Tomas Bata University in Zlín
Faculty of Applied Informatics

Doctoral Thesis

Evolutionary Synthesis of the Turing Machine's Rules

Evoluční syntéza pravidel Turingova stroje

Ing. Lukáš Kouřil

Doctoral study programme: Engineering Informatics
Supervisor: prof. Ing. Ivan Zelinka, Ph.D.

Zlín, 2012

CONTENT

ABSTRACT

This doctoral thesis is concerned with possibilities of artificial intelligence utilization for Turing machine programming. The main topic regards using Differential Evolution and Self-Organizing Migrating Algorithm as selected methods of artificial intelligence for Turing machine transition function's rules synthesis. The rules of Turing machine represent a form of program on its basis Turing machine works. Rules designing can be considered as a way of this machine programming.

The doctoral thesis consists of four parts which can be characterized as follows. The first part represents an introduction to the finite automata because Turing machines are classified as them. This part is necessary for understanding backgrounds of these machines. Highly important is a characterization of the machine on the basis of formal description. This is used as a base for rules synthesis problematics formulation in the next parts of the doctoral thesis. This first part also introduces selected algorithms of artificial intelligence. These are Differential Evolution and Self-Organizing Migrating Algorithm. The introduction to these algorithms is a key for settings of suitable parameters of selected algorithms while rules synthesis.

The second part of the doctoral thesis presents two proposed approaches to Turing machine's rules synthesis (or optimization). These approaches are "classical optimization" and "per-partes optimization". Both approaches differ from each other. Each approach has also advantages and disadvantages which herewith assess their utilization. Both of mentioned approaches are closely described in this second part.

In the third part of the doctoral thesis three selected elementary problems are introduced. These are unary addition, divisibility (exact divison) problem and primality (prime number detection) problematics. The problems are used as example tasks for Turing machine which rules we want to estimate by proposed approaches. It is utilized for analysis of rules optimization process dependence on custom settings of Differential Evolution and Self-Organizing Migrating Algorithm. This analysis is entirely fundamental not only for this part of the doctoral thesis but for the next part especially.

The last, fourth, part of the doctoral thesis represents a practical utilization of proposed approaches to programming Turing machine by artificial intelligence. As real problematics protein processing by Turing machine was chosen. Proteins are regarded as primary protein structures in this case. Evolutionary synthesis of Turing machine's rules is demonstrated from total of a twelve selected primary protein structures differing in length. As described later in the text, this problematics is so far complex that can be considered as a sufficient way of proper work proof of proposed approaches to evolutionary synthesis of Turing machine's rules as main topic of the doctoral thesis.

Keywords: Turing machine, transition function's rules, artificial intelligence, Differential Evolution, Self-Organizing Migrating Algorithm

ABSTRAKT

Tato dizertační práce se věnuje možnostem využití umělé inteligence pro programování Turingova stroje. Nosným tématem je využití Diferenciální evoluce a Samo-organizujícího se migrujícího algoritmu jako vybraných metod umělé inteligence pro syntézu pravidel přechodové funkce Turingova stroje. Pravidla Turingova stroje představují program, na jehož základě Turingův stroj pracuje. Jejich návrh lze tedy považovat za formu programování tohoto stroje.

Dizertační práce sestáva ze čtyř částí, které lze charakterizovat následovně. První část představuje úvod do konečných automatů, mezi které patří i Turingův stroj. Tato část je nezbytná pro pochopení mechanismů, na jejichž základě tento typ strojů pracuje. Velmi důležitá je charakterizace stroje na základě jeho formálního popisu, který je v dalších částech dizertační práce využit jako základ pro formulaci problému syntézy pravidel. Tato první část rovněž seznamuje s vybranými algoritmy umělé inteligence, tedy s Diferenciální evolucí a Samo-organizujícím se migrujícím algoritmem. Toto seznámení hraje důležitou roli pro vhodné nastavení parametrů těchto algoritmů při syntéze pravidel.

Druhá část dizertační práce představuje dva navržené přístupy k syntéze (nebo také optimalizaci) pravidel Turingova stroje. Těmito přístupy jsou „klasická optimalizace“ a „optimalizace po částech“. Oba tyto přístupy se zásadně liší jeden od druhého. Zároveň každý z těchto přístupů má své výhody i zápory, které zároveň určují i jejich využití. V této druhé části jsou oba tyto přístupy podrobně popsány.

Ve třetí části dizertační práci jsou uvedeny tři vybrané elementární problémy. Jde o unární součet, problém dělitelnosti bez zbytku celým číslem a problematika detekce prvočísla. Tyto problémy jsou využity jako vzorové úlohy pro Turingův stroj, jehož pravidla chceme zjistit pomocí navržených přistupů. Toho je následně využito pro analýzu závislosti procesu optimalizace pravidel na různém nastavení Diferenciální evoluce a Samo-organizujícího se migrujícího algoritmu. Tato analýza je zcela zásadní nejen pro tuto část dizertační práce, ale především pro následující.

Poslední, čtvrtá, část dizertační práce představuje praktické využití navržených přístupů k programování Turingova stroje pomocí umělé inteligence. Za reálnou problematiku bylo zvoleno zpracování proteinů pomocí Turingova stroje. Proteiny jsou v tomto případě míněny primární proteinové struktury. Evoluční syntéza pravidel Turingova stroje je demonstrována na celkem dvanácti vybraných primárních proteinových strukturách, lišících se svou délkou. Tato problematika, jak je popsáno dále v textu, je natolik komplexní, že ji lze považovat jako dostatečný způsob ověření správné činnost přístupů k evoluční syntéze pravidel Turingova stroje, jimiž se tato dizertační práce zabývá.

Klíčová slova: Turingův stroj, pravidla přechodové funkce, umělá inteligence, Diferenciální evoluce, Samo-organizující se migrující algoritmus

ACKNOWLEDGEMENTS

At this place, I would like to dearly thank to following persons:

- Supervisor of my doctoral thesis prof. Ing. Ivan Zelinka, Ph.D. who interested me in artificial intelligence and inspired me to be concerned with this amazing part of applied informatics. Also he provides many helpful advices to me.
- The head of Department of Informatics and Artificial Intelligence doc. Mgr. Roman Jasek, Ph.D. who encourage me in finalization of this doctoral thesis and assisted me whenever I needed.
- Ing. Zuzana Oplatkova, Ph.D. for her patience of my questions about artificial intelligence, doctoral thesis and other concerns related to doctoral study.
- Friends of mine from SPLK KAŠAVA for making me escaped from the hard work days. They helped me without they noticed it.

At last but not least I would like to warmly thank to my parents. They were greatly supportive for me while my doctoral study. My cordial thanks also belong to them for everything what they do for me.

Thank you. This doctoral thesis wouldn't be come into existence without you.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND SYMBOLS

AI	Artificial intelligence
DE	Differential Evolution
SOMA	Self-Organizing Migrating Algorithm
A, TM	Representation of automaton or Turing machine
Q	Set of all inner states
Σ	Set of input symbols
Δ	Set of output symbols
δ	Transition function
λ	Output function
p, q	Inner state
q_0	Initial state
q_{accept}	Accepting state
q_{reject}	Rejecting state
t_{size}	Size of data tape
s_{max}	Maximum number of computation steps
F	Set of final or accepting states
B	Blank symbol
Γ	Set of all input symbols
X	Input symbol
Y	Output symbol
D	Direction of head movement
NP	Number of population
F	Mutation constant
CR	Cross-over value
G	Number of generations
M	Number of migrations
PRT	Perturbation
PopSize	Population size
PathLength	Length of path
CFE	Cost function evaluations
A	Adenine
C	Cytosine
G	Guanine
T	Thymine
U	Uracile
DNA	Deoxyribonucleic acid
RNA	Ribonucleic acid

1 INTRODUCTION

The meaning of applied informatics can be regarded as one of building blocks of our civilization. It is an engine which drives civilization progress ahead. However it doesn't touch the present only. The past has been affected by meaning of applied informatics too. As well as it will also lead the way of civilization development in the future. If there is any extraterrestrial civilization there is surely its science which has same meaning as our applied informatics (but can be termed differently) too. I don't think it is necessary to argue into above-outspoken words. It can be considered as a small revolution when informatics has been arisen as a separate science and the applied informatics has been begun daily influence lifes of people.

The doctoral thesis which is held in your hands is aimed on applied informatics too. As considered related to this thesis, the application of informatics is related to the three areas. These are automata theory, artificial intelligence and bioinformatics. The former two areas are main topics of the doctoral thesis. The term "automata theory" can be sounded slightly boring but it is not at all. With theory of automata one can met in assorted places. It is not industry (machine tools, vehicles, consumer electronics etc.) only. The automata can also have a form of e. g. software (workflows realization, text searching and the like). The thesis considers using Turing machines which are a minor of finite automata. Turing machines are theoretical automata with abilities that common automata don't dispose (e.g. infinite size of data medium).

The second area of applied informatics related to the doctoral thesis is an artificial intelligence. In terms of public perception, the term artificial intelligence (AI) always attracts attention. This term itself is perceived as a sign of something extraordinary. As well as "AI" two letters which act as a mystical acronym. However, what is curious about these words? The probable answer is the adjective "artificial" which may be responsible for the perception of these expressions. The adjective assigns certain characteristics of human beings to the artificial non-living objects which is anomalous and can be also little strange. The strange aspect, in particular, in the consciousness of the public regarding artificial intelligence (AI) is influenced and advanced by movies all the time. AI is actually human-being-inspired only. The inspiration can be found in several methods of artificial intelligence, e.g. neural networks (neural system of brain), evolutionary algorithms (evolutionary processes in the nature), genetic algorithms (cell genetics) - subsets of evolutionary algorithms etc. The methods of artificial intelligence basically imitate actual specific processes. Due to this, AI-based methods can themselves continuously evolve and adapt to the actual solved problems. The solution to problems retrieved by methods of artificial intelligence can bring novel and unexpected results since the outlooks of AI methods regarding problems are more flexible.

The last area this thesis is aimed on is bioinformatics. Bioinformatics is relatively new science which came into existence lately. It benefits from appliances and methods of applied informatics in relation to biological problems. The meaning of applied informatics for biology is huge and can not be ommited. The problem, as the protein processing is, was selected as an one for demonstration of this doctoral thesis results practical utilization.

Altough the research described on following pages is mainly focused on concerns of applied informatics the related problems involve different ares of this science and represent new approaches for utilization artificial intelligence for programming Turing machines. The example applications to protein processing are included as well. I hope it will be interesting for readers and for practical usage especially.

2 BACKGROUND RESEARCH

Using artificial intelligence for programming Turing machines is not widely spread yet. Although it is possible to find some few mentions, scientific articles or research papers which describe attempts to use any methods of artificial intelligence these are limited to use of genetic algorithms [33], genetic programming [33] or evolutionary programming especially. It acknowledges that employing Differential Evolution and Self-Organizing Migrating Algorithm – representatives of evolutionary and memetic algorithms – for programming Turing machines as discussed in this doctoral thesis is a new unique approach to Turing machine's rules optimization. Of course, the lack of relevant research papers could be caused by unavailability of some articles in databases (ISI Web of Science[1], SpringerLink[2], ScienceDirect[3]) used in the days while working on background research, but it is less probable. Following lines briefly introduce several existing approaches used for programming Turing machines by methods of artificial intelligence selected by research papers' authors.

The first of introduced researches is concerned with evolving Turing machines from examples in eponymous article [31] by Julio Tanomaru. There are described two approaches to programming Turing machine based on using Genetic Algorithm [33]. These are termed as simple genetic algorithm approach and enhanced evolutionary approach. The Turing machines are considered as transition tables where the number of inner states can be changed dynamically. Automaton is understood as:

$$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0), \tag{2.1}$$

where:

- Q is a set of all inner states,
- Σ is a set of input symbols,
- Δ is a set of output symbols,
- δ is the transition function, whereas $\delta(p, a) = q$ and $p \in Q$, $a \in \Sigma$, $q \in Q$,
- λ is the output function, whereas $\lambda(p, a) = b$ and $p \in Q$, $a \in \Sigma$, $b \in \Delta$,
- q_0 is the initial state.

In population, the automaton is represented by next state and output symbol as follows:

$$\mu^i = [a^i_{j\,k} \epsilon\, Q | b^i_{j\,k} \epsilon\, \Delta], \tag{2.2}$$

where:

- *i = 1,..., Population size*
- *j = 1,..., n^i*
- *k = 1,..., dim(Σ)*
- n^i is number of inner states of i-th automaton
- *dim(Σ)* is number of different input symbols

In the article [31], continuous generation model was used. By crossover or mutation the population *Pop'(t)* is originated after population *Pop(t)* is duplicated at the i-th generation. Whether member of population *Pop'(t)* is originated by crossover or mutation it is expressed by ρ parameter ($0 \leq \rho \leq 1$) what is crossover ratio. There is used 2-point crossover operator when

[1] *http://apps.isiknowledge.com*
[2] *http://www.springerlink.com*
[3] *http://www.sciencedirect.com*

two of population members are selected and crossover exchanges groups of transition tables' rows. Mutation is provided by changing values within specified range. In the case of Turing machines, the formal definition is used as shown in (4.4).

Second approach described in [31] utilizes performance statistics collected for each generation. Then the dependence of operators to best automata from previous generation is observed. Also the crossover operation was excluded and three new mutation operators were used. The first one is same as described in the first approach. The second one enables to dynamic changes of inner states. The third one can discard member of population and originates a new one on the basis of performance statistics.

The results of second research were published in research paper named „Evolving Turing Machines for Biosequence Recognition and Analysis" [32] by Edgar E. Vallejo and Fernando Ramos. There is used an approach by genetic programmming [33]. The article [32] describes three slightly different experiments for evolving either Turing machine or other finite automaton for biosequence recognition. The first experiment is aimed on Turing machines. The Turing machine is considered as restricted and defined by 9-tuple:

$$TM = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}, t_{size}, s_{max}), \tag{2.3}$$

where:

- Q is a set of all inner states,
- Σ is a set of input symbols,
- Γ is a set of all data tape symbols,
- δ is the transition function, $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$,
- q_0 is an initial state, $q_0 \in Q$,
- q_{accept} is an accept state, $q_{accept} \in Q$,
- q_{reject} is a reject state, $q_{reject} \neq q_{accept}$,
- t_{size} is the data tape size
- s_{max} expresses maximum number of computation steps.

There is used genetic algorithm with tournament selection and elitism. The Turing machine is represented as genome in the form of concatenated values of the transition function using transition table. If there is considered transition table (see Table 1) same as the table published in [32]:

Table 1: Transition table of first experiment [32]

$\boldsymbol{\delta}$	**a**	**b**
q_1	(q_1, a, R)	(q_2, b, L)
q_2	(q_3, a, L)	(q_2, b, R)
q_3	(q_2, b, R)	(q_2, a, R)

then it is possible to concatenate the values of transition table to the representation of genome (2.4)

$$(q_1, a, R)\ (q_2, b, L)\ (q_3, a, L)(q_2, b, R)\ (q_2, b, R)(q_2, a, R) \tag{2.4}$$

and express it as

$$q_{0,0}a_{0,0}m_{0,0}q_{0,1}a_{0,1}m_{0,1} \dots q_{|Q|,|\Sigma|}a_{|Q|,|\Sigma|}m_{|Q|,|\Sigma|}, \tag{2.5}$$

where:

- i is a row of transition table,
- j is a column of transition table.

There are used custom-designed genetic operators because genome representation is not binary. In the research [32], the training set is compounded of biosequences accepted and randomly generated negative sequences too. Turing machine considered has 32 inner states and 8 data tape symbols.

The second experiment was aimed on two-way deterministic finite automata what is a type of Turing machine which allow recognize language and works with read-only data tape. It actually means that the input symbol is the same as the output symbol. In this experiment, there is used restricted two-way deterministic finite automaton formally defined as 8-tuple:

$$A = (Q, \Sigma, \delta, q_0, q_{accept}, q_{reject}, t_{size}, s_{max}), \tag{2.6}$$

where:

- Q is a set of all inner states,
- Σ is a set of input symbols,
- δ is the transition function, $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$,
- q_0 is an initial state, $q_0 \in Q$,
- q_{accept} is an accept state, $q_{accept} \in Q$,
- q_{reject} is a reject state, $q_{reject} \neq q_{accept}$,
- t_{size} is the data tape size
- s_{max} expresses maximum number of computation steps.

Similarly to the first experiment of [32], the genome is represented by concatenated values of transition table. In this case it looks like Table 1 and it is the same table as in [32].

Table 2: Transition table of second experiment [32]

$\boldsymbol{\delta}$	**a**	**b**
q_1	(q_1, R)	(q_2, L)
q_2	(q_3, L)	(q_2, R)
q_3	(q_2, R)	(q_2, R)

When its values are concatenated the genome has a form of:

$$(q_1, R)\ (q_2, L)\ (q_3, L)(q_2, R)\ (q_2, R)(q_2, R) \tag{2.7}$$

and it can be possible to express it as

$$q_{0,0} m_{0,0} q_{0,1} m_{0,1} \dots q_{|Q|,|\Sigma|} m_{|Q|,|\Sigma|}, \tag{2.8}$$

where:

- i is a row of transition table,
- j is a column of transition table.

Genetic operators as well as the training set and other parameters were same as in the first experiment described above.

The third experiment of [32] consists of multiple sequence alignment with evolved two-way deterministic finite automaton from the second experiment. The two-way deterministic finite automaton is used for multiple computation of each sequence at every position in the sequence. That is repeated for every positive example and common patterns of two sequences by identical behavior of the automaton can be revealed.

The last research mentioned at this place and aimed on evolving Turing machine has a title „Using Genetic Programming for Turing Machine Induction" [20]. Its authors are Amashini Naidoo and Nelishia Pillay. This research paper considers Turing machine with variable size and two data tapes. These are input data tape and output data tape. The first one is read-only. The output data tape is writeable. The input data tape encodes input string of Turing machine. The output data tape contains blank symbols only. Turing machine processes both data tapes simultaneously. Transitions between inner states are expressed by two-tuples. They represent a dependence of processing output data tape on input data tape. The first part of two-tuple says what symbol has to be read from the input data tape and what the direction of head movement is. The second part of the two-tuple expresses what symbol has to be read from output data tape, what symbol has to be written on the output data tape and what the direction of head movement is. As an example, if transition is *a/R*, *B/a/R*, it means that symbol *a* must be read from input data tape and the head must move to the right, blank symbol must be read from the output data tape, symbol *a* must be written to the output data tape and the head must move to the right. Elements of population are created by following steps. At first, the initial node (start state) is originated automatically. Node arity is specified randomly. Other states are also randomly stated as final states or not. The transitions between nodes and its children are originated by random elements selection from input signary, data tape signary and directions of head movement. Genetic operators are applied on parent nodes selected by tournament selection. The mutation operator ensures that there is selected a mutation point. The sub-node connected to this point is removed and new sub-node is originated and connected to this point. The cross-over operator randomly selects points on parent nodes. Sub-nodes connected to these points are swapped. Also inner states of sub-nodes are renumbered.

As can be seen it is possible to find several articles or research papers aimed on application of artificial intelligence methods on Turing machine adaptation. These papers consider using genetic algorithms or genetic programming for evolving Turing machines especially. The mentioned research papers also consider either elementary problems or problems with less-extensive signary for processing by evolved Turing machine. Thus it can be said that the research discussed in this doctoral thesis is unique for using Differential Evolution and Self-Organizing Migrating Algorithm for Turing machine optimization and aiming on processing highly-extensive signary problems by Turing machines evolved.

3 THESIS OBJECTIVES

The objectives of the thesis can be specified as follows:

- Design of approaches to Turing machine evolutionary programming.
- Proof of proper Turing machine evolutionary programming for processing selected example problems.
- Analysis of Turing machine evolutionary programming dependence on custom settings of Differential Evolution and Self-Organizing Migrating Algorithm.
- Proof of proper Turing machine evolutionary programming for processing proteins by using results of above-mentioned analysis.
- Software development of Turing machine in Wolfram Mathematica.

4 BRIEF INSIGHTS INTO AUTOMATA

The progress of automata theory had begun in the early 20th century mainly due to Alan Turing who had been concerned with abstraction of machines including their capabilities. Nowadays, plenty technical areas of interest, e.g. concepts and design of hardware and software, come out of knowledge of automata theory. The reason is that automata theory is based on hardware (machines or computers) but in abstract form. The principle of both abstract and non-abstract machines are quite similar, as well as software algorithms and flow of computer programs which can be compared with automata in many cases too.

Because the Turing machines are closely related with finite automata, the latter will be discussed first.

4.1 Finite automata

Finite automata [10] can be considered as event-driven machines in the form of algorithm. The events represent conditions which occur during processes or operation of finite automata. The events also act as responses to processes since all events can be regarded as consequences of previous process evolvement whereas the determination of intended process evolvement is influenced by reply to the desired response and reaction to external input.

Belonging among the above-mentioned, finite automata are basically "assembled" from two essential parts. These are events, known as states, and inputs which externally influence incoming state occurrence in terms of a future event. The inputs advance process evolvement.

It is necessary to specify one of states as initial to be able to activate the finite automaton. The automaton uses the initial state as the starting point while waiting in expectation for the first input and realizing the transitions to the next state. It is also necessary to specify the set of final or accepting states. The set contains one or more states which denote the end of the process. When the final state occurs the automaton stops activity.

There are many examples of simple finite automata. How finite automata operate can be shown on the basis of following situation. Let's consider automaton which controls the opening of a window (see Fig. 4.1). The goal of the automaton is to fully open the window. The window has four positions: closed position, first venting position, second venting position and fully opened position.

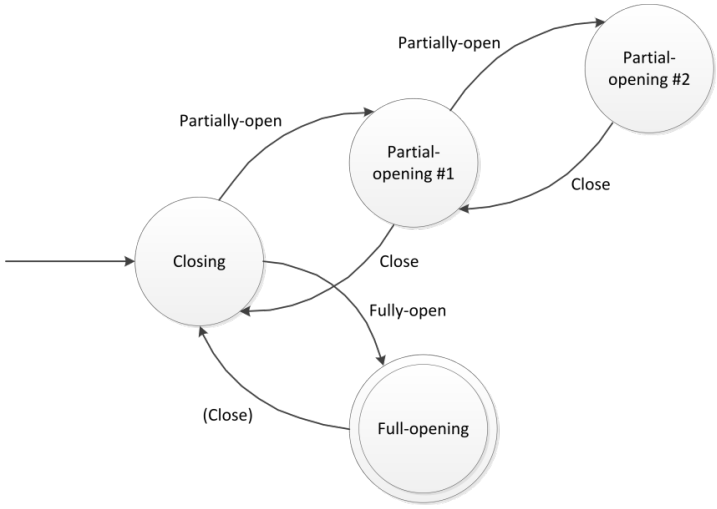

Fig. 4.1: Example of finite automaton

The opening and closing have these restrictions:

- First venting position can be realized when the window is closed
- Second venting position can be realized when the window is opened in the first venting position
- The window can be fully opened only if closed

The events or states have form of nouns, which are:

- Closing
- Partial-opening #1 (to the first venting position)
- Partial-opening #2 (to the second venting position)
- Full-opening

The inputs have the form of the following verbs:

- Fully-open
- Partially-open
- Close

Finite automata are usually illustrated by a diagram where states are represented as circles and inputs as arrows. The initial state is marked with a non-linked arrow and the final state is depicted as a double circle. An example diagram of finite automata can be seen in Fig 4.1. Before starting, the automaton is in the state "Closing" and waits for the input. If the input is "Fully-open", the automaton passes to the state "Full-opening" and the activities of the automaton finish because the state "Full-opening" is the final state. If not, it would be possible to pass to the state "Closing" with the occurrence of "Close" input. But this state is the final state thus "Close" input is written in parentheses. When the automaton is in the "Closing" state and the input is "Partially-open", the next state will be "Partial-opening #1". It is the first venting position. Now, the automaton can pass to the "Closing" state if "Close" input occurs or to the "Partial-opening #2" state when "Partially-open" input appears.

As can be seen, there are the same inputs in the diagram. These are "Close" inputs and "Partially-open" inputs. The situation is absolutely correct since these inputs can be initiated in different states. This is typical for deterministic automata [10] compared to non-deterministic automata [10], where it is possible to pass to the multiple states simultaneously (e.g. if the "Partially-open" and "Fully-open" inputs would be replaced by "Open" input only). Only deterministic finite automata will be considered further.

4.1.1 Definition of finite automata

Finite automata with deterministic behavior can be defined as 5-tuple [10]:

$$A = (Q, \Sigma, \delta, q_0, F), \tag{4.1}$$

where:

- Q is a set of all states.
- Σ is a set of input symbols.
- δ represents the transition function.
- q_0 is an initial state.
- F is a set of final states.

These parameters adapted to the example are:

- $Q \in \{"Closing", "Partial-opening\ \#1", "Partial-opening\ \#2", "Full-opening"\}$
- $\Sigma \in \{"Partially-open", "Fully-open", "Close"\}$
- δ see below.
- $q_0 = \{"Closing"\}$
- $F \in \{"Full-opening"\}$

The transition function can be expressed as:

$$\delta(q, X) = p, \tag{4.2}$$

where:

- q is a current state.
- X is an input symbol.
- p is a state the automaton passes to.

Finite automaton operates according to the transition function. It represents a reaction of the automaton to the inputs which occur in specific states. The reaction is a new future state of the automaton. If the action of above-shown example automaton is rewritten as (4.2), the result is the following system of equations.

$$\begin{aligned}
\delta("Closing", "Partially-open") &= "Partial-opening\ \#1" \quad (4.3)\\
\delta("Closing", "Fully-open") &= "Full-opening"\\
\delta("Partial-opening\ \#1", "Partially-open") &= "Partial-opening\ \#2"\\
\delta("Partial-opening\ \#1", "Close") &= "Closing"\\
\delta("Partial-opening\ \#2", "Close") &= "Partial-opening\ \#1"
\end{aligned}$$

However, the usual description of the automata is in the form of a transition table (see Table 3), which contains arguments of the transition function and the responses of the function.

Table 3: Example of the transition table

Argument of equation (4.2)		**Response of equation (4.2)**
Current state	**Current input**	**New state**
„Closing“	„Partially-open“	„Partial-opening #1“
„Closing“	„Fully-open“	„Full-opening“
„Partial-opening #1“	„Partially-open“	„Partial-opening #2“
„Partial-opening #1“	„Close“	„Closing“
„Partial-opening #1“	„Close“	„Partial-opening #1“

The 5-tuple (4.1) and the transition table together describe actual finite automaton and its operation.

4.2 Turing machines

Turing machines [10] are theoretical machines and belong among representatives of finite automata. They use a sequential approach to pursued operations. Except for complexity, they are characterized by enhancement which facilitates to provide a new way of response. Turing machines can generate output information by writing process. With regard to the character of the

Turing machines, it is even possible to use them to solve problems which are unsolvable by common appliance of informatics (e.g. because of restrictions due to limited operating memory).

As with finite automata, Turing machines can be deterministic or non-deterministic; the difference between these variations is similar as in regard to finite automata. Because in the following text only deterministic Turing machines are considered, the term "Turing machine" is related to the deterministic variant when it occurs.

Apart from deterministic and non-deterministic variants of the Turing machines, several expanded types [10] of Turing machines which differ in structure also exist. These are e.g. multi-tape Turing machines, multi-stack Turing machines, Turing machines with semi-infinite tapes etc. In this doctoral thesis, simple Turing machines with infinite tapes are considered. This variant of Turing machines simulates a machine with infinite memory (see Fig. 4.2).

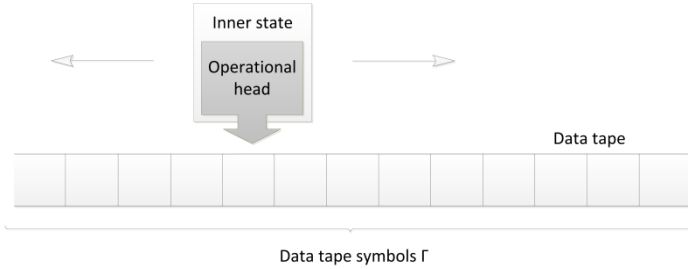

Fig. 4.2: Scheme of the Turing machine

Turing machines contain three main components which are essential to their makeup. These are:

- Data tape
  This is the data carrier of Turing machines. It serves as an information channel for input and Turing machine can use it for writing output data.
- Operational head
  The head acts as an interface between the data tape and the actual Turing machine. It can read data from the tape and write information to the data tape.
- Internal stack
  The stack contains information on inner states.

As can be seen, the structure of Turing machines is not so complex. It contains only three main components. The data tape is a storage medium which contains symbols of pre-defined signary. These symbols are subsequently processed by Turing machine. At first, they are read by the operational head and input to the machine. The symbols are processed in accordance with the transition function. Afterwards, the output is written to the data tape in the form of symbols by the operational head.

4.2.1 Definition of Turing machines

The definition of Turing machines is partially based on a formal description of finite automata (4.1) since there are several similarities between Turing machines and finite automata. Because Turing machines can generate output information, their definition includes additional parameters. The form of definition is a 7-tuple [10]:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F), \tag{4.4}$$

where:

- Q is a set of all inner states.
- Σ is a set of input symbols whereas $\subseteq \Gamma \setminus \{B\}$. These symbols represent active input information.
- Γ is a set of all data tape symbols.
- δ represents the transition function of Turing machine such as $\delta: Q \setminus F \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.
- q_0 is an initial state such as $q_0 \in Q$.
- B is a blank symbol such as $B \in \Gamma$. This symbol is „non-active" and used for filling empty spaces on the data tape, separating „active" symbols etc.
- F is a set of final states $F \subseteq Q$.

Along with the definition of Turing machines, the transition function of Turing machines is also different as can be seen above. It could be transcribed as:

$$\delta(q, X) = (p, Y, D), \tag{4.5}$$

where:

- q is a current state.
- X is an input symbol.
- p is a a state Turing machine passes to.
- Y is an output symbol which is written to the data tape.
- D is the direction of the head movement. This direction is to the left (-1) and right (1) side. Also it is possible that the head is not moved and remains (0) in its current position thus $D = \{-1,0,1\}$.

The meaning of symbols in (4.4), (4.5) which occur in conjunction with the final automata is the same as in (4.1) and (4.2).

4.2.2 How Turing machine works

Basically, Turing machines pursue simple activity. They read symbols from the data tape, process the symbols according to the transition function, write new symbols to the data tape, move the head and pass to the new inner states.

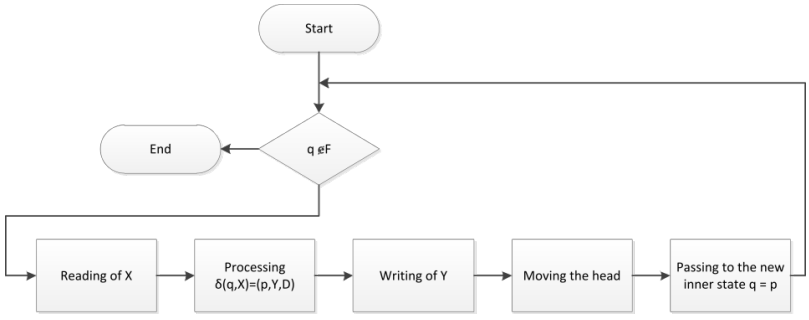

Fig. 4.3: Operating of the Turing machines

This process is repeated until a new inner state does not belong to the set of final states. The diagram of the process can be seen in Fig. 4.3.

Similarly to finite automata, the processing of symbols by Turing machines is influenced by the transition table which maps arguments of the transition function and the response of this function on Turing machine. The appearance of Turing machine's transition table depends on expression (4.5). This means that the table has five columns which correspond to the parameters q, X, p, Y and D. The content of the table can be called the rules of Turing machine's transition function.

The activity of Turing machine can be demonstrated by the following basic example where a simple Turing machine is utilized. Let's consider a problem of bit negation. The data tape contains sequences of bits. The task of Turing machine is to negate each bit encoded in the data tape. The sample data tapes can be seen in Fig. 4.4 and Fig. 4.5.

Fig. 4.4: Example of input data tape

Fig. 4.5: Example of requested output data tape

Parameters of Turing machine are:

- $Q = \{q_1, q_2\}$
- $\Sigma = \{"0", "1"\}$
- $\Gamma = \{"\#", "0", "1"\}$
- $q_0 = q_1$
- $F = \{q_2\}$

The initial head position is located at the first input symbol.

How Turing machine processes the data tape is controlled by the rules which are shown in Table 4.

Table 4 - Example of the Turing machine's transition table

Argument of expression (4.5)		**Response of expression (4.5)**		
Current state	**Loaded**	**New state**	**Record symbol**	**Direction**
q_1	„#“	p_2		
q_1	„0“	p_1	„1“	1
q_1	„1“	p_1	„0“	1

The transition table can be again rewritten in the form of an equation of the transition functions.

$$\begin{aligned} \delta(q_1, "\#") &= p_2 \\ \delta(q_1, "0") &= (p_1, "1", 1) \\ \delta(q_1, "1") &= (p_1, "0", 1) \end{aligned} \tag{4.6}$$

Let's have a look at the processing of the data tape on each step (the bold frame of the data tape marks the current head position):

1. $\delta(q_1, "0") = (p_1, "1", 1)$
   The current inner state of Turing machine is q_1 and the head is located at symbol "0" (see Fig. 4.6). According to the rules of the transition function (Table 4), Turing machine writes symbol "1" to the data tape and moves the head to the left. The inner state is not changed.

Fig. 4.6: Appearance of the data tape in step 1

2. $\delta(q_1, "1") = (p_1, "0", 1)$
   The current inner state of Turing machine is q_1 again. The head is located at symbol "1" (see Fig. 4.7). According to the rules of the transition function, Turing machine writes symbol "0" to the data tape and moves the head to the left. The inner state is not changed.

Fig. 4.7: Appearance of the data tape in step 2

3. $\delta(q_1, "1") = (p_1, "0", 1)$
   Similarly to the previous step, the current inner state of Turing machine is q_1 and the head is located at symbol "1" (see Fig. 4.8). According to the rules of the transition function, Turing machine writes symbol "0" to the data tape and moves the head to the left. The inner state is not changed again.

Fig. 4.8: Appearance of the data tape in step 3

4. $\delta(q_1, "\#") = p_2$
   The current inner state of Turing machine is q_1. The head is located at symbol "#" (see Fig. 4.9). According to the rules of the transition function, Turing machine passes to the inner state q_2. The state q_2 is final thus Turing machine finishes processing.

Fig. 4.9: Appearance of the data tape in step 4

It is not necessary to move the operational head in any direction other than toward the right in this simple example but the rules of the Turing machine can be clearly seen. They act as the program for Turing machines. This doctoral thesis is focused solely on optimization of the rules of the transition function.

5 APPLICATION OF EVOLUTIONARY ALGORITHMS

Artificial intelligence can be found in different forms. If the applications of artificial intelligence are considered the methods where the artificial intelligence is part of them are especially meant above all. In principle, these methods are inspired by natural processes interfering with life-based systems or organisms including genetics (as genetic algorithms), neural system capability (as neural networks), evolution (as evolutionary algorithms) etc. As perceivable according to the title of the chapter, there is considered a subset of methods of artificial intelligence which include evolutionary algorithms within the scope of optimization technique. The evolutionary algorithms are inspired by natural evolutionary processes. The main area of use of these algorithms is optimization which can be in the form of function approximation, pattern recognition etc.

This part of doctoral thesis describes optimization using two evolutionary algorithms. These are known as Differential Evolution [18, 19, 33, 35], which is an algorithm introduced in the 1990's by K. Price and R. Storn, and the relatively novel Self-Organizing Migrating Algorithm [33] – [35] developed by I. Zelinka. Although both are termed as evolutionary algorithms, there are a few differences between them in terms of their fundamentals and behavior in action. That is especially concerned about the latter which balances between evolutionary algorithms and memetic algorithms as will be explained later.Within this doctoral thesis' part both algorithms will be briefly presented thus it will be possible to become sufficiently familiarized with them.

5.1 Differential Evolution

This evolutionary algorithm is inspired in a significant way by the process of evolution as can be observed in nature. The principles of Differential Evolution [18, 19, 33, 35] are based on Darwin's theory and its fundamental aspects of natural occurrences. The evolution of any population is influenced by the abilities of its members to adapt to the surrounding environment where the population is spread. These abilities are mostly inherited. In addition, they come through evolvement processes during the life-cycle of every individual member of the population. The inherited abilities are a direct result from ancestors of each member. The abilities important to the survival of the members of the population depend on the actual environment where the population is spread. The abilities considered can be e.g. strength, leadership skills, ability to obtain food etc. It could be said, that the complex of abilities which affect survival express "quality" values of the individual member. The members of population with higher "quality" values have a significantly greater chance of survival in the surrounding environment.

As can be seen above, the evolution process can be considered as a form of optimization. The members of the population can be regarded as fragmentary solutions to the problem which takes the form of the surrounding environment. Optimization consists of looking for the most suitable solution to the problem thus becoming the best-adapted individual member of the population for survival in the surrounding environment.

5.1.1 Essential principles of Differential Evolution

Because the individuals of a population represent fragmentary solutions to the actual problem, it is therefore necessary to specify how each solution to various problems is encoded in the form of an individual. This is ensured by a specimen. The specimen is a vector which denotes a number of parameters of the individual and the range of these parameters. The parameters of the individuals can be imagined as the abilities of population members. At the beginning of the evolution process, the population is an assortment of individuals with randomly generated parameters according to the specimen. These individuals are parts of the first generation and are

subsequently optimized in a similar way to natural evolution. The important part of evolution is a computing of "quality" value, which is known as cost value. In accordance with the cost value, individuals are selected to advance to the next generation. These individuals "survive".

Within each generation, the individuals are subsequently processed. In addition to the currently processed individual x_i three other individuals x_{r_1}, x_{r_2} and x_{r_3} are randomly selected. The first two of randomly-selected individuals x_{r_1} and x_{r_2} are subtracted (thus the arguments for subtraction are parameters of individuals). The result of subtraction of these individuals is a differential vector. The mutation of the differential vector proceeds in the next step. The mutation takes the form of a multiplication of the differential vector and the mutation constant *F* (multiplication of one of Differential Evolution parameters and items of differential vector). The mutated differential vector is known as weighted differential vector. This vector is added to the third randomly-selected individual x_{r_3} (addition of items of weighted differential vector and parameters of individual). The result of addition is a noise vector (5.3). The final operation is a cross-over of the currently processed individual and the noise vector. The cross-over operation produces the test vector (5.1) which can be regarded as a descendant of the four individuals (currently-processed individual and three randomly-selected individuals) with inherited and evolutionary evolved abilities - parameters. Now, the cost values of the currently-processed individual and the test vector are computed. This is ensured by the cost function which is specified in accordance with the actual problem thus the parameters of individual (or items of vector) are substituted in the cost function. The individual or test vector which contains the better evaluation advances to the next generation (5.2). After processing of all individuals within the current generation, the next generation is formed and the optimization process repeats in a similar way as mentioned above.

How optimization by Differential Evolution works can be mathematically expressed [33] as follows. The computation of test vector is

$$x_{i,j}^{test} = \begin{cases} x_{r_3,j}^{G} + F \cdot \left(x_{r_1,j}^{G} - x_{r_2,j}^{G}\right) & if\ rand_j[0,1] < CR \vee j = j_{rand} \\ x_{i,j}^{G} & else \end{cases}, \quad (5.1)$$

where:

- $i = \{1, \dots, NP\}, j = \{1, \dots, D\}$D – Dimension of individual
- $r_1, r_2, r_3 \in \{1, \dots, NP\}$ Random selection of three individuals
- $r_1 \neq r_2 \neq r_3 \neq i$
- $CR \in \langle 0,1 \rangle, F \in \langle 0,2 \rangle$

The evaluation of test vector and the currently-processed individual within a generation can be formulated as (5.2). The minimization is considered in (5.2) thus lower cost value is preferred.

$$x_i^{G+1} = \begin{cases} x_i^{test} & if\ f_{cost}(x^{test}) \leq f_{cost}(x_i^{G}) \\ x_i^{G} & else \end{cases} \quad (5.2)$$

Differential Evolution has several variants which mostly differ in the way of computation of the noise vector. This doctoral thesis and following examples of optimization consider the *DE/rand/1/bin* variation of Differential Evolution. Within the scope of the mentioned variation of Differential Evolution, the noise vector is computed as (5.3).

$$v = x_{r_3,j}^{G} + F \cdot \left(x_{r_1,j}^{G} - x_{r_2,j}^{G}\right) \quad (5.3)$$

Differential Evolution algorithm contains parameters [33] which have to be set before starting the optimization process. The parameters which influence the processing of individuals can be found e.g. in equations (5.1 – 5.3). These parameters are:

- *NP* (Number of population)
  How many individuals are contained within one generation.
- *F* (Mutation constant)
  The rate of diversity when computing weighted differential vector.
- *CR* (Cross-over value)
  Influence on creating test vector.
- *G* (Generations)
  How many generations will be subsequently created until optimization ends.

As can be seen, Differential Evolution is considerably similar to natural evolution.

5.2 Self-Organizing Migrating Algorithm

The algorithm which has been described here falls into the category of evolutionary algorithms, although it is not entirely accurate. Evolutionary algorithms usually feature the ability to create new individuals from ancestors thereby its "abilities", in the form of parameters, are influenced by mutation and cross-over. During optimization by Self-Organizing Migrating Algorithm [33] – [35], new individuals are not being created and the ancestor-descendant relationship is not of concern. Thus this algorithm should rather be termed as memetic algorithm.

5.2.1 Background theory of Self-Organizing Migrating Algorithm

Self-Organizing Migrating Algorithm (SOMA) is inspired by the social behavior of cooperating individuals. It could be imagined as cooperation within migration of wildlife shoals, hunting or achieving other collective interests. At the beginning of the optimization process, there is a randomly generated initial migration according to the specimen (similar to Differential Evolution). As opposed to Differential Evolution, SOMA uses the term "migration" instead of "generation", since no new individuals are created. During that process, the individuals of the current migration are evaluated by cost function and the cost value is counted. The individual who has the better evaluation becomes the leader of other individuals within the current migration. These individuals start moving toward the leader from the surrounding environment which represents an optimization problem. Each following migration involves determination of the new leader and moving other individuals toward him thus the best and most suitable solution of the problem is subsequently revealed.

Because SOMA is not based on common principles of evolution and does not use mutation and cross-over operation, it is necessary to ensure the stochastic progression of the algorithm in an alternative way. In the case of SOMA, the mutation is replaced by perturbation which influences the movement of individuals in the surrounding environment. It means that parameters of individuals are changed by perturbation. The perturbation has a form of a unique vector (PRTVector) for each individual. This vector states the number of parameters which have to be changed for the current individual. How the PRTVector is generated can be mathematically expressed [33] as:

$$PRTVector = \begin{cases} 1 & if\ rand_j < PRT \\ 0 & else \end{cases}, \tag{5.4}$$

where:

- $j = \{1, \ldots, D\}$ D – Dimension of individual
- PRT Value of perturbation (see below)

The cross-over process of Differential Evolution is replaced by a movement of individuals in the surrounding environment and remembering the best suitable position (parameters of individual). The movement can be expressed by a directional vector [33]:

$$\vec{r} = \overrightarrow{r_0} + \overrightarrow{m}t\overrightarrow{PRTVector}, \tag{5.5}$$

where:

- $t \in \langle 0, Step, PathLength \rangle$ *Step*, *PathLength* – parameters of SOMA (see below)

The equation (5.5) can be further transcribed [33] as:

$$x_{i,j}^{M+1} = x_{i,j,start}^{M} + \left(x_{i,j}^{M} - x_{i,j,start}^{M}\right)tPRTVector_j, \tag{5.6}$$

where:

- $t \in \langle 0, Step, PathLength \rangle$ *Step*,*PathLength* – parameters of SOMA (see below)

If the items of *PRTVector* are equal to 1, the current individual moves toward the leader according to the directional vector. If certain items of *PRTVector* are equal to 0, the appropriate parameters of individual are not changed within the current migration.

There are several strategies of movement of individuals within migrations in SOMA. This chapter and all examples consider the *AllToOne* strategy, where all individuals move toward one leader.

As well as Differential Evolution, SOMA has its own parameters which have to be set before the optimization process starts. These parameters [33] are:

- *PopSize* (Size of population)
  How many individuals are contained within one migration.
- *PRT* (Perturbation value)
  Influences the generation of perturbation vector.
- *PathLength*
  Distance between halting position of the current individual and the leader.
- *Step*
  The step size of the individual movement.
- *Migrations*
  How many migrations will occur until the optimization ends.
- *AcceptedError*
  Maximal difference of the best and the worst individual in the current migration. If the difference is less than AcceptedError, the optimization ends.

SOMA algorithm can be still regarded as evolutionary although there are certain nuances between SOMA and other evolutionary algorithms. The perturbation and movement of individuals for ensuring stochastic progression can be considered to be forms of mutation and cross-over.

6 APPROACHES TO EVOLUTIONARY OPTIMIZATION OF THE RULES OF THE TURING MACHINE'S TRANSITION FUNCTION

There is no easy answer to the question: "How to evolutionary optimize the rules of the Turing machine's transition function?" In spite of this, two approaches to evolutionary optimization will be presented. These approaches differ considerably from each and are accomplishments of this doctoral research [15] – [17]. In connection with the above question and knowledge of Turing machines and selected evolutionary algorithms, both approaches offer answers to the related questions which are:

- How to encode the rules as a transition table or equation system of transition functions to a form which can be comprehensible to evolutionary algorithms?
- How to design the cost function which evaluates the individuals during optimization by Differential Evolution or SOMA?

If it is thought about what is represented by the program of Turing machine and how the above described optimization works it can be seen that above two questions are highly important. Actually, the program of Turing machine appears as rules, hence it is necessary to provide information on the transition table or the equation system of the transition functions in a suitable form for processing by evolutionary algorithms. This is because evolutionary algorithms optimize the population which is supplied by the encoded rules of Turing machine.

The approaches of optimization differ from each other by the manner in which rules are encoded and processed. The first approach presented will consider processing of rules as a whole. The second approach uses per-part processing of the rules. Each rule is optimized separately.

6.1 Classical optimization

This approach [16, 17] to optimization takes into consideration the rules as a whole. This means that all rules input to the optimization process together and are also processed as a whole. It follows that individuals within Differential Evolution or SOMA will encode rules which represent varied programs of Turing machine and the selected evolutionary algorithm will produce the best optimized and most suitable one. But it is necessary to find an effective method to do the encoding because evolutionary algorithms use numerical expressions only, whereas the rules of Turing machine are based on symbolic expressions.

6.1.1 Encoding the rules for classical optimization

If the example shown in chapter 4.2.2 (see Table 4) is remembered, it can be perceived that the arguments of the transition function are combinations of inner states of Turing machine Q and the data tape symbols Γ . Similarly, the response of the transition function can be considered as a combination of inner states of Turing machine Q, data tape symbols Γ and the direction of head movement D. These two facts could be capitalized on composition of individuals.

Let's create the vector v_{input} which contains all combinations of the transition function arguments and the vector v_{output} which contains all combinations of responses of the transition function. If the parameters of Turing machine are the same as in the example shown in chapter 4.2.2:

- $Q = \{q_1, q_2\}$,
- $\Gamma = \{"\#", "0", "1"\}$,

the vector v_{input} has 6 items (two inner states · three data tape symbols) and the vector v_{output} is compounded from 18 items (two inner states · three data tape symbols · three directions). The mentioned vectors are as follows:

$$v_{input} = \begin{Bmatrix} \{q_1, "\#"\}, \{q_1, "0"\}, \{q_1, "1"\}, \\ \{q_2, "\#"\}, \{q_2, "0"\}, \{q_2, "1"\} \end{Bmatrix} \tag{6.1}$$

$$v_{output} = \begin{Bmatrix} \{q_1, "\#", -1\}, \{q_1, "\#", 0\}, \{q_1, "\#", 1\}, \\ \{q_1, "0", -1\}, \{q_1, "0", 0\}, \{q_1, "0", 1\}, \\ \{q_1, "1", -1\}, \{q_1, "1", 0\}, \{q_1, "1", 1\}, \\ \{q_2, "\#", -1\}, \{q_2, "\#", 0\}, \{q_2, "\#", 1\}, \\ \{q_2, "0", -1\}, \{q_2, "0", 0\}, \{q_2, "0", 1\}, \\ \{q_2, "1", -1\}, \{q_2, "1", 0\}, \{q_2, "1", 1\} \end{Bmatrix} \tag{6.2}$$

These vectors can be encoded to the individual on the basis of indexes of their items. The problem is how to effectively encode these indexes to a compact form for evolutionary algorithm's individual. The values included in the individual can represent indexes of items of v_{output} whereas their positions can be matched with indexes of items of v_{input}. This may sound complicated but can be clarified by seeing Fig. 6.1.

According to the Fig. 6.1, it is possible to rewrite:

$$v_{individual} = \{1,3,4,2,5, \dots\}, \tag{6.3}$$

to the equation system of transition functions:

$$\begin{aligned} &\delta(q_1, "\#") = (q_1, "\#", -1) \\ &\delta(q_1, "0") = (q_1, "\#", 1) \\ &\delta(q_1, "1") = (q_1, "0", -1) \\ &\delta(q_2, "\#") = (q_1, "\#", 0) \\ &\delta(q_2, "0") = (q_1, "0", 0) \\ &\quad \dots \end{aligned} \tag{6.4}$$

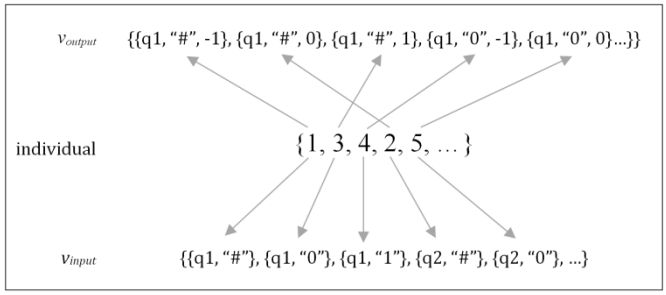

Fig. 6.1: Example of encoding complete rules

When the classical approach is used, it is necessary to set the number of inner states Q before starting the optimization process because this parameter influences vectors v_{input} and v_{output}. If the number of inner states is too low, there is a risk that it will be impossible to find suitable

rules. If the number of inner states is too high, the complexity of the optimization process rises drastically.

6.1.2 Definition of specimen

When it is clear how the individual is encoded, the specimen and boundaries of its parameters can be established. As can be seen in Fig. 6.1, the specimen will have a number of items equivalent to the length of the vector v_{input} (length of $Q \cdot$ length of Γ). Each item can gain value from the range $\langle 1;$ length of $v_{output}\rangle$. More precisely it is $\langle 1;$ length of $Q \cdot$ length of $\Gamma \cdot$ length of $D\rangle$. In the case of above-mentioned example, the specimen has six items which can gain value from the range $\langle 1;18\rangle$:

$$v_{specimen} = \{i_1, \dots, i_n\}, \tag{6.5}$$

where:

- $n = length\ of\ v_{input}$
- $i \in \langle 1, length\ of\ v_{output}\rangle$

The specimen uses the integer values for its items but the evolutionary algorithms operate over continuous space with real numbers thus it is necessary to perform conversion when decoding the result of the optimization to the form of the rules. This can be done by expressing values as the greatest integers which are less or equal to real numbers occurring in the result.

6.1.3 Designing evaluation function

The evaluation function is an important part of the optimization process realized by evolutionary algorithms as a component of cost function. Within this function the individuals are evaluated and decisions on the advantaging of individuals are made. The evaluation function which is used by classical optimization involves the following steps:

- Rewriting individuals to the form of rules.
- Initializing Turing machine using new rules.
- Starting Turing machine.
- Comparing the current output of Turing machine to the requested output.
- Evaluating individuals according to the above mentioned comparison.

Each individual from the current generation or migration is subsequently entered to the evaluation function. Then the current individual is rewritten to the form of rules. These rules are used for initialization of Turing machine and Turing machine is started. When Turing machine finishes processing the data tape, the output data tape is compared to the requested output data tape. After that the individual is evaluated on the basis of the comparison. This can be expressed as:

$$CV = \begin{cases} CV - V & if\ \Gamma_{output,i} == \Gamma_{requested,i} \\ \mathrm{CV} + V & else \end{cases}, \tag{6.6}$$

where

- CV Cost value.
- V Value which infulences computing of the cost value.
- $\Gamma_{output,i}$ Data tape symbol at i-th position of output data tape.
- $\Gamma_{requested,i}$ Data tape symbol at i-th position of requested output data tape.
- $i = 1, \dots, L$ L – length of data tape.

This is considered as optimization toward finding the minimum in (6.6) therefore the *CV* is decreased if the compared symbols are identical. Alternatively, when optimization toward finding the maximum is considered, the cost value of the individual will increase.

The process of evaluating the individual can be also depicted as can be seen in Fig. 6.2 (plus symbol means "true", minus symbol means "false"). During evaluation, the symbols of the output data tape and requested output data tape are subsequently compared (see Fig. 6.2). If the symbols are equal, the individual is advantaged (the cost value is adapted in accordance with finding the maximum or minimum by defined coefficient *V*). If the symbols are not equal, the individual is penalized by coefficient *V* (the cost value is decreased when finding the maximum or increased when finding the minimum). If the optimization is finding the minimum, the better evaluated individual has the lower cost value. If the optimization is finding the maximum, the better evaluated individual has the higher cost value.

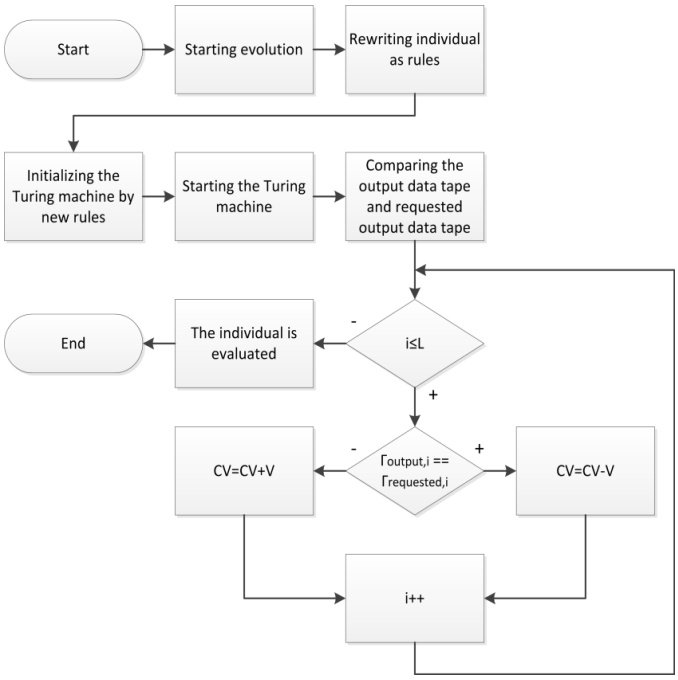

Fig. 6.2: Evaluating individuals by classical optimization

This design of the evaluation function enables optimization of the complete set of rules at once because they are encoded as one individual. When evolutionary algorithms process a generation or a migration of individuals, they optimize several sets of rules together. From the point of view of the evolutionary algorithms, these sets of rules are ordinary individuals and are processed as mentioned in chapter 5.1 and 5.2.

6.2 Per-partes optimization

This approach [15] to optimization is entirely different to the former. The individual does not encode a set of all rules but one rule only and the evolutionary algorithm does not find the best suitable set of rules in the form of the individual but separately optimizes each rule of one set and composes the ideal program for Turing machine. This means that the evolutionary algorithm tries to find the one rule which ensures preferable processing of the current symbol on the data

tape. Thus this optimization approach is known as per-partes. Therefore it is necessary to start optimization for each symbol separately. Because the former approach uses evaluation of the individual based on (6.6) and the evaluation of the individual as a whole is influenced by matching of separate symbols of the output data tape and output requested data tape, it is possible that an individual which contains several correct rules (but not all) is discarded when penalization is too high due to other rules. The above-mentioned situation is by-passed due to optimization of rules for processing each data tape symbol separately.

6.2.1 Encoding the rules for per-partes optimization

The first thing that is changed is the way in which the rules are encoded to the form of the individual. Let's consider expression of the symbol arguments of the transition function (4.5) as indexes. The equation (4.5) may be rewritten to the following form:

$$\delta(q_i, X_j) = (p_k, Y_l, D_m), \tag{6.7}$$

where:

- $q_i \in Q \quad i = 0, \dots, Length\ of\ set\ Q - 1$
- $X_j \in \Gamma \quad j = 0, \dots, Length\ of\ set\ \Gamma - 1$
- $p_k \in Q \quad k = 0, \dots, Length\ of\ set\ Q - 1$
- $Y_l \in \Gamma \qquad l = 0, \dots, Length\ of\ set\ \Gamma - 1$
- $D_m \in D\, m = -1, \dots, Length\ of\ set\ D - 2, D = \{left, none, right\}$

Now it is possible to work with integers which match indexes i, j, k, l, m instead of symbols q_i, X_j, p_k, Y_l , D_m. The encoding is simplified by expression of the rule as an individual containing only two items. These items are:

- l – index of symbol which is written to the data tape by Turing machine.
- m – index of direction of the Turing machine's head movement.

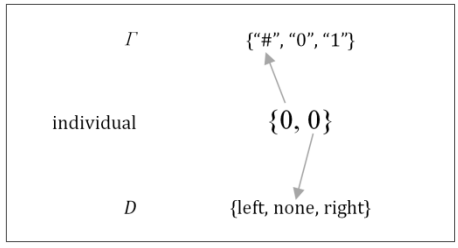

Fig. 6.3: Example of encoding one rule

An example of encoding the individual which explains above-descibed situation can be seen in Fig. 6.3.

It is possible to omit other indexes because the per-partes approach dynamically adjusts inner states. As a result, it is not necessary to set the number of inner states before starting optimization. Every movement of the Turing machine's head and processing of the data tape symbol represents a new inner state thus in (4.5) arguments q and X are known. The evolutionary algorithm finds suitable parameters Y and D. At the end of optimization, all estimated rules are compacted or reduced in order to use all combinations of inner states and data tape symbols since they are considered as arguments of the transition function. Then the p parameters are supplied in reverse order and the rules are connected together. This process is simple but it may appear unclear at first sight. The following example shows rules compositions.

Let's consider a problem of bit negation as well as in chapter 4.2.2. The input data tape and requested output data tape appear as in Fig. 4.4 and Fig. 4.5. The per-partes approach may process the input task in steps as follows:

1. $\delta(q_1, "0") = (\quad, "1", 1)$

Fig. 6.4: Appearance of the data tape in step 1

2. $\delta(q_2, "1") = (\quad, "0", 1)$

Fig. 6.5: Appearance of the data tape in step 2

3. $\delta(q_3, "1") = (\quad, "0", 1)$

Fig. 6.6: Appearance of the data tape in step 3

4. $\delta(q_4, "\#") = (\quad, "1", 1)$

Fig. 6.7: Appearance of the data tape in step 4

The arguments of the transition functions are given. The *X* parameter is influenced by the data tape symbol where the head of the Turing machine is currently located. The inner states (*q* parameter) are subsequently added in each step of head movement (as mentioned above). The responses of the transition function (right sides of the equation) are results of optimization. Within each step there is a comparison of output data tape with requested output data tape. If both data tapes are similar, the optimization process ends.

When all rules are revealed, the set of rules is reduced. The non-compacted equation system looks like:

$$\begin{aligned} &\delta(q_1, "0") = (\quad, "1", 1) \\ &\delta(q_2, "1") = (\quad, "0", 1) \\ &\delta(q_3, "1") = (\quad, "0", 1) \\ &\delta(q_4, "\#") = (\quad, "1", 1) \end{aligned} \tag{6.8}$$

As can be seen, there is a waste of inner states because all combinations of inner states and data tape symbols as arguments of the transition function are not utilized. These non-utilized combinations are e.g. (q_1;"#"), (q_1;"1"), (q_2;"#"), (q_2;"0") etc. The above-mentioned process of reduction ensures utilization of all combinations. Within the process of reduction, the *p* parameters are supplied and the connection of all rules is provided. An example of reduction is depicted as Fig. 6.8.

As can be seen, four rules were reduced to three and connected together. The final equation system of the transition functions is:

$$\begin{aligned} &\delta(q_1, "\#") = (q_2, "1", 1) \\ &\delta(q_1, "0") = (q_1, "1", 1) \\ &\delta(q_1, "1") = (q_1, "0", 1) \end{aligned} \tag{6.9}$$

In (6.9) the state q_2 is the final state. When the q_2 state is reached by Turing machine, activity of Turing machine ends.

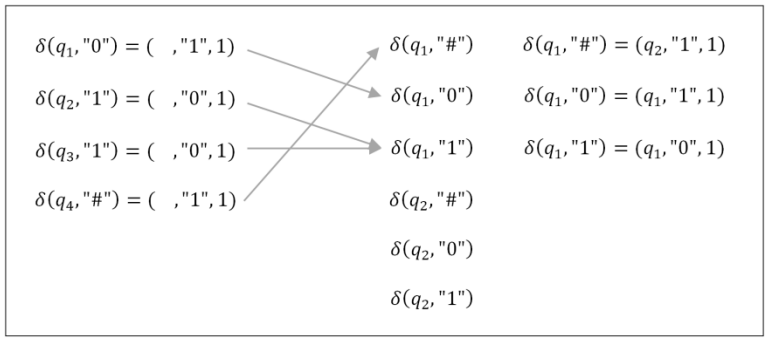

Fig. 6.8: Example of reduction of the rules

6.2.2 Definition of specimen

The encoding of rules outlines how a specimen is defined. Fig. 6.3 shows, that the specimen will have only two items. The first item represents index of the data tape symbol, the second item is an index of direction of the head movement. The index of the data tape symbol can gain value in the range $\langle 0;$ length of $\Gamma - 1\rangle$. The value of direction index is in the range $\langle -1;$ length of $D - 2\rangle$. According to the example in chapter 6.2.1, the first item is in the range $\langle 0;2\rangle$ and the second item is in the range $\langle -1;1\rangle$. The specimen will look like as follows:

$$v_{specimen} = \{l, m\}, \tag{6.10}$$

where

- $l \in \langle 0;$ length of $\Gamma - 1\rangle$,
- $m \in \langle -1;$ length of $\mathrm{D} - 2\rangle$.

As in the case of the specimen used in the classical approach to optimization (see chapter 6.1.1 and 6.1.2), it is necessary to convert the values of these items to integers since the evolutionary algorithms use real numbers. This could be ensured by e.g. expressing values as the greatest integers which are less or equal to the real numbers occurring in the result.

6.2.3 Optimization process and evaluative algorithm

The processing of individuals that is provided by the optimization process was partially described in chapter 6.2.1 in connection with encoding of rules to the form of individuals. But the optimization process which is used in the per-partes approach to optimization is yet more complex. Per-partes optimization provides:

1. Comparing output data tape and requested output data tape. If they are the same, proceed to step 4, if not, go to step 2.
2. Running evolutionary algorithm
   The individuals are processed by designed evaluative algorithm (see below).
3. When the evolutionary algorithm is finished, the result is the best suitable rule for processing the current symbol located on the data tape. The output data tape is altered by this rule. Then go to step 1.
4. Reduction of the rules.
5. End of optimization proces.

It can be illustrated as Fig. 6.9. After starting the optimization process, the output data tape and requested output data tape are compared. If the tapes are different, the evolutionary algorithm which finds the most suitable rule for processing the current symbol (necessary to be specified before starting optimization) on the data tape is started. Then the current data tape symbol and position of the head are altered according to the estimated rule. After that, both tapes are compared again. If the tapes are still different, the evolutionary algorithm starts and finds the most suitable rule for processing the symbol at the new location which is influenced by the previously estimated rule. If the data tapes are the same, the reduction of all estimated rules (as described in chapter 6.2.1) is processed and the optimization process is finished.

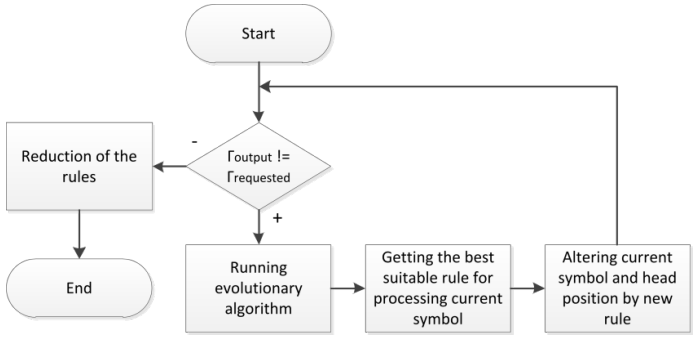

Fig. 6.9: Scheme of per-partes approach to optimization

The fundamental feature of this approach is the evaluative algorithm inner cost function. This algorithm provides a flexible way of evaluating individuals and brings another factor to the stochastic progress of the optimization process. Due to this, estimated rules which differ in processing of the data tape can be retrieved as results of optimization every time the optimization process is run. The algorithm can be best understood by Fig. 6.10.

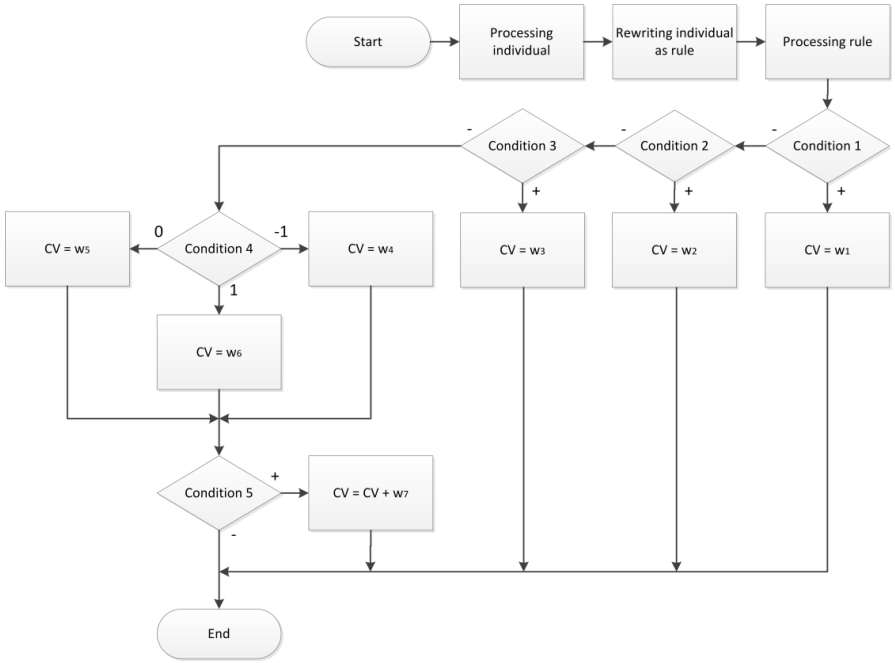

Fig. 6.10: Diagram of the evaluative algorithm

There are five essential conditions in the evaluative algorithm. These conditions are shown and explained in Table 5.

Table 5: Conditions which are parts of the evaluative algorithm

Condition	Meaning	Question
1	Checking new location of the head.	Is the new position of the head out of data tape?
2	Verification of last three symbols contained on the data tape and performed movements.	Are they the same?
3	Comparison of the new output symbol and requested output symbol.	Aren't they the same?
4	Checking the direction of head movement.	What is the direction of head movement?
5	Verification of the current output symbol-	Is it the blank symbol?

Optimization proceeds in five steps. At first, it is verified, whether the new rule which is encoded as the current individual will cause the head of the Turing machine to move out of data tape (if there are limitations of the data tape). If yes, the individual is penalized by weight w_1 and further evaluation stops. If no, comparison proceeds of the output symbol and direction of the head movement encoded for the current individual and last two previous symbols and directions of head movement. If they are the same, the individual is penalized by weight w_2 (right direction of the head movement is preferred thus repeating this direction is not penalized) and evaluation is finished. If not, the current output symbol and the requested output symbols are compared. If they are not same, the individual is penalized by weight w_3 and evaluation is finished again. If they are the same, the direction of head movement is checked. Thanks to weights w_4, w_5 and w_6 the preference of certain directions can be set. As the last step, if the current output symbol is the blank symbol, the individual can be penalized by weight w_7.

As can be seen, the evaluative algorithm is really flexible by penalization or giving preference to individuals by various initializations of weights. The features of the evaluative algorithm can be summarized as:

- Possibilities to prefer certain directions of head movement.
- Possibilities to accept even incorrect rules assuming correction in next steps of processing.
- Abilities to override circular rules and repeating of the output symbols and directions of head movement.
- Possibility to penalize a blank symbol.

The features mentioned above allow for rules of Turing machine optimization for solving complex problems quickly and effectively using the per-partes approach to optimization.

7 SELECTED EXAMPLES

Up to this point there was spoken theoretically about the evolutionary optimization of the rules of the Turing machine's transition function. In this part of the doctoral thesis, three examples of the previously-mentioned problems will be shown. The examples of using evolutionary optimization involve different tasks which were processed and solved by Turing machine. These tasks are:

- Unary addition
- Divisibility
- Primality

Above mentioned tasks or problems are very simple and represent basic mathematical operations in unary number system. These tasks will be used as simple problem representatives for rules estimation by the classical approach. The tasks will be also used while analyzing the influence of custom settings of evolutionary algorithms to the results and progress of optimization process (see chapter 8).

A description of the problem and settings of Turing machine will be given at each example task. (There are considered two variants of Turing machines used differentiating by number of inner states.)

7.1 Unary addition

The problem of unary addition [16, 17, 51] is simple and consists of the addition of two numbers expressed by the unary number system. Let's consider the mathematical example 2+3. If rewritten to the unary number system, it will appear as:

11 + 111

The result in unary number system is:

11111

It is necessary to encode specification of this example on the data tapes. The initial data tape looks like:

...	#	#	#	1	1	#	1	1	1	#	#	#	#	...

Fig. 7.1: Initial data tape of unary addition problem

and the requested output data tape:

...	#	#	#	1	1	1	1	1	#	#	#	#	#	...

Fig. 7.2: Requested output data tape of unary addition problem

The settings of Turing machine are following:

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_6\} \mid Q = \{q_1, \dots, q_{16}\}$
- $\Sigma = \{"1"\}$
- $\Gamma = \{"\#", "1"\}$
- $q_0 = q_1$
- $B = "\#"$
- $F = \{q_6\} \mid F = \{q_{16}\}$

The initial head position is located before the first input symbol. The number of inner states was established experimentally.

7.2 Divisibility

Divisibility [16, 17, 51] is the task in which it is requested to divide two numbers exactly. Division is again solved using the unary number system. In this example, the mathematical example 4/2 is considered. The initial data tape looks as in the case of unary addition:

...	#	#	#	1	1	#	1	1	1	1	#	#	#	...

Fig. 7.3: Initial data tape of divisibility problem

The task of Turing machine is to find whether these numbers are exactly divisible. If they are, Turing machine should alter the data tape to the following form:

...	#	#	#	1	1	X	1	1	1	1	#	#	#	...

Fig. 7.4: Requested output data tape of divisibility problem

The settings of Turing machine are:

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_6\} \mid Q = \{q_1, \dots, q_{16}\}$
- $\Sigma = \{"1", "X"\}$
- $\Gamma = \{"\#", "1", "X"\}$
- $q_0 = q_1$
- $B = "\#"$
- $F = \{q_6\} \mid F = \{q_{16}\}$

The head is initially positioned at the middle blank symbol between arguments represented by input symbols. The number of inner states was set according to previous observations.

7.3 Primality

The problem of primality [16, 17, 51] consists of finding out whether a number encoded on the data tape is a prime number or not. This example also considers a number which is expressed in the unary number system and it is number 5. The initial data tape appears as in Fig. 7.5.

...	#	#	#	#	1	1	1	1	1	#	#	#	#	...

Fig. 7.5: Initial data tape of primality problem

If the number which is encoded on the data tape is a prime number, Turing machine indicates it by placing the symbol “X” before the first input symbol and thus before the number. Also the other input symbols are replaced by blank symbols (see Fig. 7.6).

...	#	#	#	X	#	#	#	#	#	#	#	#	#	...

Fig. 7.6 :Requested output data tape of primality problem

The settings of Turing machine are:

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_6\} \mid Q = \{q_1, \dots, q_{16}\}$
- $\Sigma = \{"1", "X"\}$
- $\Gamma = \{"\#", "1", "X"\}$
- $q_0 = q_1$
- $B = "\#"$
- $F = \{q_6\} \mid F = \{q_{16}\}$

The initial position of the head is right before the first input symbol. As in the previous examples, the number of inner states was established experimentally.

8 EFFECT OF CUSTOM SETTINGS OF SELECTED EVOLUTIONARY ALGORITHMS ON EVOLUTIONARY-ESTIMATED PROGRAMMING

Dependence of evolutionary process on custom settings of evolutionary algorithm used is a key issue which should be considered before discussing practical utilization of described methods to real problems. It is apparent that settings of selected evolutionary algorithms influence an estimation process of the transition function's rules in a critical way. There is meant accuracy and execution time of optimization process especially including other apects of the estimation process too. These are important factors of the estimating process thus it is necessary to analyze a dependence of custom settings of the evolutionary algorithms on the process of estimation.

The analysis is aimed on settings of Differential Evolution and Self-Organizing Migrating Algorithm while estimating rules for processing selected examples (see chapter 7) by the classical optimization (see chapter 6.1). The classical optimization is highly dependent on settings of evolutionary algorithm thus the per-partes optimization is not involved in analysis. (Except settings of evolutionary algorithm, the per-partes optimization is also significantly influcenced by other factors mentioned in chapter 6.2.3.) The results of analysis will be used for specifying the settings of evolutionary algorithms for the per-partes optimization of real problems discussed in next part of the doctoral thesis. The issue of custom settings of evolutionary algorithms and rules estimation was partially described here [12].

8.1 Methodology

As problems to processing by the Turing machine thus rules estimation, there were considered selected examples (see chapters 7.1 – 7.3). These problems were subsequently processed by the classical optimization which utilized Differential Evolution and Self-Organinzing Migrating Algorithm. While estimation process there were changed settings of mentioned evolutionary algorithms and rate of successfull estimation and execution time were measured. Parameters changed can be found in following tables.

Table 6: Custom settings of Differential Evolution

Parameter	Period	Values	Initial
NP	$\langle 10D, 100D \rangle$	$10D - 100D$	$10D$
F	$\langle 0,2 \rangle$	0 – 2	0.9
CR	$\langle 0,1 \rangle$	0 – 1	0.2
G	customizable	100 – 1000	100

Table 7: Custom settings of Self-Organizing Migrating Algorithm

Parameter	Period	Values	Initial
PopSize	$\langle 10, \text{customizable} \rangle$	$D - 10D$	D
PRT	$\langle 0,1 \rangle$	0 – 1	0.1
PathLength	$\langle 1.1,5 \rangle$	1.1 – 5	3
Step	$\langle 0.11, PathLength \rangle$	0.11 – 3	0.3
Migrations	$\langle 10, \text{customizable} \rangle$	100 – 1000	100

Parameter *D* represents a dimension of the problem – it is a number of cost function's arguments – or length of specimen vector.

While analysing dependence of evolutionary algorithms' custom settings on evolutionary estimation, one only parameter was subsequently changed. Other parameters were set on initial values. The periods of parameters change were stated in accordance with recommandations in [33] and can be seen in Table 6 and Table 7. When analysis of current parameter was done, parameter changed was set on initial value and other parameter was analyzed. When analyzing *NP*, *G* and *Migrations* parameters, the step of change was 100. In the case of analyzing *PopSize*, the step of change was equal to *D*. While considering other parameters, the step of change was 0.1.

When specifying initial parameters, there was necessary take into account number of cost function evaluations. The number of cost function evaluations depends on several parameters of Differential Evolution and Self-Organizing Migrating Algorithm therefore it is perceptible that the initial settings must comply the condition of approximately equal number of cost functions evaluations while changing parameters being analyzed of both algorithms.

Number of cost function evaluations in the case of Self-Organizing Migrating Algorithm can be expressed as [33]:

$$CFE_{SOMA} = \frac{(PopSize-1)\cdot PathLength\cdot Migrations}{Step} \tag{8.1}$$

Number of cost function evaluations in the case of Differential Evolution can be induced as:

$$CFE_{DE} = NP \cdot G \tag{8.2}$$

If considering encoding the rules of the classical approach and definition of the specimen (see chapter 6.1.1 and 6.1.2), the dimension of the problems is equal to:

$$D = (Q - 1) \cdot \Gamma \tag{8.3}$$

Number of inner states is decremented by number one, because the inner states involve the final state too. This state is omitted for dimension computations.

When initial settings (Table 6 and Table 7) are used with connection to specifications of examples used (Table 8), number of cost function evaluations (CFE) while analyzing parameters which influcence number of *CFE* can be seen in following Table 9 and Table 10. There are considered *NP – PopSize* and *G – Migrations* which can be understood as equivalent.

Table 8: Specifications of examples used

Example id	Example	$Q - 1$	Γ	D
1	Unary addition	$\{q_1, \dots, q_5\}$	{"#","1"}	10
2	Unary addition	$\{q_1, \dots, q_{15}\}$	{"#","1"}	30
3	Divisibility	$\{q_1, \dots, q_5\}$	{"#","1","X"}	15
4	Divisibility	$\{q_1, \dots, q_{15}\}$	{"#","1","X"}	45
5	Primality	$\{q_1, \dots, q_5\}$	{"#","1","X"}	15
6	Primality	$\{q_1, \dots, q_{15}\}$	{"#","1","X"}	45

In following tables, CFE_{min} and CFE_{max} represent number of cost function evaluations for minimal and maximal value of analyzed parameter.

Table 9: Number of cost function evaluations for parameters NP and G of Differential Evolution

	NP		***G***	
Example Id	CFE_{min}	CFE_{max}	CFE_{min}	CFE_{max}
1	10000	100000	10000	100000
2	30000	300000	30000	300000
3	15000	150000	15000	150000
4	45000	450000	45000	450000
5	15000	150000	15000	150000
6	45000	450000	45000	450000

Table 10: Number of cost function evaluations for parameters PopSize and Migrations of Self-Organizing Migrating Algorithm

	PopSize		***Migrations***	
Example Id	CFE_{min}	CFE_{max}	CFE_{min}	CFE_{max}
1	9000	99000	9000	90000
2	29000	299000	29000	290000
3	14000	149000	14000	140000
4	44000	449000	44000	440000
5	14000	149000	14000	140000
6	44000	449000	440000	440000

8.2 Results

As stated above, there were two goals of analysis. These were answers to following questions:

- What is accuracy of optimization proces for custom settings of selected evolutionary algorithms?
- What is execution time of optimization proces for custom settings of selected evolutionary algorithms?

The first question can be regarded as the most important. On the basis of these results, concrete evolutionary algorithm and its settings will be selected for processing real problems (discussed in the next part of the doctoral thesis). The results aimed on execution time are less important (but could be very interesting on the other hand) thus they will be only used as secondary criterion for process of evolutionary algorithm and its settings selection.

8.2.1 Dependence of optimization process on DE's NP parameter

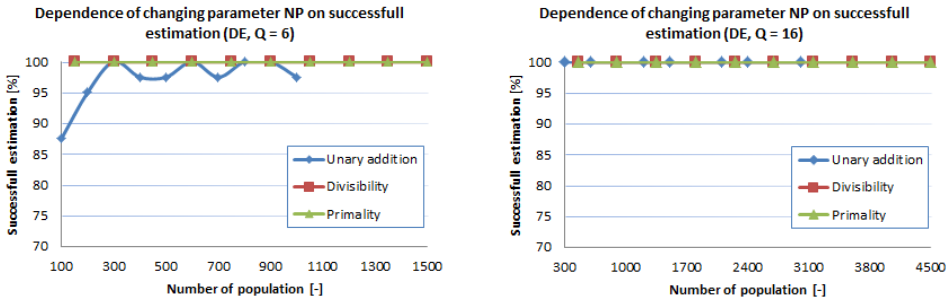

Fig. 8.1: Dependence of successfull estimation on changing DE's NP parameter

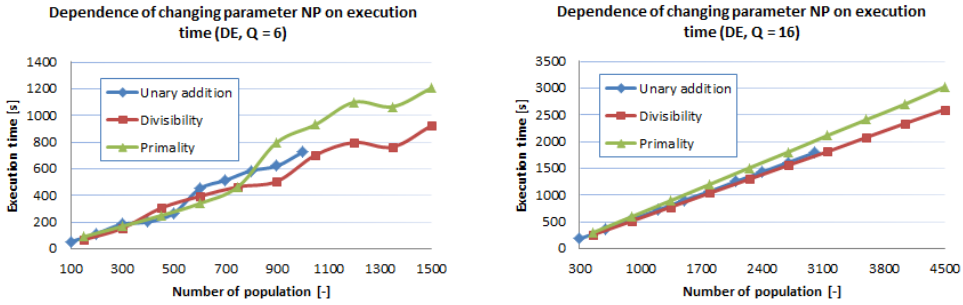

Fig. 8.2: Dependence of execution time on changing DE's NP parameter

As can be seen at Fig. 8.1, in the case of $Q = 6$ it can be said that increasing of population means the higher percentual successfull rate. This is evident for unary addition. When the divisibility and primality are processed, number of population which is equal to $10D$ is sufficient as well as in the case of $Q = 16$ and all problems processed.

Fig. 8.2 depicts strong dependence of execution time on the number of population. If number of population is increasing the execution time is increasing too.

Recommandation: $NP \in \langle 300, 100D \rangle$ and higher dimension of problem.

8.2.2 Dependence of optimization process on DE's F parameter

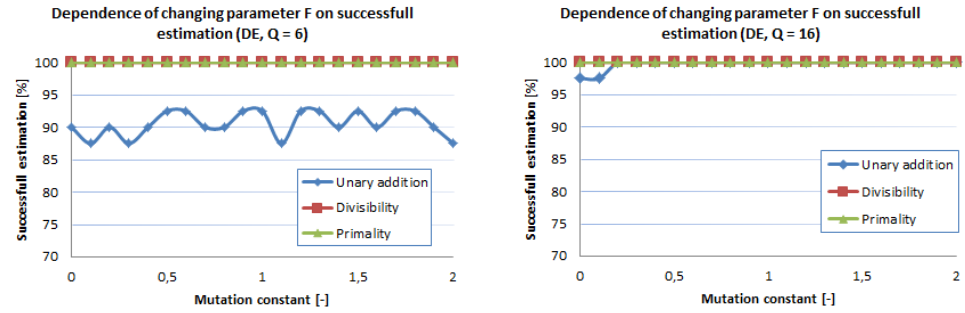

Fig. 8.3: Dependence of successfull estimation on changing DE's F parameter

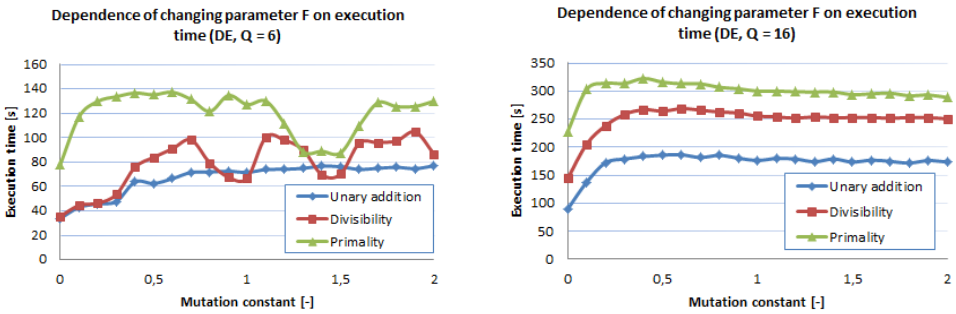

Fig. 8.4: Dependence of execution time on changing DE's F parameter

Analysis of dependence of successfull estimation on changing DE's parameter F (Fig. 8.3) revealed that estimation while changing parameter F is strongly dependent on dimension of the problem (see results for unary addition, divisibility and primality). In the case of unary addition, the dimension of the problem is 10 and the successfull estimation oscillates around 90%. Divisibility and primality problems have dimension equal to 15 what is enough for 100% successfull estimation. Important factors are number of inner states and data tape symbols of problem – see (8.3) and Fig. 8.3 for $Q = 16$.

Execution time of optimization process is not dependent on changing F parameter, thus while considering F parameter execution time can be omitted.

Recommandation: $F \in \langle 0.3,2 \rangle$ and higher dimension of problem.

8.2.3 Dependence of optimization proces on DE's CR parameter

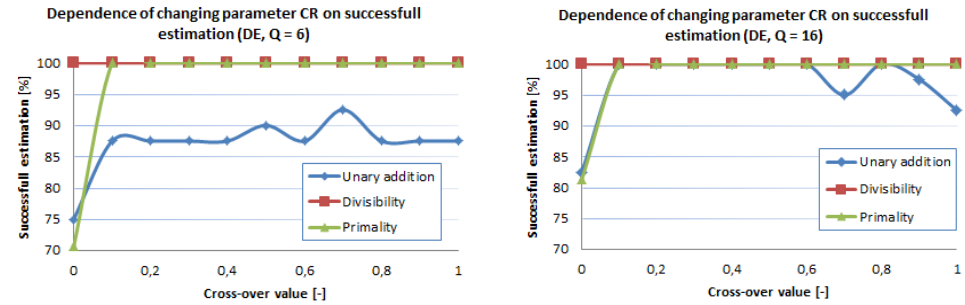

Fig. 8.5: Dependence of successfull estimation on changing DE's CR parameter

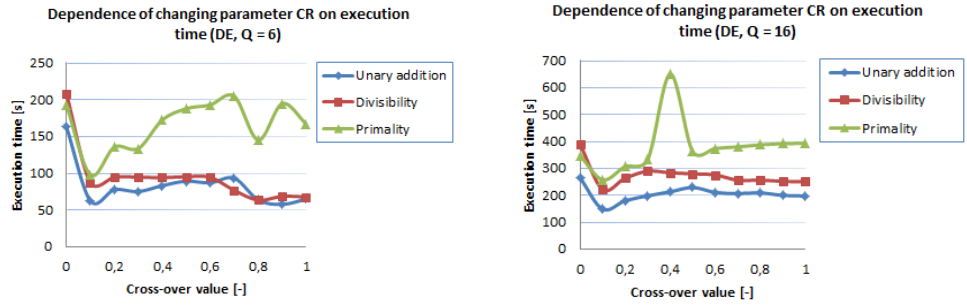

Fig. 8.6: Dependence of execution time on changing DE's CR parameter

Estimation while changing parameter *CR* can be regarded as highly dependent on dimension *D*. It is perceptible thanks to analysis and Fig. 8.5. In the case of unary addition and $Q = 16$, the dimension which is equal to 30 is even not enough. Except that, CR parameter also significantly influnce the successfull estimation in this case.

As well as in the changing parameter *F*, execution time of optimization proces is not dependent on changing *CR* parameter, thus while considering *CR* parameter execution time can be omitted.

Recommandation: $CR \in \langle 0.1, 0.6 \rangle$, higher dimension of problem.

8.2.4 Dependence of optimization proces on DE's G parameter

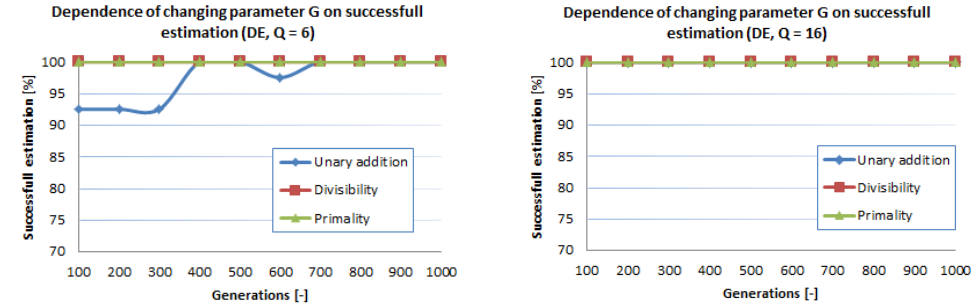

Fig. 8.7: Dependence of successfull estimation on changing DE's G parameter

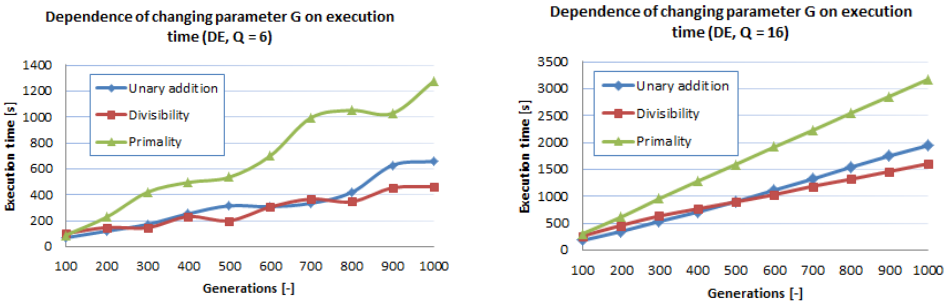

Fig. 8.8: Dependence of execution time on changing DE's G parameter

Progression of estimation proces while changing parameter *G* can be liken to situation when changing parameter *NP* has been analyzed. Except unary addition with $Q = 16$, the successfull estimation is always equal to 100% as can be seen at Fig. 8.7. Similarly to previous situations, dimension of the problem influence the successfull estimation significantly.

Execution time is highly dependent on parameter *G* thus number of generations as well as in the case of parameter *NP* changing. See Fig. 8.8 for details.

Recommandation: $G \in \langle 400, 1000 \rangle$, higher dimension of problem.

8.2.5 Dependence of optimization process on SOMA's PopSize parameter

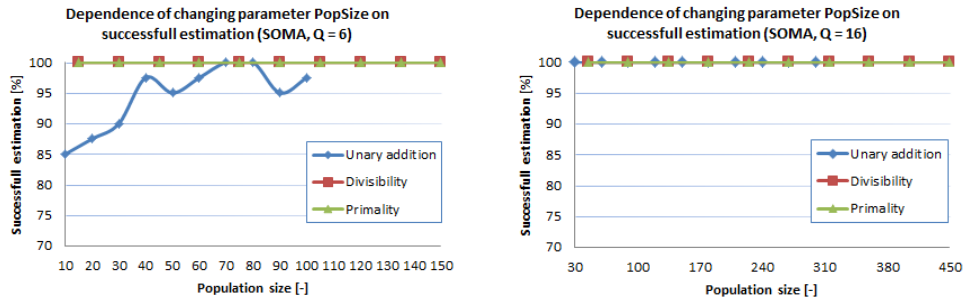

Fig. 8.9: Dependence of successfull estimation on changing SOMA's PopSize parameter

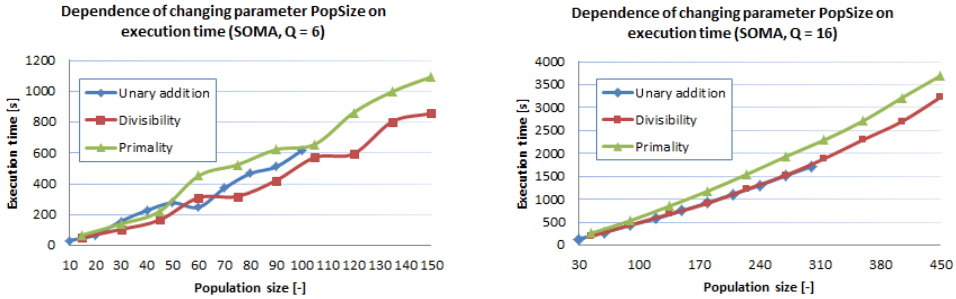

Fig. 8.10: Dependence of execution time on changing SOMA's PopSize parameter

As can be seen at Fig. 8.9, changing SOMA's *PopSize* parameter influences estimation process in the same way as changing DE's *NP* parameter. It means that successfull estimation of lower dimension problems can be positively influenced by increasing PopSize parameters especially when $Q = 6$. In the case of problems with higher dimension, PopSize which is equal to lower bound is sufficient.

Changing PopSize parameter significantly influences execution time thus there can be observed dependence of execution time on *PopSize* parameter as it is depicted at Fig. 8.10.

Recommandation: $PopSize \in \langle 80{,}10D \rangle$ and higher dimension of problem.

8.2.6 Dependence of optimization process on SOMA's PRT parameter

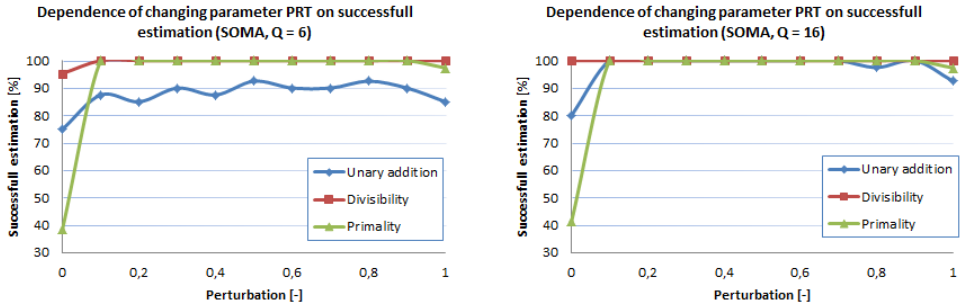

Fig. 8.11: Dependence of successfull estimation on changing SOMA's PRT parameter

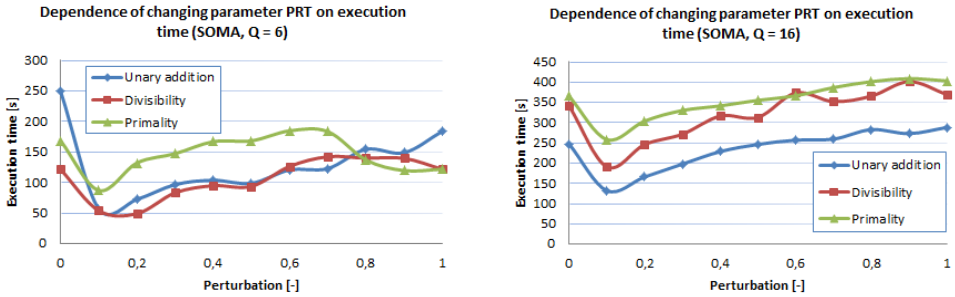

Fig. 8.12: Dependence of execution time on changing SOMA's PRT parameter

In the case of changing *PRT* parameter it can be said that successfull estimation is not only influenced by dimension of the problem but selection of the *PRT* parameter is highly important too. Higher dimension can bring better progress of estimation process but *PRT* parameter and its selection can still significantly influence results. This state is illustrated at Fig. 8.11.

Influence of execution time by changing *PRT* parameter is not too important in this case thus can be omitted.

Recommandation: $PRT \in \langle 0.2, 0.7 \rangle$, higher dimension of problem.

8.2.7 Dependence of optimization process on SOMA's PathLength parameter

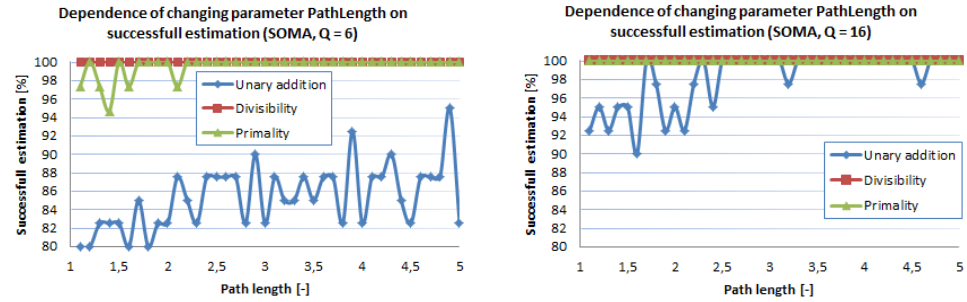

Fig. 8.13: Dependence of successfull estimation on changing SOMA's PathLength parameter

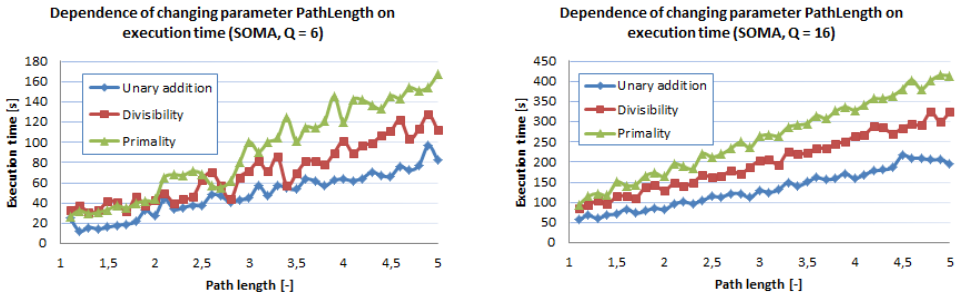

Fig. 8.14: Dependence of execution time on changing SOMA's PathLength parameter

In this case, importance of *PathLength* parameter and dimension of the problrm can be perceptible. The rate of successfull estimatation of unary addition whose dimension is lower than dimension of other problems is worse. This situation can be partially improved by selection

of suitable *PathLength* parameter value. But this improvement can be not enought in some cases as can be seen at right part of Fig. 8.13.

Execution time is strongly dependent on changing *PathLength* parameter. If value of *PathLength* parameter is incresed the execution time is increased too.

Recommandation: $PathLength \in \langle 2.5,5 \rangle$ and higher dimension of problem.

8.2.8 Dependence of optimization proces on SOMA's Step parameter

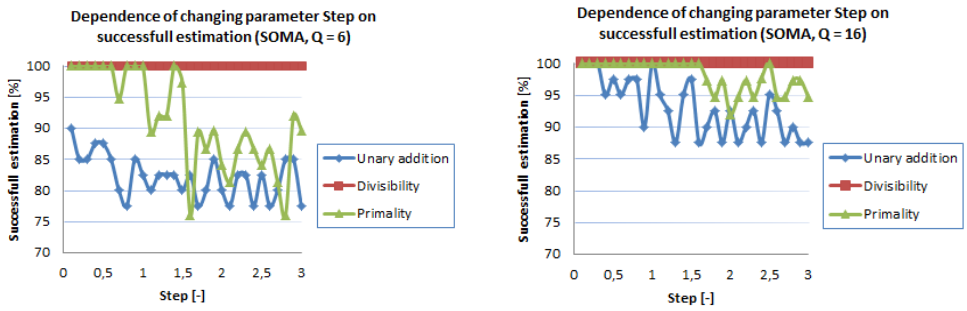

Fig. 8.15: Dependence of successfull estimation on changing SOMA's Step parameter

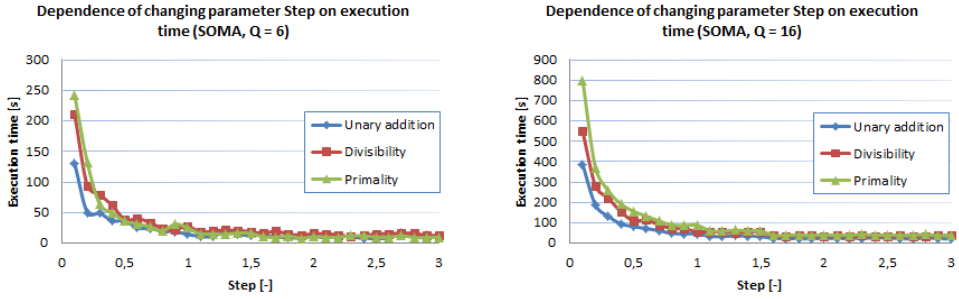

Fig. 8.16: Dependence of execution time on changing SOMA's Step parameter

The influence of increasing *Step* parameter has opposite effect than increasing other parameters. In this case the rate of successfull estimation is decreased when the value of *Step* parameter gets higher. The dependence of suitable *Step* value on estimation is significant as well as dependence of problem dimension. It can be said that high dimension of the problem is really fundamental for successfull estimation as can be seen at Fig. 8.15.

Descending tendency of execution time when increasing *Step* parameter can be observed here as well as in the case of successfull estimation.

Recommandation: $Step \in \langle 0.11,1.5 \rangle$, high dimension of problem.

8.2.9 Dependence of optimization process on SOMA's Migrations parameter

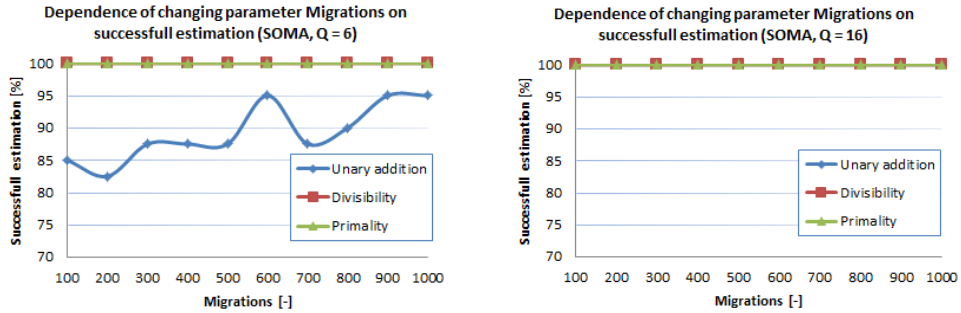

Fig. 8.17: Dependence of successfull estimation on changing SOMA's Migrations parameter

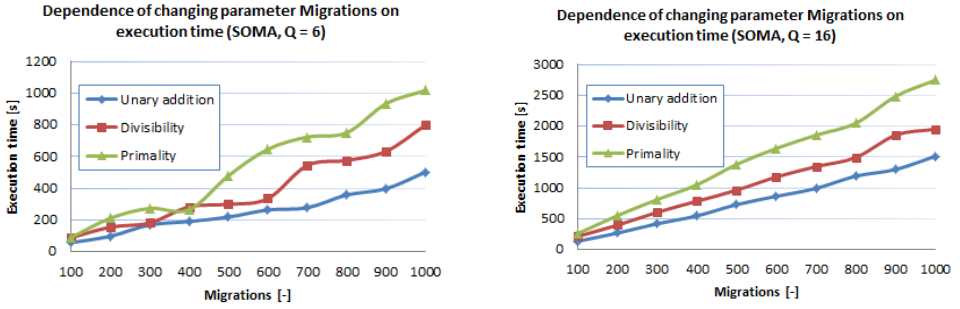

Fig. 8.18: Dependence of execution time on changing SOMA's Migrations parameter

Changing of the *Migrations* parameter has similar effect as in the case of changing *PopSize* parameter (see Fig. 8.17 and Fig. 8.9). Increasing of this parameter can positively influence the rate of successfull estimation as well as increasing of problem dimension. If unary addition is considered there is perceptible that its dimension which is equal to 10 ($Q = 6$) is not enough for 100% successful estimation.

Influence of changing *Migrations* parameter to execution time is similar again as in the case of changing *PopSize* parameter. When *Migrations* parameter is increasing, execution time is increasing too (see Fig. 8.18 and Fig. 8.10).

Recommandation: $Migrations \in \langle 600,1000 \rangle$, higher dimension of problem

8.3 Analysis conclusion

Analysis revealed several important depencence aspects of optimization process for estimating Turing machine's rules. Main goal of the analysis was specifiying suitable parameters of evolutionary algorihm selected for optimization in the next part of the doctoral thesis where processing real problems is considered. These parameters stated according to the analysis can be found in Table 11 and Table 12.

Table 11: DE parameters stated according to analysis recommandation

Parameter	Period	Values
NP	$\langle 10D, 100D \rangle$	$30D - 100D$
F	$\langle 0,2 \rangle$	0.3 – 2
CR	$\langle 0,1 \rangle$	0.1 – 0.6
G	customizable	400 – 1000

Table 12: SOMA parameters stated according to analysis recommandation

Parameter	Period	Values
PopSize	$\langle 10, \text{customizable} \rangle$	$8D - 10D$
PRT	$\langle 0,1 \rangle$	0.2 – 0.7
PathLength	$\langle 1.1,5 \rangle$	2.5 – 5
Step	$\langle 0.11, PathLength \rangle$	0.11 – 1.5
Migrations	$\langle 10, \text{customizable} \rangle$	600 – 1000

As can be seen in chaper 8.2, except parameters of evolutionary algorithms successfull estimation is strongly dependent on dimension of the problem. Dimension is specified as (8.3) and influenced by number of inner states of Turing machine and number of data tape symbols. It can be presumed that arguments of (8.3) will be sufficiently high in the case of real problem representation thus factor of dimension ensures successfull progression of estimation process without sole affecting optimization by parameters of evolutionary algorithm. Suitable settings of evolutionary algorithm used can be understood as "an insurance" if the dimension doesn't ensure successful estimation. Results of the analysis also proved that selection of the evolutionary algorithm used is not a key aspect of optimization process. Both selected evolutionary algorithms are entirely equivalent in the case of optimization Turing machine's rules. The results of successfull estimation are similar without dependence of evolutionary algorithm used.

During analysis there were optimized a large set of rules for processing example problems by Turing machine. Selected representants of these rules can be found in Appendix A.

9 PRACTICAL UTILIZATION

While discussing practical utilization of evolutionary synthesis of the Turing machine's rules there can be found many of applications in the form of problems related to utilization of Turing machines. Fundamental advantage of using approaches described in chapter 6 is to simply and reliably ensure programming Turing machine for processing even the most complex tasks. In chapter 8, there was proceeded analysis consisted in optimization of rules for processing several elementary problems. But the power of using evolutionary approaches to programming Turing machine is a possibility to use methods described in previous parts of the doctoral thesis for estimation and optimization Turing machine's rules for processing real problems which are not elementary at all. As a representant of highly complex real problems protein processing was selected. From the view of using Turing machines, the task of estimation the rules for processing proteins is very complicated thus can be regarded as suitable for proving abilities of described approaches to evolutionary synthesis to successfully estimate proper Turing machine's rules.

9.1 Proteins essentials

Proteins [2, 3, 8, 36] can be regarded as the most important for all organisms which live and generally exist on the Earth. It is thanks to a key feature of proteins which influences processes belonging to life build blocks. This key feature is entirely uniqe for proteins. As life build blocks anyone can imagine prepositions for life genesis, keeping the life, controlling organisms' internal and external processes, metabolism, immune system etc. Cell structures, respiration, muscles, skin – these are results of protein features and activities as well as accelerators of biochemical processes – enzymes [1, 36], which are fundamentals of internal processes. Proteins have impact to all life-based activities.

9.1.1 Proteins as Turing machine's data tapes

In order to explain how to transcribe proteins as the Turing machine's data tapes it is necessary to breifly introduce protein biosynthesis.

Base elements of proteins are macromolecule of deoxyribonucleic acid (DNA) [5, 26, 36] or ribonucleic acid (RNA) [6, 26, 36] eventually. DNA macromolecule has a form of double helix [28, 36]. Strands of double helices are made of sugar-phosphate residues and serve for retaining sequences of four bases [36]. These are adenine (A), guanine (G), cytosine (C) and thymine (T). Combinations of residues and bases are termed as nucleotides [25, 36] or nucleosides [25, 36]. It depends on condition whether base is combined with sugar and phosphate residues (nucleotide) or sugar residue only (nucleoside). These combinations are termed as adenosine, guanosine, cytidine and thymidine. Both strands of DNA helix are held together by hydrogen bonds [22, 36] which originate between following pairs of bases:

- A – T
- G – C
- T – A
- C – G

Basis transformation to the form of protein is two-step process composed by:

- Transcription
- Translation

Transcription [9, 29, 36] is a process which can be liken to DNA replication [24, 36]. During DNA replication a new complementary strand of DNA helix is created by DNA polymerase

enzyme [36] in accordance with base pairing regarding one of strands. New DNA macromolecule identical to ancestor is created (see Fig. 9.1).

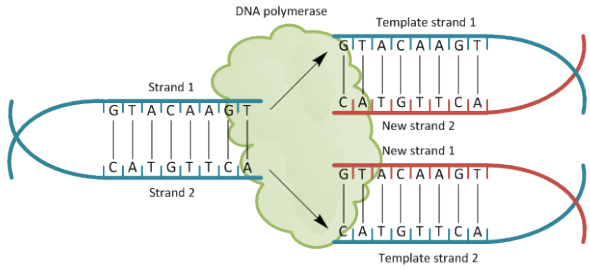

Fig. 9.1: Scheme of DNA replication

Transciption differs from DNA replication by enzyme. Instead of DNA polymerase, RNA polymerase [36] is used. It is perceptible that the result is not identical DNA macromolecule but RNA macromolecule. The process of transcription is similar to replication. One of strands serves as template thus it is termed as non-coding because contains complementary bases [22]. Another strand is termed as coding. Because new strand is created according to template strand its appereance is identical to coding strand but thymine is replaced by uracile (U) (see Fig. 9.2).

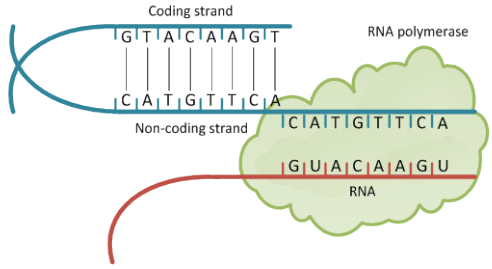

Fig. 9.2: Scheme of DNA transcription

Contrary to DNA macromolecules, RNA macromolecules are compounded of one strand only. Also appereance of RNA macromolecules is much less regular than appereance of DNA macromolecules.

RNA originated from DNA by transcription is termed as messenger RNA (mRNA) and contains “simple” transcription of DNA information. It is not possible to synthesize proteins from mRNA directly.

Second step of protein origination is a translation [9, 36]. During translation nucleotides contained in mRNA strand are processed as triplets termed as codons [23, 36]. There are 64 different codons which are related to combinations of four nucleotides in mRNA. Three of these codens (UAA, UAG and UGA) represent so-called stop-codons [36] which denotes end of protein. Other codons are considered as amino acids [7, 36] as well as AUG codon which is protein opening codon. Because there are only 20 common amino acids it is clear that several different codons (or combinations of nucleotides) can represent same amino acid (e.g. GCU, GCC, GCA and GCG codons stand for alanin amino acid).

After translation, there are obtained amino acids which have direct impact to proteins because proteins are biopolymers consisted of amino acid sequences. Amino acids are molecules compounded of two functional groups [8, 36]. These are amine group NH_2 and carboxylic acid group COOH (see Fig. 9.3).

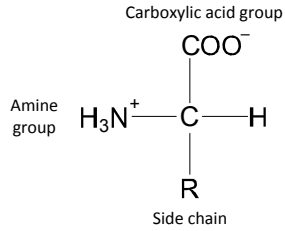

Fig. 9.3: General form of amino acid

Thanks to above-mentioned functional groups amino acids can be chained together thus proteins can be formed. Because amine group is a base which allows to react with acids and neutralize them and carboxylic acid is an oposite part of molecule, it is possible to be reaction between amino acids arisen by condensation process. This process enables to originate a peptide C-N bond [27, 36] between amino acids. Sequenced amino acids forms polypeptide chains [36] which are regarded as primary protein structures [21, 36].

Except full name of amino acids, they can be referred by abbreviations. These are 3-letter or 1-letter. It is perceptible that 1-letter abbreviations of amino acid names are the most suitable in the case of processing by Turing machines. Thus the Turing machine's data tape symbols can simply appear as following parts of Turing machine formal definition:

- $\Gamma = \begin{Bmatrix} "\#","A","C","D","E","F","G","H","I","K","L", \\ "M","N","O","Q","R","S","T","V","W","Y" \end{Bmatrix}$
- $B = "\#"$

9.2 Protein processing by Turing machine

Employing Turing machines at protein processing is a representant of real highly complex and interesting problem. The protein processing as it is considered related to this doctoral thesis and using Turing machines involves protein reconstruction independly on origin protein structure. It means processing polypeptide chains containing randomly positioned amino acids to the form of requested primary protein structures by Turing machine. The optimized Turing machine's rules which ensure above-mentioned protein processing enable e.g. simple protein description resulted from general set of amino acids, reconstruction of corrupted primary protein structures etc.

As mentioned protein processing is a highly complex problem. If the dimension is expressed as (8.3) it is possible to obtain:

$$D = (Q - 1) \cdot 21 \tag{9.1}$$

If the numer of inner states Q is equal to 6, dimension of the problem is 105. In the case of $Q = 16$, dimension is significantly increased to 315. If it is considered that maximal dimension of some problems used while analysis was 45 and maximal execution time was nearly 1 hour (see Fig. 8.10), time consumption could by higly unsatisfactory in the case of $D = 105$ or even $D = 315$. Additionaly it is necessary to discuss whether number of inner states equal to 16 is sufficient for such complex problem as it is protein processing. If the Q is set too low there is a risk of impossibility to successfully optimize Turing machine's rules. Fortunately, issue of stating Q parameter is only related to using classical optimization (see chapter 6.1). If the per-partes optimization (see chapter 6.2) is used, number of inner states is adapting dynamically during optimization. Also dimension of the problem is considerably decreased without influence of successfull estimation thanks to per-partes approach to optimization (see chapter 6.2 for more details). Thus the per-partes optimization is selected in relation to protein processing.

There are two prepositions for using per-partes optimization which have to be carefully considered. These are:

- Selection of evolutionary algorithm and its parameters
- Per-partes optimization's weights (see chapter 6.2.3) selection

The answer to first preposition is rather simple. Analysis approved that Turing machine's rules estimation process is not directly dependent on evolutionary algorithm selected. Instead of that the dependence is on settings of evolutionary algorihm. From this point of view Differential Evolution and Self-Organizing Migrating Algorithm are entirely equivalent thus Differential Evolution will be selected as algorithm used in relation to the per-partes optimization. Parameters of Differential Evolution will be specified according to Table 11 and previous experiences. Selected values can be seen in Table 13.

Table 13: Parameters of Differential Evolution for protein processing

Parameter	Recommanded value	Values
NP	30*D* – 100*D*	200
F	0.3 – 2	0.9
CR	0.1 – 0.6	0.3
G	400 – 1000	1000

Second preposition is more important because it influences optimization process and Turing machine operations while processing proteins. It can be said that the per-partes optimization's weights provides answers to questions contained in Table 5 (also see Fig. 6.10). By changing weights it is possible to completely alter behavior of Turing machine while processing proteins or other problems too. There is a lot of ways how to set these weights. At this place the set of weights which was experimentally obtained and successfully approved many times (e.g. here [14]) will be used. This set is contained in Table 14. Detailed exaplanation of meaning of these weights can be found in chapter 6.2.3.

Table 14: Per-partes optimization's weights

Weight	Value
1	4000
2	2000
3	1000
4	-2000
5	-1000
6	-3000
7	1000

Concrete specification of the protein processing tasks can be formulated as reconstruction of proteins from the set of amino acids. It means that it is requested to evolutionary estimate rules which enable to transcribe a data tape containing sequence of randomly positioned amino acids to the form of primary structure of selected protein whereas both of sequences have the same length. There were selected twelve proteins [30, 40] whose primary protein structures will be

used for processing. These primary protein structures differ from each other by length thus additional variability and complexity increasing is ensured.

All algorithms which will be used (Turing machine, Differential Evolution, per-partes optimization algorithm) are written in functional programming language F# [50]. The reason is that the effectivity of F# programming language is much higher than algorithm implementations in Wolfram Mathematica used till this time as it was proven [13]. Turing machine and Differential Evolution implemented in F# are parts of the F# Artificial Intelligence Library [11].

Following subsections include information on sequences used, summary of obtained 5 results of processing each polypeptide sequence and analysis of head movement while processing amino acid sequences.

Shared Turing machine settings for all results are:

- $\Sigma = \begin{Bmatrix} "A","C","D","E","F","G","H","I","K","L", \\ "M","N","O","Q","R","S","T","V","W","Y" \end{Bmatrix}$
- $\Gamma = \begin{Bmatrix} "\#","A","C","D","E","F","G","H","I","K","L", \\ "M","N","O","Q","R","S","T","V","W","Y" \end{Bmatrix}$
- $q_0 = q_1$
- $B = "\#"$
- Initial position of the head is at the fifth left symbol contained in the data tape

Data tapes look like:

- $tape = \{"\#","\#","\#","\#","\#", \ldots sequence \ldots, "\#","\#","\#","\#","\#"\},$

where *sequence* is an amino acid protein-coding part of test sequence described in following subsections.

Parameters of Differential Evolution were set according to Table 13. Weights of the per-partes optimization algorithm can be seen in Table 14. Images of the proteins which are shown below were created in Geneious v5.5 [4] software application.

9.2.1 2J01

Structure of the thermus thermophilus 70s ribosome complexed with mRNA, tRNA and Paromomycin (part 2 of 4) [30, 48] – chain 1. Length of sequence is 98 amino acids.

Seq. 1: 2J01 chain 1 sequence

```
MSKVCEISGKRPIVANSIQRRGKAKREGGVGKKTTGISKRRQYPNLQKVRVRVAG
QEITFRVAASHIPKVYELVERAKGLKLEGLSPKEIKKELLKLL
```

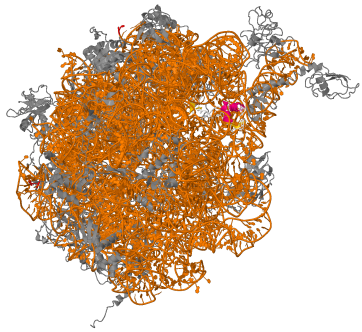

Fig. 9.4: Image of 2J01 with highlighted chain 1

Seq. 2: Test sequence for 2J01

```
TYNDFEWEQMLQRAVNSFDQGSWGLEKMTTYGGMSPNWQVRQVPWFLTLDGMEEP
PQGFWKEKVQHWSGANLKVRFFFFDYHQLHASVVCNHRIMWYE
```

Table 15: Summary of 5 obtained results of processing 2J01 chain 1

Result no.	Q	F	Steps	Rate* [%]
1	$\{q_1, \dots, q_{10}\}$	$\{q_{10}\}$	116	100
2	$\{q_1, \dots, q_{16}\}$	$\{q_{16}\}$	127	100
3	$\{q_1, \dots, q_{13}\}$	$\{q_{13}\}$	125	100
4	$\{q_1, \dots, q_{15}\}$	$\{q_{15}\}$	124	100
5	$\{q_1, \dots, q_{34}\}$	$\{q_{34}\}$	170	100
Average successfull processing amino acids contained in sequence [%]				100

* It means percentual rate of all correctly processed amino acids in sequence

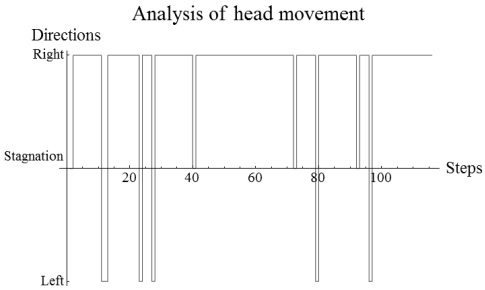

Fig. 9.5: Analysis of head movement for 1J01 chain 1 result 1

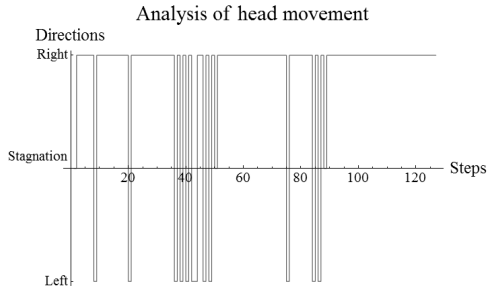

Fig. 9.6: Analysis of head movement for 1J01 chain 1 result 2

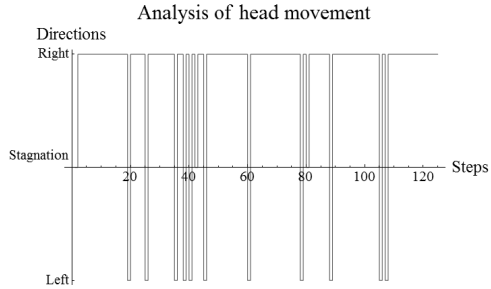

Fig. 9.7: Analysis of head movement for 1J01 chain 1 result 3

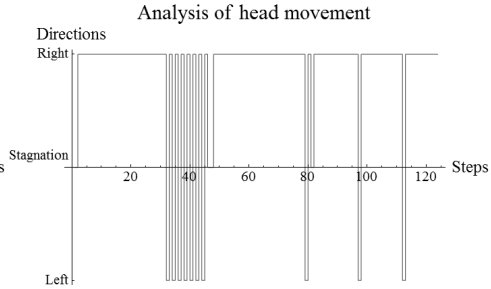

Fig. 9.8: Analysis of head movement for 1J01 chain 1 result 4

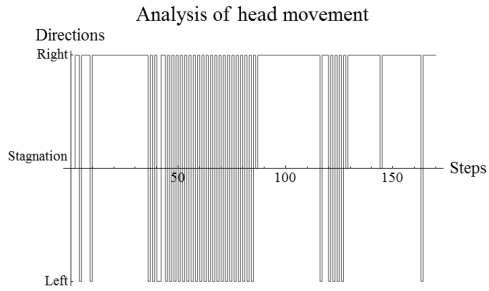

Fig. 9.9: Analysis of head movement for 1J01 chain 1 result 5

9.2.2 1AOI

Complex between nucleosome core particle (H3, H4, H2A, H2B) and 146 bp long DNA fragment [30, 38] – chain A. Length of sequence is 116 amino acids.

Seq. 3: 1AOI chain A sequence

```
LATKAARKSAPATGGVKKPHRYRPGTVALREIRRYQKSTELLIRKLPFQRLVREI
AQDFKTDLRFQSSAVMALQEASEAYLVALFEDTNLCAIHAKRVTIMPKDIQLARR
IRGERA
```

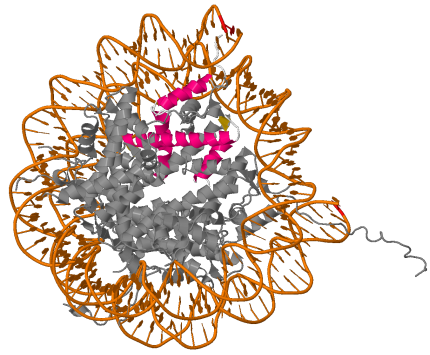

Fig. 9.10: Image of 1AOI with highlighted chain A

Seq. 4: Test sequence for 1AOI

```
QPCESKNFCVEWNTECQLWNVTEGDTPPATESCMNNNKEPQEDSVEKCFKYGSIM
SPPVKDIFCAKMKRVFRKPNNKMDEPRFPHYMCVCYPRYNPFCMKMRCQMKNVIN
DPFLQE
```

Table 16: Summary of 5 obtained results of processing 1AOI chain A

Result no.	Q	F	Steps	Rate* [%]
1	$\{q_1, \dots, q_{22}\}$	$\{q_{22}\}$	175	100
2	$\{q_1, \dots, q_{24}\}$	$\{q_{24}\}$	171	99.14
3	$\{q_1, \dots, q_{14}\}$	$\{q_{14}\}$	143	100
4	$\{q_1, \dots, q_{15}\}$	$\{q_{15}\}$	132	100
5	$\{q_1, \dots, q_{14}\}$	$\{q_{14}\}$	142	100
Average successfull processing amino acids contained in sequence [%]				99.83

* It means percentual rate of all correctly processed amino acids in sequence

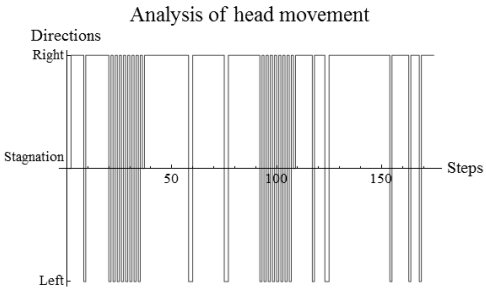

Fig. 9.11: Analysis of head movement for 1AOI chain A result 1

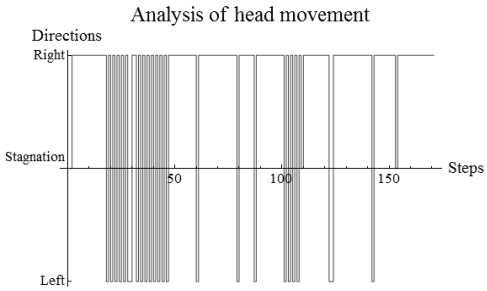

Fig. 9.12: Analysis of head movement for 1AOI chain A result 2

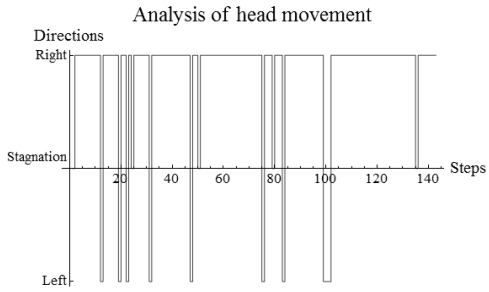

Fig. 9.13: Analysis of head movement for 1AOI chain A result 3

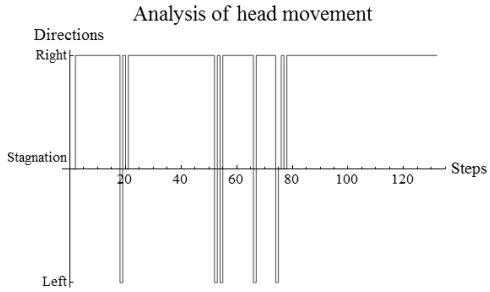

Fig. 9.14: Analysis of head movement for 1AOI chain A result 4

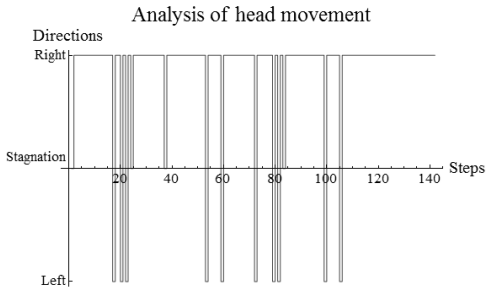

Fig. 9.15: Analysis of head movement for 1AOI chain A result 5

9.2.3 1LIT

Human lithostathine [30, 44] – chain A. Length of sequence is 144 amino acids.

Seq. 5: 1LIT chain A sequence

```
QEAQTELPQARISCPEGTNAYRSYCYYFNEDRETWVDADLYCQNMNSGNLVSVLT
QAEGAFVASLIKESGTDDFNVWIGLHDPKKNRRWHWSSGSLVSYKSWGIGAPSSV
NPGYCVSLTSSTGFQKWKDVPCEDKFSFVCKFKN
```

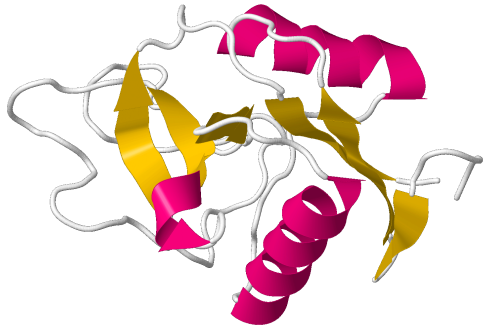

Fig. 9.16: Image of 1LIT with highlighted chain A

Seq. 6: Test sequence for 1LIT

```
FFYGKHMSQIVVWRGIDNWAQKKSFMDTVIPCEIKKNRIPATHVASQFHCFIKTM
HCVPPCYPYPKGIRIGFENNSCFALDRSKNIHMIATNGCQWFLSRCIFQGWAPEF
DREEGMETSVAYEKEPHQPNFRTYRDVQKSEWPD
```

Table 17: Summary of 5 obtained results of processing 1LIT chain A

Result no.	Q	F	Steps	Rate* [%]
1	$\{q_1, \ldots, q_{57}\}$	$\{q_{57}\}$	285	100
2	$\{q_1, \ldots, q_{42}\}$	$\{q_{42}\}$	263	97.92
3	$\{q_1, \ldots, q_{51}\}$	$\{q_{51}\}$	279	99.30
4	$\{q_1, \ldots, q_{57}\}$	$\{q_{57}\}$	315	99.30
5	$\{q_1, \ldots, q_{44}\}$	$\{q_{44}\}$	276	99.30
Average successfull processing amino acids contained in sequence [%]				99.16

* It means percentual rate of all correctly processed amino acids in sequence

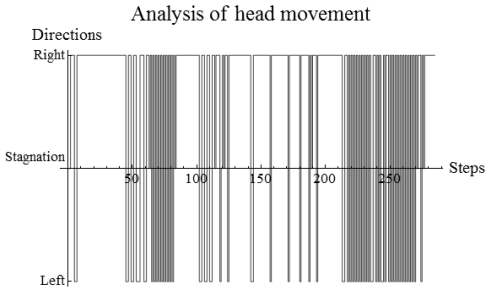

Fig. 9.17: Analysis of head movement for 1LIT chain A result 1

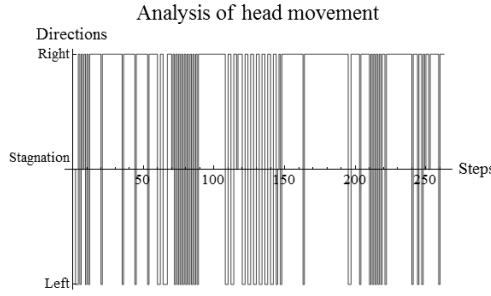

Fig. 9.18: Analysis of head movement for 1LIT chain A result 2

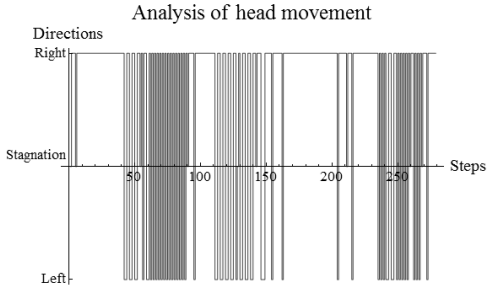

Fig. 9.19: Analysis of head movement for 1LIT chain A result 3

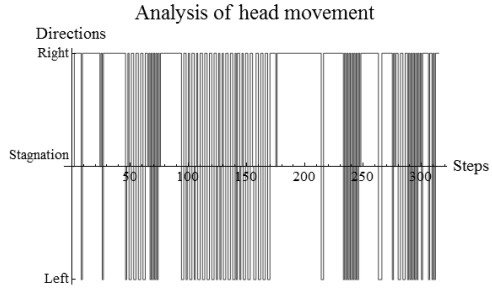

Fig. 9.20: Analysis of head movement for 1LIT chain A result 4

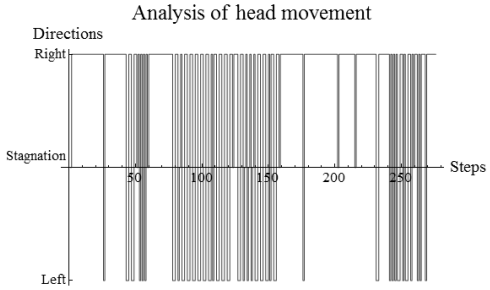

Fig. 9.21: Analysis of head movement for 1LIT chain A result 5

9.2.4 1B08

Lung surfactant protein D (SP-D) [30, 39] – chain A. Length of sequence is 158 amino acids.

Seq. 7: 1B08 chain A sequence

```
EAEAGSVASLRQQVEALQGQVQHLQAAFSQYKKVELFPNGQSVGEKIFKTAGFVK
PFTEAQLLCTQAGGQLASPRSAAENAALQQLVVAKNEAAFLSMTDSKTEGKFTYP
TGESLVYSNWAPGEPNDDGGSEDCVEIFTNGKWNDRACGEKRLVVCEF
```

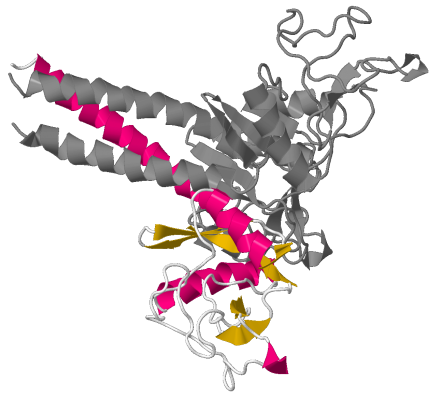

Fig. 9.22: Image of 1B08 with highlighted chain A

Seq. 8: Test sequence for 1B08

```
PQIGSGDDIDKEALQAICWKQMGSIIGRMEIPRFHYWVVVWDYAYHIERPHHPPAQ
EKTEACGSHMPSYRWLIIFSDYLNCEEAPKTSYSTMDKGSCILGAEMPKKCEYQWW
FKGYPALNRYIRPAKCPHQKIRRYVITWIQDGNMCMNMGQDYCKNW
```

Table 18: Summary of 5 obtained results of processing 1B08 chain A

Result no.	Q	F	Steps	Rate* [%]
1	$\{q_1, \dots, q_{78}\}$	$\{q_{78}\}$	373	98.73
2	$\{q_1, \dots, q_{66}\}$	$\{q_{66}\}$	351	96,84
3	$\{q_1, \dots, q_{61}\}$	$\{q_{61}\}$	337	99.37
4	$\{q_1, \dots, q_{82}\}$	$\{q_{82}\}$	376	98.73
5	$\{q_1, \dots, q_{81}\}$	$\{q_{81}\}$	366	99.37
Average successfull processing amino acids contained in sequence [%]				98.61

* It means percentual rate of all correctly processed amino acids in sequence

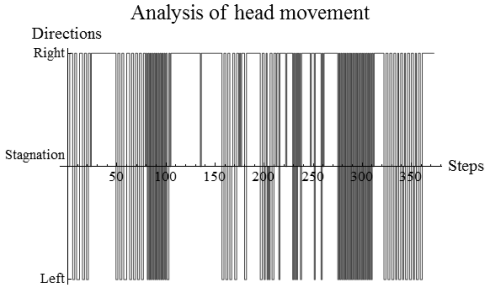

Fig. 9.23: Analysis of head movement for 1B08 chain A result 1

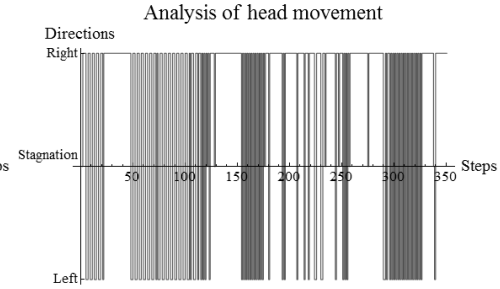

Fig. 9.24: Analysis of head movement for 1B08 chain A result 2

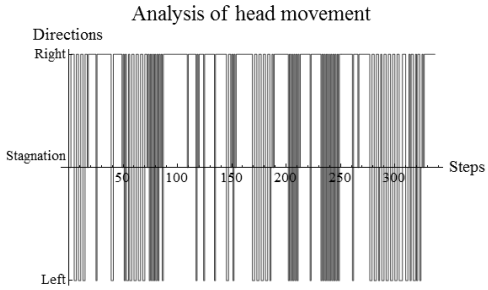

Fig. 9.25: Analysis of head movement for 1B08 chain A result 3

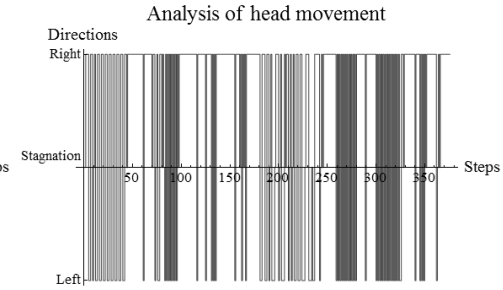

Fig. 9.26: Analysis of head movement for 1B08 chain A result 4

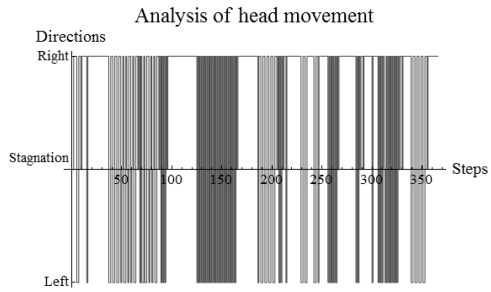

Fig. 9.27: Analysis of head movement for 1B08 chain A result 5

9.2.5 1B09

Human C-reactive protein complexed with phosphocholine [40, 41] – chain A. Length of sequence is 206 amino acids.

Seq. 9: 1B09 chain A sequence

```
QTDMSRKAFVFPKESDTSYVSLKAPLTKPLKAFTVCLHFYTELSSTRGYSIFSYA
TKRQDNEILIFWSKDIGYSFTVGGSEILFEVPEVTVAPVHICTSWESASGIVEFW
VDGKPRVRKSLKKGYTVGAEASIILGQEQDSFGGNFEGSQSLVGDIGNVNMWDFV
LSPDEINTIYLGGPFSPNVLNWRALKYEVQGEVFTKPQLW
```

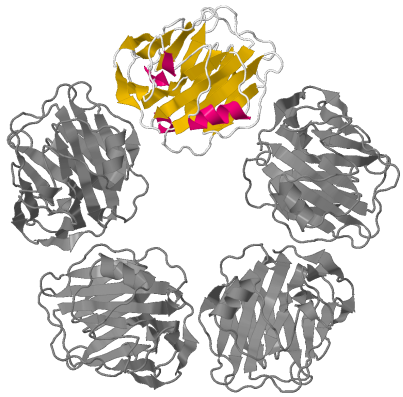

Fig. 9.28: Image of 1B09 with highlighted chain A

Seq. 10: Test sequence for 1B09

```
VWQGEKRCCEFMICYFIYTLTLFIPVYSVSTIERQYQELNRRRTTLCGMDIKEGS
AGCAGKVTRCKGCIHKQEHEASEGKDITVEFRMCKASSFTKLFDVDLLMCTTNCD
TYWVHQPLPHRTEVKEAIIINRYSWMTVNRDYMKEEDKGYHSAIIWYVAAKYLTT
RTDMPESVPWAYMMFNWRCHWFKGPCMENDIAIFHINVGGM
```

Table 19: Summary of 5 obtained results of processing 1B09 chain A

Result no.	Q	F	Steps	Rate* [%]
1	$\{q_1, \dots, q_{41}\}$	$\{q_{41}\}$	310	98.54
2	$\{q_1, \dots, q_{113}\}$	$\{q_{113}\}$	481	99.03
3	$\{q_1, \dots, q_{68}\}$	$\{q_{68}\}$	370	97.57
4	$\{q_1, \dots, q_{61}\}$	$\{q_{61}\}$	366	98.06
5	$\{q_1, \dots, q_{87}\}$	$\{q_{87}\}$	413	98.54
Average successfull processing amino acids contained in sequence [%]				98.35

* It means percentual rate of all correctly processed amino acids in sequence

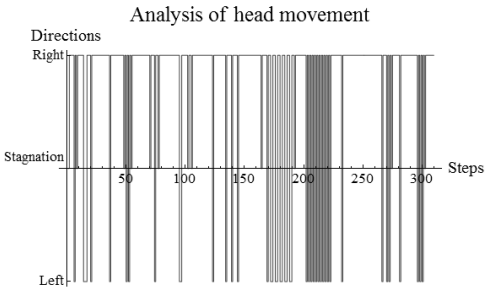

Fig. 9.29: Analysis of head movement for 1B09 chain A result 1

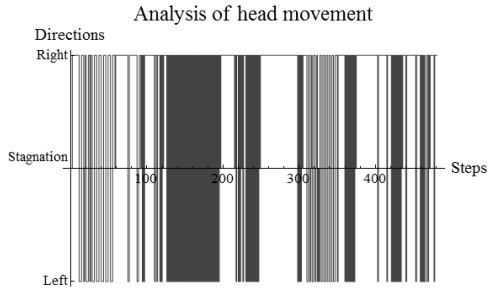

Fig. 9.30: Analysis of head movement for 1B09 chain A result 2

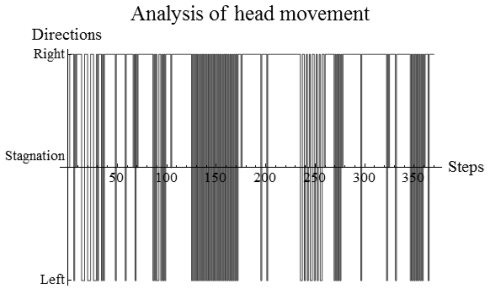

Fig. 9.31: Analysis of head movement for 1B09 chain A result 3

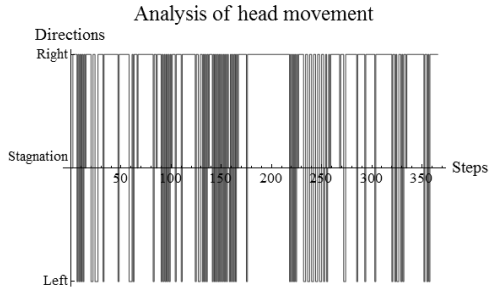

Fig. 9.32: Analysis of head movement for 1B09 chain A result 4

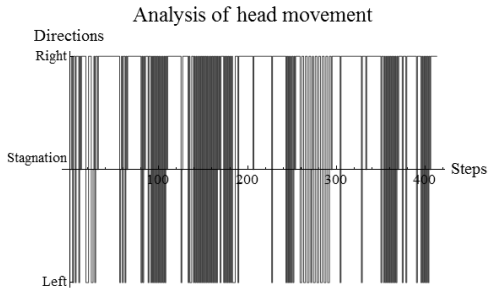

Fig. 9.33: Analysis of head movement for 1B09 chain A result 5

9.2.6 1TUP

Tumor suppressor P53 complexed with DNA [30, 45] – chain A. Length of sequence is 219 amino acids.

Seq. 11: 1TUP chain A sequence

```
SSSVPSQKTYQGSYGFRLGFLHSGTAKSVTCTYSPALNKMFCQLAKTCPVQLWVD
STPPPGTRVRAMAIYKQSQHMTEVVRRCPHHERCSDSDGLAPPQHLIRVEGNLRV
EYLDDRNTFRHSVVVPYEPPEVGSDCTTIHYNYMCNSSCMGGMNRRPILTIITLE
DSSGNLLGRNSFEVRVCACPGRDRRTEEENLRKKGEPHHELPPGSTKRALPNNT
```

Fig. 9.34: Image of 1TUP with highlighted chain A

Seq. 12: Test sequence for 1TUP

```
MFQEDDDFEISQHSIHWAIWGDGVFVGSYAYVHINWEFVSCIHPINIANQIATAV
VDLAVEYYIAQNSNFKHRTMRQCPGTWNPSCARVKDVVYAIPHLHCYWCFQTKLN
LTTSDSFDRQTYCLYVFGTKEGPNKISHADLENFQHEPQYSHSEVANMQLHKFVE
RTECNAIRALHWDTQKDYKMKHFSFARDGFWFVYYMNMYALEDRQSMKQNWFP
```

Table 20: Summary of 5 obtained results of processing 1TUP chain A

Result no.	***Q***	***F***	**Steps**	**Rate* [%]**
1	$\{q_1, \dots, q_{36}\}$	$\{q_{36}\}$	288	99.09
2	$\{q_1, \dots, q_{45}\}$	$\{q_{45}\}$	323	98.63
3	$\{q_1, \dots, q_{46}\}$	$\{q_{46}\}$	334	100
4	$\{q_1, \dots, q_{32}\}$	$\{q_{32}\}$	308	100
5	$\{q_1, \dots, q_{35}\}$	$\{q_{35}\}$	310	100
Average successfull processing amino acids contained in sequence [%]				99.54

* It means percentual rate of all correctly processed amino acids in sequence

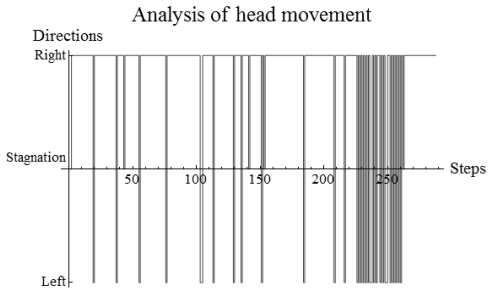

Fig. 9.35: Analysis of head movement for 1TUP chain A result 1

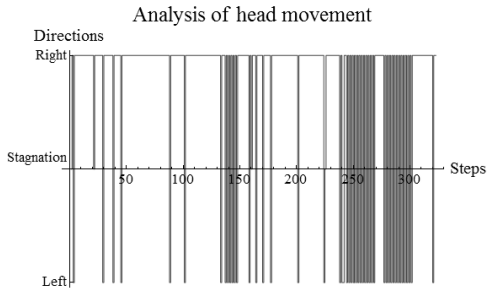

Fig. 9.36: Analysis of head movement for 1TUP chain A result 2

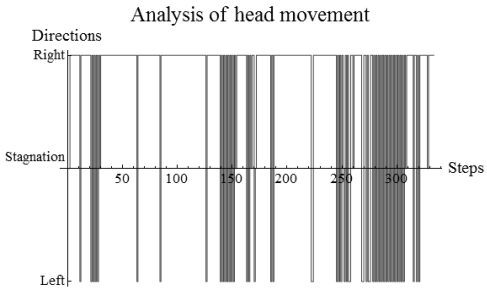

Fig. 9.37: Analysis of head movement for 1TUP chain A result 3

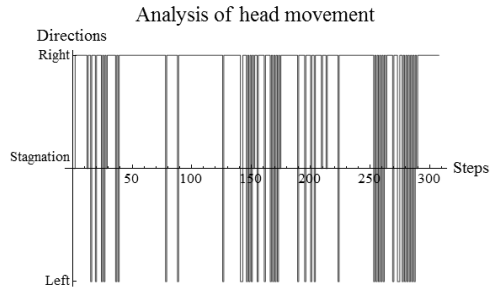

Fig. 9.38: Analysis of head movement for 1TUP chain A result 4

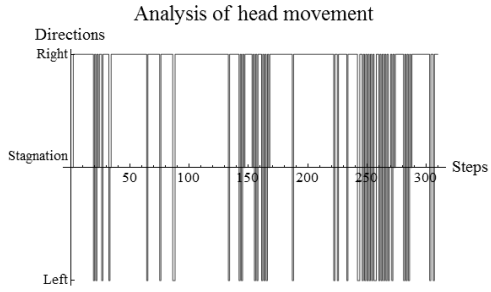

Fig. 9.39: Analysis of head movement for 1TUP chain A result 5

9.2.7 1YAR

Structure of archeabacterial 20S proteasome mutant D9S – PA26 complex [30, 46] – chain A. Length of sequence is 233 amino acids.

Seq. 13: 1YAR chain A sequence

```
MQQGQMAYSRAITVFSPDGRLFQVEYAREAVKKGSTALGMKFANGVLLISDKKVR
SRLIEQNSIEKIQLIDDYVAAVTSGLVADARVLVDFARISAQQEKVTYGSLVNIE
NLVKRVADQMQQYTQYGGVRPYGVSLIFAGIDQIGPRLFDCDPAGTINEYKATAI
GSGKDAVVSFLEREYKENLPEKEAVTLGIKALKSSLEEGEELKAPEIASITVGNK
YRIYDQEEVKKFL
```

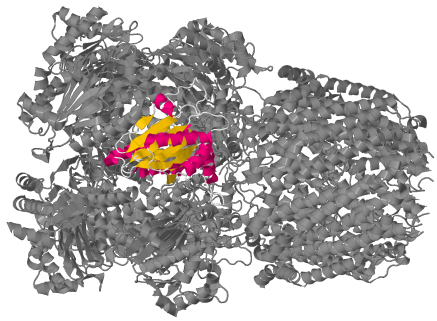

Fig. 9.40: Image of 1YAR with highlighted chain A

Seq. 14: Test sequence for 1YAR

```
CVPGIFPYQASNAFEISKMAPYYFKNKRNVKINPVAKYTFEWASGPGKNKNYWDV
PLMYHLSWHIHHRDNRDSMKFFHLMIDNKHCMYCIMWNPSSKHNHLERYLQWGTW
NMHEHMMLSITRYPAFPMLVTKPMSEQNPNECDATPRCMKQHCSDDFWQPFEKGS
ARRERFTFRMQWTYMRLQSFRANHLCRTKFTLKMLQEAVNWCQWMYRFFHDPCER
FKANTAVEPPIFG
```

Table 21: Summary of 5 obtained results of processing 1YAR chain A

Result no.	Q	F	Steps	Rate* [%]
1	$\{q_1, \dots, q_{41}\}$	$\{q_{41}\}$	353	100
2	$\{q_1, \dots, q_{59}\}$	$\{q_{59}\}$	386	98.71
3	$\{q_1, \dots, q_{66}\}$	$\{q_{66}\}$	416	99.57
4	$\{q_1, \dots, q_{57}\}$	$\{q_{57}\}$	409	99.57
5	$\{q_1, \dots, q_{27}\}$	$\{q_{27}\}$	309	97.85
Average successfull processing amino acids contained in sequence [%]				99.14

* It means percentual rate of all correctly processed amino acids in sequence

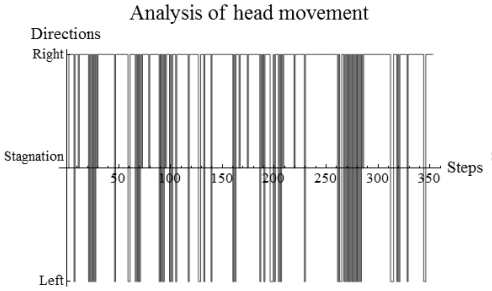

Fig. 9.41: Analysis of head movement for 1YAR chain A result 1

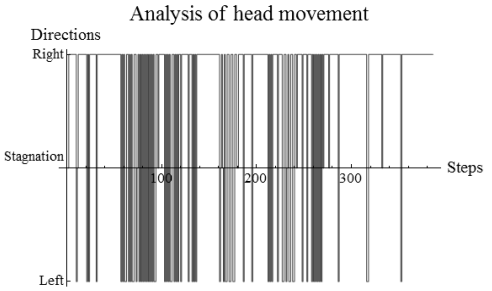

Fig. 9.42: Analysis of head movement for 1YAR chain A result 2

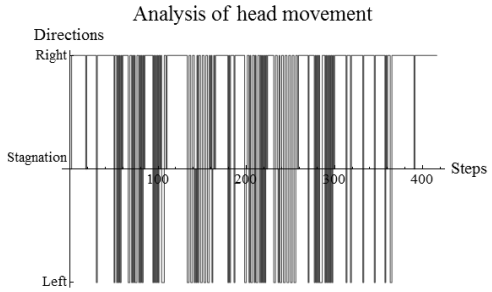

Fig. 9.43: Analysis of head movement for 1YAR chain A result 3

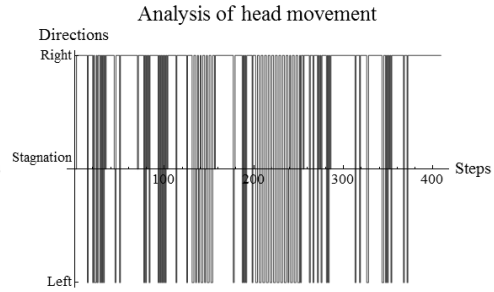

Fig. 9.44: Analysis of head movement for 1YAR chain A result 4

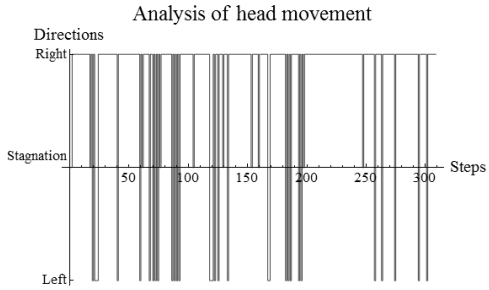

Fig. 9.45: Analysis of head movement for 1YAR chain A result 5

9.2.8 1FNT

Crystal structure of the 20S proteasome from yeast in complex with the proteasome activator PA26 from Trypanosome Brucei at 3.2 Angstroms resolution [30, 43] – chain A. Length of sequence is 252 amino acids.

Seq. 15: 1FNT chain A sequence

```
MSGAAAASAAGYDRHITIFSPEGRLYQVEYAFKATNQTNINSLAVRGKDCTVVIS
QKKVPDKLLDPTTVSYIFCISRTIGMVVNGPIPDARNAALRAKAEAAEFRYKYGY
DMPCDVLAKRMANLSQIYTQRAYMRPLGVILTFVSVDEELGPSIYKTDPAGYYVG
YKATATGPKQQEITTNLENHFKKSKIDHINEESWEKVVEFAITHMIDALGTEFSK
NDLEVGVATKDKFFTLSAENIEERLVAIAEQD
```

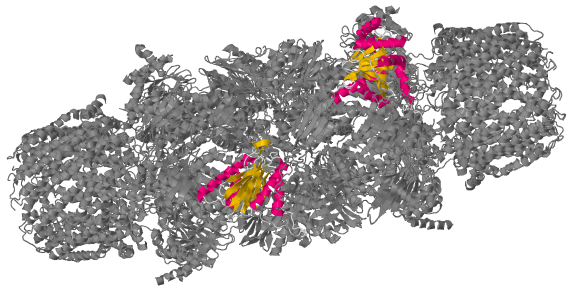

Fig. 9.46: Image of 1FNT with highlighted chain A

Seq. 16: Test sequence for 1FNT

```
KDWSRGPCDTHHWRVILPYTTINKNDGQKYSYHNQGHIHPRYMSKLCNMWESSSY
ETFAGFDAEMENVPMKRPLWQMSSISMKKWCLLPLRWDIWCFCKSAKMCENNPAA
EQRIWFTIMSEGADTKGTCCIFTPPPECCCTYMNVALPQHWASARQPNQMRGWML
NVTMVAALWESWTGIHVNFMMGHNMNYLQNPYCRAVMWKNHLPRIAEMNQHDPCF
YRIQCTLCCVYKNTMIRGYNPVITHFQLVDDT
```

Table 22: Summary of 5 obtained results of processing 1FNT chain A

Result no.	Q	F	Steps	Rate* [%]
1	$\{q_1, \dots, q_{120}\}$	$\{q_{120}\}$	597	97.62
2	$\{q_1, \dots, q_{111}\}$	$\{q_{111}\}$	537	98.02
3	$\{q_1, \dots, q_{123}\}$	$\{q_{123}\}$	562	98.81
4	$\{q_1, \dots, q_{104}\}$	$\{q_{104}\}$	460	99.21
5	$\{q_1, \dots, q_{149}\}$	$\{q_{149}\}$	616	99.60
Average successfull processing amino acids contained in sequence [%]				98.65

* It means percentual rate of all correctly processed amino acids in sequence

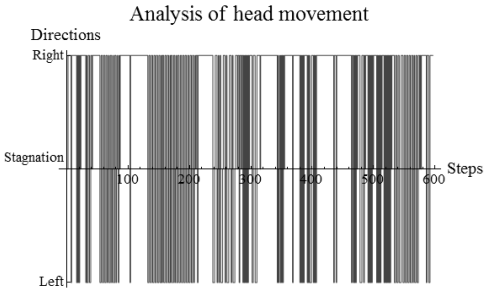

Fig. 9.47: Analysis of head movement for 1FNT chain A result 1

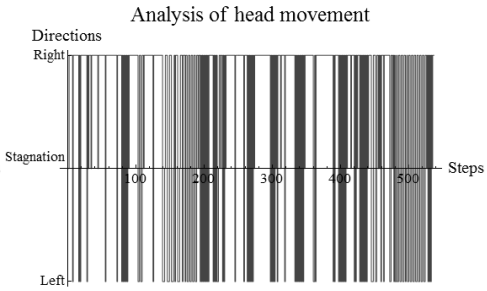

Fig. 9.48: Analysis of head movement for 1FNT chain A result 2

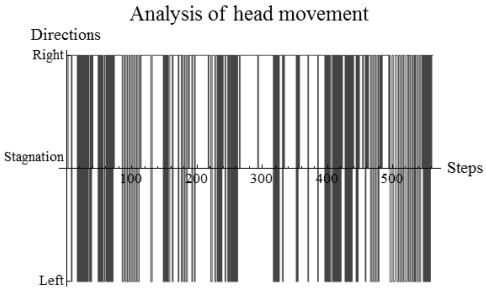

Fig. 9.49: Analysis of head movement for 1FNT chain A result 3

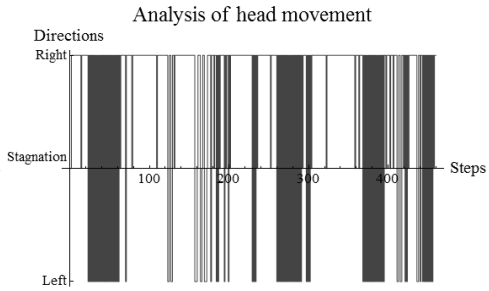

Fig. 9.50: Analysis of head movement for 1FNT chain A result 4

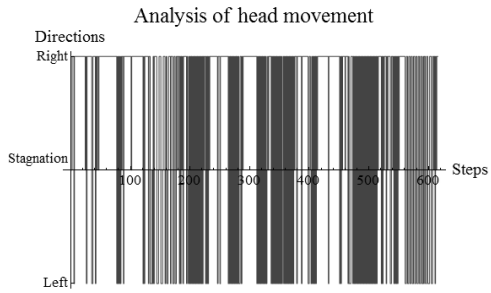

Fig. 9.51: Analysis of head movement for 1FNT chain A result 5

9.2.9 2J00

Structure of the Thermus Thermophilus 70S ribosome complexed with mRNA, tRNA and paromomycin [30, 47] – chain B. Length of sequence is 256 amino acids.

Seq. 17: 2J00 chain B sequence

```
MPVEITVKELLEAGVHFGHERKRWNPKFARYIYAERNGIHIIDLQKTMEELERTF
RFIEDLAMRGGTILFVGTKKQAQDIVRMEAERAGMPYVNQRWLGGMLTNFKTISQ
RVHRLEELEALFASPEIEERPKKEQVRLKHELERLQKYLSGFRLLKRLPDAIFVV
DPTKEAIAVREARKLFIPVIALADTDSDPDLVDYIIPGNDDAIRSIQLILSRAVD
LIIQARGGVVEPSPSYALVQEAEATETPEGESEVEA
```

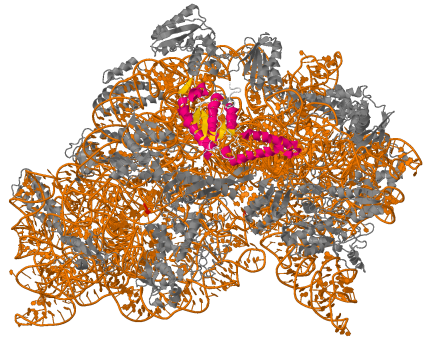

Fig. 9.52: Image of 2J00 with highlighted chain B

Seq. 18: Test sequence for 2J00

```
CSNACPLFLFYANWDAEVINSSMTKSQMPTRLPLRSLFNWLVKSPFCYGYKWHMC
MNQDGSHVSTVESCGSEMDFNDFFIDCIIYKLNPFWRYVGLKPEHTSQMGKKYQD
QRVCPTAINFYFMEIWYLKPDHNWRNWQACMIWSPFLRYGTSYYVSGEPKETDQS
TNCMFMQTIPQPKWESFMVQQQWVAYQGNNPLQCLVFKRLTYQQPYPDIKDFIYC
DHAAFIKLTMDESESATCGSWQICPDHLRQLIEVEW
```

Table 23: Summary of 5 obtained results of processing 2J00 chain B

Result no.	Q	F	Steps	Rate* [%]
1	$\{q_1, \dots, q_{88}\}$	$\{q_{88}\}$	483	99.61
2	$\{q_1, \dots, q_{80}\}$	$\{q_{80}\}$	488	98.83
3	$\{q_1, \dots, q_{50}\}$	$\{q_{50}\}$	425	98.83
4	$\{q_1, \dots, q_{97}\}$	$\{q_{97}\}$	512	98.44
5	$\{q_1, \dots, q_{82}\}$	$\{q_{82}\}$	492	98.44
Average successfull processing amino acids contained in sequence [%]				98.83

* It means percentual rate of all correctly processed amino acids in sequence

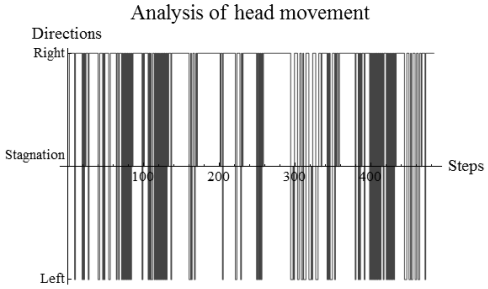

Fig. 9.53: Analysis of head movement for 2J00 chain B result 1

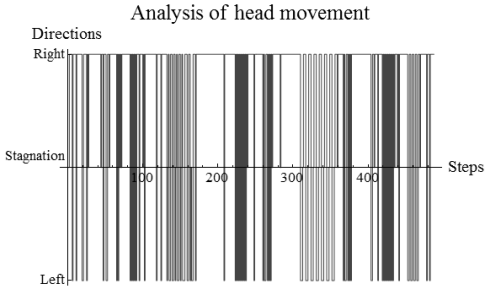

Fig. 9.54: Analysis of head movement for 2J00 chain B result 2

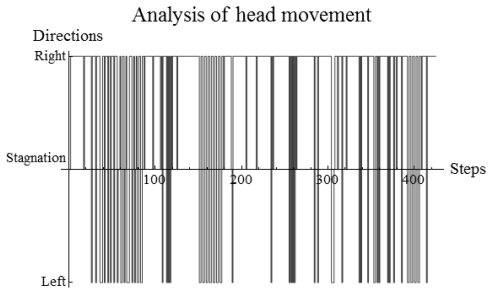

Fig. 9.55: Analysis of head movement for 2J00 chain B result 3

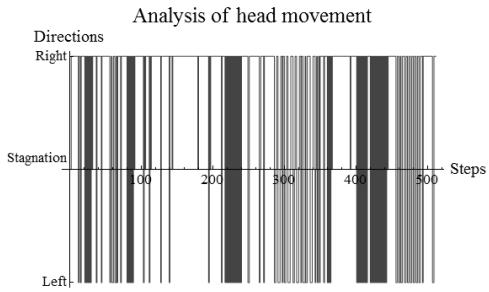

Fig. 9.56: Analysis of head movement for 2J00 chain B result 4

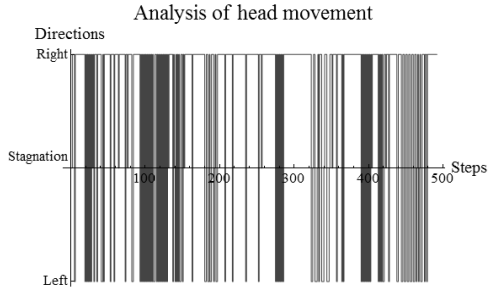

Fig. 9.57: Analysis of head movement for 2J00 chain B result 5

9.2.10 1A4Y

Ribonuclease inhibitor – angiogenin complex [30, 37] – chain A. Length of sequence is 460 amino acids.

Seq. 19: 1A4Y chain A sequence

```
SLDIQSLDIQCEELSDARWAELLPLLQQCQVVRLDDCGLTEARCKDISSALRVNP
ALAELNLRSNELGDVGVHCVLQGLQTPSCKIQKLSLQNCCLTGAGCGVLSSTLRT
LPTLQELHLSDNLLGDAGLQLLCEGLLDPQCRLEKLQLEYCSLSAASCEPLASVL
RAKPDFKELTVSNNDINEAGVRVLCQGLKDSPCQLEALKLESCGVTSDNCRDLCG
IVASKASLRELALGSNKLGDVGMAELCPGLLHPSSRLRTLWIWECGITAKGCGDL
CRVLRAKESLKELSLAGNELGDEGARLLCETLLEPGCQLESLWVKSCSFTAACCS
HFSSVLAQNRFLLELQISNNRLEDAGVRELCQGLGQPGSVLRVLWLADCDVSDSS
CSSLAATLLANHSLRELDLSNNCLGDAGILQLVESVRQPGCLLEQLVLYDIYWSE
EMEDRLQALEKDKPSLRVIS
```

Fig. 9.58: Image of 1A4Y with highlighted chain A

Seq. 20: Test sequence for 1A4Y

```
RTAHPKGFPNVQDGGNISFQMSYWHMLNSLNYYKPVGADATVFQCLPEPNFPIFH
HHKGMHHWYPFWYAINWAGCIQHLTCIKPKIEIRRIVCFQTCQWQRVEMIPHYQD
NKEGEVLWDESNSKFQALLHYIGAKEASWIFVYAMHPCTIGCHEICPSWVNAIYN
IVQHWWIAWYHDREVPSAAYCKRHTHTHHCFLSEPRGKKVPFRCMKMTIPSAEGM
GNLFLAKFDPNWNQQKGVVWQVHPAAMQIPCAGWMVINMGFPGCYTKDSKLEDKH
WTWRQWFHITHCFQSNPDVYHADDFKNFNGYIKTNFKWGNISASLMEQNMEPFIG
IIWAQGICITFMHWDITVEMRKTFEGRTYVSPCGRVVWASRRWNCAGQYSQRPPT
QMRIHCAIQFHWKINDLFYQTCTYVMHMWNCLKFSSSDMCLPNNEACCPIPDIDN
HKTGAVSPMLTWGQSLMRWP
```

Table 24: Summary of 5 obtained results of processing 1A4Y chain A

Result no.	Q	F	Steps	Rate* [%]
1	$\{q_1, \ldots, q_{166}\}$	$\{q_{166}\}$	865	98.69
2	$\{q_1, \ldots, q_{159}\}$	$\{q_{159}\}$	885	99.13
3	$\{q_1, \ldots, q_{169}\}$	$\{q_{169}\}$	928	98.69
4	$\{q_1, \ldots, q_{130}\}$	$\{q_{130}\}$	792	99.13
5	$\{q_1, \ldots, q_{147}\}$	$\{q_{147}\}$	875	99.78
Average successfull processing amino acids contained in sequence [%]				99.08

* It means percentual rate of all correctly processed amino acids in sequence

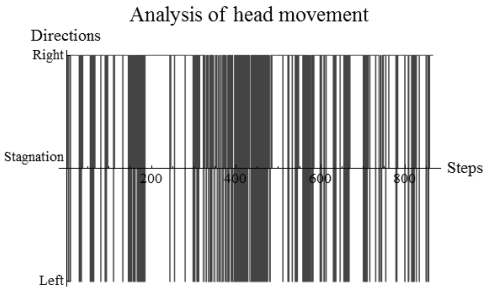

Fig. 9.59: Analysis of head movement for 1A4Y chain A result 1

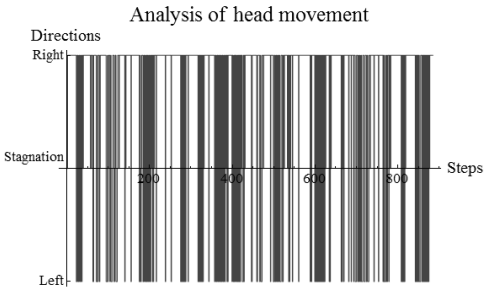

Fig. 9.60: Analysis of head movement for 1A4Y chain A result 2

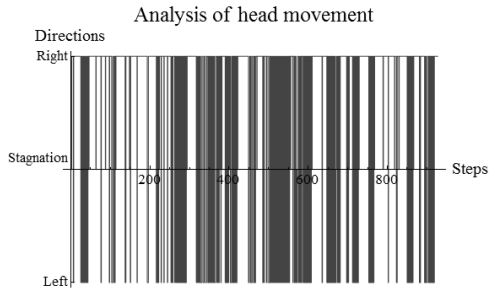

Fig. 9.61: Analysis of head movement for 1A4Y chain A result 3

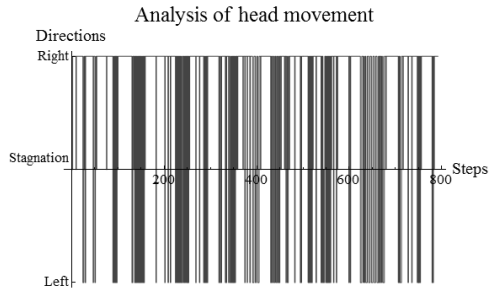

Fig. 9.62: Analysis of head movement for 1A4Y chain A result 4

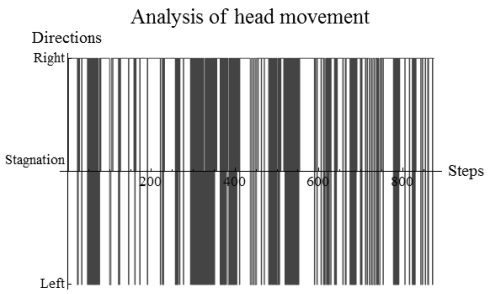

Fig. 9.63: Analysis of head movement for 1A4Y chain A result 5

9.2.11 1BMF

Bovine mitochondrial F1 - atpase [30, 42] – chain A. Length of sequence is 510 amino acids.

Seq. 21: 1BMF chain A sequence

```
QKTGTAEVSSILEERILGADTSVDLEETGRVLSIGDGIARVHGLRNVQAEEMVEF
SSGLKGMSLNLEPDNVGVVVFGNDKLIKEGDIVKRTGAIVDVPVGEELLGRVVDA
LGNAIDGKGPIGSKARRRVGLKAPGIIPRISVREPMQTGIKAVDSLVPIGRGQRE
LIIGDRQTGKTSIAIDTIINQKRFNDGTDEKKKLYCIYVAIGQKRSTVAQLVKRL
TDADAMKYTIVVSATASDAAPLQYLAPYSGCSMGEYFRDNGKHALIIYDDLSKQA
VAYRQMSLLLRRPPGREAYPGDVFYLHSRLLERAAKMNDAFGGGSLTALPVIETQ
AGDVSAYIPTNVISITDGQIFLETELFYKGIRPAINVGLSVSRVGSAAQTRAMKQ
```

VAGTMKLELAQYREVAAFAQFGSDLDAATQQLLSRGVRLTELLKQGQYSPMAIEE
QVAVIYAGVRGYLDKLEPSKITKFENAFLSHVISQHQALLGKIRTDGKISEESDA
KLKEIVTNFLAGFEA

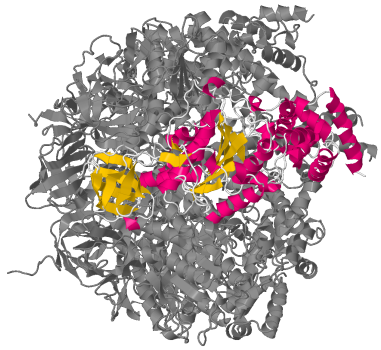

Fig. 9.64: Image of 1BMF with highlighted chain A

Seq. 22: Test sequence for 1BMF

YGWNPGYPDESYGAICYAVIHMELVNINDWFVTPIGTIREAHWNEYDLFVEAELC
RMSYCCVGMYYYTKCFVKTWRQIWPKLSTDHQYDAQCGFCFRKGWHPTTCVPTDE
RQYYHDSYRERWIAFPEMMEYPVKYPMQHEQCPSDFLLNIYHCNYWMFGCRNAFW
YWEVRFATLVMDRKAGKCRNLRCEWPSAKGKHSGHSTNRSWGIIRWGFLNRVDLE
IDMYFFFSAMSRYVCMCARACYLQSVNHTVIRMGRPYWLDHVFFEFDLHRIRDAA
YVMLRAVGCTHGWILCAPLVWWHNCSVTYYSVDMEMCIWRYRGYQLRFWTWNGIR
VATPKNHTQLRGSKNFFHRWESFKVPYHFMLWLLKESFDIGKPMMNKQTSQRNAA
LDRDVFAEPCDIQEDDRYHGSPMRHWRHEYKRDFFQSAHIYHNHNRYCRRKHLVY
HRWPTRLRHRHAPHACWDGMEWQSIRRMYGEAYKMAWCPTVCWYACRQEAVNRPR
SIMFITSDYDASWCH

Table 25: Summary of 5 obtained results of processing 1BMF chain A

Result no.	***Q***	***F***	**Steps**	**Rate* [%]**
1	$\{q_1, \dots, q_{110}\}$	$\{q_{110}\}$	814	99.41
2	$\{q_1, \dots, q_{142}\}$	$\{q_{142}\}$	934	99.02
3	$\{q_1, \dots, q_{114}\}$	$\{q_{114}\}$	872	98.63
4	$\{q_1, \dots, q_{142}\}$	$\{q_{142}\}$	912	99.22
5	$\{q_1, \dots, q_{142}\}$	$\{q_{142}\}$	952	99.22
Average successfull processing amino acids contained in sequence [%]				99.10

* It means percentual rate of all correctly processed amino acids in sequence

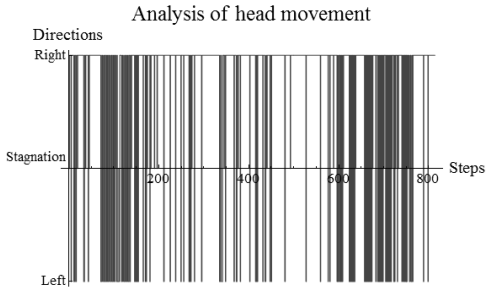

Fig. 9.65: Analysis of head movement for 1BMF chain A result 1

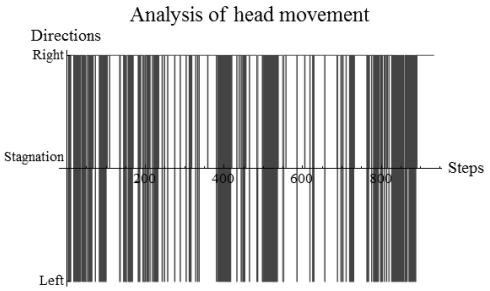

Fig. 9.66: Analysis of head movement for 1BMF chain A result 2

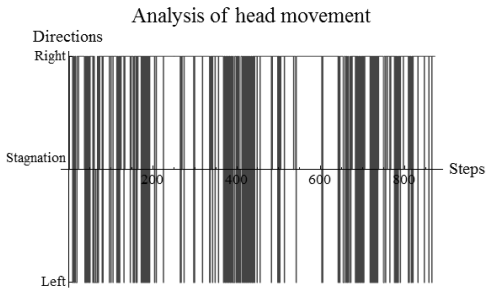

Fig. 9.67: Analysis of head movement for 1BMF chain A result 3

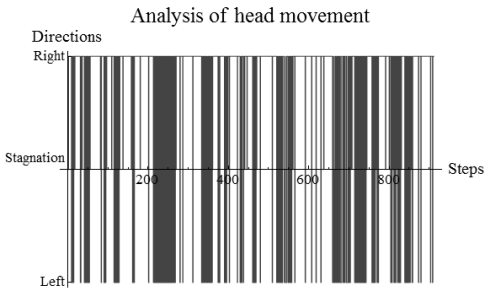

Fig. 9.68: Analysis of head movement for 1BMF chain A result 4

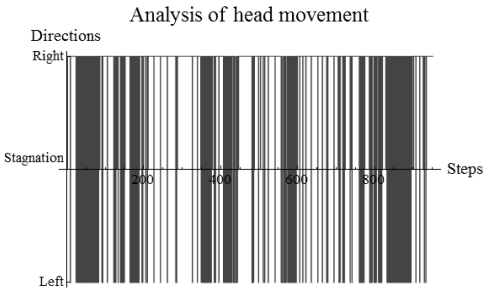

Fig. 9.69: Analysis of head movement for 1BMF chain A result 5

4.1.1. 2ZV4

The structure of rat liver vault at 3.5 Angstrom resolution [30, 49] – chain N. Length of sequence is 861 amino acids.

Seq. 23: 2ZV4 chain N sequence

```
MATEEAIIRIPPYHYIHVLDQNSNVSRVEVGPKTYIRQDNERVLFAPVRMVTVPP
RHYCIVANPVSRDTQSSVLFDITGQVRLRHADQEIRLAQDPFPLYPGEVLEKDIT
PLQVVLPNTALHLKALLDFEDKNGDKVMAGDEWLFEGPGTYIPQKEVEVVEIIQA
TVIKQNQALRLRARKECFDREGKGRVTGEEWLVRSVGAYLPAVFEEVLDLVDAVI
```

LTEKTALHLRALQNFRDLRGVLHRTGEEWLVTVQDTEAHVPDVYEEVLGVVPITT
LGPRHYCVILDPMGPDGKNQLGQKRVVKGEKSFFLQPGERLERGIQDVYVLSEQQ
GLLLKALQPLEEGESEEKVSHQAGDCWLIRGPLEYVPSAKVEVVEERQAIPLDQN
EGIYVQDVKTGKVRAVIGSTYMLTQDEVLWEKELPSGVEELLNLGHDPLADRGQK
GTAKPLQPSAPRNKTRVVSYRVPHNAAVQVYDYRAKRARVVFGPELVTLDPEEQF
TVLSLSAGRPKRPHARRALCLLLGPDFFTDVITIETADHARLQLQLAYNWHFELK
NRNDPAEAAKLFSVPDFVGDACKAIASRVRGAVASVTFDDFHKNSARIIRMAVFG
FEMSEDTGPDGTLLPKARDQAVFPQNGLVVSSVDVQSVEPVDQRTRDALQRSVQL
AIEITTNSQEAAAKHEAQRLEQEARGRLERQKILDQSEAEKARKELLELEAMSMA
VESTGNAKAEAESRAEAARIEGEGSVLQAKLKAQALAIETEAELERVKKVREMEL
IYARAQLELEVSKAQQLANVEAKKFKEMTEALGPGTIRDLAVAGPEMQVKLLQSL
GLKSTLITDGSSPINLFSTAFGLLGLGSDGQPPAQK

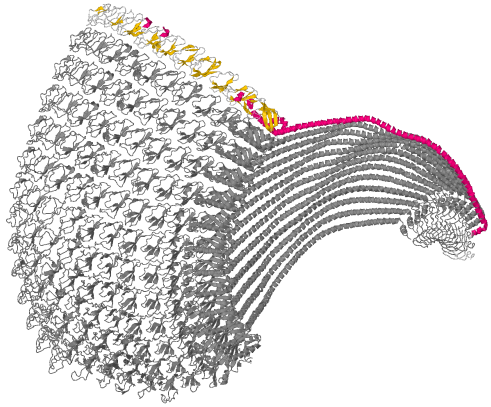

Fig. 9.70: Image of 2ZV4 with highlighted chain N

Seq. 24: Test sequence for 2ZV4

LWWFIHANHAWYGMTQNTYCKIFILWLQASCYCVVNCYQCFGCQSVHYVLIYKQF
EMVFDWIGNQRDWSARPWHQMLKTILVHKACYTCVHPEQCMTAILALMEINSLYV
NSQYLPEAHGKMLMMTEIWISMGDGDPTTCHETIEQVNSGSAEWQGYFMNPVDDA
NWAMKQMRLAKAIRTAVMSIKWVRLKLIAWDHAHNQQTSLMNPCIKLCVAYPFEI
FFEKESWVSPIYIVFRNMHWACASVCFPKEWMQDNPHPHTAIAVFYSLTLLQSNE
RKERLIGIHWCTENNESYDKIEDSGYCLWYTMYCGRQVVWEMYNTWWWTCHYYKF
GENGFTMDRQREWRYLRSGILYEGHWDWGADYSVFFFMRWSHPCGQMYGQPEMKE
RGNGFFCLTWEIYCRGQFSSLMYKNHSVRALFAIITHNCAKINVAVEHQPCFVRE
QMDGGPCCQWYLFQTGADWQPNPFCESLDKTFLEDIYNAILGEPMFSFMIRYSHA
FCQPKSWQKLQRWFTCGVISGTDSVNQQMLSNFQFMALAVKWENNREMLCMTDMN
SPQFGNSECYPCPPFSTPSARHCDVFRTMLWSKQSWHHKVMRNVGYLPNQYECFN
WNESSNAWTEHPVVKCFPPICNLHYPFFMRTNISAIGDFWDEGVQKSRFGGTLVQ
QSHNSLANFHTAMEEDCCHGGQPKNYAQQILSAVPCMTQDYKVVLHMWPYKLDHD
TLAMDVGILNPRVRLTECEFKEYYNSTYTWWWKCCKHPTDEMDPTSNMNWLFIER
YWMLIATNYIGEDYHRNPNICIDNAEGVIKVGFLGWYDLNLPWSPFGLWYVTTIC
PAQYEKPYKAEIERELMGTEMSMMEPQPPDSKSDKM

Table 26: Summary of 5 obtained results of processing 2ZV4 chain N

Result no.	Q	F	Steps	Rate* [%]
1	$\{q_1, \dots, q_{173}\}$	$\{q_{173}\}$	1410	99.54
2	$\{q_1, \dots, q_{144}\}$	$\{q_{144}\}$	1347	99.42
3	$\{q_1, \dots, q_{119}\}$	$\{q_{119}\}$	1246	99.30
4	$\{q_1, \dots, q_{150}\}$	$\{q_{150}\}$	1334	99.77
5	$\{q_1, \dots, q_{154}\}$	$\{q_{154}\}$	1252	99.65
Average successfull processing amino acids contained in sequence [%]				99.54

* It means percentual rate of all correctly processed amino acids in sequence

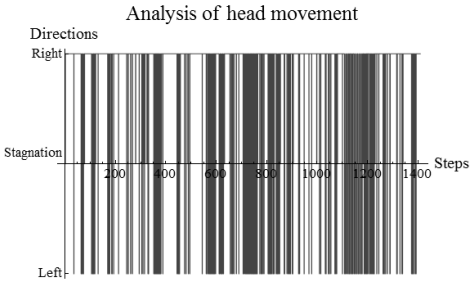

Fig. 9.71: Analysis of head movement for 2ZV4 chain N result 1

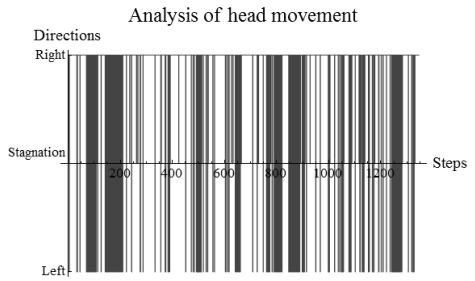

Fig. 9.72: Analysis of head movement for 2ZV4 chain N result 2

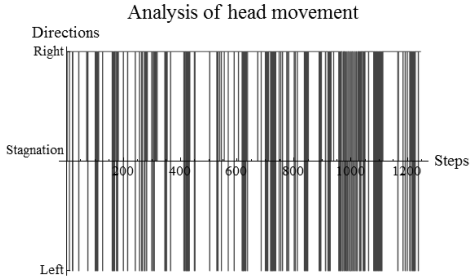

Fig. 9.73: Analysis of head movement for 2ZV4 chain N result 3

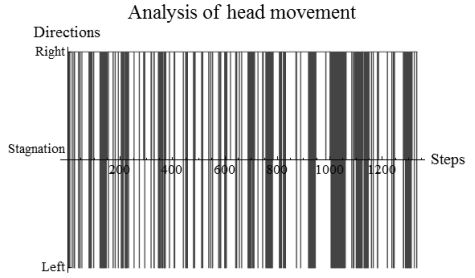

Fig. 9.74: Analysis of head movement for 2ZV4 chain N result 4

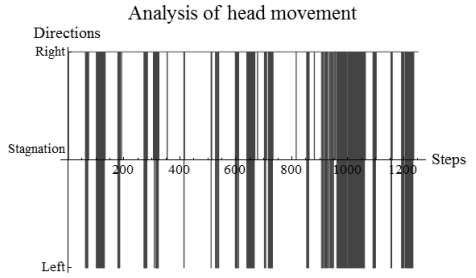

Fig. 9.75: Analysis of head movement for 2ZV4 chain N result 5

10 CONCLUSION

In this doctoral thesis the evolutionary synthesis of the rules of the Turing machine's transition function was described and introduced. Except background research which represents analysis of present utilization of artificial intelligence for programming Turing machine, doctoral thesis provides two short studies which are necessary for doctoral thesis' topic dealt with. These are study of finite automata theory focused on Turing machines and study of selected artificial intelligence methods aimed on Differential Evolution and Self-Organizing Migrating Algorithm. Thanks to mentioned studies it was possible to specify conception of Turing machine software implementation in Wolfram Mathematica and Turing machine evolutionary programming. The software implementation of Turing machine was important for the doctoral research because it was used as parts of optimization algorithms approaches. Also it was used for visualization of Turing machine behavior. The doctoral research was further aimed on outlining possibilities of using artificial intelligence for evolutionary optimization and its applications to the rules of the Turing machine's transition function. There was necessary to deal with problematics of proper representation of Turing machine rules for processing by evolutionary algorithms and conception of cost function of evolutionary algorithms. Within the scope of the doctoral research, two approaches to Turing machine evolutionary programming were designed and described in this doctoral thesis. As well as conceptions of algorithms used in cost functions of selected evolutionary algorithms there were introduced. The approaches are classical optimization and per-partes optimization. The former is suitable for estimating the rules of Turing machine when processing not very complex problems because it optimizes the complete set of rules at once. The latter estimates each rule separately thus it is also possible to obtain the rules of Turing machine when processing highly complex problems. On the previous pages there were brough the proof of proper Turing machine evolutionary programming for processing selected example problems by former of designed approaches and analysis based on above proof. The analysis was concerned with Turing machine evolutionary programming dependence on custom settings of Differential Evolution and Self-Organizing Migrating Algorithm. The analysis represents highly important part of doctoral thesis because shows influence of selected evolutionary algorithms and their settings on Turing machine evolutionary programming process. As final part of doctoral thesis conception of primary protein structures processing by evolutionary programmed Turing machine was introduced. The final part is a key part of the doctoral thesis and represents practical utilization of the topic the doctoral thesis deals with. The final part also comprehends proofs of proper Turing machine evolutionary programming for processing proteins on twelve selected primary protein structures by the latter of designed approaches.

As background reseach presented in introductory part of the doctoral thesis shown, currently it is not yet quitely common to use methods of artificial intelligence to estimation of the Turing machines rules except very little of cases. Therefore it can be presumed that approaches described in this doctoral thesis and related research papers and articles published are entirely novel and uniqe especially if Differential Evolution and Self-Organizing Migrating Algorithm are considered as optimization methods.

11 LITERATURE

[1] BERING, C. L. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 2. p. 54-56. Enzymes. ISBN 0-02-8659319.

[2] BLACK, S., The Origin of Proteins. *The American Biology Teacher*. 2004, vol. 66, no. 6, p. 408-408.

[3] DOEBLER, S. A. The Dawn of the Protein Era. *BioScience*. 2000, vol. 50, no. 1, p. 15-20.

[4] DRUMMOND, A. J., ASHTON, B., BUXTON, S., CHEUNG, M., COOPER, A., DURAN, C., FIELD, M., HELED, J., KEARSE, M., MARKOWITZ, S., MOIR, R., STONES-HAVAS, S., STURROCK, S., THIERER, T., WILSON, A., *Geneious v5.5*, 2011. Available from: http://www.geneious.com.

[5] GROVER, N. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 2. p. 7-8. Deoxyribonucleic Acid (DNA). ISBN 0-02-8659319.

[6] GROVER, N. *Chemistry: Foundations and Applications.* Macmillan Reference USA, 2004. vol. 4. p. 84-85. Ribonucleic Acid. ISBN 0-02-8659319.

[7] HOLME, T. A. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 1. p. 44-45. Amino Acid. ISBN 0-02-8659319.

[8] HOLME, T. A. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 4. p. 34-38. Proteins. ISBN 0-02-8659319.

[9] HOLME, T. A. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 4. p. 39-43. Protein Synthesis. ISBN 0-02-8659319.

[10] HOPCROFT, J. E., MOTWANI, R., ULLMAN, J. D. *Introduction to Automata Theory, Languages and Computation*. 2nd s.l., Pearson Education, 2000. ISBN 0-201-44124-1.

[11] KOURIL, L. F# Artificial Intelligence Library. In *Codeplex* [online]. © 2012 [cit. 2011-09-08]. Available from: http://fsai.codeplex.com.

[12] KOURIL, L., JASEK, R. Dependence of Evolutionary-Programming the Turing Machine on Settings of Differential Evolution. In *Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium.* Vienna, Austria, 2011, p. 1537-1538. ISBN 978-3-901509-83-4, ISSN 1726-9679.

[13] KOURIL, L., JASEK, R. Comparison of Algorithms for Evolutionary-Programming the Turing Machine. In *Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium*. Vienna, Austria, 2011, p. 1539-1540. ISBN 978-3-901509-83-4, ISSN 1726-9679.

[14] KOURIL, L., JASEK, R., MOTYL, I. A Description of the Protein Structures by Evolutionary-Programmed Turing Machine. In *Recent Advances in Signal Processing, Computational Geometry and System Theory - Proceedings of the 11th WSEAS International Conference on Signal Processing, Computational Geometry and Artificial Vision (ISCGAV '11)*. Florence, Italy, 2011, p. 278-283. ISBN 978-61804-027-5.

[15] KOURIL, L., ZELINKA, I. An Evaluative Algorithm for Per-Partes-Programming the Turing Machine. In *Proceedings of the 17th International Conference on Soft Computing MENDEL 2011*. Brno, Czech Republic, 2011, p. 30-37. ISBN 978-80-214-4302-0.

[16] KOURIL, L., ZELINKA, I. Evolutionary Synthesis of Rules for Programming the Turing Machine. *Odborny vedecky casopis Trilobit* [online]. 2010, no. 2 [cit. 2012-04-10]. ISSN 1804-1795. Available from: http://trilobit.fai.utb.cz/evolutionary-synthesis-of-rules-for-programming-a-turing-machine.

[17] KOURIL, L., ZELINKA, I. Evolutionary-Estimated Programming the Turing Machine by Differential Evolution. In *Proceedings of the 16th International Conference on Soft Computing MENDEL 2010*. Brno, Czech Republic, 2010, p. 41-48. ISBN 978-80-214-4120-0.

[18] LAMPINEN, J., ZELINKA, I. *New Ideas of Optimization*. 1st London, McGraw-Hill, 1999. p. *127-146.* Mechanical Engineering Design Optimization by Differential Evolution. ISBN 007-709506-5.
[19] MAYER, D. G., KINGHORN, B. P., ARCHER, A. A. Differential evolution - an easy and efficient evolutionary algorithm for model optimization. *Agricultural Systems.* 2005, vol. 83, no. 3, p. 315-328.
[20] NAIDOO, A., PILLAY, N. Using Genetic Programming for Turing Machine Induction. In *Proceeding EuroGP '08 Proceedings of the 11th European Conference on Genetic Programming*. Springer-Verlag, 2008, p. 350-361. ISBN 3-540-78670-8.
[21] ROBERTS-KIRCHHOFF, E. S. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 4. p. 32. Primary Structure. ISBN 0-02-8659319.
[22] SCOVELL, W. M. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 1. p. 105-107. Base Pairing. ISBN 0-02-8659319.
[23] SCOVELL, W. M. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 1. p. 237-239. Codon. ISBN 0-02-8659319.
[24] SCOVELL, W. M. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 2. p. 20-22. DNA Replication. ISBN 0-02-8659319.
[25] SCOVELL, W. M. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 3. p. 186-187. Nucleotides. ISBN 0-02-8659319.
[26] SCOVELL, W. M. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 3. p. 182-186. Nucleic Acids. ISBN 0-02-8659319.
[27] SCHWABACHER, A. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 3. p. 227. Peptide Bond. ISBN 0-02-8659319.
[28] SULLIVAN, D. M. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 2. p. 23-24. Double Helix. ISBN 0-02-8659319.
[29] SULLIVAN, D. M. *Chemistry: Foundations and Applications*. Macmillan Reference USA, 2004. vol. 4. p. 43-44. Protein Translation. ISBN 0-02-8659319.
[30] SWAMINATHAN, J., VELANKAR, S. Interesting Protein Structures. In: *Workshop Tutorials* [online]. PDBe – Protein Data Bank Europe, ©2012 [cit. 2012-04-24]. Available from: http://www.ebi.ac.uk/pdbe/docs/Tutorials/workshop_tutorials/InterestingProteinStructures.html
[31] TANOMARU, J. Evolving Turing Machines from Examples. In *Proceeding AE '97 Selected Papers from the Third European Conference on Artificial Evolution*. Springer-Verlag, 1998, p. 167-182. ISBN 3-540-64169-6.
[32] VALLEJO, E. E., RAMOS, F. Evolving Turing Machines for Biosequence Recognition and Analysis. In *Proceeding EuroGP '01 Proceedings of the 4th European Conference on Genetic Programming*. Springer-Verlag, 2001, p. 192-203. ISBN 3-560-41899-7.
[33] ZELINKA, I., OPLATKOVA, Z., SEDA, M., OSMERA, P., VCELAR, F. *Evolucni vypocetni techniky - principy a aplikace*. Praha: BEN - technicka literatura, 2008. ISBN 80-7300-218-3.
[34] ZELINKA, I. *New Optimization Techniques in Engineering*. Springer-Verlag, 2004. p. 167-218. Chapter 7. SOMA - Self-Organizing Migrating Algorithm. ISBN 3-540-20167X.
[35] ZELINKA, I. *Umela inteligence v problemech globalni optimalizace*. Praha: BEN – technicka literatura, 2002. ISBN 80-7300-069-5.
[36] ZVELEBIL M., BAUM, J. O. *Understanding Bioinformatics*. Garland Science, Taylor & Francis Group, LLC, 2008. ISBN 0-8153-4024-9.

[37] 1A4Y – Ribonuclease Inhibitor – Angiogenin Complex. *PDBe Protein Data Bank Europe* [online]. ©2012 [cit. 2012-04-27]. Available from: http://www.ebi.ac.uk/pdbe-srv/view/entry/1a4y/primary.
[38] 1AOI – Complex between Nucleosome Core Particle (H3, H4, H2A, H2B) and 146 BP Long DNA Fragment. *PDBe Protein Data Bank Europe* [online]. ©2012 [cit. 2012-04-27]. Available from: http://www.ebi.ac.uk/pdbe-srv/view/entry/1aoi/primary.
[39] 1B08 – Lung Surfactant Protein D (SP-D). *PDBe Protein Data Bank Europe* [online]. ©2012 [cit. 2012-04-27]. Available from: http://www.ebi.ac.uk/pdbe-srv/view/entry/1b08/primary.
[40] 1B09 – Human C-Reactive Protein Complexed with Phosphocholine. *RCSB Protein Data Bank* [online]. ©2012 [cit. 2012-04-24]. Available from: http://www.pdb.org/pdb/explore/explore.do?structureId=1b09.
[41] 1B09 – Human C-Reactive Protein Complexed with Phosphocholine. *PDBe Protein Data Bank Europe* [online]. ©2012 [cit. 2012-04-27]. Available from: http://www.ebi.ac.uk/pdbe-srv/view/entry/1b09/primary.
[42] 1BMF – Bovine Mitochondrial F1 – Atpase. *PDBe Protein Data Bank Europe* [online]. ©2012 [cit. 2012-04-27]. Available from: http://www.ebi.ac.uk/pdbe-srv/view/ entry/1bmf/primary.
[43] 1FNT – Crystal Structure of the 20S Proteasome from Yeast in Complex with the Proteasome Activator PA26 from Trypanosome Brucei at 3.2 Angstroms Resolution. *PDBe Protein Data Bank Europe* [online]. ©2012 [cit. 2012-04-27]. Available from: http://www.ebi.ac.uk/pdbe-srv/view/entry/1fnt/primary.
[44] 1LIT – Human Lithostathine. *PDBe Protein Data Bank Europe* [online]. ©2012 [cit. 2012-04-27]. Available from: http://www.ebi.ac.uk/pdbe-srv/view/entry/1lit/primary.
[45] 1TUP – Tumor Supressor P53 Complexed with DNA. *PDBe Protein Data Bank Europe* [online]. ©2012 [cit. 2012-04-27]. Available from: http://www.ebi.ac.uk/pdbe-srv/view/entry/1tup/primary.
[46] 1YAR – Structure of Archeabacterial 20S Proteasome Mutant D9S – PA26 Complex. *PDBe Protein Data Bank Europe* [online]. ©2012 [cit. 2012-04-27]. Available from: http://www.ebi.ac.uk/pdbe-srv/view/entry/1yar/primary.
[47] 2J00 – Structure of the Thermus Themrophilus 70S Ribosome Complexed with MRNA, TRNA and Paromomycin (Part 1 of 4). *PDBe Protein Data Bank Europe* [online]. ©2012 [cit. 2012-04-27]. Available from: http://www.ebi.ac.uk/pdbe-srv/view/entry/2j00/primary.
[48] 2J01 – Structure of the Thermus Thermophilus 70S Ribosome Complexed with MRNA, TRNA and Paromomycin (Part 2 of 4). *PDBe Protein Data Bank Europe* [online]. ©2012 [cit. 2012-04-27]. Available from: http://www.ebi.ac.uk/pdbe-srv/view/entry/2j01/primary.
[49] 2ZV4 – The Structure of Rat Liver Vault at 3.5 Angstrom Resolution. *PDBe Protein Data Bank Europe* [online]. ©2012 [cit. 2012-04-27]. Available from: http://www.ebi.ac.uk/pdbe-srv/view/entry/2zv4/primary.
[50] Microsoft F# Developer Center [online]. ©2012 [cit. 2012-02-03]. Available from: http://www.fsharp.net.
[51] Turing Machines implemented in JavaScript. *The Alan Turing Internet Scrapbook* [online]. [Cit. 2011-04-15]. Available from: http://www.turing.org.uk/turing/scrapbook/tmjava.html.

12 PUBLICATIONS

12.1 Conference proceedings and journals

1. KOURIL, L., POSPISILIK, M., ADAMEK, M., JASEK, R. Application of Differential Evolution for Audio Transformers Optimization. *Int. J. of Circuits, Systems and Signal Processing*. 2012, vol. 6, no. 3, p. 231-240. ISSN 1998-4464.
2. KOURIL, L., POSPISILIK, M., ADAMEK, M., JASEK, R. Flat Coil Optimizer in the Meaning to Coil Optimization. *Int. J. of Circuits, Systems and Signal Processing*. 2012, vol. 6, no. 4, p. 241-248. ISSN 1998-4464.
3. POSPISILIK, M., KOURIL, L., ADAMEK, M., ZELINKA, I., JASEK, R. Custom Winding Ratio Analysis of Evolutionary Optimized Audio Transformer. In *Proceedings of International Conference on Prediction, Modeling and Analysis of Complex Systems NOSTRADAMUS 2012*. Ostrava, Czech Republic, 2012. Accepted for publication.
4. POSPISILIK, M., KOURIL, L., OTAHAL, J., ADAMEK, M. Proposal on Intelligent Wearable Sensor Suit. In *Proceedings of the 16th WSEAS International Conference on Circuits and Systems*. Kos, Greece, 2012, p. 564-567. ISBN 978-1-61804-108-1.
5. POSPISILIK, M., KOURIL, L., ADAMEK, M., ZELINKA, I., JASEK, R. SOMA-Based Audio Transformers Optimization. In *Proceedings of the 18th International Conference on Soft Computing MENDEL 2012*. Brno, Czech Republic, 2012, p. 326-331. ISBN 978-80-214-4540-6.
6. POSPISILIK, M., KOURIL, L., ADAMEK, M. Contactless ECG Scanning Device Hardware Design Proposal. In *Proceedings of the XX IMEKO World Congress*. Busan, Republic of Korea, 2012. Accepted for publication.
7. KOURIL, L., POSPISILIK, M., ADAMEK, M., JASEK, R. Audio Transformers Optimization by Means of Evolutionary Algorithm. In *Proceedings of 5th WSEAS World Congress on Applied Computing Conference*. Faro, Portugal, 2012, p. 133-138. ISBN 978-1-61804-089-3.
8. KOURIL, L. POSPISILIK, M. ADAMEK, M., JASEK, R. Coil Optimization with Aid of Flat Coil Optimizer. In *Proceedings of 5th WSEAS World Congress on Applied Computing Conference*. Faro, Portugal, 2012, p. 124-127. ISBN 978-1-61804-089-3.
9. KOURIL, L., JASEK, R. Dependence of Evolutionary-Programming the Turing Machine on Settings of Differential Evolution. In *Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium*. Vienna, Austria, 2011, p. 1537-1538. ISBN 978-3-901509-83-4. ISSN 1726-9679.
10. KOURIL, L., JASEK, R. Comparison of Algorithms for Evolutionary-Programming the Turing Machine Implemented in Wolfram Mathematica and Microsoft .NET Framework. In *Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium*. Vienna, Austria, 2011, p. 1539-1540. ISBN 978-3-901509-93-4. ISSN 1726-9679.
11. POSPISILIK, M., KOURIL, L., ADAMEK, M. Comparison of Diferential Evolution and Self-Organising Migrating Algorithm at Flat Inductor Optimisation. In *Proceedings of the XIII Annual International Conference Internet, Competiveness and Organizational Security – Process Management and the Use of Modern Technologies*. Zlin, Czech Republic, 2011, p. 323-327. ISBN 978-80-7454-012-7.
12. POSPISILIK, M., KOURIL, L., ADAMEK, M. Approach to Electrical Circuitry Designing by Employing Differential Evolution. In *Proceedings of the XIII Annual International Conference Internet, Competitiveness and Organizational Security – Process Management and the Use of Modern Technologies*. Zlin, Czech Republic, 2011, p. 328-330. ISBN 978-80-7454-012-7.

13. KOURIL, L., JASEK, R., MOTYL, I. A Description of the Protein Structures by Evolutionary-Programmed Turing Machine. In *Recent Advances in Signal Processing, Computational Geomatry and System Theory – Proceedings of the 11th WSEAS International Conference on Signal Processing, Computational Geometry and Artificial Vision (ISCGAV '11)*. Florence, Italy, 2011, p. 278-283. ISBN 978-1-61804-027-5.
14. POSPISILIK, M., KOURIL, L., MOTYL, I., ADAMEK, M. Single and Double Layer Spiral Planar Inductors Optimisation with the Aid of Self-Organising Migrating Algorithm. In *Recent Advances in Signal Processing, Computational Geometry and System Theory – Proceedings of the 11th WSEAS International Conference on Signal Processing, Computational Geometry and Artificial Vision (ISCGAV '11)*. Florence, Italy, 2011, p. 272-277. ISBN 978-1-61804-027-5.
15. KOURIL, L., ZELINKA, I. An Evaluative Algorithm for Evolutionary-Estimated Per-Partes-Programming the Turing Machine. In *Proceedings of the 17th International Conference on Soft Computing MENDEL 2011*. Brno, Czech Republic, 2011, p. 30-37. ISBN 978-80-214-4302-0.
16. POSPISILIK, M., KOURIL, L., ADAMEK, M. Planar Inductor Optimised by Evolutionary Algorithm. In *Proceedings of the 17th International Conference on Soft Computing MENDEL 2011*. Brno, Czech Republic, 2011, p. 38-43. ISBN 978-80-214-4302-0.
17. KOURIL, L., ZELINKA, I. Evolutionary Synthesis of Rules for Programming the Turing Machine. *Odborny vedecky casopis Trilobit* [online]. 2010, no. 2 [cit. 2012-04-10]. ISSN 1804-1795. Available from: http://trilobit.fai.utb.cz/evolutionary-synthesis-of-rules-for-programming-a-turing-machine
18. KOURIL, L., ZELINKA, I. Evolutionary-Estimated Programming the Turing Machine by Differential Evolution. In *Proceedings of the 16th International Conference on Soft Computing MENDEL 2010*. Brno, Czech Republic, 2010, p. 41-48. ISBN 978-80-214-4120-0.
19. POSPISILIK, M., KOURIL, L., ADAMEK, M. Employing Self-Organizing Migrating Algorithm at Electrical Circuitry Designing. In *Proceedings of the 16th International Conference on Soft Computing MENDEL 2010*. Brno, Czech Republic, 2010. ISBN 978-80-214-4120-0.
20. KOURIL, L., ZELINKA, I. Moznosti predikce aminokyselinovych sekvenci v proteinech. In *FOBIA Conference 2009 – Book ofAbstracts*. Zlin, Czech Republic.
21. PROCHAZKA, M., KOURIL, L., ZELINKA, I. Classification and Prediction by Decision Trees and Neural Networks. In *Proceedings of the 15th International Conference on Soft Computing MENDEL 2009*. Brno, Czech Republic, 2009. ISBN 978-80-214-3884-2.

12.2 Other publications

1. HECZKO, M., KOURIL, L., MALANIK, D. Disease-Simulation Environment. In *Ceske narodni finale Imagine Cup 2009 – Sbornik souteznich praci*. Praha, Czech Republic, 2009.

12.3 Software

1. POSPISILIK, M., KOURIL, L. Flat Coil Optimizer. Tomas Bata University in Zlin, Faculty of Applied Informatics, Zlin, 2011.

12.4 Supervised or consulted diploma thesis

1. STUSAK, V. Rizeni pohybu autonomniho robota pomoci umele inteligence, MSc. Project, 2012.
2. STUSAK, V. Testovaci prostredi pro vyuku kurzu Microsoft IT Academy na UTB ve Zline, Bc. Project, 2011.

12.5 Internal Grant Agency projects

1. KOURIL, L., ZELINKA, I. IGA/FAI/2012/053 Aplikace evolucnich algoritmu pri navrhu a optimalizaci elektrickych obvodu, 2012.
2. KOURIL, L., LEBEDIK, A., ZELINKA, I. IGA/42/FAI/10/D Evolucni synteza biomolekularnich struktur, 2010 – 2011.

13 AUTHOR'S CURRICULUM VITAE

Education

- Tomas Bata University in Zlin, Czech Republic, Faculty of Applied Informatics
  Master's study (2008)
  Program: Engineering Informatics
  Diploma thesis: „Computer Viruses and Artificial Intelligence"
  Diploma thesis was awarded by 3rd prize at international student contest STOC 2008
- Tomas Bata University in Zlin, Czech Republic, Faculty of Applied Informatics
  Bachelor's study (2006)
  Program: Engineering Informatics
  Bachelor thesis: „Comparing of Web Technologies ASP, ASP.NET and PHP"

Certifications

- Microsoft Certified Technology Specialist
  70-620: Configuring Microsoft Windows Vista Client
- Microsoft Certified Professional
  70-270: Installing, Configuring and Administering Microsoft Windows XP Professional

Professional student activities

- Microsoft Student Partner
  Tomas Bata University in Zlin, Czech Republic
- Participation in student contest „Nejlepsi podnikatelsky zamer Zlinskeho kraje 2011/2012"
  Bussiness plan was awarded as the 3rd best.
- Participation in international student contest Imagine Cup 2009
  Project SimuLIVE – The Desease-Simulation Environment was awarded by 2nd prize at national finals, Czech Republic
- Lectures focused on software development
  Tomas Bata University in Zlin, Czech Republic

Awards

- MCTS: Microsoft Windows Vista, Configuration Charter Member
- Certificate of recognition for contribution to the Czech Microsoft Research's Machine Translation Engine Evaluation for Visual Studio 2010.

Research activities

- Principal researcher
  1. 2011: Project IGA/FAI/2012/053 „Aplikace evolucnich algoritmu pri navrhu a optimalizaci elektrickych obvodu"
  2. 2010-2011: Project IGA/42/FAI/10/D „Evolucni synteza biomolekularních struktur"
- Member of project team
  3. 2011-2012: Project CEBIA-Tech No. CZ.1.05/2.1.00/03.0089. 2nd reseach team „Security research"

APPENDICES

Appendix A: Examples of selected rules estimated during analysis
Appendix B: Description of Turing machine software implementation in Wolfram Mathematica

Appendix A: Examples of selected rules estimated during analysis

- Unary addition

$\delta(q_1, "\#") = (q_2, "\#", 0)$
$\delta(q_1, "1") = (q_3, "1", 1)$
$\delta(q_2, "\#") = (q_3, "\#", 1)$
$\delta(q_2, "1") = (q_4, "1", 1)$
$\delta(q_3, "\#") = (q_4, "1", 1)$
$\delta(q_3, "1") = (q_1, "1", 1)$
$\delta(q_4, "\#") = (q_5, "\#", -1)$
$\delta(q_4, "1") = (q_1, "\#", -1)$
$\delta(q_5, "\#") = (q_6, "\#", -1)$
$\delta(q_5, "1") = (q_5, "\#", 1)$

- Divisibility

$\delta(q_1, "\#") = (q_1, "\#", 1)$
$\delta(q_1, "1") = (q_4, "1", 0)$
$\delta(q_1, "X") = (q_1, "1", 0)$
$\delta(q_2, "\#") = (q_7, "1", 1)$
$\delta(q_2, "1") = (q_1, "1", 0)$
$\delta(q_2, "X") = (q_8, "\#", 0)$
$\delta(q_3, "\#") = (q_3, "1", 0)$
$\delta(q_3, "1") = (q_8, "X", 0)$
$\delta(q_3, "X") = (q_5, "X", 0)$
$\delta(q_4, "\#") = (q_1, "1", 1)$
$\delta(q_4, "1") = (q_5, "1", 1)$
$\delta(q_4, "X") = (q_1, "\#", -1)$
$\delta(q_5, "\#") = (q_2, "1", 0)$
$\delta(q_5, "1") = (q_3, "1", 1)$
$\delta(q_5, "X") = (q_4, "X", 1)$
$\delta(q_6, "\#") = (q_1, "X", 1)$
$\delta(q_6, "1") = (q_4, "X", 0)$
$\delta(q_6, "X") = (q_7, "\#", -1)$
$\delta(q_7, "\#") = (q_5, "\#", 1)$
$\delta(q_7, "1") = (q_4, "\#", 0)$
$\delta(q_7, "X") = (q_5, "1", 0)$

- Primality

$\delta(q_1, "\#") = (q_{20}, "X", 0)$
$\delta(q_1, "1") = (q_{15}, "X", 1)$
$\delta(q_1, "X") = (q_4, "X", 1)$
$\delta(q_2, "\#") = (q_{15}, "X", 0)$
$\delta(q_2, "1") = (q_{13}, "1", -1)$
$\delta(q_2, "X") = (q_{25}, "\#", 0)$
$\delta(q_3, "\#") = (q_{15}, "\#", -1)$
$\delta(q_3, "1") = (q_{15}, "1", 0)$
$\delta(q_3, "X") = (q_{25}, "X", 0)$
$\delta(q_4, "\#") = (q_{21}, "X", -1)$
$\delta(q_4, "1") = (q_{19}, "\#", -1)$
$\delta(q_4, "X") = (q_{17}, "\#", -1)$
$\delta(q_5, "\#") = (q_{21}, "X", -1)$
$\delta(q_5, "1") = (q_{11}, "X", 1)$
$\delta(q_5, "X") = (q_{22}, "\#", -1)$
$\delta(q_{11}, "\#") = (q_{25}, "1", -1)$
$\delta(q_{11}, "1") = (q_{22}, "1", 0)$
$\delta(q_{11}, "X") = (q_3, "\#", -1)$
$\delta(q_{12}, "\#") = (q_{10}, "1", 0)$
$\delta(q_6, "\#") = (q_{18}, "X", 0)$
$\delta(q_6, "1") = (q_{16}, "\#", -1)$
$\delta(q_6, "X") = (q_{20}, "X", 1)$
$\delta(q_7, "\#") = (q_{15}, "X", -1)$
$\delta(q_7, "1") = (q_{20}, "1", -1)$
$\delta(q_7, "X") = (q_{10}, "1", 0)$
$\delta(q_8, "\#") = (q_{19}, "1", -1)$
$\delta(q_8, "1") = (q_{10}, "\#", 0)$
$\delta(q_8, "X") = (q_{13}, "\#", 0)$
$\delta(q_9, "\#") = (q_{14}, "X", 0)$
$\delta(q_9, "1") = (q_{13}, "1", 0)$
$\delta(q_9, "X") = (q_9, "1", 0)$
$\delta(q_{10}, "\#") = (q_{14}, "\#", 0)$
$\delta(q_{10}, "1") = (q_{16}, "1", -1)$
$\delta(q_{10}, "X") = (q_5, "1", 1)$
$\delta(q_{18}, "\#") = (q_{17}, "X", 0)$
$\delta(q_{18}, "1") = (q_1, "\#", 1)$
$\delta(q_{18}, "X") = (q_7, "X", 1)$
$\delta(q_{19}, "\#") = (q_3, "\#", -1)$

$\delta(q_{12}, "1") = (q_{12}, "\#", -1)$
$\delta(q_{12}, "X") = (q_9, "X", 0)$
$\delta(q_{13}, "\#") = (q_{11}, "X", -1)$
$\delta(q_{13}, "1") = (q_{11}, "\#", -1)$
$\delta(q_{13}, "X") = (q_5, "X", 1)$
$\delta(q_{14}, "\#") = (q_8, "\#", 1)$
$\delta(q_{14}, "1") = (q_4, "X", 0)$
$\delta(q_{14}, "X") = (q_{12}, "1", 0)$
$\delta(q_{15}, "\#") = (q_3, "X", 1)$
$\delta(q_{15}, "1") = (q_{18}, "\#", 1)$
$\delta(q_{15}, "1") = (q_2, "\#", -1)$
$\delta(q_{16}, "X") = (q_{22}, "1", -1)$
$\delta(q_{16}, "1") = (q_{13}, "1", -1)$
$\delta(q_{16}, "X") = (q_7, "X", 1)$
$\delta(q_{17}, "\#") = (q_5, "X", -1)$
$\delta(q_{17}, "1") = (q_5, "X", -1)$
$\delta(q_{17}, "X") = (q_{22}, "\#", 1)$

$\delta(q_{19}, "1") = (q_9, "X", 0)$
$\delta(q_{19}, "X") = (q_{23}, "1", 1)$
$\delta(q_{20}, "\#") = (q_{10}, "1", -1)$
$\delta(q_{20}, "1") = (q_{10}, "\#", -1)$
$\delta(q_{20}, "X") = (q_{15}, "\#", 0)$
$\delta(q_{21}, "\#") = (q_4, "1", -1)$
$\delta(q_{21}, "1") = (q_{12}, "X", 0)$
$\delta\ (q_{21}, "X") = (q_{10}, "\#", 0)$
$\delta(q_{22}, "\#") = (q_7, "X", 1)$
$\delta(q_{22}, "1") = (q_4, "X", 1)$
$\delta(q_{22}, "X") = (q_{11}, "1", 1)$
$(q_{23}, "\#") = (q_6, "X", 0)$
$\delta(q_{23}, "1") = (q_{15}, "1", 0)$
$\delta(q_{23}, "X") = (q_{18}, "1", -1)$
$\delta(q_{24}, "\#") = (q_4, "\#", 1)$
$\delta(q_{24}, "1") = (q_{14}, "\#", 1)$
$\delta(q_{24}, "X") = (q_5, "\#", 1)$

- Proteins

  These rules are not published in the doctoral thesis because of their extensiveness.

Appendix B: Description of Turing machine software implementation

Turing machine is implemented as Turing machine simulator (see figure below) in Wolfram Mathematica. It is created as CDF[4] (Computable Document Format) file thus it can be run in free Wolfram CDF Player[5] or can be placed at web pages thanks to CDF plugin for web browsers.

Turing machine simulator consists of two parts. The first part located at the top of application window contains input fields for Turing machine settings as are number of innter states, data tape symbols, initial state, accepting state, initial head position, data tape, and rules of transition function. If Run button is pressed, the simulation will proceed. The Step slider allows manual moving the head operations of the Turing machine.

The second part of the application windows located at bottom contains information on current step and rule. There are also included preview of output data tape with current position of Turing machine's head and graph which depicts head movement.

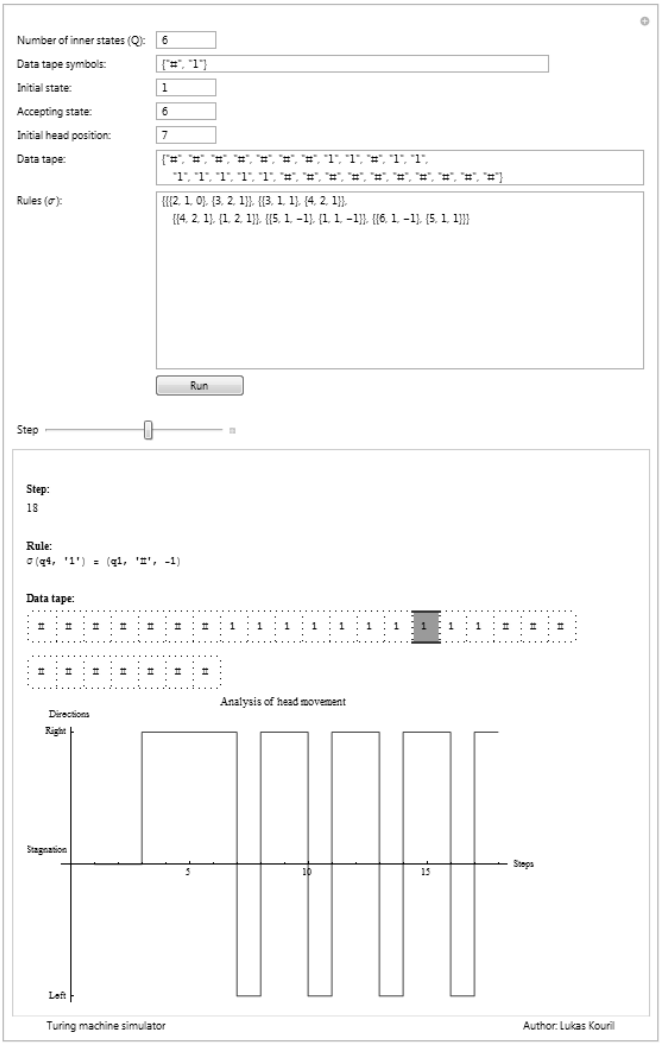

Fig.: Application window of Turing machine simulator.

[4] *http://www.wolfram.com/cdf/*
[5] *http://www.wolfram.com/cdf-player/*