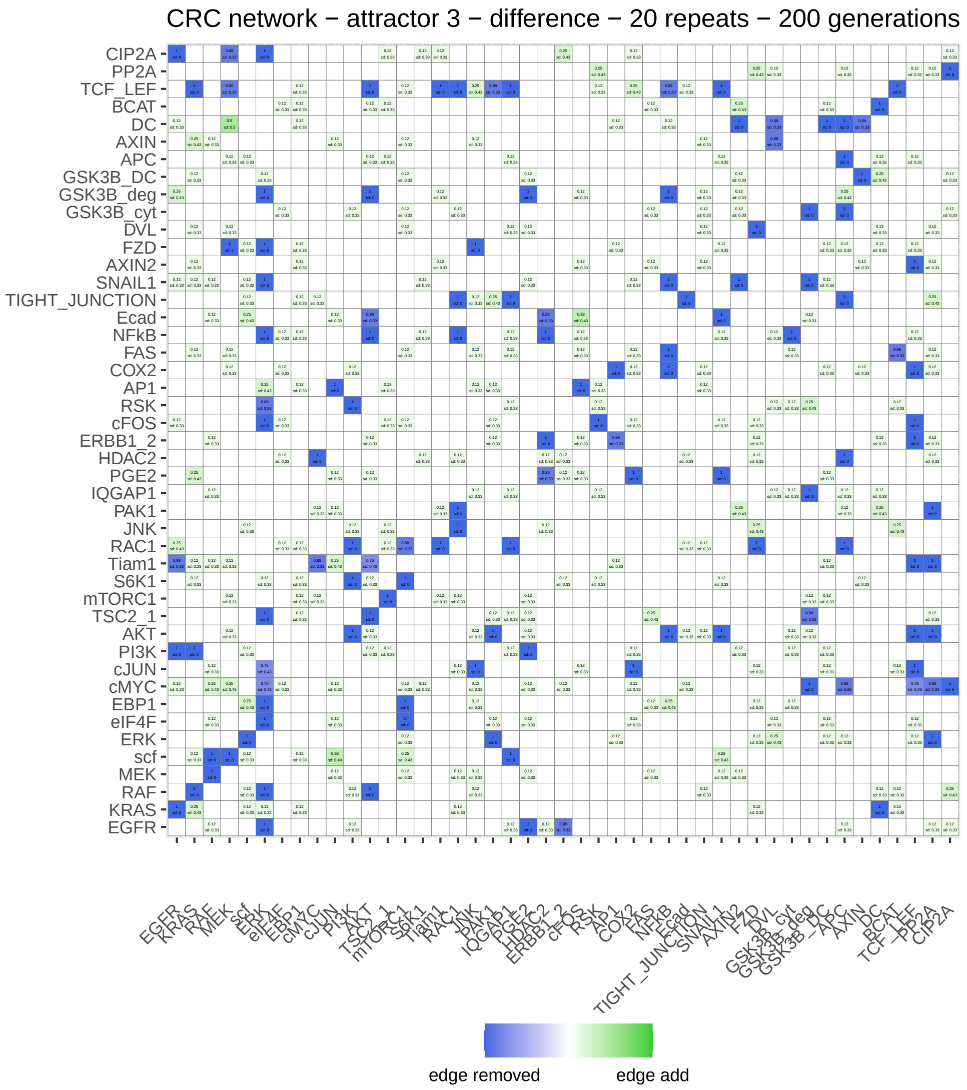


Figure B.3: Differences adjacency matrix of the coevolved CRC network with attractor 3 compared with the original network. Blue are deleted edges, green are added edges. In white are edges with no differences between the networks. The numbers shows the mean and standards derivation.

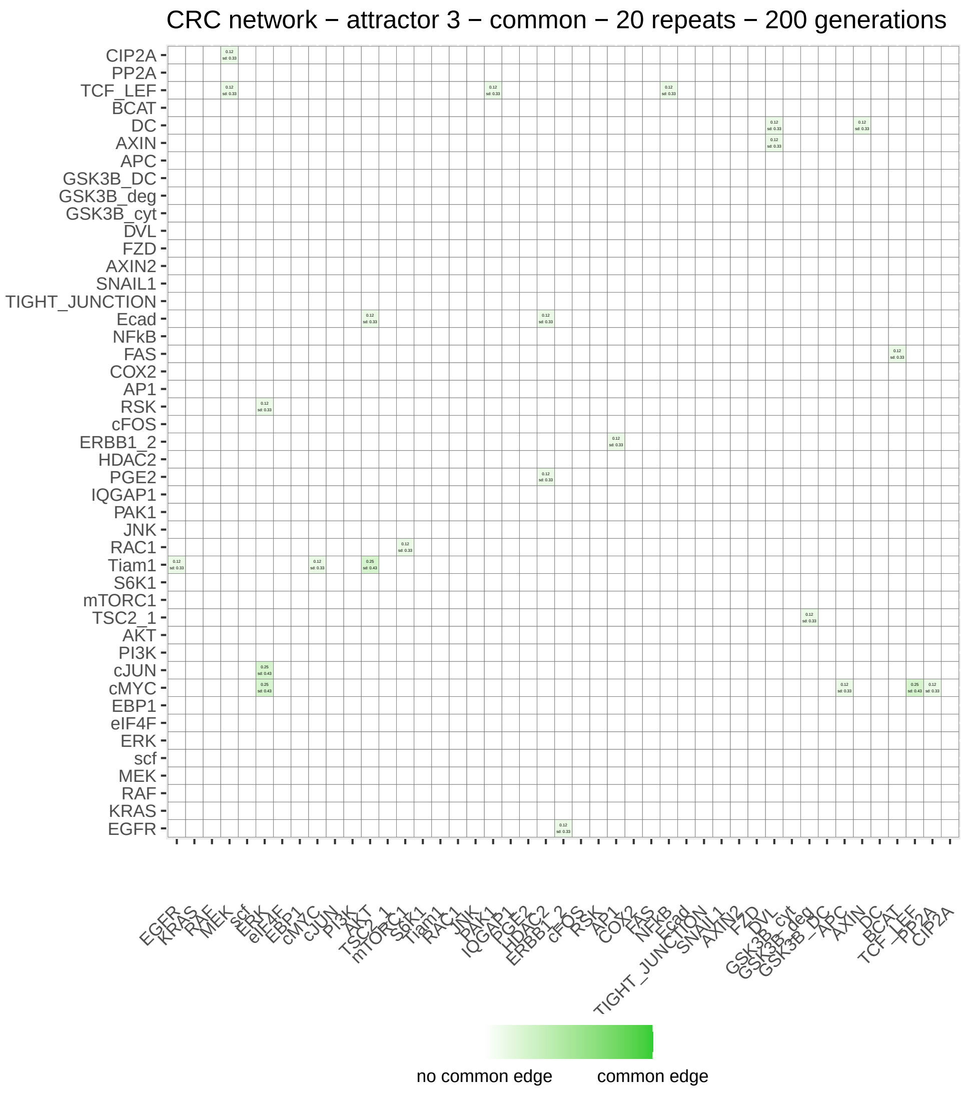


Figure B.4: Common edges adjacency matrix of the coevolved CRC network with attractor 3 compared with the original network. Green indicates common edges. The numbers shows the mean and standard derivation.

B.2.2 Attractor 4

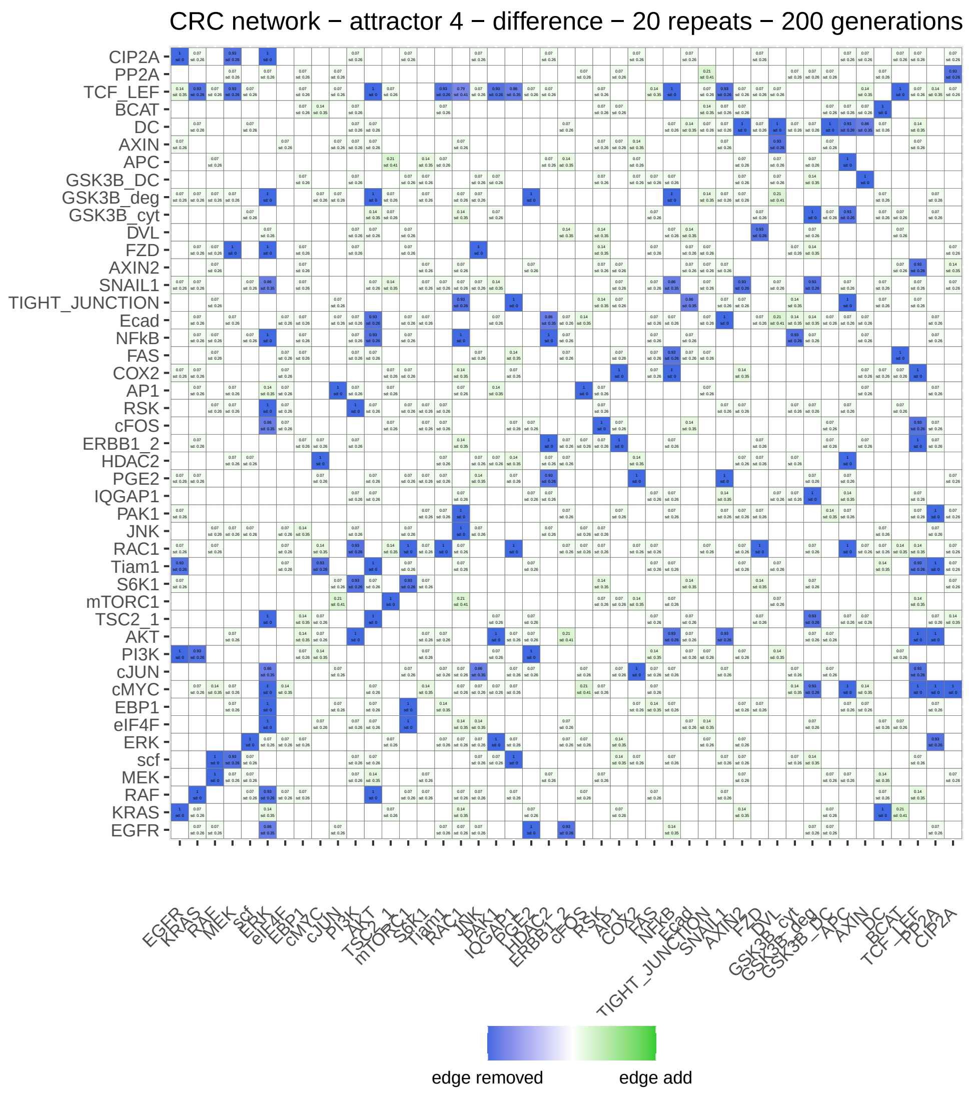


Figure B.5: Differences adjacency matrix of the coevolved CRC network with attractor 4 compared with the original network. Blue are deleted edges, green are added edges. In white are edges with no differences between the networks. The numbers shows the mean and standard derivation.

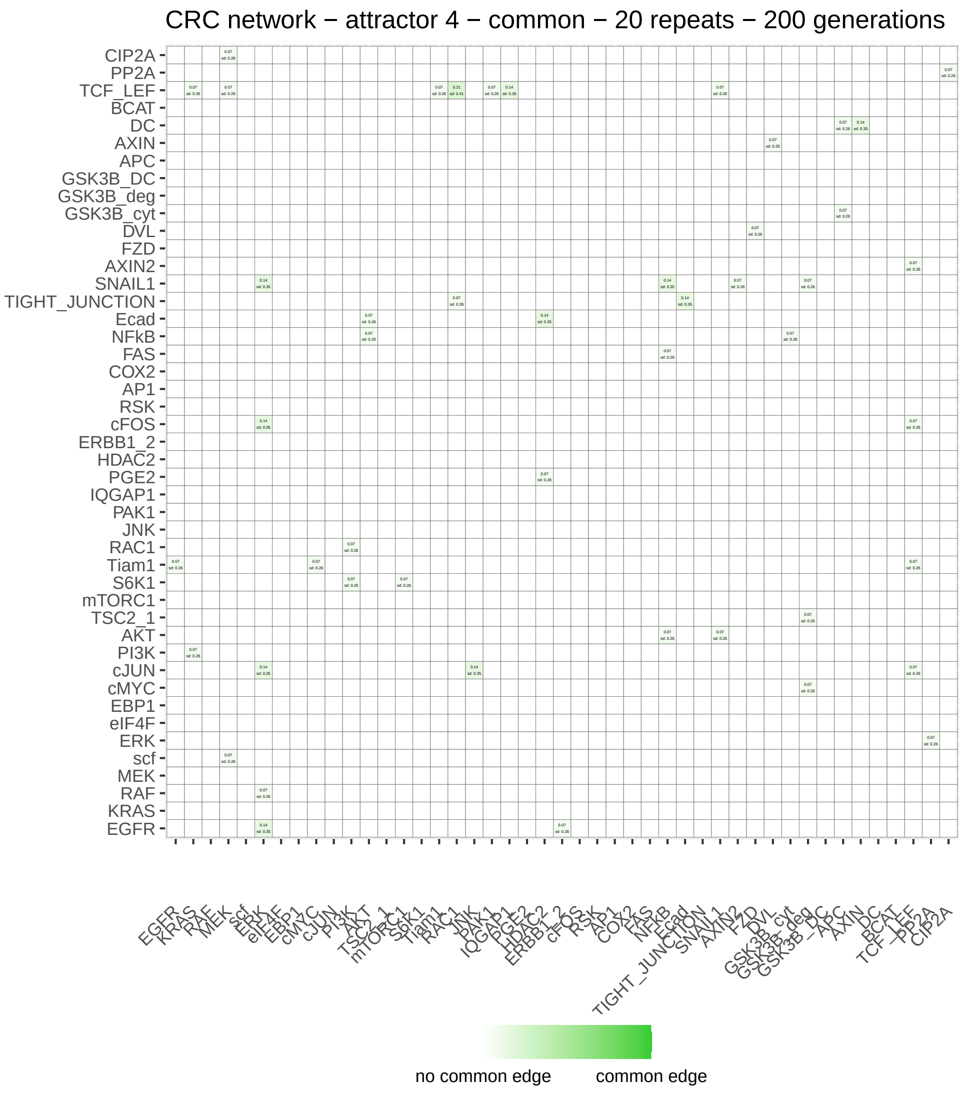


Figure B.6: Common edges adjacency matrix of the coevolved CRC network with attractor 4 compared with the original network. Green indicates common edges. The numbers shows the mean and standard derivation.

B.2.3 Attractor 4 - Simulation with large population size

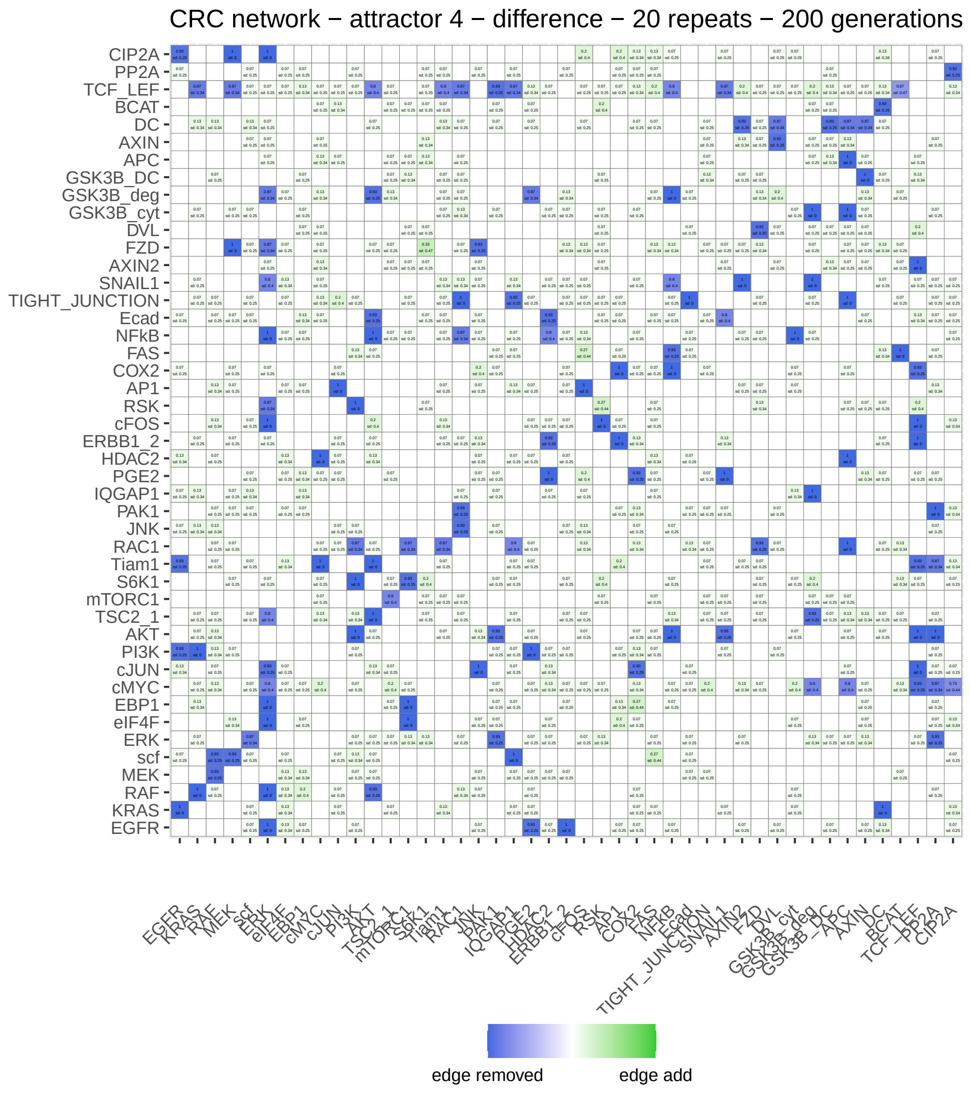


Figure B.7: Differences adjacency matrix of the coevolved CRC networks compared with the original CRC network. With a population size of 400 and 15 mutations per individual and generation in the CRC strain. Exchange of initial states from igfWnt to CRC each 100 generations. Blue are deleted edges, green are added edges. In white are edges with no differences between the networks. The numbers shows the mean and standard derivation.

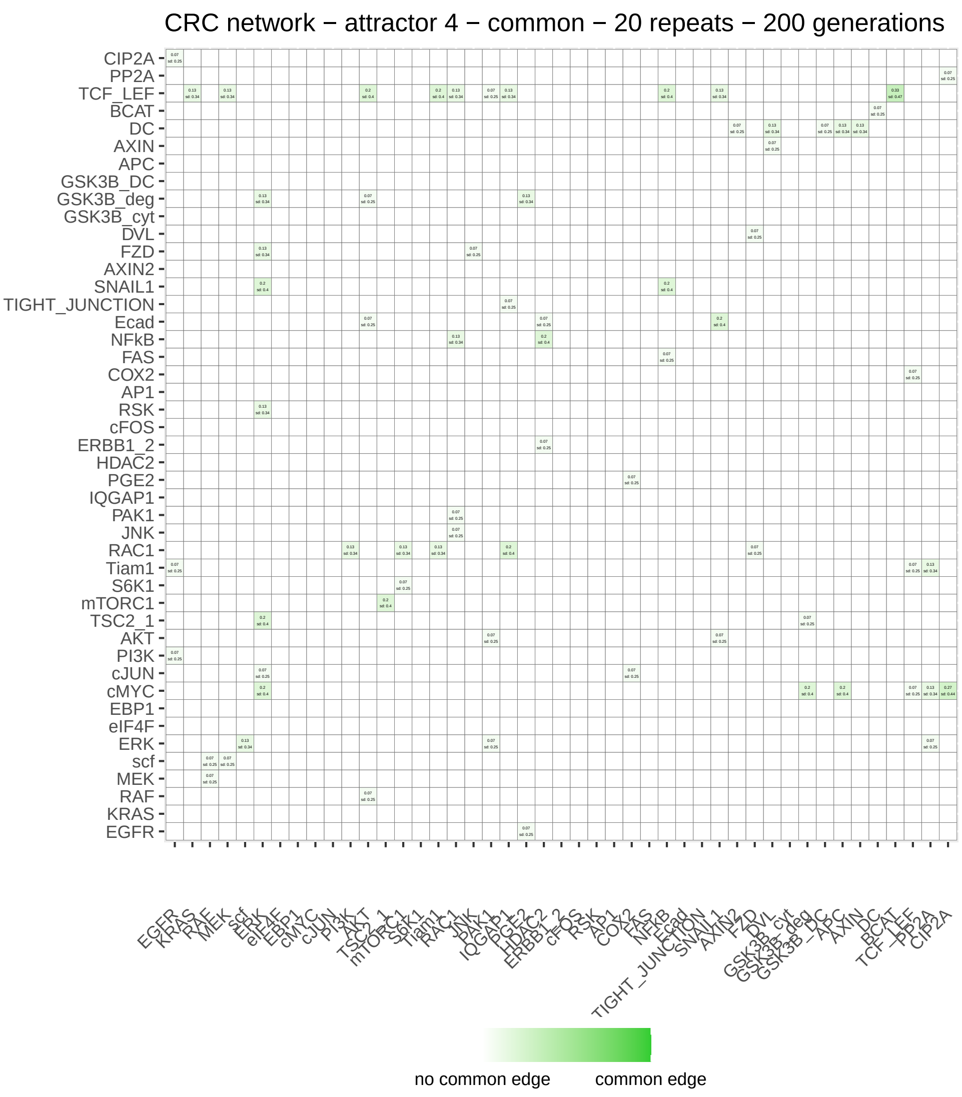


Figure B.8: Common edges adjacency matrix of the coevolved CRC networks compared with the original CRC network. With a population size of 400 and 15 mutations per individual and generation in the CRC strain. Exchange of initial states from igfWnt to CRC each 100 generations. Green indicates common edges. The numbers shows the mean and standard derivation.

B.2.4 Influence of crossover

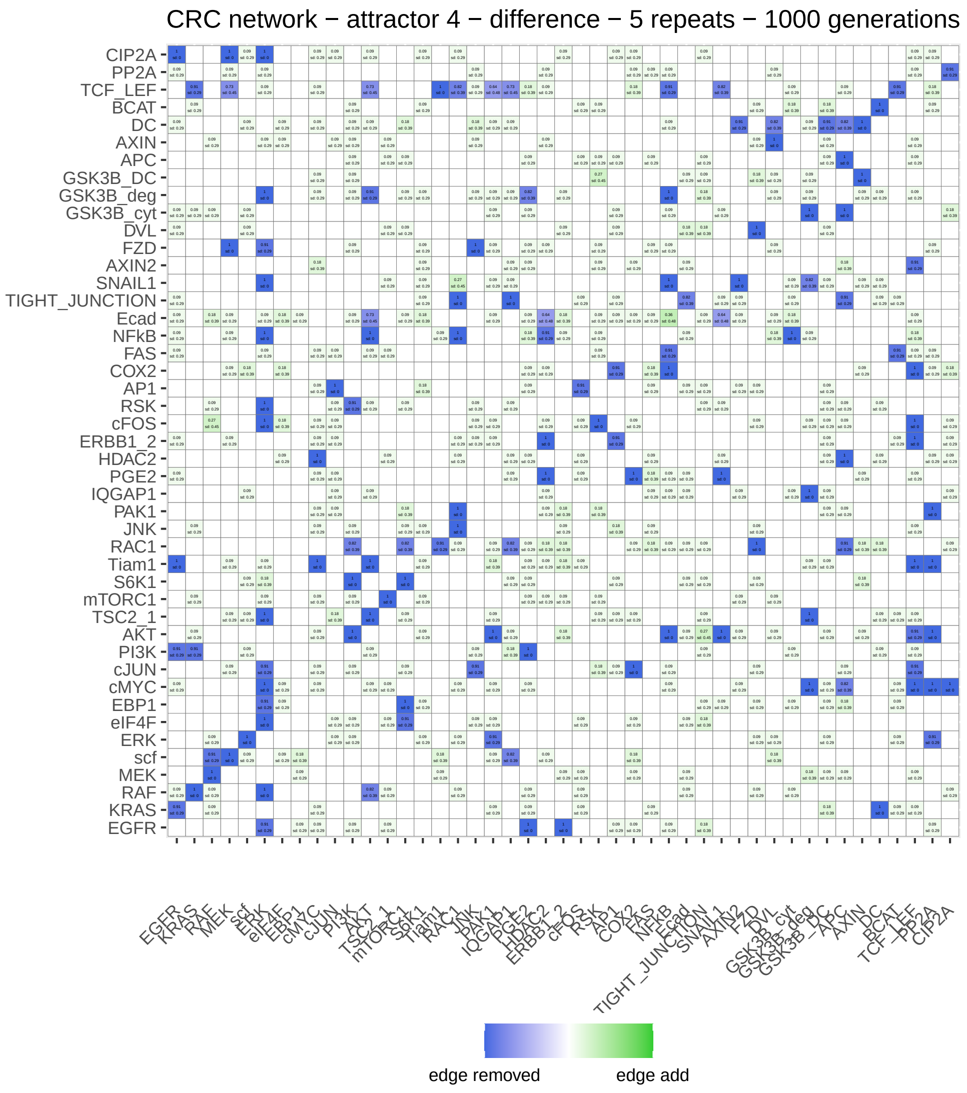


Figure B.9: Differences adjacency matrix of the coevolved CRC networks compared with the original CRC network. Used 150 individuals per generation and 50% cross-over operations . Blue are deleted edges, green are added edges. In white are edges with no differences between the networks. The numbers shows the mean and standard derivation.

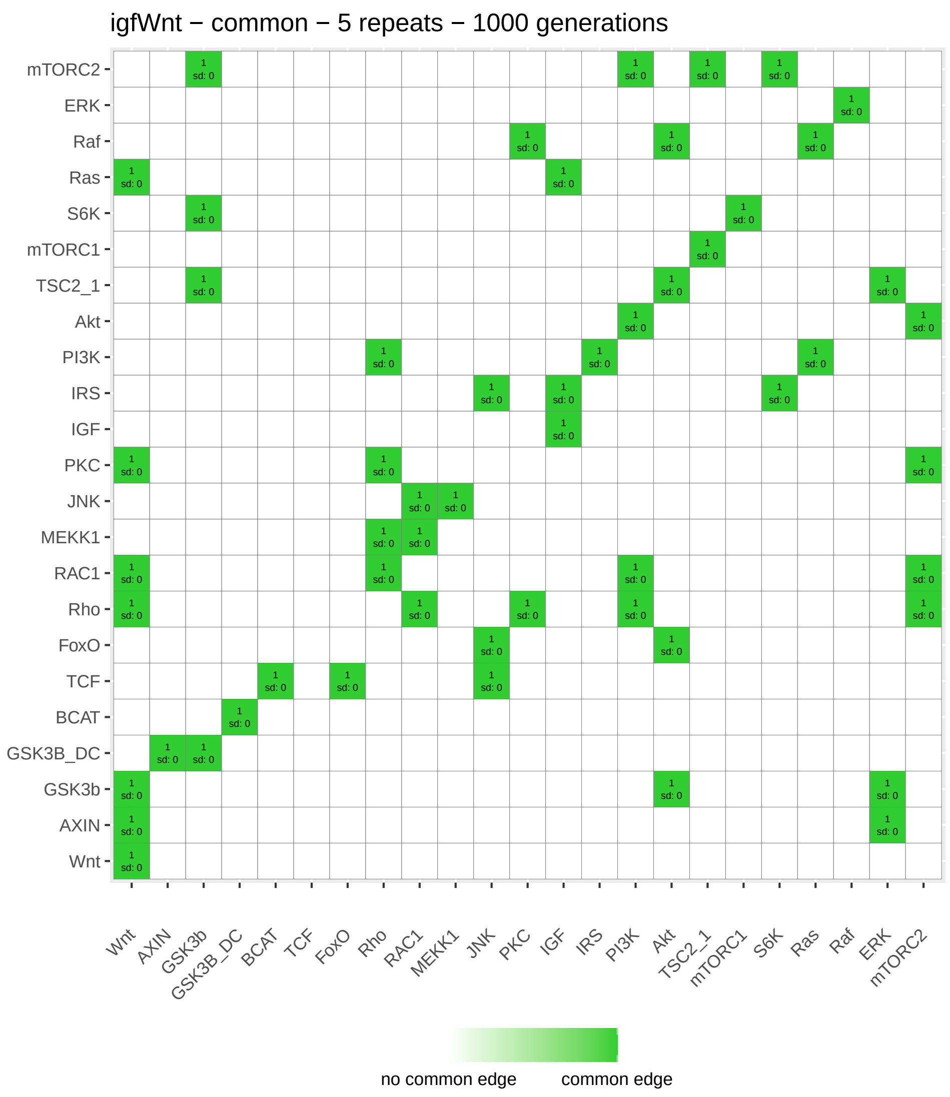


Figure B.10: Common edges adjacency matrix of the coevolved igfWnt networks compared with the original ifgWnt network with the second target attractor definition. Green indicates common edges. The numbers shows the mean and standard derivation.

B.3 Annex: Biologically plausible experiment - Masking missing gene values

Attractor 1

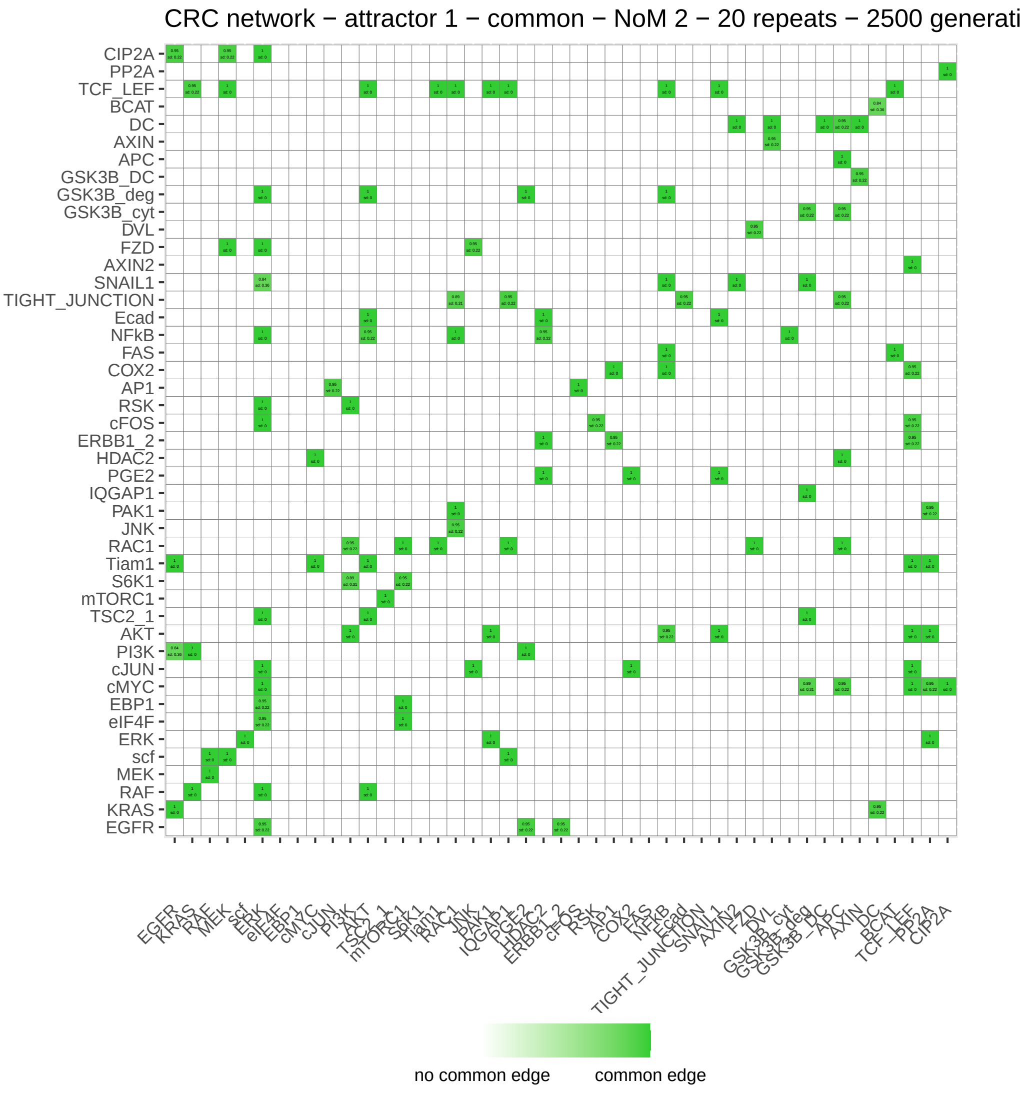


Figure B.11: Common edges adjacency matrix of the coevolved CRC networks with attractor definition 1 compared with the original CRC network. The mutation rate (NoM) is 2 without crossover. Green indicates common edges. The labelling shows the mean and standard derivation.

Attractor 2

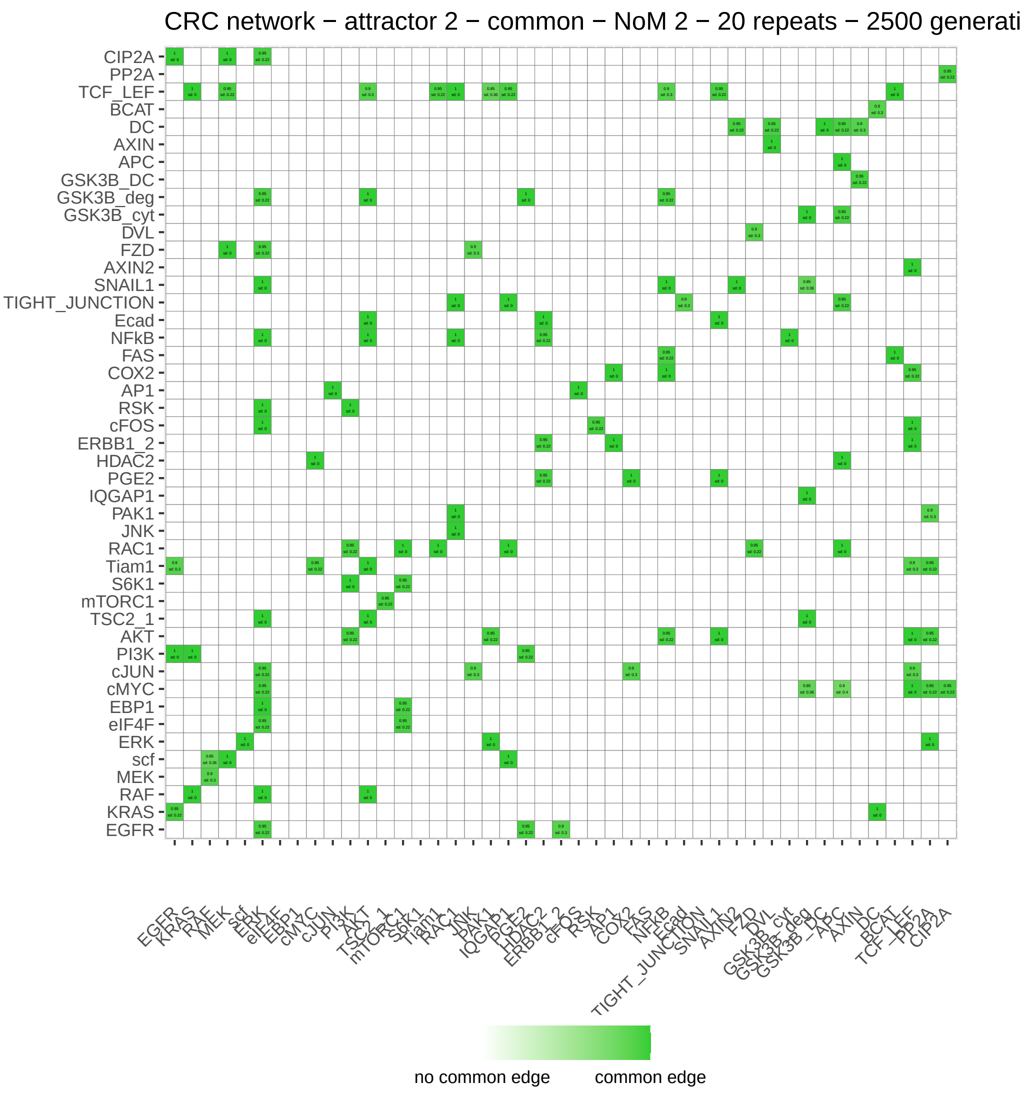


Figure B.12: Common edges adjacency matrix of the coevolved CRC networks with attractor definition 2 compared with the original CRC network. Number of Mutations (NoM) = 2. Green indicates common edges. The labelling shows the mean and standard derivation.

Attractor 3

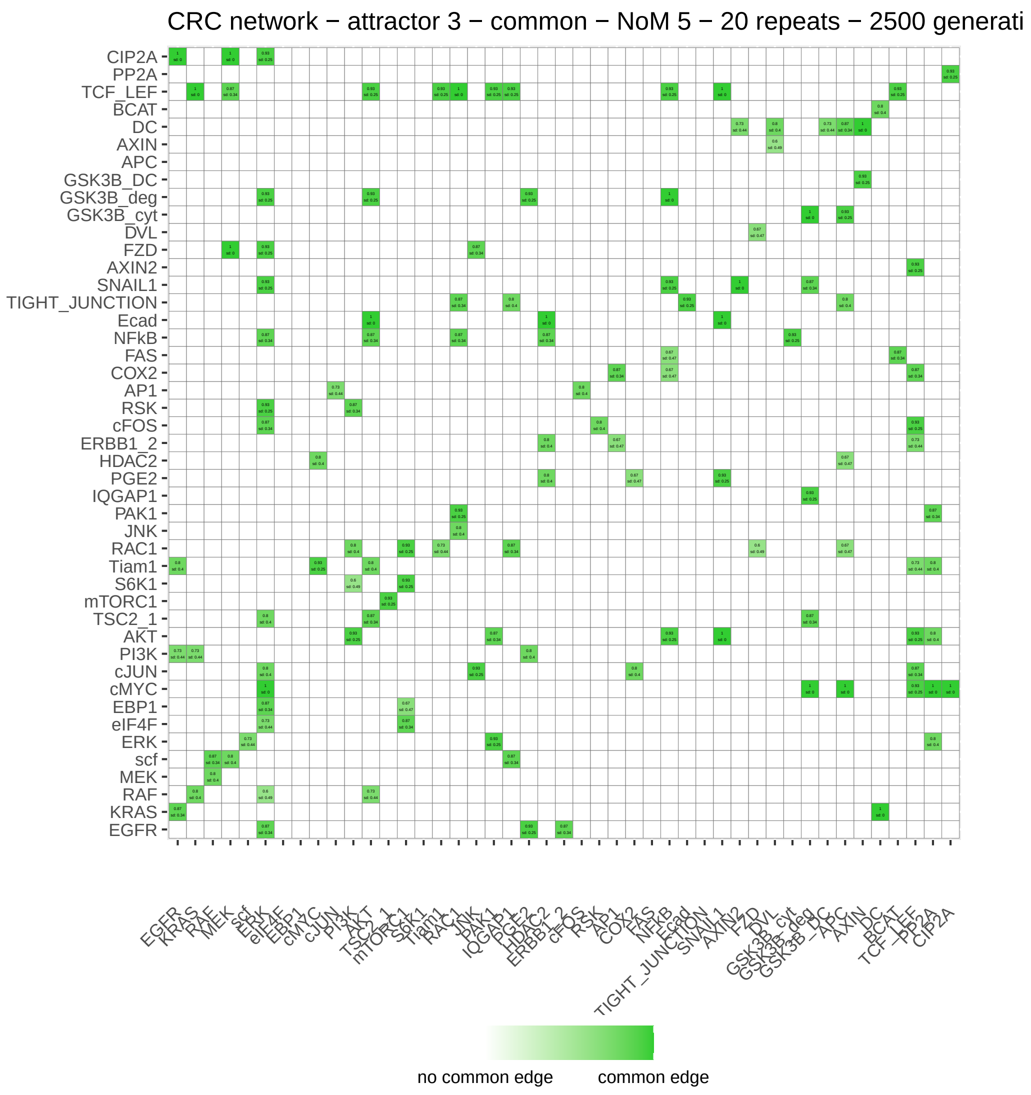


Figure B.13: Common edges adjacency matrix of the coevolved CRC networks with attractor definition 3 compared with the original CRC network. Number of Mutations = 5. Green indicates common edges. The labelling shows the mean and standard derivation.

Attractor 4

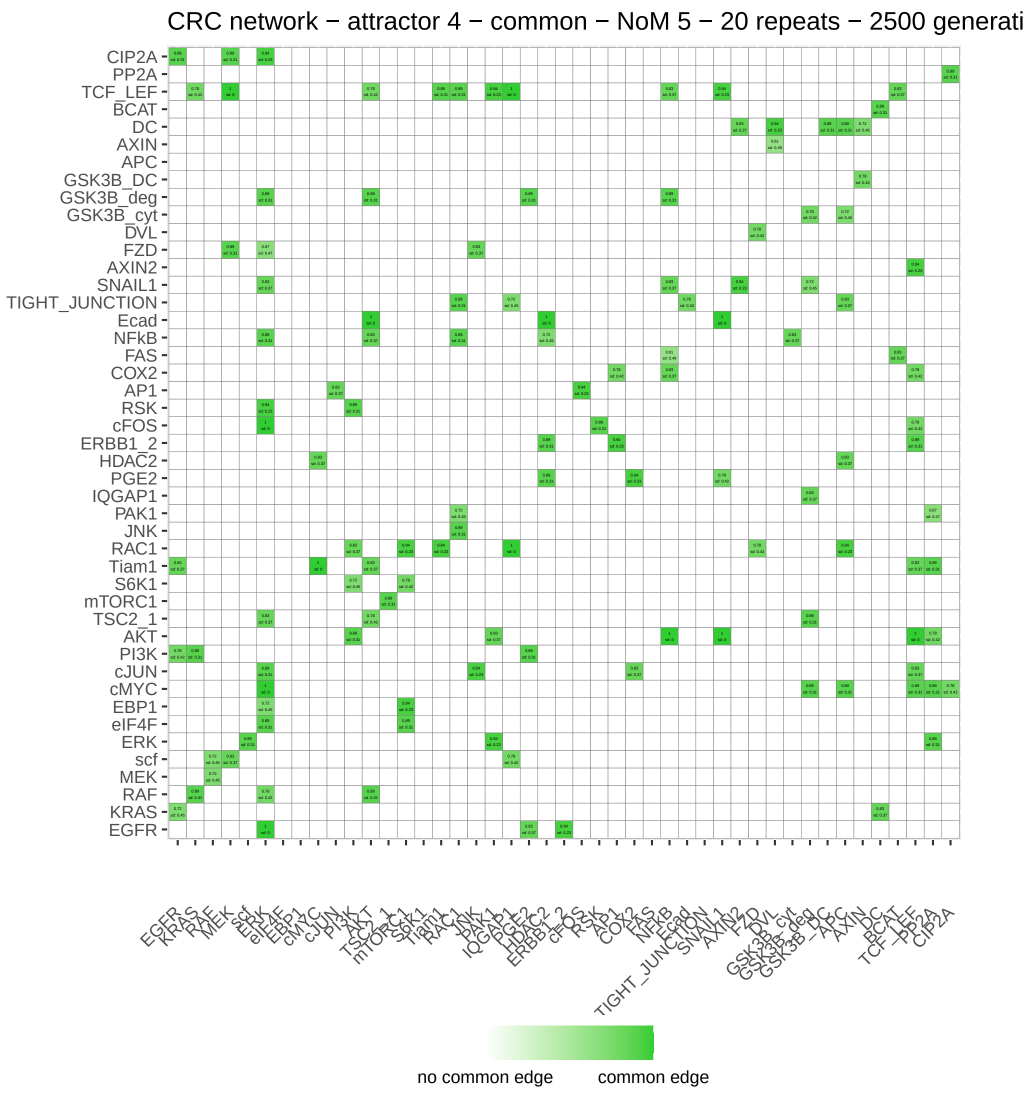
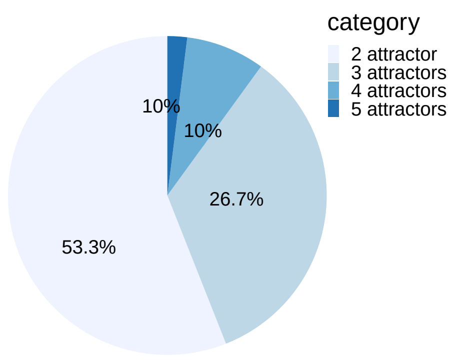
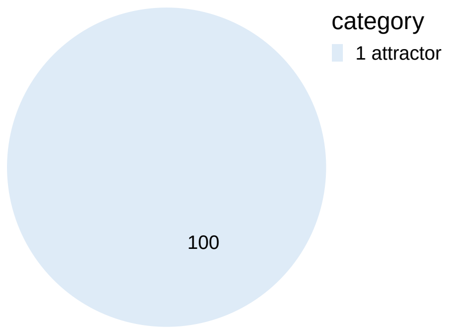


Figure B.14: Common edges adjacency matrix of the coevolved CRC networks with attractor definition 4 compared with the original CRC network. Number of Mutations = 5. Green indicates common edges. The labelling shows the mean and standard derivation.

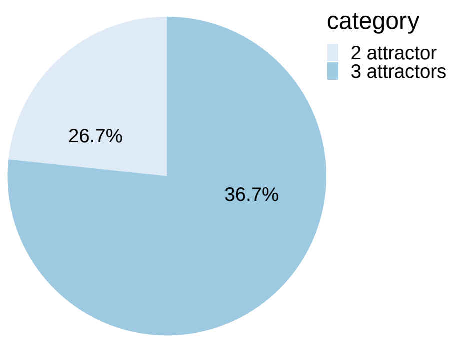
B.3.1 Influence of different sharing operators



(a) Overwrite

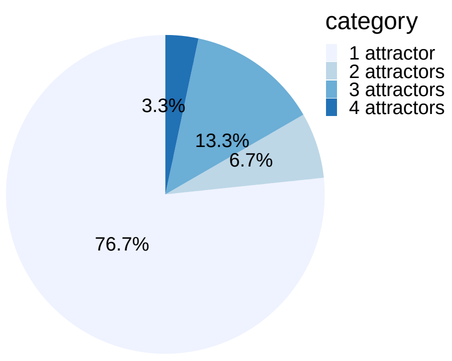


(b) Append

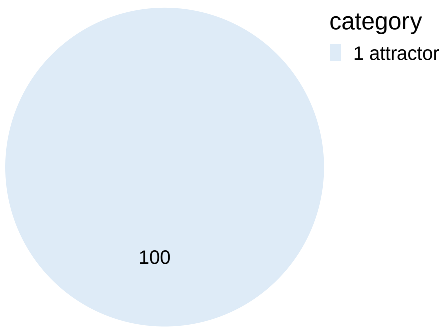


(c) Append 4

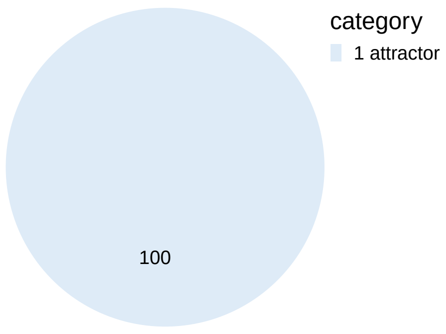
Figure B.15: Number of attractors - with exchange modes *overwrite*, *append* and *append 4* for the CRC network with attractor definition 2.



(a) Overwrite



(b) Append



(c) Append 4

Figure B.16: Number of attractors - with exchange modes *overwrite*, *append* and *append 4* for the CRC network with attractor definition 3.

Differences adjacency matrix for attractor 4

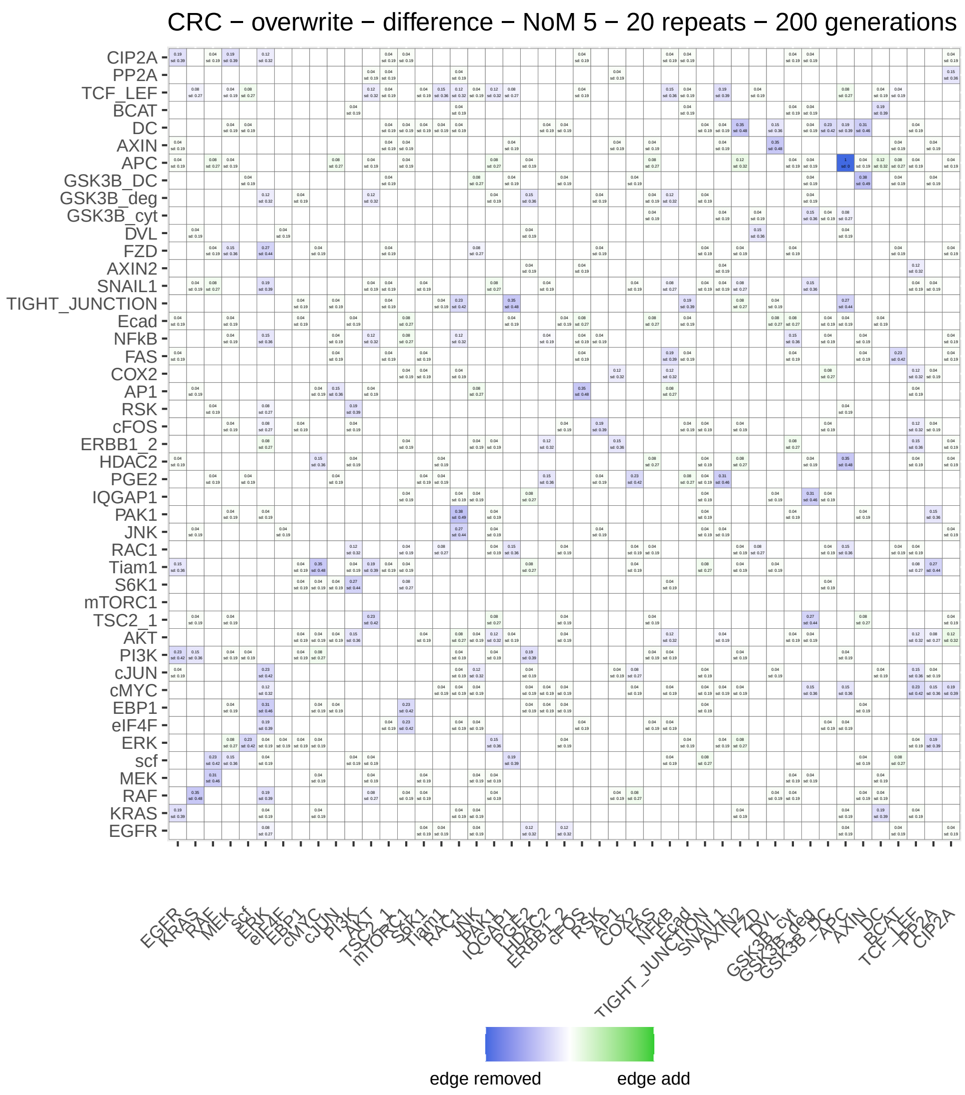


Figure B.17: Differences adjacency matrix of the coevolved CRC networks compared with the original CRC network. Initial states exchange with the *overwrite* operator. The labeling shows the mean and standard derivation.

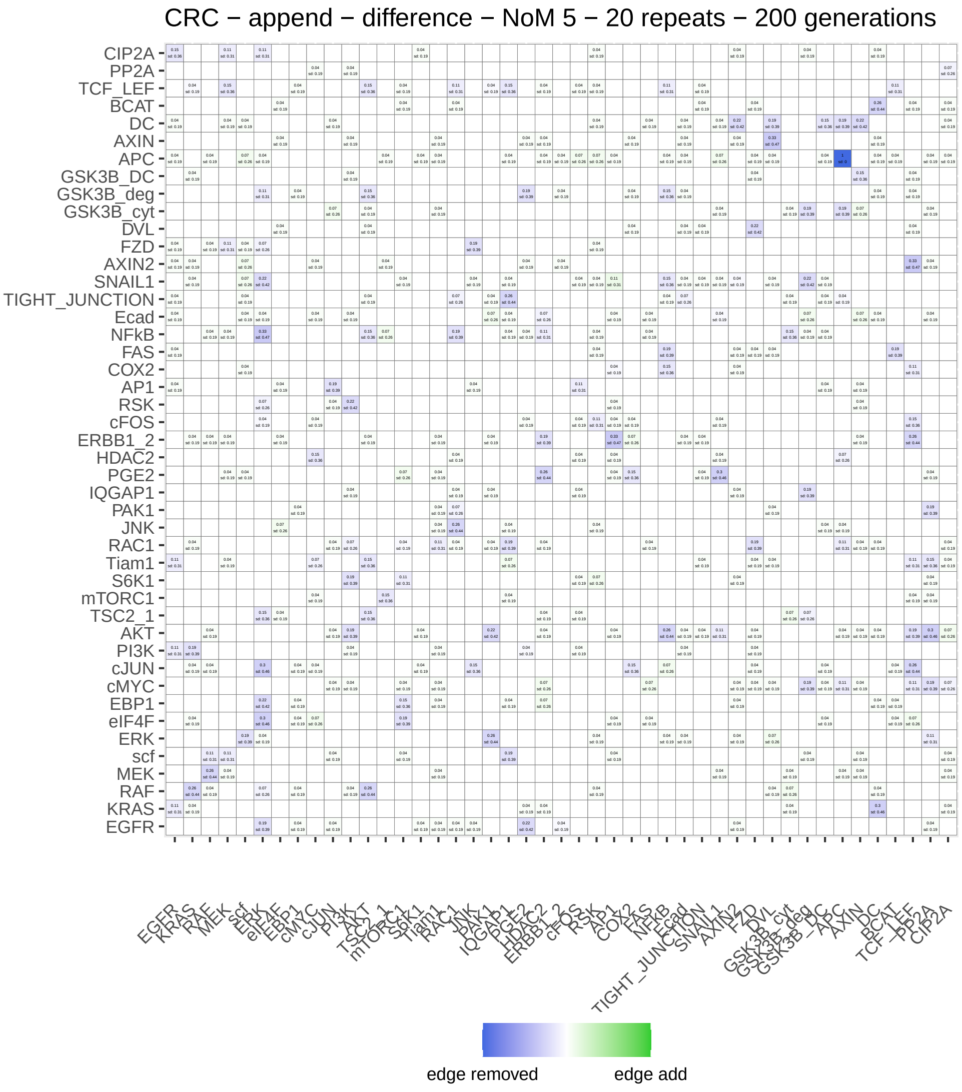


Figure B.18: Differences adjacency matrix of the coevolved CRC networks compared with the original CRC network. Initial states exchange with the *append* operator. The labeling shows the mean and standard derivation.

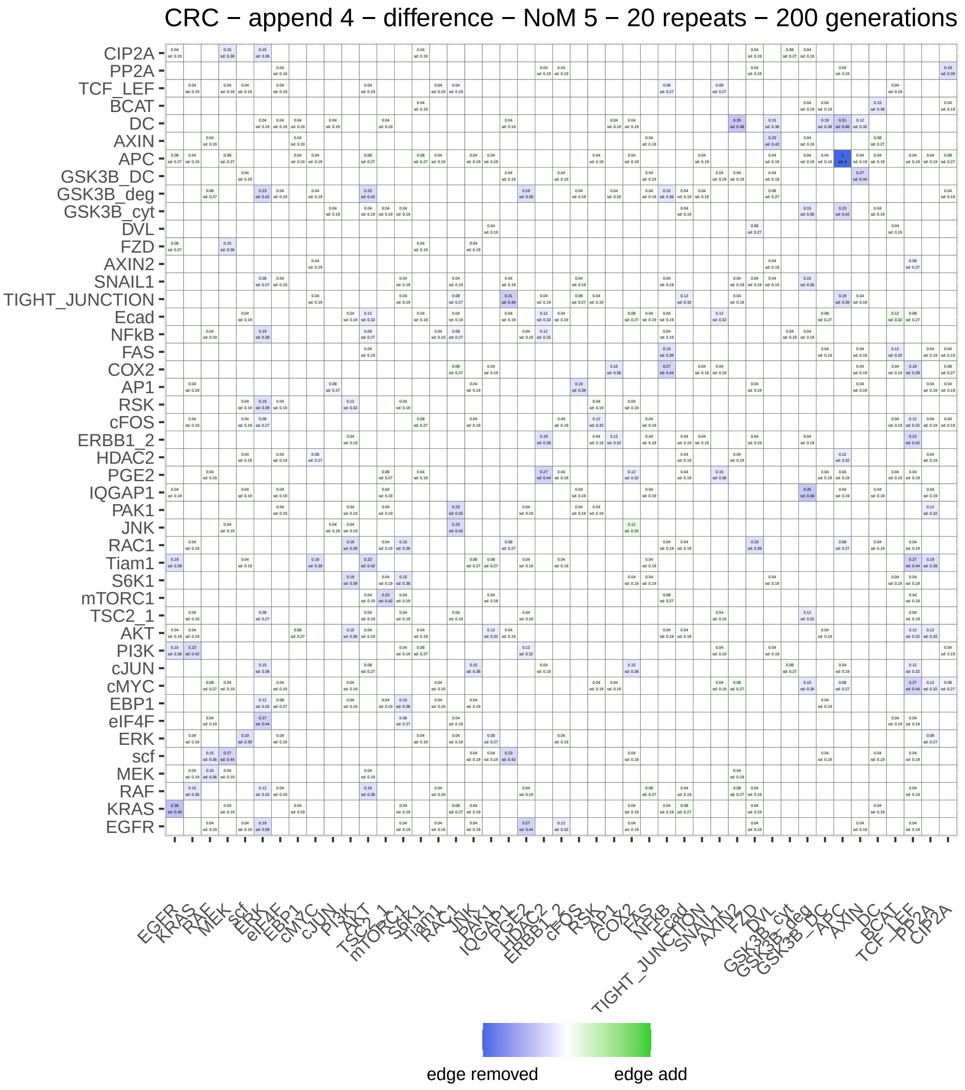


Figure B.19: Differences adjacency matrix of the coevolved CRC networks compared with the original CRC network. Initial states exchange with the *append 4* operator. The labeling shows the mean and standard derivation.

Bibliography

[1] Neil A. Campbell and Jean B. Reece. *Biology*. 8th. Pearson Benjamin Cummings, 2008.

[2] S.A. Kauffman. *The origins of order. Self-organization and selection in evolution*. 1th. https://doi.org/10.1046/j.1420-9101.1994.7040518.x: Oxford University Press, 2003.

[3] Miguel Nicolau and Marc Schoenauer. "Evolving Scale-Free Topologies using a Gene Regulatory Network Model". In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. 2008, pp. 3747–3754. DOI: 10.1109/CEC.2008.4631305.

[4] B. Alberts, D. Bray, and K. Hopkin. *Lehrbuch der Molekularen Zellbiologie*. 4th. Graw, J in Wiley-VCH., 2012.

[5] D. Zhou et al. "Boolean networks for modeling and analysis of gene regulation". In: *Ulmer Informatik-Berichte* 10.10 (2009), pp. 1–26. DOI: 10.1109/JPROC.2002.804686.

[6] C. Cros. *Complex and Adaptive Dynamical Systems*. 1th. Springer, 2015.

[7] J. Schwab et al. "Concepts in Boolean network modeling: What do they all mean?" In: *Computational and Structural Biotechnology* 18.1 (2020), pp. 571–582. DOI: https://doi.org/10.1016/j.csbj.2020.03.001.

[8] H.d. Jong. "Modeling and Simulation of Genetic Regulatory Systems: A Literature Review". In: *Journal of Computational Biology* 9.1 (2002), pp. 67–103. DOI: 10.1089/10665270252833208.

[9] G. Karlebach and R. Shamir. "Modelling and analysis of gene regulatory networks". In: *Nature Reviews Molecular Cell Biology* 9.1 (2008), pp. 770–780. DOI: 10.1038/nrm2503.

[10] S.A. Kauffman. "Metabolic stability and epigenesis in randomly constructed genetic nets". In: *Journal of Theoretical Biology* 22.3 (1969), pp. 437–467. DOI: `https://doi.org/10.1016/0022-5193(69)90015-0`.

[11] S.A. Kauffman and L.G. Glass. "The logical analysis of continuous, non-linear biochemical control networks". In: *Journal of Theoretical Biology* 39.1 (1973), pp. 103–129. DOI: `https://doi.org/10.1016/0022-5193(73)90208-7`.

[12] M. Hopfensitz et al. "Attractors in Boolean networks: a tutorial". In: *Computational Statistics* 28.1 (2012), pp. 19–36. DOI: `https://doi.org/10.1007/s00180-012-0324-2`.

[13] Adrien Fauré et al. "Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle". In: *Bioinformatics* 22.14 (July 2006), e124–e131. ISSN: 1367-4803. DOI: `10.1093/bioinformatics/btl210`. eprint: `https://academic.oup.com/bioinformatics/article-pdf/22/14/e124/614634/btl210.pdf`. URL: `https://doi.org/10.1093/bioinformatics/btl210`.

[14] Christoph Müssel, Martin Hopfensitz, and Hans A. Kestler. "BoolNet—an R package for generation, reconstruction and analysis of Boolean networks". In: *Bioinformatics* 26.10 (Apr. 2010), pp. 1378–1380. ISSN: 1367-4803. DOI: `10.1093/bioinformatics/btq124`. eprint: `https://academic.oup.com/bioinformatics/article-pdf/26/10/1378/16892352/btq124.pdf`. URL: `https://doi.org/10.1093/bioinformatics/btq124`.

[15] A. Saadatpour, I. Albert, and R. Albert. "Attractor analysis of asynchronous Boolean models of signal transduction networks". In: *Journal of Theoretical Biology* 266.4 (2010), pp. 641–656. DOI: `https://doi.org/10.1016/j.jtbi.2010.07.022`.

[16] I. Shmulevich, E.R. Dougherty, and Wei Zhang. "From Boolean to probabilistic Boolean networks as models of genetic regulatory networks". In: *Proceedings of the IEEE* 90.11 (2002), pp. 1778–1792. DOI: `10.1109/JPROC.2002.804686`.

[17] Y. Kwon, J. Kim, and K. Cho. "Dynamical Robustness against Multiple Mutations in Signaling Networks". In: *IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS* 13.5 (2015), pp. 996–1002. DOI: `10.1109/TCBB.2015.2495251`.

[18] U. Schöning. *Logic for comuter scientists*. 1th. Abt. Theroretische Informatik, Universität Ulm: Birkhäuser, 1989.

[19] Stuart Kauffman et al. "Genetic networks with canalyzing Boolean rules are always stable". In: *Proceedings of the National Academy of Sciences* 101.49 (2004), pp. 17102–17107. DOI: 10.1073/pnas.0407783101. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.0407783101. URL: https://www.pnas.org/doi/abs/10.1073/pnas.0407783101.

[20] C. Gershenson. "Guiding the self-organization of random Boolean networks". In: *Theory in Biosciences* 131.3 (2012), pp. 1611–7530. DOI: https://doi.org/10.1007/s12064-011-0144-x.

[21] D. Orlando et al. "Global control of cell-cycle transcription by coupled CDK and network oscillators". In: *Nature* 453.7197 (2008), pp. 1476–4687. DOI: https://doi.org/10.1038/nature06955.

[22] Fangting Li et al. "The yeast cell-cycle network is robustly designed". In: *Proceedings of the National Academy of Sciences* 101.14 (2004), pp. 4781–4786. DOI: 10.1073/pnas.0305937101. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.0305937101. URL: https://www.pnas.org/doi/abs/10.1073/pnas.0305937101.

[23] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence : from natural to artificial systems*. Oxford Univ. Press, 1999.

[24] K. DeJong. *Evolutionary Computation – A Unified Approach*. MIT Press, 2006.

[25] R. Poli, W. Langdon, and N. McPhee. "A Field Guide to Genetic Programming". In: *Genetic Programming and Evolvable Machines* 10.2 (2009), pp. 1573–7632. DOI: https://doi.org/10.1007/s10710-008-9073-y.

[26] X. Yang. *Nature-Inspired Computation and Swarm Intelligence : Algorithms, Theory and Applications*. Elsevier Science & Technology, 2020.

[27] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer, 2015.

[28] John N. Thompson. "Concepts of coevolution". In: *Trends in Ecology & Evolution* 4.6 (1989), pp. 179–183. ISSN: 0169-5347. DOI: https://doi.org/10.1016/0169-5347(89)90125-0. URL: https://www.sciencedirect.com/science/article/pii/0169534789901250.

[29] Charles Darwin. *On the Origin of Species by Means of Natural Selection*. or the Preservation of Favored Races in the Struggle for Life. London: Murray, 1859.

[30] Paul R. Ehrlich and Peter H. Raven. "Butterflies and Plants: A Study in Coevolution". In: *Evolution* 18.4 (1964), pp. 586–608. ISSN: 00143820, 15585646. URL: `http://www.jstor.org/stable/2406212` (visited on 05/12/2022).

[31] Daniel H. Janzen. "When is it Coevolution?" In: *Evolution* 34.3 (1980), pp. 611–612. ISSN: 00143820, 15585646. URL: `http://www.jstor.org/stable/2408229` (visited on 05/12/2022).

[32] Johann Peter Murmann. *Knowledge and competitive advantage: The coevolution of firms, technology, and national institutions*. Cambridge University Press, 2003.

[33] Christopher D Rosin and Richard K Belew. "New methods for competitive coevolution". In: *Evolutionary computation* 5.1 (1997), pp. 1–29.

[34] Tatsuya Akutsu et al. "A System for Identifying Genetic Networks from Gene Expression Patterns Produced by Gene Disruptions and Overexpressions". In: *Genome Informatics* 9 (1998), pp. 151–160. DOI: `10.11234/gi1990.9.151`.

[35] Julian D. Schwab and Hans A. Kestler. "Automatic Screening for Perturbations in Boolean Networks". In: *Frontiers in Physiology* 9 (2018). ISSN: 1664-042X. DOI: `10.3389/fphys.2018.00431`. URL: `https://www.frontiersin.org/article/10.3389/fphys.2018.00431`.

[36] Elena Dubrova and Maxim Teslenko. "A SAT-Based Algorithm for Finding Attractors in Synchronous Boolean Networks". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 8.5 (2011), pp. 1393–1399. DOI: `10.1109/TCBB.2010.20`.

[37] Ilya Shmulevich and Stuart A. Kauffman. "Activities and Sensitivities in Boolean Network Models". In: *Phys. Rev. Lett.* 93 (4 2004), p. 048701. DOI: `10.1103/PhysRevLett.93.048701`. URL: `https://link.aps.org/doi/10.1103/PhysRevLett.93.048701`.

[38] S. Bornholdt. “Boolean network models of cellularregulation: prospects and limitations”. In: *J. R. Soc. Interface* 5 (2008), pp. 85–94. DOI: 10.1098/rsif.2008.0132.focus. URL: https://doi.org/10.1098/rsif.2008.0132.focus.

[39] R. Aly. *Population-based simulation of Boolean networks*. 1th. Ulm University, 2017.

[40] M. Zwolinski. *Digital System Design with VHDL*. 2th. Perarson, 2004.

[41] The climate corporation. *Claypoole: Threadpool tools for Clojure. Last accessed April 1st 2017.* https://github.com/TheClimateCorporation/claypoole. 2013. URL: https://github.com/TheClimateCorporation/claypoole.

[42] Michael Fogus and Chris Houser. *The Joy of Clojure: Thinking the Clojure Way*. 1 edition. Greenwich, Conn: Manning Publications, Apr. 7, 2011. 360 pp. ISBN: 978-1-935182-64-1.

[43] Mariana Recamonde Mendoza and Ana Lúcia C. Bazzan. “Evolving Random Boolean Networks with Genetic Algorithms for Regulatory Networks Reconstruction”. In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO ’11. Dublin, Ireland: Association for Computing Machinery, 2011, 291–298. ISBN: 9781450305570. DOI: 10.1145/2001576.2001617. URL: https://doi.org/10.1145/2001576.2001617.

[44] Shuhua Gao et al. “Learning Asynchronous Boolean Networks From Single-Cell Data Using Multiobjective Cooperative Genetic Programming”. In: *IEEE Transactions on Cybernetics* 52.5 (2022), pp. 2916–2930. DOI: 10.1109/TCYB.2020.3022430.

[45] Philip Tan and Joc Cing Tay. “Evolving Boolean Networks to Find Intervention Points in Dengue Pathogenesis”. In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. GECCO ’06. Seattle, Washington, USA: Association for Computing Machinery, 2006, 307–308. ISBN: 1595931864. DOI: 10.1145/1143997.1144053. URL: https://doi.org/10.1145/1143997.1144053.

[46] Sîrbu A., Ruskin H., and Crane M. “Comparison of evolutionary algorithms in gene regulatory network model inference”. In: *BMC Bioinformatics* 11.1 (2010), pp. 1471–2105. DOI: 10.1186/1471-2105-11-59. URL: https://doi.org/10.1145/1143997.1144053.

[47] Ney Lemke, Jose'e C.M. Mombach, and Bardo E.J. Bodmann. “A numerical investigation of adaptation in populations of random boolean networks”. In: *Physica A: Statistical Mechanics and its Applications* 301.1 (2001), pp. 589–600. ISSN: 0378-4371. DOI: `https://doi.org/10.1016/S0378-4371(01)00372-7`. URL: `https://www.sciencedirect.com/science/article/pii/S0378437101003727`.

[48] Miguel Nicolau and Marc Schoenauer. “Evolving Scale-Free Topologies using a Gene Regulatory Network Model”. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. 2008, pp. 3747–3754. DOI: `10.1109/CEC.2008.4631305`.

[49] Christoph Fretter, Agnes Szejka, and Barbara Drossel. “Perturbation propagation in random and evolved Boolean networks”. In: *New Journal of Physics* 11.3 (2009), p. 033005. DOI: `10.1088/1367-2630/11/3/033005`. URL: `https://doi.org/10.1088/1367-2630/11/3/033005`.

[50] Szejka A. and Drossel B. “Evolution of canalizing Boolean networks”. In: *The European Physical Journal B* 56.4 (2007), pp. 1434–6036. DOI: `10.1140/epjb/e2007-00135-2`. URL: `https://doi.org/10.1140/epjb/e2007-00135-2`.

[51] Tamara Mihaljev. “Dynamics and evolution of random Boolean networks”. en. PhD thesis. Darmstadt: Technische Universität, 2008. URL: `http://tuprints.ulb.tu-darmstadt.de/1188/`.

[52] A.P. Quayle and S. Bullock. “Modelling the evolution of genetic regulatory networks”. In: *Journal of Theoretical Biology* 238.4 (2006), pp. 737–753. ISSN: 0022-5193. DOI: `https://doi.org/10.1016/j.jtbi.2005.06.020`. URL: `https://www.sciencedirect.com/science/article/pii/S002251930500278X`.

[53] Larry Bull. “Evolving Boolean Networks on Tunable Fitness Landscapes”. In: *IEEE Transactions on Evolutionary Computation* 16.6 (2012), pp. 817–828. DOI: `10.1109/TEVC.2011.2173578`.

[54] GórskiP., Czaplicka A., and Hołyst J. “Coevolution of information processing and topology in hierarchical adaptive random Boolean networks”. In: *The European Physical Journal B* 89.2 (2016), pp. 1434–6036. DOI: `10.1140/epjb/e2015-60530-6`. URL: `https://doi.org/10.1140/epjb/e2015-60530-6`.

Name: Michel Lutz Matricel number: 1048353

Declaration

I declare that I have written this thesis independently and have not used any sources or aids other than those indicated .

Ulm, ...

Michel Lutz