University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Anna Gogolińska

Algorithms Inspired by Petri Nets in Modeling of Complex Biological Systems

PhD dissertation

Supervisor

prof. dr hab. Wiesław Nowak

Faculty of Physics, Astronomy and Informatics
Nicolaus Copernicus University in Toruń

May, 2015

Author's declaration:
aware of legal responsibility I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

....
date *Anna Gogolińska*

Supervisor's declaration:
the dissertation is ready to be reviewed

....
date *prof.dr hab. Wiesław Nowak*

Algorithms Inspired by Petri Nets in Modeling of Complex Biological Systems

Abstract

In this dissertation new tools for the fields of bioinformatics, data mining and operational research are developed. New algorithms were inspired by Petri nets. Petri nets (PNs) belong to mathematical modeling languages and are used in science and technology. The PNs typically have a form of a bipartite graph with two kinds of nodes: places and transitions, but they may be represented as matrices as well. The aims of the thesis were to exploit the properties and to extend the applications of PN in a new area of modeling of complex biological systems. Apart from the applications of classical, timed and priority-based PNs, a new type of the PNs – the random priority-based Petri nets - has been proposed for the first time. In those networks transitions are fired randomly and the probability of firing is proportional to the priority of the transition. This property is strongly desirable in the molecular dynamics simulations (MD) considered here. Moreover, the priority-based networks were tailored to aimed applications.

A new, improved, PN based model of the immune system (IS) has been developed. The t-invariant analysis of the model has been performed. All t-invariant, t-clusters and MCT sets were found, and their biological meanings were identified. The model is correct, since it has the CTI property – every transition is a part of at least one t-invariant. Selected phenomena and diseases were added to the model, such as fever, ageing, infection of the HIV virus, Adult-Onset Immunodeficiency Syndrome disease and Autism Spectrum Disorders. The responses of the IS to such external stimuli were modeled. Particularly pioneering is the study of the correlation between fever and autism.

A possibility of parallelization in PN studies is discussed. A novel algorithm for parallel Petri net simulations using the CUDA/GPU technology has been developed and tested. Our algorithm outperforms classical ones when used for very large PNs.

In order to generate adequate biological data, sets MD computer simulation study of two complexes were performed: the grass pollen and its antibody and the chemokine MCP-1 and its antibody. Both classical MD and steered MD trajectories were calculated. Simulations substantiated the validity of certain experimental techniques based on Atomic Force Microscopy. Huge sets ($>10^6$) of structural data were further classified using our new Petri networks. MD trajectories are difficult to analyze. Therefore, completely new groups of methods of an MD trajectory analysis were formulated. Three algorithms were designed: OPOA (one place one atom/amino acid), OPOC (one place one conformation) and CON (contact algorithm) and their computational time complexity were analyzed. All three algorithms can generate classical, timed, priority-based or random priority-based Petri nets. In the thesis many biological examples of the generated PNs and their interpretations are presented. Such results have never been proposed before.

Our studies show that PN formalism can be a powerful tool useful in bioinformatics and, in particular, in MD simulations analysis.

Keywords: Petri nets, computer modeling, algorithms, molecular dynamics analysis, immune system, concurrent algorithms, GPU, CUDA, data mining, clustering algorithms

ACM classification:
Software and its engineering → Software system models → Petri nets
Theory of computation → Design and analysis of algorithms
Applied computing → Life and medical sciences → Bioinformatics
Applied computing → Life and medical sciences → Computational biology → Molecular structural biology

Algorytmy oparte o formalizm sieci Petriego w modelowaniu złożonych systemów biologicznych

Abstract

W rozprawie wprowadzone i opisane są nowe narzędzia obliczeniowe z dziedziny bioinformatyki, wydobywania danych oraz badań operacyjnych. Powstałe algorytmy oparto o formalizm sieci Petriego. Sieci Petriego należą do matematycznych języków modelowania, są używane w nauce i technice do reprezentowania oraz analizy skomplikowanych układów. Sieci te mają formę grafu dwudzielnego z dwoma typami wierzchołków: miejscami i tranzycjami, bywają reprezentowane jako macierze. Celem rozprawy było zbadanie możliwości nowego wykorzystania sieci Petriego w biologii obliczeniowej czy bioinformatyce strukturalnej oraz poszerzenie ich zastosowań o modelowanie złożonych systemów biologicznych. Poza sieciami klasycznymi, sieciami z czasem i z priorytetami, użyty został, sformułowany na potrzeby tych badań, nowy typ sieci Petriego – sieci losowo-priorytetowe. W sieciach tych prawdopodobieństwo realizacji tranzycji jest wprost proporcjonalne do jej priorytetu. Taka własność jest konieczna w modelowaniu trajektorii dynamiki molekularnej (MD) rozważanych w części rozprawy. Ponadto typowe sieci priorytetowe zostały zmodyfikowane pod kątem zastosowań w modelowaniu MD.

Przy użyciu sieci Petriego stworzono ulepszony komputerowy model układu odpornościowego, który został następnie poddany tzw. analizie t-niezmienników. Wszystkie t-niezmienniki, t-klastry i zbiory MCT zostały w tym modelu zidentyfikowane i określono ich znaczenie biologiczne. Model ma własność CTI – każda tranzycja wchodzi w skład przynajmniej jednego t-niezmiennika, co potwierdza jego poprawność. Do modelu dodano modyfikacje kodujące wybrane zjawiska i choroby: wpływ gorączki, starzenie, infekcja choroby AIDS i Adult-Onset Immunodeficiency Syndrome (AOIS) oraz autyzm. Poprzez odpowiednie symulacje zbadano wpływ tych czynników na zachowanie się nowego modelu układu odpornościowego. Szczególnie pionierskie są przedstawione tu badania związku między gorączką a autyzmem.

W rozprawie dyskutowany jest problem zrównoleglenia symulacji dynamiki sieci Petriego. Opracowano i przetestowano nowy algorytm symulacji z wykorzystaniem technologii CUDA/GPU. Testy pokazały, że dla dużych sieci algorytm ten jest znacznie wydajniejszy niż klasyczny.

W celu wygenerowania testowych danych przeprowadzono symulacje dynamiki molekularnej (z ang. *molecular dynamics* MD) dwóch kompleksów białkowych: pyłku trawy z przeciwciałem (alergie) oraz chemokiny MCP-1 z przeciwciałem (autyzm). Wykonano zarówno klasyczne symulacje MD jak i symulacje metodą sterowanej MD. Wyniki potwierdzają potencjalną użyteczność nowych eksperymentalnych metod diagnostycznych opartych na mikroskopii sił atomowych. Własne zbiory danych strukturalnych zostały przeanalizowane w ramach nowo zaproponowanego podejścia do analizy danych opartego o sieci Petriego. Trajektorie MD są na ogół trudne do interpretacji, dlatego też zaproponowano trzy dedykowane algorytmy generowania sieci Petriego na podstawie trajektorii MD nazwane: OPOA (jedno miejsce - jeden atom/aminokwas), OPOC (jedno miejsce - jedna konformacja) oraz CON (algorytm śledzenia kontaktów). Wszystkie trzy algorytmy mogą generować różne sieci, m.in.: klasyczne sieci Petriego, sieci z czasem, sieci priorytetowe oraz losowo-priorytetowe. W rozprawie przedstawiono przykłady wygenerowanych sieci wraz z ich analizą i interpretacją. Taka metoda analizy masywnych danych MD ma szereg zalet, a jak dotąd nigdy nie była używana.

Przeprowadzone badania pokazują, że formalizm sieci Petriego może być potężnym narzędziem w bioinformatyce, a szczególności w analizie wyników symulacji dynamiki molekularnej.

Słowa kluczowe: sieci Petriego, modelowanie komputerowe, algorytmy, analiza dynamiki molekularnej, układ odpornościowy, algorytmy równoległe, GPU, CUDA, data mining, algorytmy klasteryzujące

ACM classification:
Oprogramowanie i jego inżynieria → Modele systemów oprogramowania → Sieci Petriego
Teoria obliczeń → Projektowanie i analiza algorytmów
Informatyka Stosowana →Życie i nauki medyczne →Bioinformatyka
Informatyka Stosowana →Życie i nauki medyczne →Biologia obliczeniowa
→Molekularna biologia strukturalna

Acknowledgements

My work was financially supported by grants: N202 262038 (Ministry of Science and Higher Education), N519 578138 (Ministry of Science and Higher Education) and "Krok w przyszłość V" (kujawsko-pomorskie Voivodeship).

I would like to express gratitude to my parents, who have supported me a lot. Without them the preparation of this thesis would not be possible.

I would like to thank my supervisor prof. dr hab. Wiesław Nowak and colleagues from the Faculty of Physics, Astronomy and Informatics: mgr inż. Marcin Dąbrowski, mgr inż. Rafał Jakubowski, dr Karolina Mikulska, dr Łukasz Pepłowski, mgr inż. Jakub Rydzewski for they support and advices.

Niniejsza praca praca była wspierana finansowo przez granty: N202 262038 (Ministerstwo Nauki i Szkolnictwa Wyższego), N519 578138 (Ministerstwo Nauki i Szkolnictwa Wyższego) oraz "Krok w przyszłość V" (Województwo Kujawsko-Pomorskie).

Chciałabym szczególnie podziękować moim Rodzicom, którzy bardzo mnie wspierali. Bez nich powstanie tej rozprawy nie byłoby możliwe.

Chciałabym podziękować mojemu promotorowi profesorowi Wiesławowi Nowakowi oraz kolegom z Wydziału Fizyki, Astronomii i Informatyki Stosowanej: mgr inż. Marcinowi Dąbrowskiemu, mgr inż. Rafałowi Jakubowskiemu, dr Karolinie Mikulskiej, dr Łukaszoi Pepłowskiemu, mgr inż Jakubowi Rydzewskiemu za ich wsparcie i rady.

Contents

Introduction

Computer science brings a new quality to life. Our well-being is based not only on excellent hardware technology, but on progress in algorithms as well. Science in general, and biology in particular, profit also from the computer revolution. Complex biological systems are studied using sophisticated computer models. By a biological system I understand a population, an organism, a physiological system, a tissue, a cell or even a biomolecule such as a protein or a piece of nucleic acid. However, many crucial phenomena are still poorly understood and they need strong efforts in all fields, including computer science. Mathematical models of the complex systems should grasp their main features from the physical reality and transfer them into mathematical entities. Efficient manipulations on these objects require dedicated, advanced algorithms [1]. In particular, graph-based techniques have attracted the attention of computer scientists in recent years [2-4].

During my undergraduate studies I got interested in concurrent systems and parallel processing. I have learnt a simple but powerful formalism of Petri nets [5] in 2009. This mathematical language may be used in science (chemistry, biology, engineering) and industry [6]. At the same time, I developed my admiration to biology and physiology. I have found that it is possible to connect those two "fascinating fields". There are numerous papers on PN in biology-related problems [7-9]. However, there were no applications of PN in the important field of structural bioinformatics or computational biology. In this area, huge structural data sets are analyzed [10]. Graph-based techniques facilitate an analysis of data structures and a presentation of results. Dynamical phenomena, such as a time evolution of a system structure, are routinely modeled using computers, but such an analysis is rarely based on graphs or nets. What is particularly popular, also in Poland, are computer simulations of proteins and nucleic acids dynamics [11-12]. Therefore, I decided to explore the utility of PN in this field of science in my PhD thesis.

The immediate goal of this thesis was to develop new ideas based on the PN formalism and provide new computer research tools and algorithms for biological/structural data representation and analysis. That goal has been achieved. Hopefully, my models and methods will contribute to computer science, computational biomolecular modeling, bioinformatics and will find wider application in these areas of research.

To this end I have analyzed several standard, model complex biological systems (a model of human immune system, an antigen-antibody protein pair, a multi-domain transport protein), and have developed new PN models of the whole systems and/or dynamics of biomolecules. I have modeled and studied computationally numerous aspects of the immune system related to common diseases: autism spectrum disorder, AIDS, Adult-Onset Immunodeficiency Syndrome (AOIS), etc. I have

worked out new algorithms for the construction of PN and have proposed several diverse ways of modeling dynamical states using the nets based approach. In order to improve the efficiency of computational methods, I have developed new algorithms for dynamical studies of PN using parallelism provided by graphical processing units (GPUs). Those new ways of dynamical studies are an important and original part of my work.

In the first chapter of the thesis the description of the Petri nets is presented. This chapter contains basic definitions about the networks and theory of t-invariant analysis. Extended types of the Petri nets (PNs) are described: stochastic, continuous, colored, timed, priority-based and random-priority based. To the priority-based PNs, new definitions and modifications are added. New, random-priority based PNs invented by myself are introduced for the first time. In those networks the probability of transition firing is proportional to the priority of that transition.

In the second chapter the PN model of the immune system (IS) is presented. This part of the thesis contains inter alia biological description of the immune system functions and the phenomena added to the existing model: fever, ageing and the diseases such as AIDS, AOIS and autism spectrum disorder (ASD). For every phenomenon which was tested using the IS model, the way of mapping it into the network model is presented. The results obtained from the PN simulations for every feature added as well as the conclusions are described. In the last part of the chapter t-invariant analysis of the PN model is performed.

The third chapter contains a description of the parallel algorithm of the simulation of the PN, tailored for the GPU. The algorithm is implemented using the CUDA technology. The CUDA architecture is briefly described. The presented algorithm consists of two parts: a preprocessing and the simulation, are both discussed in details. The algorithm was tested using different PNs and different graphical cards, thus the computational efficiency analysis results are presented in this part as well.

In the fourth chapter the descriptions of the molecular dynamics (MD) and steered MD (SMD) studies of the two antigen-antibody complexes are presented. Those complexes are: a pollen from timothy grass Phl p2 with its antibody, and chemokine MCP-1 with its antibody. In the computer simulation study the complexes were dissociated by adding pulling forces. The effects of different directions of the external forces were tested. The results of modeling, together with proper statistical and bioinformatical discussion, are presented in the chapter.

The last chapter opens quite new areas of PN formalism applications. It contains information about a possible usage of the Petri nets in MD simulations analysis. After a general overview of the problem the types of the PNs used in my research are presented. Three, dedicated, newly designed algorithms are described. They are named: One Place One Atom (OPOA, where one place represents localization of

one atom), OPOC, where one place corresponds to one conformation of the molecule, CON, where one place represents contacts between two amino acids. The examples of the PN generated using these algorithms and their analysis are presented. New algorithms can generate PNs of different types. In order to perform the further analysis the MD/SMD simulations, the algorithm had to be adapted to simulate different types of the networks and to use data obtained from OPOA, OPOC or CON algorithms. The method of a generation of a special, so called, PDB file was based on the PNs generated earlier. The PN simulation algorithm and an algorithm suitable for the PDB file generation have been developed and they are also described in the Chapter 5.

The research on ASD was performed within a project NCN N519 578138 lead by Prof . Włodzisław Duch, Chair of Computer Sciences N. Copernicus Univ. Torun, Poland, MD simulations of proteins were a part of NCN N N202 262038 project supervised by Prof. Wiesław Nowak, Institute of Physics, N. Copernicus Univ. Torun, Poland. The search for an effective ASD genetic test was supported by kujawsko-pomorskie Voivodeship. The results were presented at 13 international conferences (i.e. in Great Britain, Germany, Hungary, Poland) and have been published in 4 papers (see: Appendix C). New papers are in preparation.

In summary, my thesis introduces new types Petri Nets, presents new algorithms for large biological data representations and analysis. I have developed new tools for the immune system pathologies analysis and have suggested new net-based methods of molecular dynamics data scrutiny. I hope that my ideas and computer codes will enrich a spectrum of computer science tools offered to the society.

The PN model of the IS, the implementation the GPU algorithm of PNs simulation, the program which allows to perform complete MD analysis using PNs and examples data are available at http://www-users.mat.umk.pl/~leii/thesis/.

Chapter 1. **Petri nets - a short review**

1.1 Basic definitions

Petri nets (PN) formalism belongs to the mathematical languages created to describe the distributed systems. The first concepts of Petri nets [13] were proposed by Carl Adam Petri in 1939. He proposed the currently common graphic representation for nets and presented their application in chemical processes. His famous dissertation "Kommunikation mit Automaten" (Communication with Automata), published in 1962 [14], is considered as the first introduction of Petri nets to science. In that work PNs were used to synchronize communicating automata. This puts the Petri nets among the oldest modeling techniques of the computer science. Indeed, many modeling methods have been proposed over the last years, like for example ordinary differential equations (ODEs) [15], process calculi [16], Boolean networks [17], Bayesian networks [18], stochastic equations [19], or cellular automata [20]. In contrast to some other techniques, which were favored for a short time and then forgotten, Petri nets have kept their place as one of the well-established modeling techniques [21]. Since 1962 the Petri nets theory has been greatly developed, some theoretical questions have been posed and solved, and many subclasses of PN have been developed in order to improve specialist systems' modeling [6]. Due to their simplicity and universality PNs have been applied in many branches of science. The main field of applications of PNs modeling is engineering. Here Petri nets-based models are used to solve different-scales problems like, for example, production scheduling [22-23], deadlock control of automated manufacturing [24-25] or even traffic jump control [26-27]. They are also often applied in computer science, for studying the properties of communication protocols [28-29], multimedia architecture [30-31] or in artificial intelligence [32-33].

Since the nineties Petri nets have been also applied in modeling of biological systems. Pioneers in this field were Reddy [34-35] and Hofestädt [36]. Reddy et al. represented metabolic pathways as Petri nets, and illustrated some properties, for example, liveness, reachability, and invariant properties. They created a PN model of fructose metabolism. Hofestädt presented PN describing the isoleucine biosynthesis in E. coli and illustrated the metabolic process depending on the expressed genes. He gave examples of modeling biosynthesis, protein biosynthesis, and cell communication processes. After those first works many diverse PN applications have been published. Petri nets have been used to model metabolic systems [4], signal transduction pathways [37] and assembly processes of complexes [38]. They are applied also in modeling of gene regulations networks [39-40]. Petri nets are popular and useful tools in modeling in medicine and chemistry [9]. However, to the best of my knowledge PNs have not been exploited in the field of biomolecular computer simulations. Thus, the present thesis aims to fill this gap.

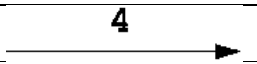
Petri nets have a form of the bipartite graph with two kinds of nodes: places and transitions. Any two places or two transitions cannot be connected by the edge.

Def. 1. A Petri net graph is a 4-tuple *(P,T,F,W)*, where:

- *P* is a finite set of places.
- *T* is a finite set of transitions (or actions), such that $P \cap T = \emptyset$
- *F* is a set of directed arcs, satisfying: $F \cap (P \times P) = F \cap (T \times T) = \emptyset$ (the place may be connected with the transition or the transition with the place; two places or two transitions cannot be connected)
- $W:F \rightarrow \{1,2,3...\}$ is a weight function assigned to arcs. The weight of one is assigned to an arc as a default.

On the plot of the network places are represented by circles, transitions by squares, arcs by arrows. Weights are represented by numbers placed near to arcs. The default weight is 1 and it is usually omitted in the plot. The commonly accepted graphical representations of elements are shown in **Table 1**.

Table 1. Elements of the Petri net and their graphical symbols.

Element	Symbol
Place	
Transition	
Arc with weight four	
Arc with weight one	
Token	

Places usually correspond to the objects or states, and transitions usually represent events.

We do not have any actions in PN created according to **Def. 1** - it is only a steady framework. In order to have actions, we need tokens. Tokens may be located in places. If a place contains a token it is named marked, if the place is empty it is named unmarked. Places may contain one or more tokens. A distribution of tokens over the places of a net is called a marking.

Def. 2. A Marking is a mapping $M:P \rightarrow \{0, 1, 2, 3, ...\}$.

Def. 3. A Petri net is a quintuple (P, T, F, W, M_0) where M_0 is initial marking, *P, T, F, W* like in **Def. 1**.

Def. 4. For each element $t \in T$ we can define the set of input places $\bullet t = \{p \in P; (p, t) \in F\}$ - the set of places from which arcs run to transition *t* and the set of output places $t\bullet = \{p \in P; (t, p) \in F\}$ - the set of places to which arcs run from transition *t*.

Def. 5. Transition *t* may fire (it is called firabled[1] or enabled) in a marking *M* if the number of tokens in every input place *p* of transition *t* is equal or greater than the weight *W(p)* assigned to the arc between the place *p* and the transition *t* in the marking *M*.

Def. 6. The set of all enabled transitions in a marking *M* is denoted as *enb(M)*.

Transition *t* consumes tokens from its input places *p* and puts them into output places *q* – the number of tokens transferred is described by the weights of arcs involved so firing of a transition changes the marking.

Def. 7. We say that firing of transition *t* transfer a marking *M* into a marking *M'*, when transition *t* is fired in marking *M* and it leads to a new marking $M - w_{\bullet t,\, t} + w_{t,\, t\bullet}$, where $w_{\bullet t,\, t}$ mark weights of arcs between input places of transition *t* and the transition *t*, and $w_{t,\, t\bullet}$ mark weights of arcs between the transition *t* and its output places. Both $w_{\bullet t,\, t}$ and $w_{t,\, t\bullet}$ are represented as vectors of the length $|P|$. The new marking is defined as $M' = M - w_{\bullet t,\, t} + w_{p,t\bullet}$ The transfer of the marking is denoted: *MtM'*.

Tokens in a place mean that an object which is represented by the place is present in the model, and the number of tokens indicates how many copies of the object are present. Transitions, as it was mentioned above, represent events. Transitions transfer a token from one place to another. This transfer corresponds to a physical change of an object into another. Transitions may also represent a change of the states of some objects.

The number of tokens is not constant, some transitions may put more or fewer tokens into their output places than they consume. In particular, a transition may not have any input places and it can fire without any restrictions and produce tokens. Also, a transition may not have any output places and it can consume tokens only.

Firing of transitions is a concurrent process, transitions which do not have common places may fire at the same time. If some transitions have common input places, they may compete for the tokens, and firing of one transition may cause that another transition will not be longer enabled.

Def. 8. Two transitions t_1 and t_2 are in the soft conflict if they have at least one common input place:

$$\bullet t_1 \cap \bullet t_2 \neq \emptyset. \tag{1.1}$$

Def. 9. Two transitions t_1 and t_2 are in the conflict if they have at least one common place (input or output):

$$(\bullet t_1 \cup t_1 \bullet) \cap (\bullet t_2 \cup t_2 \bullet) \neq \emptyset. \tag{1.2}$$

[1] Word "firabled" is correct and often used in PN publications.

The concept of the conflict is very important in some types of Petri nets. Information about conflicts can be gathered into the conflict matrix.

Def. 10. The conflict matrix is a matrix $Conf = (\alpha_{ij})_{m\times m}$ where m is size of T and:

$$\alpha_{\mathrm{ij}} = \begin{cases} 1, if\ t_i and\ t_j are\ in\ the\ conflict \\ 0, otherwise \end{cases}. \tag{1.3}$$

1.2 Algebraic representation and t-invariants.

The Petri net can be represented in an algebraic approach in the form of two matrices with integer coefficients: an input matrix and an output matrix. The input matrix represents arcs from transitions to places and its coefficients are equal to the weights of the arcs. Moreover, the output matrix represents arcs from places to transitions and its coefficients are also equal to the weights of the arcs. The algebraic representation is bijection, based on the matrices one can reproduce the PN and the opposite.

Def. 11. Let $PT = (P, T, F, W, M_0)$ be a Petri net, where P, T, F, W like in **Def. 1**, M_0 like in **Def. 3**. The input matrix is a matrix $C^+ = (a_{ij})_{n\times m}$, where:

$$a_{ij} = \begin{cases} W_{a_j p_i}, if\ p_i \in a_j^{\bullet} \\ 0\ otherwise \end{cases}. \tag{1.4}$$

The output matrix is a matrix $C^- = (\alpha_{ij})_{n\times m}$, where:

$$a_{ij} = \begin{cases} W_{p_i a_j}, if\ p_i \in {}^{\bullet}a_j \\ 0\ otherwise \end{cases}. \tag{1.5}$$

The input and output matrices may be used to calculate the incidence matrix.

Def. 12. Let $PT = (P, T, F, W, M_0)$ be a Petri net, where P, T, F, W like in **Def. 1**, M_0 like in **Def. 3**, C^+ and C^- like in **Def. 11**. The incidence matrix is a matrix $C = (a_{ij})_{n\times m}$, where $C = C^+ - C^-$.

The element α_{ij} of the incidence matrix represents the token's change at place p_i by firing of transition a_j. The incidence matrix N is necessary to define one of the most important property of the biological Petri nets: t-invariants.

Def. 13. T-invariant is a vector $x \in N_l$ (where $l = |T|$), satisfying: $C \cdot x = 0$ [9].

The t-invariant contains transitions of the PN and firing all transitions from one t-invariant will reproduce a given marking. Sometimes the transitions have to be fired a few times within the same t-invariant in order to keep the marking unchanged. Thus firing all transitions from the t-invariant will not change the marking of the network.

T-invariants are usually binary vectors, $(x)_i$ is equal to zero if the transition t_i is not included into t-invariant x, or it equals one if the transition t_i is included in t-invariant x. Sometimes other values are used to mark the transition as included into a t-invariant, for example, the values which describe how many times a transition should be fired within the t-invariant.

Def. 14. The support of the t-invariant x is given by: $supp(x) = \{t_i \in T:(x)_i \neq 0\}$.

T-invariants are very important in the network analysis. For example, in biology one t-invariant should correspond to one biological process or a pathway. All t-invariants should have a biological meaning, if some t-invariants do not have any biological sense it suggests the error in the model. In rare cases a t-invariant, which does not have a corresponding biological process, may indicate a novel property, like it was discussed in the paper [4]. The analysis of the t-invariants and its meaning is one of the most important parts of the PN based model examination.

Def. 15. The Petri net is covered by t-invariants (CTI) if all transitions of the network are contained in an t-invariant.

The CTI is a crucial property of the biological Petri net. The transition, which is not a part of at least one t-invariant, may be suspected of being a false and unwanted process. It may also indicate uncontrolled accumulation of the tokens. If PN is not CTI, it cannot be a correct model. On the other hand, if the Petri net is created with care and based on knowledge, the CTI property is a strong prerequisite for the correctness of the model [9].

One transition may be a part of a few t-invariants. According to the **Def. 13**, it is not required that the transition which is part of a t-invariant has to be enabled in the initial marking M_0 or by other transitions from the same t-invariant. The calculating of the t-invariants is an algebraic operation on the incidence matrix C and it does not require "real" firing of the transitions.

Although the biological meaning of t-invariants is a crucial process for the network analysis, sometimes the set of t-invariants may contain hundreds of elements and it will be not possible to determine the biological representation of them. However, special concepts in Petri nets analysis are available, which may limit the number of elements that have to be analyzed.

The first are Maximal Common Transition Sets (MCT-sets). The MCT-set is a set of transitions which occur always together with each other in the considered set of t-invariants [9]. An example of a table of t-invariants and two MCT-sets are shown in **Table 2**.

Table 2. Examples of transitions and their participation into t-invariants. Two MCT-sets are present: {t1, t3} and {t5, t6}.

	Inv1	Inv2	Inv3	Inv4	Inv5
t1	0	1	0	1	0
t2	0	1	0	0	0
t3	0	1	0	1	0
t4	0	1	0	1	1
t5	1	0	1	0	0
t6	1	0	1	0	0

Transitions inside one MCT-set do not have to be connected by places, but may be situated in different parts of the network. MCT-sets represents a kind of building blocks of the networks. Their biological meaning should be checked and they may represent reactions which show a similar behavior [9].

The second concept in Petri nets analysis are t-clusters. The most similar t-invariants may be connected into t-clusters. "Similar" in this case means that the t-invariants have many common transitions. To calculate t-clusters the distance matrix *D* has to be created.

Def. 16. Let *I* be the set of all t-invariants of the network, $k = |I|$ and $x_i, x_j \in I$ are t-invariants. The distance matrix is a matrix $D = (d_{ij})_{k \times k}$, where $s(x_i, x_j)$ is Tanimoto coefficient [41]:

$$d_{ij} = 1 - s(x_i, x_j) = \frac{|\, supp(x_i) \cap supp(x_j)|}{|\, supp(x_i) \cup supp(x_j)|}. \tag{1.6}$$

Of course, coefficients other than the Tanimoto coefficient for similarity measuring may be used, but in the book [9] this method is recommended.

Now, when the distance matrix is calculated, one of the clustering algorithm may be used to merge the most similar objects, it does not have to be created especially for the t-clusters. Many such algorithms have been developed, they are compared in [42] and in [9] authors recommended the UPGMA (Unweighted Pair Group Method with Arithmetic Mean) algorithm.

The UPGMA algorithm [43] can be used to merge any type of similar clusters and it uses the distance matrix. In each iteration the most similar objects are merged and the new distances between a newly created object and other objects are calculated. In the PN case t-clusters or t-invariants can be merged. The distance between two objects: C_a and C_b can be calculated by the formula:

$$\Delta_{ab} = \frac{1}{|C_a| * |C_b|} * \sum_{x_i \in C_a, x_j \in C_b} d_{ij}. \quad (1.7)$$

The algorithm stops when one object is obtained. In order to get more resulting clusters, the cut parameter is introduced. The cut parameter is a value expressed in percent and it denotes the greater distance below which the objects will be merged. If the smallest distance in the system is greater than the cut-off parameter, the objects will not be merged and the algorithm will stop. The choice of the cut-off parameter depends on a Petri network, as for one model one parameter value will be satisfying, while for another one a different value will be required. The cut-off parameter also depends on the user's preferences, such as how many results (small or large dataset) are suitable and therefore acceptable for further analysis.

During the operation, the UPGMA algorithm generates a dendrogram as well. The dendrogram describes the correlations between the resulting t-clusters. The t-clusters are leaves, and the objects with the smallest distances are connected. The dendrogram describes which t-clusters will be connected in the next iterations, if the cut-off parameter is larger. Using the dendrogram the logical composition of the model and the relationship between t-clusters and t-invariant can be analyzed.

1.3 Simulation of Petri nets

The simulation of the Petri net is one of the methods of analyzing properties of the network. If the model correctly represents the features of the modeled system then during the simulation the dynamical properties of the system may be noticed and studied. For the simulation we need the network and a number of steps which will describe the length of the simulations. One step of the simulation is one firing of transitions.

Algorithm 1. Basic algorithm of the simulation of the Petri net:

Input: The input and output matrices, both *n x m* matrices, where *n* is the number of places and *m* is the number of transitions. The initial marking presented as a vector of length *n*, where *i*-th element of the vector is a marking of a *i*-th place. The number of steps of the simulation: *k*.

Output: The marking *M*, presented as vector of length *n*, obtained after *k* steps of the simulation.

Steps:

```
1. begin
2. currentStep := 0;
3. do
4.    begin
5.    t := findEnabledTransitions();
```

```
6.    if t <> NULL then
7.          begin
8.          fire(t);
9.          currentStep := currentStep + 1;
10.         end
11.   end
12.   while(currentStep < k) AND (t <> NULL);
13.   end
```

The main part of the **Algorithm 1** is a do-while loop (lines 3-12). Before the loop the *currentStep* - variable which store the number of the current step is reset. In the loop the enabled transition *t* is found (line 5). If the transition *t* exists (it may happen that no transition is enabled) it is fired (line 8) and the number of the step is increased. The loop is finished when *k* steps are performed or the death marking is reached (no transition is enabled). Two additional functions are used in the algorithm. The first is *fire()*, in this function the marking of input and output places of the transition *t* is changed, according to weights of the arcs and definition of transition's firing. The second is *findEnabledTransition()* which is more complex. Here the enabled transition is found – there is no other way to do that as checking the marking of the every input places of following transition since the enabled one is found. Transitions may be checked in some order or may be randomly chosen.

The pessimistic time complexity of the **Algorithm 1** is the following. The loop (lines 3-12) will execute at most *k* times. The firing of a transition requires *n* steps, where *n* is the number of places, because all input and output places of the transition *t* have to be found and their marking must be changed. In the algorithm only the input and output matrices are used, so the whole row of each matrix has to be checked to found which places are connected to *t*. However, even when any other data structure will be used still the pessimistic complexity of the firing will be the same, because in some networks all transitions may be connected with all places. During the finding of the enabled transition in the most pessimistic case all transition have to be checked and the last one will be enabled, so it will require *m* operations, where *m* is the number of transitions. Checking each transition is similar to the firing and it takes *n* operations. It should be noticed that this is a very pessimistic case and typically an enabled transition will be found earlier. The pessimistic complexity of the algorithm is therefore $O(k\,n + k\,m\,n) = O(k\,m\,n)$.

Other algorithms for the same problem can be created. One possible option is:

Algorithm 2. A basic algorithm of the simulation of the Petri net – a version with the set of enabled transitions.

Input: The input and output matrices, the initial marking and the number of steps *k* are the same like in the **Algorithm 1**. The conflict matrix is like in **Def. 10**.

Output: The same like in **Algorithm 1**.

Steps:

```
1. begin
2. currentStep := 0; Enabled := ∅;
3. do
4.    begin
5.    Enabled := findEnabledTransitions();
6.    if Enabled <> ∅ then
7.          begin
8.          t ∈ Enabled;
9.          fire(t);
10.         currentStep := currentStep + 1;
11.         end
12.   end
13.while(currentStep < k) AND (Enabled <> ∅);
14.end
```

The algorithm is similar to **Algorithm 1**, the most important difference is the usage of the set *Enabled*, which is a set of all enabled transitions in the current marking. At the beginning the set *Enabled* is empty, in the do-while loop (lines 3-13), which is also the main part of the algorithm, every enabled transition is added to the set (line 5). If the set is not empty, the transition *t* from the set is chosen and it is fired. The transition may be chosen randomly or in another way. The loop finishes when the desired number of steps *k* is reached or no transition is enabled. The firing of the transition *t* may be the same like in the previous algorithm. The function *findEnabled()* is the most complex part of the algorithm. In order to find all enabled transitions every transition may be checked and if it is enabled, it will be added to the set. However, it is not an effective way and it has to be performed only once during the first step of the simulation. In the next iteration we have to check only those transitions which have some common places (or a place) with the previously fired transition. Only those transitions may change their status, others will be still enabled or not enabled. Information on which transitions have some common places with the previously fired one can be found in the conflict matrix. Only for those transitions which are in the conflict with the previously fired ones the marking of their input places must be checked.

The pessimistic time complexity of the **Algorithm 2** is: the do-while loop will execute *k*-times. The finding of all enabled transitions during the first iteration requires *m* operations to check every transition multiplied by *n* operations for every transition to control markings of its input places. During the next iterations, checking every element from the row of the conflict matrix which describes the conflicts of the previously fired transition also requires *m* operations. For those which are in a conflict, *n* calculations have to be performed in order to verify if they are enabled. In a typical situation not all transitions will be in a conflict and the checking of the marking will be performed not so often, however in the pessimistic complexity we have to assume the worst-case scenario, where every transition is in conflict with the rest of transitions and the complexity of

findEnabled() function will be still *m* times *n*. The firing of the transition can be performed in *n* steps. Together the complexity is of *O(k m n + k n) = O(k m n)* - the same as in **Algorithm 1**. However, in **Algorithm 2** the conflict matrix is necessary, which consumes *m x m* amount of a memory, and the calculation of the conflict matrix is time consuming (the basic algorithm has a complexity $O(m^2 n)$).

Summarizing, the first algorithm of the simulation is better. Indeed, in classical Petri nets it is more efficient, but in some extensions it cannot be used. For example, in timed Petri nets the terms enabled and firebled are not the same, and in order to fire a transition the set of all enabled transitions has to be found first. Also in other extensions of the Petri nets the set *Enabled* has to be calculated to find a transition to fire. In such cases **Algorithm 2** (with some modifications) should be used.

1.4 Types and extensions of Petri nets

The general concept of the Petri nets is very simple, however in many applications it is not sufficient. Basing on the canonical definitions (**Def. 3** and **Def. 5**) many extensions of Petri nets were developed, adjusted to specified applications. In the next sections a few types of Petri nets are described, but other extensions, not included here, also exist [44].

1.4.1 Stochastic Petri nets

Stochastic Petri nets (SPNs) are basically similar to classical place/transition PNs. However, in SPN enabled transitions fire with an exponentially distributed time delay. They were proposed by researchers active in the field of applied stochastic modeling. They idea was presented in two doctoral thesis of S. Natkin [45] and M. K. Molloy [46]. Those works were performed simultaneously and independently almost in the same time.

The basic definition of the SPN is following [47]:

Def. 17. An SPN is a six-tuple: $SPN = (P, T, I, O, M_0, \Lambda)$, where (P, T, I, O, M_0) is the marked PN (the same like in **Def. 3**) underlying the SPN, which as usual comprises:

- a set of places P
- a set of transitions T
- a set of input arcs $I \in P \times T$,
- a set of output arcs $O \in T \times P$,
- an initial marking M_0

and $\Lambda = (\lambda_1, \lambda_2, ..., \lambda_n)$ is an array of (possibly marking dependent) firing rates associated with transitions.

A firing delay is associated with each transition. It specifies the amount of time that must elapse before the transition can fire. This firing delay is a random variable with negative exponential probability distribution function [47]. According to [7] SPN definition may be expanded to the weights (as in **Def. 3**). The rules of transitions firing in SPN have to be changed [47]. During the firing process each enabled transition has to sample an instance of the random firing delay from the associated probability distribution function. The transition which samples the minimum firing delay is the one which will fire. The marking is changed according to standard rules.

1.4.2 Hybrid Petri nets

Hybrid Petri nets contain classical, discrete places and transitions as well as continuous places and transitions [44], so the definition of the continuous PN must be introduced.

In the continuous PN places may be real numbers and firing of the transitions is carried out like a continuous flow. In a continuous PN, one can have a quantity of firing which is not an integer. In a continuous PN, one can have a quantity of firing which is not an integer [44]. Such situation is presented in **Fig. 1**.

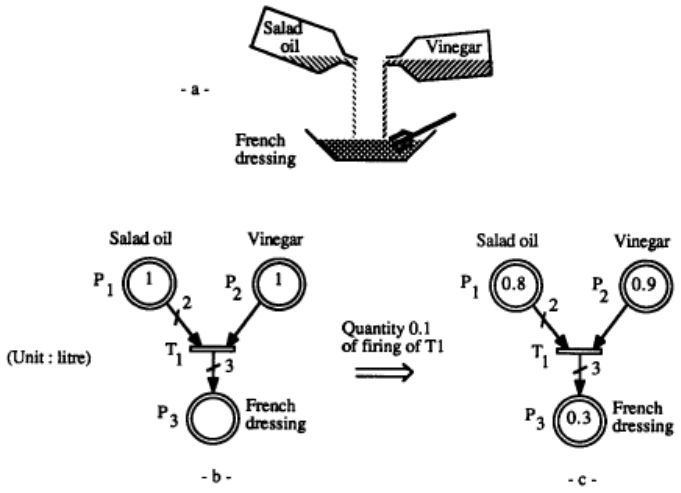


Fig. 1. An example of continuous PN (from [44]).

An example continuous PN represents a creation of French dressing from salad oil and vinegar in the ration of two. For example [44], if the quantity of firing is $x = 0.1$, the marking in **Fig. 1b** is transformed into marking presented in **Fig. 1c**. In the hybrid Petri net additional places and transitions may be added to Petri net presented in **Fig. 1** which can be marked, discrete Petri net places or transitions. Continuous and discrete elements can be connected. For example if transitions T1 from **Fig. 1** have additional discrete input place and the weight of the arc between them is one, then T1 can fire only when this discrete place is not empty and

transition T1 will consume one token from this place during firing. The definition of hybrid Petri nets is presented in [39].

Def. 18. We denote a *hybrid PN* as $Q = (P, T, h, Pre, Post, M_0)$, where P and T are the sets of *places* and *transitions* respectively; $h : P \cup T \rightarrow \{D, C\}$ indicates for every place or transition whether it is a discrete or continuous one. A non-negative integer called the *number of token* is always associated with a discrete place and a non-negative real numbers called the mark is always associated with a continuous place. $Pre(P_i, T_j)$ ($Post(P_i, T_j)$) is a function that define arc from a place P_i (a transition T_j) to a transition T_j (a place P_i), where the arc has a weight of non-negative integer (non-negative real value) if $h(P_i) = D$ ($h(P_i) = C$). *Pre* and *Post* functions must meet the following criterion: if P_i and T_j are a place and transition such that P_i is discrete and T_j is continuous then $Pre(P_i, T_j) = Post(P_i, T_j)$ must be verified; M_0 is a mapping from the set of places to the set of non-negative integers or the set of non-negative real numbers called the initial marking.

1.4.3 Colored Petri nets

In some modeled systems it may be necessary to distinguish tokens between themselves. That is the reason why colors were introduced to Petri nets. In colored Petri nets (CPN) [48] information can be attached to each token as a token-color, each transition can occur in several ways represented by different occurrence-colors and the relation between an occurrence-color and the token-colors involved in the occurrence of the transition is defined by functions attached to the arcs. An example of such network is presented in **Fig. 2** [49].

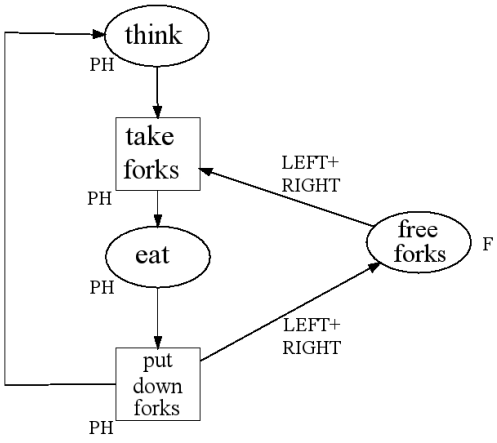


Fig. 2. Colored Petri net describing the philosopher system (from [49]).

Presented CPN describes the five philosopher problem. One philosopher can think or eat, but for eating he requires two neighboring forks. The set PH is a set of colors representing the individual philosopher, F is a set representing the fork. LEFT and RIGHT are functions which map each philosopher color into the color of its left and right fork respectively [49]. Those functions provide that firing of "take

forks" transition with color $c \in$ PH removes two tokens from "free forks" with colors LEFT(c) $\in$ F and RIGHT(c) $\in$ F. Similar situation occurs for "put down forks". Formal definition of CPNs is following [49]:

Def. 19. Let A be a nonempty set and D be $\mathbb{N}$ or $\mathbb{Z}$. By $[A \to D]_f$ we denote the set of functions $g \in [A \to D]$, where the support $\{a \in A \mid g(a) \neq 0\}$ is finite. For finite A we have $[A \to D]_f = [A \to D]$.

A colored Petri net is 5-tuple CPN = *(P, T, C, W, M_0)*, where:

- *P* is set of places
- *T* is set of transitions
- $P \cup T = \emptyset, P \cap T \neq \emptyset$
- *C* is the color-function defined from $P \cup T$ into nonempty sets
- *W* is the incidence-function defined on $P \times T$ such that $W(p, t) \in [C(t) \to [C(p) \to \mathbb{Z}]_f]$ for all $(p, t) \in P \times T$
- M_0 is the initial marking, is a function defined on *P*, such that $M(p) \in [C(p) \to \mathbb{N}]_f$ for all $p \in P$

1.4.4 Timed Petri nets

In the classical Petri nets (**Def. 3**) one of the enabled transitions fires and we do not try to control which one. However, sometimes it is necessary to add some order to a Petri net to represent modeled events better and that is why time is added to the networks. Numerous time extensions of Petri nets were developed [50], in some approaches time is added to transitions, in other to places [50], but all are called Timed Petri nets. The most popular are two formalisms: Ranchamdani's Timed Petri nets (RTPN) and Merlin Time Petri nets (MTPN) [51]. In MTPN times intervals are associated with the transitions. However, for my studies the Ranchamdani's Timed Petri are much more suitable and their general idea will be presented, following the paper [51].

Def. 20. A timed Petri net is a six-tuple: *(P, T, F, W, M_0, f)*, where *P, T, F, W, M_0* are like in **Def. 3** and $f: T \to \mathbb{R}^+$ is a firing time function, which assigns a positive real number, called firing time, to each transition.

Now, when the new element was added to the definition of the Timed Petri net, also the firing rule have to be changed to adjust to a new concept. Not only does the firing time have to be added to the network, but also some representation of the current time state of the system is necessary. The firing time function is something like canonical, initial time necessary to fire a transition and some type of a clock is required to measure if this firing time has been reached.

Def. 21. The clock state [51] is a pair (*M, V*), where *M* is a marking and V is a clock valuation function, $V: enb(M) \to \mathbb{R}^+$.

The clock state is defined only for transitions which are enabled and it represents time necessary to fire the transition. That amount of time will elapse during

transition's firing. The initial clock state s_0 for each transition is equal to the value of the firing time function: $s_0 = (M_0, V_0)$, where $V_0(t) = f(t)$, for $t \in enb(M_0)$. After the initiation of the clock for a newly enabled transition the clock state for this transition is subject to change, as long as the transition is enabled - this mimics the passage of time. Because of that now it is necessary to distinguish between newly enabled transitions and transitions that were enabled in the previous markings, thus the concept of the new enabled transition is introduced [51].

Def. 22. A transition $t \in V$ is a new enabled after firing transition t_f at marking M which leads to marking M' if (i) it was not enabled in M and it is enabled in M' or (ii) if it is the former fired transition t_f and it is still enabled. The set of new enabled transitions is denoted as $new(M')$.

Now we have to distinguish between the enabled and firabled transitions. In the previous definitions those terms were used alternatively and had the same meaning. This was possible because in the classical Petri nets only a basic firing rule (**Def. 5**) referring to the marking of the input places is used, and no other rules are present. Now, when the transition satisfies the basic condition about the marking of its input places, it will be called an enabled, but it is not enough to fire. To fire the transition has to satisfy additional conditions and only when they are fulfilled the transition is called firabled and may fire. Usually many transitions are enabled but among them only a few transitions are firabled, sometimes only one. The firabled transition in Timed Petri net is [51]:

Def. 23. The transition $t_f \in V$ is firabled in marking M when: $t_f \in enb(M)$ and $V_M(t_f) \leq V_M(t_i) \underset{t_i \in enb(M)}{\forall}$.

According to **Def. 23**, a transition is firabled when the value of the clock assigned to the transition is the smallest in all enabled transitions. This value of the clock is denoted as τ and it represents the time which elapses while firing a firabled transition. When this time elapses the clock state of other transitions should be reduced by this value. So, when the firing of the transition t_f at marking M leads to marking M' then the clock state will be as follows:

$$V_{M'} = \begin{cases} f(t) \; for \; t \in new(M') \\ V_M - \tau \; for \; t \in enb(M') \setminus new(M') \end{cases}. \tag{1.8}$$

When a few transitions are firabled, at a given moment just one has to be chosen (in any way) to be fired.

1.4.5 Priority-based Petri nets

Adding priorities is another type of Petri nets modification [52]. The priority is added to transitions and strongly affects a sequence in which the transitions are fired.

Def. 24. The priority-based Petri net is 6-tuple: *(P, T, F, W, M_0, prio)*, where *P, T, F, W* are the same like in **Def. 1** and M_0 is like in **Def. 3**. *prio* is the function which assigns a natural number, called "priority" to the transitions: $prio: T \rightarrow \mathbb{N}$.

Now the conditions under which transitions may fire have to be changed. In this type of Petri nets we have to distinguish between a transition enabled and firabled. Like before, a transition is enabled if it satisfies conditions for a required number of tokens in its input places. A transition is firabled if it is enabled and additionally it fulfills extra features connected to the extension of the PN.

Def. 25. A transition *t* is firabled in the priority-based Petri net defined in **Def. 24** when it is enabled and there are not any enabled transitions with the higher priority [52]:

$$t \in firable \Leftrightarrow t \in enb \; and \; \overset{\forall}{_{t_j \in enb}} \; prio(t_j) \leq prio(t). \tag{1.9}$$

When the transition fires it transfers tokens like in the classical Petri net. So, in the priority-based Petri nets only the transition with the highest priority at a time may fire. It is easy to image the network in which two transitions with a very high priority create a cycle and one of them is enabled. They will fire alternately and other transitions, even if they are enabled, will not have an opportunity to fire. Other parts of the network (apart from those two transitions in the cycle) will not be even necessary. Other unique characteristic of such a network appears when one starts simulations of the Petri net several times from the same initial marking. Then the same sequence of transitions firing will be observed because a value of the priority function does not change, since it is assigned to the network in the initial marking. Therefore, the same transition will always have the highest priority and it will be fired in every simulation.

Petri nets with the firing rule like in **Def. 25** may be useful in some cases, however, in my study Petri nets with a less deterministic sequence of transitions' firing are required (a more detailed description of suitable PN is presented in Chapter 5). Here, I propose a modified version of **Def. 25**.

Def. 26. Let K_j be the set of enabled transitions which are in soft conflict with a transition t_j, including also the transition t_j. K_j is called a conflict set of transition t_j.

Firing of transition t_j in a marking *M* which leads to a marking *M'* may cause the transitions from K_j, which according to the definition were enabled in the marking *M*, may be no longer enabled in the marking *M'*.

Lemma 1. Firing of a transition t_j in a marking *M* which leads to a marking *M'* will not change any enabled transition $t_k \notin K_j$ in the marking *M* into transition not enabled in the marking *M'*.

Proof. A few relations between input and output places of transitions t_j and t_k are possible. The first - they do not have any common places – the firing of the

transition t_j will not change anything for the transition t_k. The second - they have common output places, so the firing of the transition t_j will change a marking of output places, but the output places do not play any role in classifying the transition t_k as enabled. The next case - some (maybe only just one) input places of the transition t_j are output places of the transition t_k. Like before, changing a marking of the output places does not affect the firing of the transition t_k. The last case - some output places (maybe only one) of the transition t_j are input places of the transition t_k. The transition t_j will only add tokens to the input places of the transition t_k. Since t_k was enabled earlier, all its input places contain enough tokens, so it will still be enabled. The case when both transitions have common input places (one common place is enough) is not possible because $t_k \notin K_j$.

□

Now we are ready to define a new rule for firing transitions in the priority-based Petri net.

Def. 27. A transition t_j is firabled in the priority-based Petri net defined in **Def. 24** when it is enabled and there are no any transitions with a higher priority in its conflict set:

$$t \in firable \Leftrightarrow\ t_j \in enb\ and \underset{t_i \in K_j}{\forall} prio(t_i) \leq prio(t_j). \tag{1.10}$$

Moreover, if there is a transition with a higher priority in K_j is becomes firabled.

$$\underset{t_i}{\exists}\, t_i \in\ K_j\ \wedge \underset{t_k \in K_j, k \neq i}{\forall} prio(t_i) \geq\ prio(t_k) \Rightarrow t_i\ is\ firable. \tag{1.11}$$

So, if we want to fire a transition, we should choose the transition t_j, which is enabled, then we should define its conflict set, check priorities and finally choose the transition with the highest priority from the set. The transition t_j may be chosen in different ways, for example, randomly. A random selection provides more diverse sequences of transitions' firing during Petri net simulation than implicated by the definition **Def. 25**.

We may consider the problem of firing of transitions also in a different way. First, we choose the conflict set and we fire a transition with the highest priority from the set. This does not impact other conflict sets, as a consequence of **Lemma 1**. So, in every step of the simulation the number of possible fired transition is equal to the number of conflict sets.

The sequence of firing of transitions based on **Def. 27** is obviously different from the one implicated by **Def. 25**, however, some similarity is kept.

Lemma 2. A transition t_k which in the marking M satisfies the equation (1.11), according to **Def. 27**, (a) will be fired in this marking or (b) if other transition has been fired and this has led to the marking M', then in the marking M' the transition t_k will be still enabled.

Proof. If in the marking M t_k there is the transition chosen as a candidate to fire or any other transition from the conflict set K_k is chosen then, since t_k is the transition with the highest priority from all the ones enabled, t_k will fire. If the other transition (i.e. from the other conflict set) has been fired and has led to the marking M' then, according to **Lemma 1**, t_k is still enabled.

□

One can see that even after firing some other transitions, the transition with the highest priority will still be enabled and will have a chance to fire. Moreover, this transition will be eventually fired because, according to Lemma 2, no other transition may change its ability to fire. In some situations both rules of firing in the priority-based Petri net **Def. 25** and **Def. 27** become the same, for example, when only one place which is an input place for only one transition, is marked, then the conflict set and a set of all enabled transitions are the same. To avoid such situation a new type of the Petri nets - random priority-based Petri nets is developed here.

1.4.6 Random priority-based Petri nets

The random priority-based Petri nets are very similar to the classical priority-based Petri nets. However, they do not have an unwanted property of full determinism. This feature means that when we start a simulation from the same initial marking several times, we will obtain the same sequence of transitions' firing.

Def. 28. The random priority-based Petri net is 7-tuple: $(P, T, F, W, M_0, prio, X)$, where P, T, F, W, M_0 and $prio$ are like in **Def. 24**. X is a random variable with continuous uniform distribution on the interval <0, 1>.

Like previously, also here a transition enabled should be distinguished from a firabled one. A transition is enabled if its input places contain at least so many tokens as the weights of corresponding arcs, which connect the input places and the transition. To become firabled the transition has to satisfy an additional condition imposed by the type of the Petri net. In a random priority-based PN a conflict set will be used but its definition **Def. 26** has to be modified.

Def. 29. The conflict set of transition t_j is a set with a linear order K_j, that contains transitions which are enabled and are in a soft conflict with t_j (including t_j).

A new concept here is a linear order. Numerous definitions are possible, for example, an order resulting from the arbitrary numbering of transitions in the network. Also other linear orders are possible, for example, imposed by a user within a given conflict set. We need to stress that if one order is accepted it should be used during the entire operation of finding a firabled transition.

Def. 30. Let $S_{j,k}$ be the sum of values of priority function of k first elements of the conflict set K_j according to its linear order. S_j is the sum of values of priority function of all elements of the conflict set K_j.

Def. 31. A firing value *f* is a value selected from the uniform random variable *X* in the random priority-based Petri nets.

Def. 32. The transition t_j is firable if it is enabled and it fulfills one of the following conditions:

- it is not in any soft conflict ($|K_j| = 1$)
- it is in a soft conflict ($|K_j| > 1$). Then, the firing value *f* is selected from the uniform random variable *X*. Now, the transition t_j may fire if:
  - $0 \leq f < \frac{prio(t_j)}{S_j}$ if the transition t_j is the first in the K_j (according to its order).
  - $\frac{S_{j,n-1}}{S_j} \leq f \leq \frac{S_{j,n}}{S_j}$ if the transition t_j is *n*-th element of the K_j (according to its order).

If f does not belong to the above intervals, then a transition t_i exists which is the *x*-th element in the set K_j and it satisfies the condition:

$$\exists_{x \in \mathbb{N}} \frac{S_{j,x-1}}{S_j} \leq f \leq \frac{S_{j,x}}{S_j},$$

the transition t_i becomes firabled.

In other words, usually when we are looking for a firable transition, we have to select one enabled transition. This selected transition determines its conflict set. A single transition from this set will be fired basing on the random firing value *f* (not necessary the chosen one!). It will be just that transition whose priority (projected into the interval <0, 1>) encompasses the drawn firing value *f*. Therefore, the larger the projected interval is, the bigger the probability of the firing. Moreover, higher values of the priority function (of the transition) result in larger intervals. In the random priority-based Petri net the probability of firing of the transition is proportional to the value of the priority function, so transitions with high priorities will fire more often than transitions with smaller priorities. However, every transition has an opportunity to fire, even the transition with the smallest priority. It cannot be foreseen which sequence of transitions' firing will occur in this type of PN. However, the sequence is not fully random, since each sequence has a predefined probability. Sequences of the transitions with a higher priority will appear more often in PN simulations. This property is strongly desirable in my work.

Chapter 2. **The immune system as a model of biological system**

2.1 Introduction

Humans and other animals live in an unsafe environment, full of microorganisms like bacteria, viruses, parasites and fungi. They may be very dangerous for higher organisms like mammals. In order to avoid that threat, during millions of years of the evolution, several lines of effective biological defense systems have been developed. The central role in this protection plays the immunological system (IS) [53-54]. Such a system may be modeled computationally. Developing of new network-based methods useful for IS study was one of the goal of the present thesis. Here a short description of IS is presented.

The IS usually works quite well, but sometimes perturbations in its activity create health problems. Since I have experienced such a situation personally (frequent infections), I devoted much interest to the IS structure and functioning. Knowing more about the physiology of IS prompted me to develop novel computer models of IS. PNs seemed to be a perfect tool for this purpose. I have learnt that some scientists had already initiated research in this field [55-57]. Good computer model of IS may facilitate IS research and may lead to new information useful in medicine and/or biology. In particular, various pathological states of IS may be computationally studied in an effective and controlled way.

This was the main motivation for the formulation of our new Petri net based model of IS. The model, more elaborated than the ones published before [55-57], has been already presented at many international conferences and in two papers [58-59]. Here we add new aspects: the modeling of AIDS, symptoms of the AOIS disease and the studies of IS ageing. Technical details are described in the paragraph 2.4, and the main features of IS are described in the next paragraph. One should note that our model presents mostly an adaptive IS and a part of the innate IS. This chapter has some "biological" character, however, introducing this information is crucial for understanding the significance of my PN programming efforts and computer experiments performed.

The immune system is a complicated set of cells, proteins and interactions between them, critical for proper functioning of higher living organisms [53]. The immune response is a complex and multistage process which involves many different mechanisms. In order to better understand and comprehend such a complicated system as IS two ways of classifications of IS components were introduced. It should be noted that those divisions are not strict, as some elements may occur in many parts or other IS components may be difficult to classify. Nevertheless, the divisions (classifications) are generally accepted, present in the literature, and they enable better understanding of the immune response.

The first division of the IS distinguishes between two parts: cellular and humoral response [54]. The cellular response is directed against pathogens living in cells for example, against viruses which enter cells, modify their DNA and multiply inside the cells. The cellular immunity involves the activation of macrophages, natural killer cells (NK), antigen-specific cytotoxic T-lymphocytes (Tc), and the release of various cytokines in response to an antigen. The humoral response is specialized against pathogens living in body fluids. The basic elements of the humoral immunity are antibodies produced in the cells of the B lymphocyte (B cell). In both the cellular and humoral immunity helper T lymphocytes (Th cells) are involved.

The second division of IS concerns an innate and an adaptive immune system [53-54]. The innate immune system provides an immediate defense against the infection in a non-specific way. The elements of this system can recognize the pathogens as "alien" cells and respond to them in a non-specific way. It is evolutionary mechanism older and faster than the adaptive immune system. The innate immune response is necessary for activation the adaptive immune response. Elements of the innate immune system are mast cells, macrophages, neutrophils, dendritic cells, basophils, eosinophils, natural killer cells. The adaptive immune system can recognize a pathogen and can respond in a specific way. It consists of highly specialized cells, but its reaction is slower. Elements of the adaptive immune system are helper T cells (Th), cytotoxic T cells (Tc), B cells and antibodies produced by them [60].

2.2 Immune response

The first step of the adaptive immune response is an antigen presentation. Many type of cells may present an antigen (B cells, dendritic cells, macrophages, infected regular body cells), but professional antigen presenting cells (APC) are dendritic cells (DCs). They have two types of major histocompatibility complex (MHC) proteins: MHC class I and class II. The DCs migrate to various body tissues and there they sample the surrounding environment. If DCs are infected by intracellular antigens, the antigen is preprocessed inside the cell and their fragments are presented on the surface of the DCs in a complex with the MHC class I. The same mechanism is present in all infected body cells and it is not specific for DCs. The MHC class I molecules present the antigen to Tc cells. DCs also phagocyte exogenous pathogens. The exogenous antigen is preprocessed inside the cell and its fragments are presented on the surface of the DCs in the complex with the MHC class II. The MHC class II molecules are recognized by Th cells [61]. When DCs find the antigen they migrate to lymph nodes and there second step of the immune response takes place.

DCs via MHC class II present antigens and activate native helper T lymphocytes. Th cells need two signals - two sets of corresponding receptors to have a proper activation. One signal is not enough, it may indicate autoimmune response and a Th cell after only one activation signal becomes inactive. A native Th cell

maturates, produces cytokines and differentiate into helper T cells type 1 (Th1) or type 2 (Th2). The type of Th cell depends on cytokines produced by DCs and a type of an antigen. Both types of the Th cells produce cytokines, but specific for the type. Th1 produce cytokines which inhibit the humoral response and promote the cellular response. They are necessary during the activation of the cytotoxic T lymphocytes. Th2 produce cytokines which inhibit the cellular response and promote the humoral response.

The basic elements of the cellular response are the cytotoxic T cell (Tc cells). A Tc cell is activated by APC via MHC class I and it also requires two signals - two sets of corresponding receptors for the activation. After the activation the Tc cell kills APC cells (because they are infected), and stimulated by Th1 cytokines it can multiply. New, active Tc lymphocytes leave the lymph nodes and migrate to find infected body cells. The infected body cells present the antigen via MHC class I and that is why Tc cells can find them. When Tc cell recognizes the infected cell it induces the apoptosis of the infected cell.

B cells are present in the organism and they poses receptors on their surface which correspond to antibodies produced by the B cell. The B cell without an activation may produce IgM and IgD antibodies. If those antibodies recognize the antigen, receptors of a B cell will also recognize it. The B cell, which recognizes the antigen, presents it and migrates to lymph nodes. There the B cell presents the antigen to Th2 cells and if Th2 cells recognize it, Th2 activate the B cell and B lymphocyte produces IgG, IgE or IgA antibodies. The production of the antibodies may be additionally stimulated by cytokines produced by the Th2 cells.

The steps described above are mechanisms of the adaptive immune system. The macrophages are a part of the innate immune system. They circulate through the body and phagocyte antigens. They also produce a few kinds of cytokines, which have an impact on different types of the immune response.

2.3 Phenomena present in immune system

Fever is one of the interesting biological phenomena related to IS. According to the Britannica Encyclopedia fever is abnormally high bodily temperature or a disease of which an abnormally high temperature is characteristic [62]. A fever may be caused by substances called pyrogens. One of the most common pyrogens is lipopolysaccharide (LPS) present in bacteria. The presence of the LPS causes production of cytokines: IL-1, IL-6 and TNF-α, which induce an increase of the thermoregulatory set point in the hypothalamus. Some elements of the immune system are sensitive to temperature increase. The fever does not cause any new mechanisms in IS, it only stimulates different processes and increases their rates [63]. During the fever dendritic cells migrate faster to a place of the infection [64]. At a higher temperature Tc lymphocytes are more active [63, 65], their differentiation, proliferation and persistence are stimulated [63, 66-70]. The fever

also facilitates the Th cell activation [69]. The elevated body temperature does not have any direct impact on B lymphocytes, but through an activation and a stimulation of Th2 lymphocytes, it stimulates the B cells and increases the production of the antibodies [63, 65, 71].

Other phenomenon added to our IS model was ageing of the IS. Ageing has an impact on different elements of the immune systems. The first is reducing the production of IL-2 and INF-γ by Th lymphocytes. The second effect is decreasing the amount of native T and B cells. The next feature related to ageing is smaller expression of CD28 (and TCR) receptors and an increase in a number of "memory" T cells [72-73]. So, the ageing has mostly adverse impact on the IS. The larger number of the memory T cells may be beneficial, however, they may hinder the immune response against unknown infections – together with a smaller number of the native lymphocytes. Many scientists suggested that having the IS capable preserving its functions during years, is a good prognostic factor for human longevity [72].

We were also interested in diseases directly related to IS like AIDS and AOIS. AIDS means Acquired Immunodeficiency Syndrome and it is caused by the HIV virus. When the HIV virus infects the organism, it attacks Th cells and multiplies inside those cells. The number of Th lymphocytes decreases and the immune response cannot develop property. Without a proper immune response even normally harmless infections become lethal, for example, a common cold may transform into dangerous pneumonia. Moreover, AOIS (Adult-Onset Immunodeficiency Syndrome) is a newly discovered disease [74] which was reported for the first time in Thailand and Taiwan. Its symptoms are similar to AIDS: frequent and severe infections which should be harmless during a normal immune response. Patients with AOIS were tested for the presence of HIV virus, however they were not infected by the virus. The authors of the paper [74] found the presence of anti-INF-γ antibodies in 88% of Asian adults with multiple opportunistic infections. Those antibodies bind to INF-γ - one of the cytokines necessary to develop a proper immune response. The low level of INF-γ paralyzes the immune system.

Autism Spectrum Disorder (ASD) is a range of severe medical conditions without a known cure. It is a very serious disease and its symptoms are: communication impairments, social deficits, problems with the acceptance of changes, repetitive behaviors, etc. Reviews tend to estimate a prevalence of 6 per 1000 for autism spectrum disorders, however the number of autism cases continues to grow during the last years [75].

The etiology of ASD is not known, and a few risk factors have been identified. Genes' mutations are perhaps one of the most probable causes [76]. Many gene mutations (>100) have been found to be correlated with ASD, and this correlation is confirmed by the epidemiology. The prevalence of ASD in siblings of autistic

children is approximately 15 to 30 times greater than the rate in the control population [77], and the prevalence in monozygotic twins is even higher (60-94%) [78-79].

However, some authors suggest that the correlation between the prevalence of ASD and brotherhood is caused by the same environmental risk factors [80]. One of the recently postulated possible ASD risk factors is related to the immune system [80-85]. In papers [83-84] authors suggest that the maternal immune response during a pregnancy may be a cause of ASD. Indeed, many animal models of ASD have been achieved by the activation of pregnant females' immune system [83]. Some of the cytokines from the maternal circulatory system may be transported across the placenta and induce inflammation in the placenta or the fetus [82-83], which in turn may cause a distribution in neurodevelopment in the offspring [82, 85]. Another consequence of the maternal inflammation during the pregnancy may be deregulation of the immune system in the autistic people [81-83], which is revealed by altered levels of some cytokines. **Table 3** contains the summary of information about the cytokines, showing which levels are elevated in the ASD hosts.

Table 3. A list of cytokines exhibiting elevated levels in ASD hosts

Cytokine	Source	Target	Stimulation of other cytokines	Comment
IL-4	Th0, Th2	Th0->Th2, B, Macrophages		Anti-inflammatory
IL-5	Th0, Th2	B		
IL-13	Th0, Th2	B, Monocytes		
IL-2	Th0, Th1	Th0->Th1, Tc		
INF-γ	Th0, Th1	Tc, NK, Macrophages	IL-1, IL-6, TNF	Antiviral; cause of fever
TNF-α	Macrophages, Monocytes	Macrophages, T and B with other cytokines	IL-2, INF-γ, IL-1, IL-6 (Macrophages)	Cause of fever; pro-inflammatory; triggered by LPS and INF-γ
IL-1β	Macrophages, Monocytes	T, B	IL-2, INF-γ, IL-1, IL-6 (Macrophages)	Cause of fever; pro-inflammatory; triggered by LPS and INF-γ
IL-6	Macrophages, Monocytes	B, T->Tc		Cause of fever; triggered by IL-1 (mostly) and INF-γ, TNF, LPS
MCP-1	Macrophages, Monocytes	T, B		Pro-inflammatory

Another premise for the association between ASD and IS may be observation that in some autistic cases fever improves the condition of children with ASD [86]. The children with elevated temperatures were reported to exhibit longer attention spans and increased social interactions than were typically observed in normal temperature conditions. The symptoms remission occurred at the onset of fever and

persisted for 1 – 3 days following after return of the temperature to the normal level. The reason of such effect of the fever is not well studied, two hypotheses were developed. The first hypothesis refers to locus coeruleus-noradrenergic (LC-NA) system [87]. Other authors suggest that the fever phenomenon is correlated with fluctuations in levels of many cytokines during fever [83, 88-89]. The cytokines are very important in the communication between the central nervous system and the immune system, many of them participate in normal neural development and function [82]. Therefore in our PN based model of IS we included major cytokines and studied cytokines levels in normal and elevated temperature conditions in the context of ASD.

2.4 The model

In 2004 Petri nets were used for the first time to create a simple model of the immune system by Na *et al*. [55]. They used fuzzy-continuous PNs formalism to create three networks describing Th cells proliferation, humoral and cellular response. However, their model was very general, as it was limited only to main immune system's cell types and their interactions. It is also not up-to-date, since in 2004 many immune mechanisms were not known, as well as the role of cytokines in IS. In 2005 and 2006 the authors added cytokines to the model, however only to the networks which describe lymphocyte Th1 and Th2 selection step [56-57]. Inspired by that work, we have developed a much more elaborated model of the IS. This chapter is partially based on the results presented in *Gogolinska, A. and W. Nowak, "Petri Nets Approach to Modeling of Immune System and Autism", in Artificial Immune Systems, Springer Berlin / Heidelberg, 2012, pp. 86-99.*

In 2009 the first version of my model of the IS, inspired by Na *et al.* was created. That model was also rather simple and limited solely to the major elements of the immune system. During next years I have developed and extended the model by adding new elements, mechanisms and features [58]. Initially it consisted of four parts (sub-networks), which may be studied separately or as one, large network. The studies of all networks together are more interesting and bring more results. In this model I have used the classical Petri network formalism and focused on the adaptive immune response. The four parts of the model are: (A) reaction of dendritic cells, (B) Th cells proliferation, (C) cellular response and (D) humoral response. In 2011 a new part of the immune response was added to the model [59]: (E) the reaction of the macrophages. The macrophages are classified as the part of the innate immune response, however, they were necessary to model the immune system compromised in the people with ASD. The whole model is presented in **Fig. 3** and **Fig. 4**.

Part (A) describes the reaction of the DC cells. In general, it represents two linear pathways of the preprocessing of the antigen. Linear here means that if the first transition from the pathway is activated then the next transitions will fire in the established order. Choosing the pathway depends on type of the antigen: external

or internal. Then through the MHC class I or II the antigen is presented to Tc cells or Th cells. Part (B) represents the Th cell proliferation. DC cell presents the antigen, and if two signals of an activation are present a Th cell proliferates into Th1 or Th2 lymphocytes, and they produce their characteristic cytokines. The next transitions connect Th1 cells with the cellular response and Th2 cells with the humoral response. Part (C) shows an activation of the Tc lymphocytes by DC cells and their development. Tc cells require to maturation cytokines produced by Th1 lymphocytes. Additionally the cytokines produced by macrophages stimulate Tc lymphocytes for faster development. The mature Tc cells kill body cells infected by viruses in two ways. The body cell may be infected by a "regular" virus or by the HIV virus (only Th cells may be infected by that virus). Part (D) describes humoral response during which B cells produce antibodies. If they recognize and bind to an external antigen, the B cell may become active. With the participation of Th2 cells, the B lymphocyte starts the production of the new types of the antibodies. The production may be stimulated by the presence of cytokines produced by Th2 cells or macrophages. The description of the whole network is presented in **Table 4** and **Table 5**. The model file is also available at http://www-users.mat.umk.pl/~leii/thesis/.

The model was created basing on text books and other papers [53-54, 61]. Because of the difficulties in collecting the appropriate values of parameters like, for example, the amount of INF-γ which is produced by Th1 cells, the model is qualitative. The correct numbers necessary to create a quantitative model are not measurable experimentally at this moment and are not known, thus it is impossible to develop such a model. As a result, the PN model of IS cannot give answers to questions like: how many cells or proteins are present in the system. However, the model can answer other biologically important questions, for example: whether in given conditions the amounts of cells or proteins increasing or decreasing, what changes are expected when the amount of some substances is increasing (or decreasing), etc. Moreover, the weights in the model are usually default (equal 1) to avoid adding a quantitative data. Only in a few cases customized weights were necessary, for example, special weight were used in a transition representing proliferation of the cell. On rare occasions the weights were added arbitrary to highlight the stimulation of some processes, and they were usually small integers like two or five.

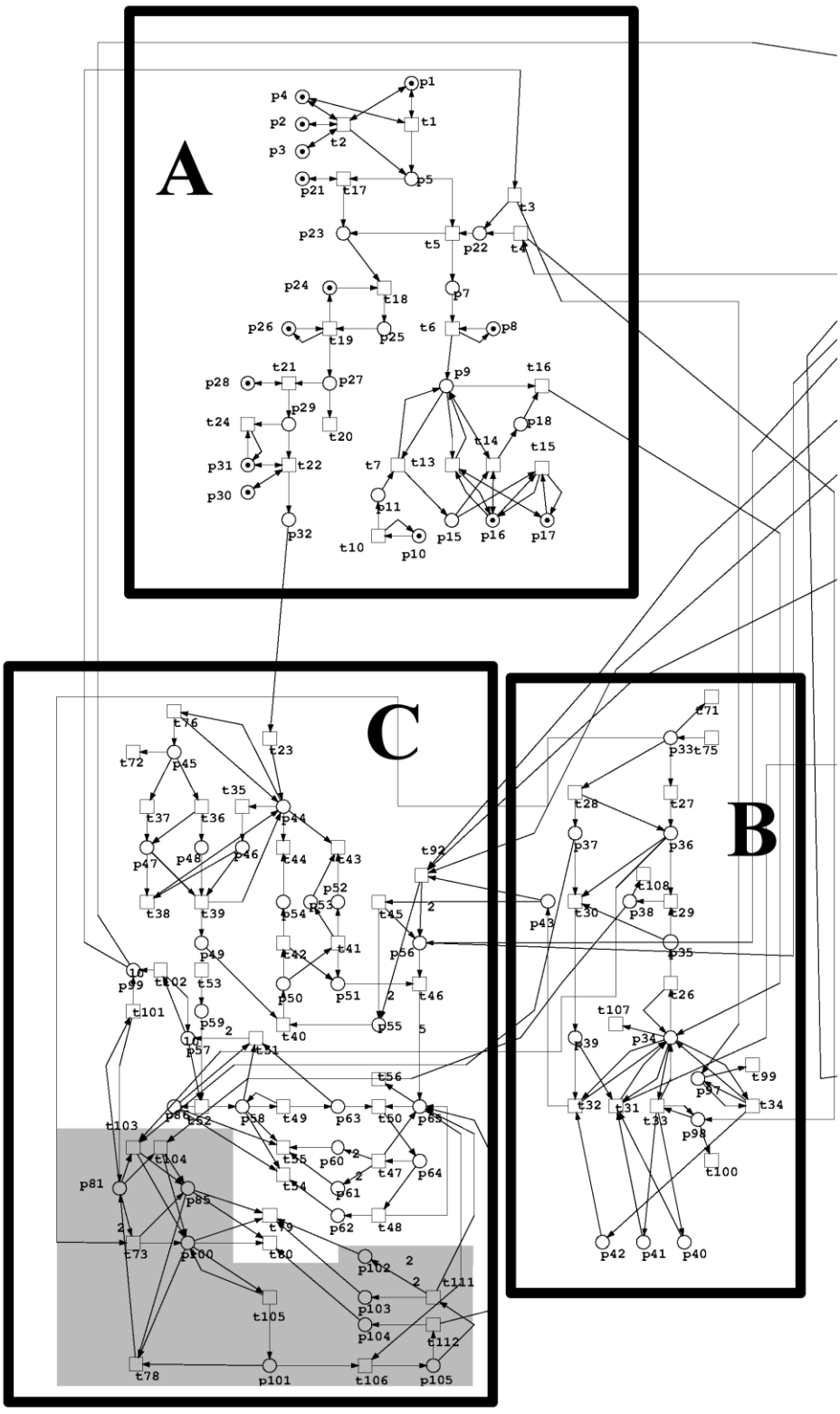


Fig. 3. The first part of the PN model of the IS.

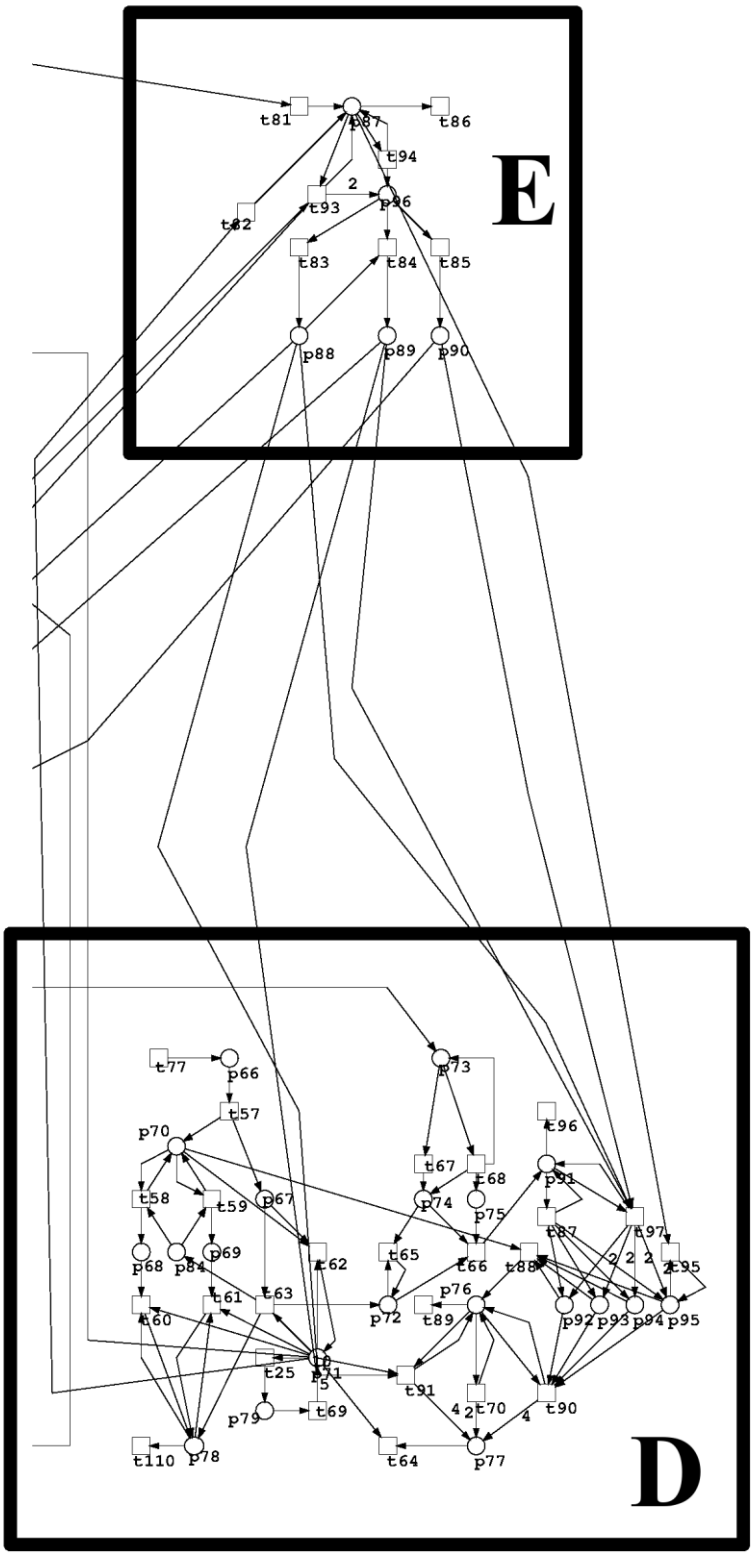


Fig. 4. The second part of the PN model of the IS.

The model was created using public domain software SNOOPY [90]. One should note that this program is not appropriate for the analysis of PN networks. Two types of analysis were performed: the t-invariant analysis and the net simulations. The network simulations' protocol was as follows: the initial state for the simulation is an initial marking, from the enabled transitions set just one is randomly chosen and fired, the marking is changing, then the next enabled transition is chosen. One firing is one step of the simulation and after a given number of steps the simulation stops. During the simulation a marking of a chosen place is monitored and remembered. The same scheme was repeated many times (i.e. five hundred or one thousand) and from all runs the average marking of the chosen place was calculated. All those operations were performed by Java software written by myself. The results of the simulations are shown in the paragraph 2.5. T-invariant and t-clusters were calculated using public domain software Mona Lisa [91]. The output files were processed by my own Java software. MCT-sets were also calculated by the same original software.

A new useful features were added to the model on an advanced stage of this project: fever, HIV infection, AOIS and ASD. All were described above in the paragraph 2.3. Unfortunately, in order to test the majority of them the changes in the network structure are required. However, the changes are rather minute and are limited only to tuning of some selected weights. Furthermore, a number of features which should be added to the model depends on the user, who can modify all relevant transitions or only selected ones.

Table 4. Description of places, which occur in the model.

ID	Place description	ID	Place description
1	Dendritic cell	54	Receptor FasL
2	Fragment of cell	55	Cytokine IL-2
3	Immunosuppressant	56	Cytokine INF-gamma
4	Cytokines	57	Antigen
5	DC in place of inflammation	58	Infected body cell
6	exogenous antigen	59	Body cell
7	Early endosome	60	Perforin
8	Proton pump	61	Granzymes
9	Late endosome	62	Receptor FasL
10	Protease	63	Complex MHC I + antigen
11	Active protease	64	Active Tc
12	Protease	65	New Tc cells
13	Protease	66	pre-B
14	Protease	67	Lymphocyte B with IgM and IgD receptors
15	Antigen after processing	68	Antibody IgM
16	MHC class II	69	Antibody IgD
17	CLIP protein	70	B producing antibodies
18	Complex MHC II + antigen	71	Antigen
19	DC with complex MHC II + antigen on its surface	72	B presenting antigen

21	Endogenous antigen	73	Th2
22	Antigen absorption	74	Receptor TCR
23	Antigen in cytoplasm	75	Molecule C154
24	Ubiquitin	76	B producing other antibodies
25	Protein with ubiquitin	77	Other antibodies: IgG or IgE or IgA
26	Proteasome	78	Connection of antigen and receptor
27	Peptides	79	Antigen that is dividing
28	TAP protein	80	Endogenous antigen
29	Complex antigen + TAP	81	HIV
30	Chaperone proteins	82	Cyclosporine
31	MHC class I	83	Lymphocyte Th, that can't produce cytokines
32	Complex MHC I + antigen	84	Recognition
33	Lymphocyte Th	85	Infection of HIV
34	Dendritic cell with MHC II complex	86	Infection of other virus
35	Complex MHC II + antigen and molecules CD80/86 on dendritic cell's surface	87	Macrophage
36	TCR receptor on Th cell's surface	88	IL-1
37	Molecule CD28	89	IL-6
38	Th in state of anergy	90	TNF-α
39	Lymphocyte Th0	91	Active Th2
40	Cytokine IL-12	92	IL-4
41	Cytokine INF-gamma	93	IL-5
42	Cytokine IL-4	94	IL-13
43	Th1	95	IL-10
44	Dendritic cell	96	Macrophage which produces cytokines
45	Lymphocyte Tc	97	Cellular response
46	Complex MHC I + antigen and molecules CD80/86 on dendritic cell's surface	98	Humoral response
47	TCR receptor on Tc cell's surface	99	Virus (HIV or other)
48	Molecule CD28	100	Cell infected by HIV
49	Initially activated Tc	101	Complex MHC I + antigen on HIV infected cell
50	Activated Tc	102	Perforin
51	Proliferated Tc	103	Granzymes
52	Perforin	104	Receptor FasL
53	Granzymes	105	Active Tc (against HIV infected cell)

Table 5. Description of transitions, which occur in the model.

ID	Transition description	ID	Transition description
1	Receiving signals type 1 of danger	57	Baturation of lymphocyte B
2	Receiving signals type 2 of danger	58	IgM production
3	Endocytosis of endogenous antigen	59	IgD production
4	Endocytosis of exogenous antigen	60	Binding of antigen by antibody
5	Absorption	61	Binding of antigen by antibody
6	pH decreasing	62	Unidentified antigen
7	Processing of protease	63	Identified antigen
8	New antigen	64	Binding of antigen by antibody

9	New antigen	65	Th2 doesn't recognize antigen
10	Activation of protease	66	Th2 recognizes antigen
11	Activation of protease	67	Suitable molecules are present on cell's surface
12	Activation of protease	68	Suitable molecules are present on cell's surface
13	Disconnection	69	Arrival of antigen
14	Connection of antigen and MHC II	70	Antibodies production
15	Absence of reaction between MHC II and antigen	71	No activation
16	Transport to cell's surface	72	No activation
17	Infection	73	Virus infection
18	Ubiquitination	74	Action of immunosuppressive drug
19	Proteolysis	75	New Th lymphocyte arrives
20	Complete degradation	76	New Tc lymphocyte arrives
21	Transport of antigen	77	New B lymphocyte arrives
22	Connection of antigen and MHC I	78	Development of HIV
23	Transport to cell's surface	79	Induction of apoptosis
24	Absence of reaction between MHC I and antigen	80	Induction of apoptosis
25	Division	81	Phagocytosis
26	Suitable molecules are present on cell's surface	82	Phagocytosis
27	Suitable molecules are present on cell's surface	83	Production of cytokine
28	Suitable molecules are present on cell's surface	84	Production of cytokine
29	Connection of receptors without activation (only 1 signal)	85	Production of cytokine
30	Connection of receptors and activation (2 signals)	86	Death of macrophage
31	Th cell's stimulation and proliferation	87	Production of cytokines
32	Th cell's stimulation and proliferation	88	Activation of B cell
33	Cytokines production	89	Death of B cell
34	Cytokines production	90	Production of antibodies
35	Suitable molecules are present on cell's surface	91	Production of antibodies
36	Suitable molecules are present on cell's surface	92	Production of cytokines
37	Suitable molecules are present on cell's surface	93	Activation
38	Connection without activation (only 1 signal)	94	Activation
39	Connection and activation (2 signals)	95	Deactivation
40	Continued activation	96	Death of cell
41	Degranulation	97	Cytokine production
42	Receptor activation	98	Death of DC cell
43	Induction of apoptosis	99	Death of DC cell
44	Induction of apoptosis	100	Death of DC cell
45	Cytokines production	101	Presence of HIV virus
46	Proliferation	102	Presence of virus (not HIV)
47	Degranulation	103	Infection of Th cell by HIV

48	Receptor activation	104	Infection of Th cell by HIV
49	Antigen presentation	105	Antigen presentation
50	Activation	106	Activation
51	Development of antigen	107	Death of DC cell
52	Infection	108	Death of Th cell
53	New body cell	109	Death of Th cell
54	Induction of apoptosis	110	Antigen is killed
55	Induction of apoptosis	111	Degranulation
56	Death of cell	112	Receptor activation

The new functionalities which were added to the model were used to study some events and phenomena. The model is very flexible, new elements can be easily added to check another interactions or impacts of different factors. Those which are presented in the present work gave very interesting results and each problem should be treated as a separate study, however, these case studies can also illustrate the usefulness of the model and the advantages the Petri nets themselves. I want to stress that the model has a potential to extension.

2.5 Simulations of immune system

2.5.1 Fever and ageing

As it was noted in paragraph 2.3 fever does not cause new interactions in the immune response, but only stimulates some mechanisms listed above. Those mechanisms were recognized in the model and the transitions corresponding to them were identified. The appropriate transitions are shown in **Table 5** in red. The effect of fever on the IS model was also presented in the paper [58].

The most straightforward and adequate way to demonstrate a stimulating effect of fever on the transitions is changing selected weights. An increase in the weights of the arcs leading from transitions to places reflects the quantitative effect of the fever. The mechanism of IS activity remains the same as observed in a normal body temperature, however greater weights simulate a greater efficiency of the mechanisms.

One should note that the accurate statistical data on how much the fever affects the immune response are not known yet, so experimental information how much the weights should be changed is not available. As a result, the weights in PN IS model were changed arbitrarily, by the factor of two. Once firm data are available, these weights may be easily adjusted.

It was crucial to monitor the populations of viruses and Tc cells during the immune response. A number of simulations of the model were performed, during a viral infection both without elevated body temperature ("normal" state) and with fever. In order to simulate the fever the weights of transitions affected by the fever selected from the part of the model related to the cellular response were increased

by the factor of two. These transitions are: t32, t39, t46, t50 - all shown in red in **Table 5**. For each of the fever-affected transitions a set of simulations was performed and one extra simulation with the effect of the fever on all selected transitions. During the simulations the numbers of viruses (results shown in the **Fig. 5a**) and Tc cells (**Fig. 5b**) were calculated.

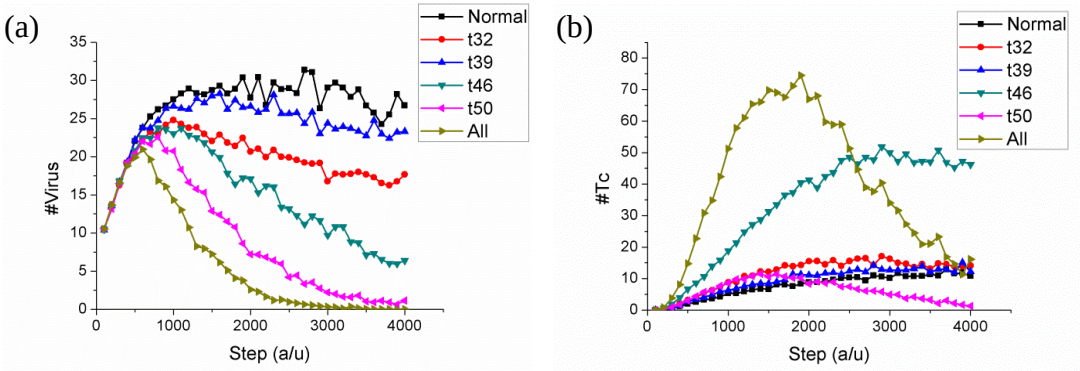


Fig. 5. The results of the simulations of the model with and without fever (a model without "macrophages" part). Different transitions connected with the fever were used. (a) A time evolution of the virus population during the fever. (b) The effects of the fever on the number of Tc lymphocytes.

Looking at **Fig. 5a** one can see that without fever the number of viruses is the highest. It can be even inferred that the IS cannot stop the infection. However, in the **Fig. 7** the time of simulation is longer and finally the number of viruses is reduced. The lowest impact on the number of viruses is observed when changing the weight of the transition t39 – the connection and activation of the Tc lymphocytes. The middle-level reduction of the number of viruses is seen for transitions t32 - Th1 cell's stimulation and proliferation as more Th1 cells stimulate Tc cells better and for t46 - proliferation of Tc cells. As a result it seems obvious that more Tc lymphocytes will kill more viruses. The transition t50 has the biggest effect in the model, which denotes a direct, temperature induced, activity of lymphocytes Tc during killing the viruses. Furthermore, the infection is defeated the most quickly when all the assumed effects of fever are present. The general result is that the fever has reduced the number of viruses and helped to stop the infection.

In **Fig. 5b** the number of Tc lymphocytes is shown for comparison. The biggest amount of Tc cell is observed when all effects of the fever are present. It is obvious because almost all transitions affected by the fever result in an increasing number of Tc cells. What is interesting is the decreasing amount of Tc lymphocytes. However, if **Fig. 5b** is compared with **Fig. 5a,** it can be postulated that from some point of time during the simulation the virus is not present in the system anymore, so the immune response is inhibited and the number of Tc cells should be reduced. A large impact on the number of Tc lymphocytes is observed for the fever effect of transition t46 - this transition represents proliferation of Tc cells, so a bigger number of Tc cells is normal. The number is still quite large even at the end of

simulation, because the infection is still present in the model (**Fig. 5a**). For the transitions t32 and t36 the number of Tc cells is only slightly bigger than in the case without the fever. Those results correspond to the **Fig. 5a**. What is interesting is the effect of the fever observed in the number of Tc cells for the transition t50, as initially the number of Tc cells was similar to the case without the fever. It can be explained by the fact that the transition t50 does not have any impact on the number of Tc cells but only on their activity. Later the number of Tc cells decreases but it corresponds to the results from **Fig. 5a** - the infection is controlled.

The process of ageing has diverse effects on IS. Thus, the elements affected by age were added to the model using different methods depending on the type of the change in the IS. A reduction of the production of IL-2 and INF-γ was introduced by changing weights between p43, t45 and t92. As a result of those changes two Th lymphocytes in the "old" IS will produce the same amount of cytokines like the ones in the "young" IS. The number of native T and B lymphocytes has been decreased due to additional transitions, which can consume tokens from places p33, p45 and p66. Those places represent native lymphocytes and are input places for the newly added transitions. The transitions do not have other input places and any output place. A similar method was used to reduce the expression of CD28 and TCR receptors. New, "consuming" transitions were added to the model. Places p47 and p48, p36 and p37, which represent CD28 and TCR receptors, became input places for those new transitions. The most difficult task was the introduction of memory Tc cells to the model. The activity of the memory cells is quite a complex part of the real immune system and its full representation in the model would require large changes in the PN. In order to avoid such remodeling a simple yet working mechanism was suggested, namely a new transition which can only add tokens to the places p65 representing mature Tc cells, has been added. This mechanism does not strictly represent the nature of ageing processes, it only provides a desirable effect.

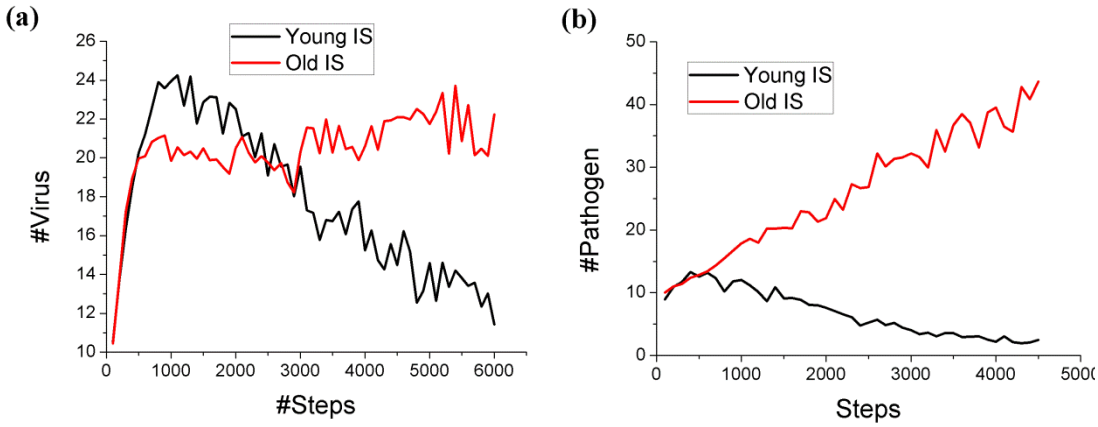


Fig. 6. The number of viruses (a) and pathogens (b) calculated during simulations of the cellular (a) and humoral (b) response in the model of "young" and "old" IS.

Simulations for the cellular and humoral response in the model of "young" and "old" IS were performed. The results are presented in **Fig. 6**. The model of the "young" IS is our standard model with the "macrophages" part added, used in all simulations presented later on in this chapter. The model of the "old" IS is the modified model described in this section. One can see that the model of the "young" IS, in both cellular and humoral responses, can control an infection, which is manifested by a decreasing the number of pathogens. However, in the model of the "old" IS the infection cannot be controlled, especially in the humoral response (**Fig. 6b**) case. In the cellular response the situation is better, probably due to a positive effect of the memory Tc cells. However, even this positive effect is not strong enough to defeat the infection.

2.5.2 AIDS and AOIS

The HIV infection is added to the model by inserting new places and transitions. They are shown in **Fig. 3** on a grey background. This part of the model contains places and transitions which represent an infection of Th cells by the HIV virus, the virus replication and possible killing of the HIV virus by the immune system. The introduction of the HIV infection to the model is done by changing the initial marking M_0. If one wants to add the HIV virus to the model, tokens have to be added to the place p81, which represents the HIV virus. When the place p81 is empty the HIV virus is not present in the immune system and other properties can be studied without the influence of the HIV infection.

Such a method of the introduction of the HIV infection to the model was chosen as the most reasonable one. The simplest and the most obvious way to add the HIV to the PN of IS model is adding a new place and a transition, or maybe a few transitions, which will be connected to places corresponding to the Th lymphocytes and will consume tokens from them. It was also the base of the HIV infection in the model. The disadvantage of such an approach is the absence of the reaction of IS. In the model such reaction has been added.

Let us recall that the AOIS disease is caused by the lack of cytokine INF-γ. The cytokine is bound by abnormal antibodies. In order to model the disease in PN probably the simplest method is to add a new place, which represents the abnormal antibodies and a transition which represents the binding to INF-γ. This mechanism was tested and the results are presented in **Fig. 7** as a cyan line (labeled as "AOIS new trans"). However, in the paper [74] the results on the amounts of the INF-γ in healthy and ill subjects were presented and such a simple approach in the model would not be able to represent those numbers. That is why the other method was also tasted, namely, the weight from the place p56 (represents the INF-γ) to the transition t46 was increased from 1 to 4. It means that for four copies of INF-γ one is active and can induce the reaction. Those numbers are in accordance with the results from [74]. The results from the simulations are shown in **Fig. 7** as a black line (AOIS w4). The number of viruses increases very fast, but it is not a linear

growth, around one thousand of steps an inflection point can be observed and after the inflection point the growth is slower. However, even this result was not satisfactory because of fast growth. Thus, two other cases were studied: the weight between the place p56 and the transition t46 was set to 2 and 3. In the model with the weight 2 the number of viruses is the smallest from all AOIS studied cases. In **Fig. 7** it is shown as a light blue line (AOIS w2). The inflection point is observed around the same number of steps, but it is more explicit. For the weight 3 the number of viruses grows faster than for the model with the weight of 2 and slower than for the weight 4 and it is presented as red line in **Fig. 7** (AOIS w3). An interesting observation is a non-linear correlation between the weight and the number of viruses. Changing the weight from 2 to 3 causes large changes in the amount of viruses. A further change of the weight causes much smaller changes in the number of viruses.

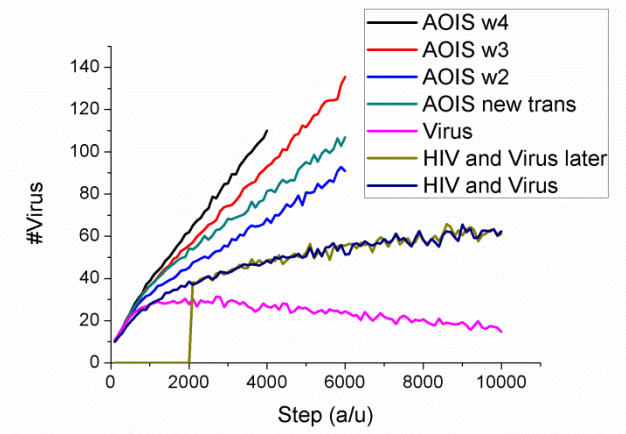


Fig. 7. Time evolution of the virus population during AIDS and AOIS diseases modeled by a PN model of the immune system.

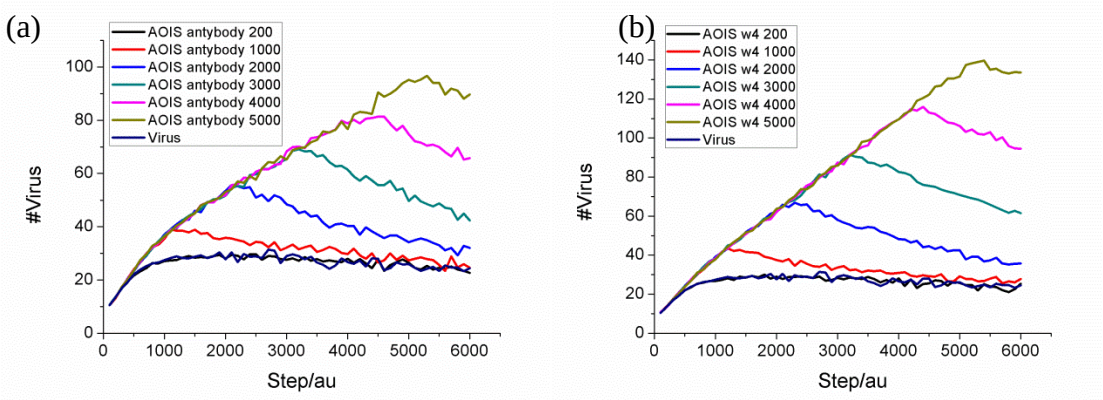


Fig. 8. The time evolution of the virus population during AOIS, which was introduced to the model by (a) new transition, (b) changing of the weight of the arc between the place p56 to the transition t46 to 4, and during the INF-γ treatment. Different moments of starting the treatment are presented – the number in the legend shows the step number of the simulation when the INF-γ started to affect the model.

In **Fig. 7** we can also see the time evolution of the virus population in other situations. The pink line corresponds only to the presence of the virus (other, not HIV) and the normal immune response. As one can see, initially the number of viruses groves, but later it is decreased by the immune response. During the infection with the HIV virus (dark blue and brown lines) the number of viruses grows and at the end of the simulation is it significantly larger than during the normal immune response. It should be underlined that only the number of "normal" viruses (not HIV) is calculated. In summary, when the IS is healthy it can defeat an infection, but during the HIV infection (leading to AIDS) or AOIS the system is not able to control the infection.

Our model was used to check if INF-γ treatment would inhibit the AOIS symptoms. The following protocol was applied: three types of AOIS introduction into the model were chosen – (i) a new transition which represents the binding of abnormal antibodies, (ii) the weight of 4 of the arc from the place p56 to the transition t46 and (iii) the weight of 2 of the same arc. During the simulation the cytokine INF-γ was added to the model by changing a marking of the place p56 every one hundred steps of the simulation. Different moments during the simulations when the INF-γ was for the first time added to the model as well as different amounts of INF-γ were tested. The results are shown in **Fig. 8**.

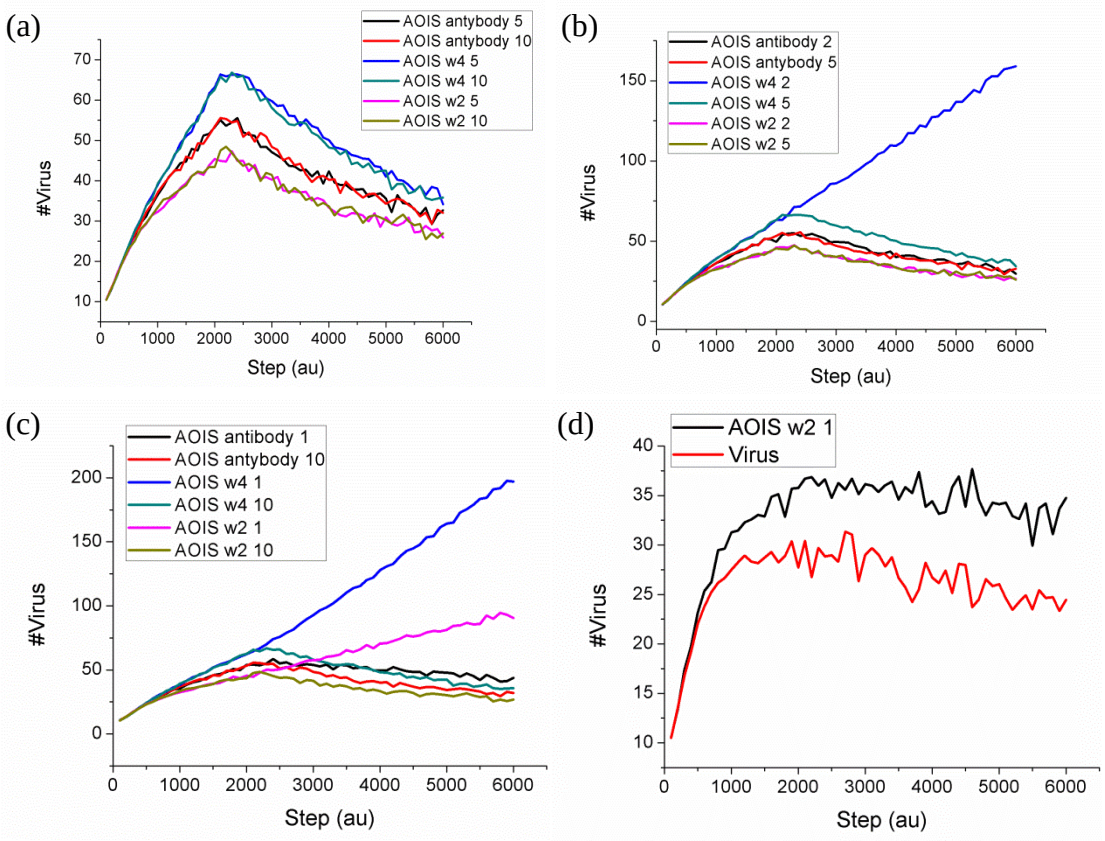


Fig. 9. The amount of a virus during simulations observed for three different ways of AOIS introduction to the model (see the text) and for different doses of INF-γ.

In **Fig. 8** the amount of the virus monitored during two types of simulation is presented. In the panel (a) the AOIS was added to the model by a new transition, in (b) by changing the weight of an arc between the place p56 and the transition t46 (from 1 to 4). Here presented are the data for simulations with the changing time points when the INF-γ treatment was added. The results show the IS can indeed control the infection if the additional INF-γ is added to the model. We see that the amount of viruses is reduced and it goes back close to the level observed during the normal immune response. Similar results were observed for the changing of the weight of arc between place p56 and transition t46 from 1 to 2. For all three ways of AOIS modeling 10.000 steps long simulations were performed for one particular case, when the INF-γ treatment was stated after 2000 steps. Long simulations results confirm the observations from the shorter simulations (results not shown). It should be noted that the model presents only the IS functions, it does not consider other body systems and it cannot give an answer which amount of viruses may be lethal. It is possible that the model shows an optimistic state that the IS can still control the virus infection when the INF-γ treatment was started, but the infection could already have destroyed other body systems.

Interesting results were seen when varying amounts of INF-γ were added to the model. Four cases were checked. The default amount of INF-γ added to the model was ten and results in **Fig. 8** are calculated for that dose of INF-γ. Doses five, two and one were also tested. All three modes of AOIS presence in the model were checked and the same staring point of the simulated treatment was used. The results are presented in **Fig. 9**. The simulations show that for INF-γ amount of 10 and 5 the results are the same - the IS can control the virus infection - **Fig. 9a**. In **Fig. 9b** the data from simulations with dose of INF-γ five and two are compared. One can see that for the weight four set to the arc between the place p56 and the transition t46 the dose of two is not sufficient. The same is observed for a dose one in combination with the weight two of this arc. For the test with the new transition, which represents the anti-INF-γ antibodies the results show that any dose is sufficient, but this case was studied more deeply. The number of viruses in the healthy IS (red line, **Fig. 9d**) are compared to the results of the IS modeling with AOIS introduced by the new transition. It was assumed that the INF-γ treatment had started after two hundred steps of simulation and the INF-γ dose was set to one (black line, **Fig. 9d**). One can see that although the IS with AOIS can control the infection, the level of viruses is higher than in the healthy IS.

The general conclusion from this computational experiments is the following: the immune system can function well when the INF-γ is applied, even when it is a long time after the infection onset. However, the correct IS functioning depends on the dose of INF-γ in the organism.

2.5.3 ASD

There are strong evidences of a correlation between Autism Spectrum Disorder (see paragraph 2.3) and the immune system. The PN IS model was used to test how levels of cytokines are changing during fever and ASD. I have addressed the basic question: is it possible that those changes are responsible for transient improvements of condition of autistic children? Cytokines whose levels were reported to be elevated in autistic cases are presented in **Table 3**. Unfortunately, data from the literature are often inconsistent [81] and so many different parameters will make outcomes from the simulations unclear. Three cytokines were chosen to be elevated in the model of ASD: IL-1, IL-6 and TNF-α. This selection is consistent, all chosen signaling molecules are produced by macrophages, they are also discussed in the papers [83, 85]. Because the chosen cytokines are produced by macrophages, the macrophage activity should have been added to the IS model. Reactions of macrophages are represented in the part (E) of the model (**Fig. 4**) and it shows a phagocytosis of the antigen and the production of cytokines by the macrophages.

Four types of PN IS simulations were preformed: (1) non-autistic, (2) non-autistic with fever, (3) autistic, (4) autistic and fever, each four thousands steps long. During the simulations the numbers of the following cytokines were calculated: INF-γ and IL-1, IL-6, TNF-α - last three were chosen to be elevated in the autistic case. The results are shown in **Fig. 10**. Those results were also presented in our paper [59].

In **Fig. 10a** an average amount of (probably) the most important cytokine IL-1 is presented for all the cases studied. In almost every simulation both the cellular and humoral responses were present, otherwise it is denoted in the text. One can see that the amount of IL-1 in non-autistic cases with fever and without it is similar and significantly smaller than the level in the autistic model. Indeed, the amount of IL-1 in the autistic case grows constantly. However, in an autistic subject with fever the concentration of IL-1 does not grow constantly, but achieves a smaller constant level than in the case without fever and more similar to the non-autistic cases. IL-1 exhibits pro-inflammatory activity and according to [82, 92] a high level of IL-1 is not good for the nervous system. In conclusion, for IL-1 the effect of fever is positive.

However, one can see in **Fig. 10a** that in the autistic subject the problem of homeostasis is present. Two new transitions were added to the model, which represent collectively all remaining mechanisms of degradation/inactivation of cytokines INF-γ and TNF-α, respectively. Results from the simulations of the model with those two new transitions were shown in **Fig. 10b**. Also, the results presented for other cytokines (Fig c-f) were obtained from the modified model. In **Fig. 10b** the general behavior of the levels of IL-1 is the same like in **Fig. 10a**, however the concentration in the autistic case without fever is more stable.

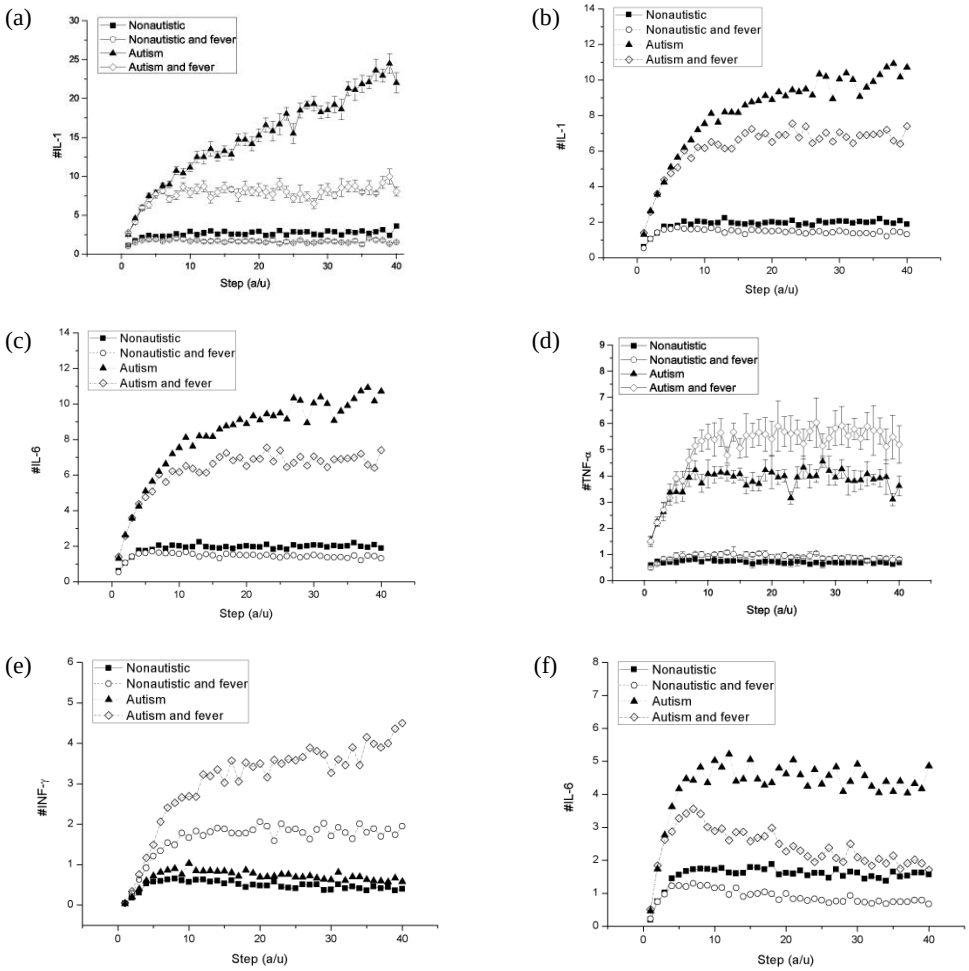


Fig. 10. Results of ASD simulations: (a) levels of IL-1, bars represent statistical errors (b) levels of IL-1 with additional transitions, (c) levels of IL-6, (d) levels of TNF-α, (e) levels of INF-γ, (f) levels of IL-6 but only during the humoral response (from [59]).

In **Fig. 10c** the amount of IL-6 is presented. The plots are very similar to the IL-1 simulations, which is clear because the production of IL-6 depends on the level of IL-1 [61]. IL-6 also may be dangerous for the nervous system and cause the inflammation state in the Central Nervous System [82, 92]. Therefore, the effect of fever is beneficial in the model. However, this case was monitored also in much longer simulations (data not shown) and still the homeostasis did not occur for all cases studied. Probably our straightforward method of IL-6 destruction is too simple.

The next cytokine observed during the simulations was TNF-α. In **Fig. 10d** data shows that in "autistic" simulations with and without the fever the amount of the cytokine is significantly bigger than in non-autistic cases. Other observation is a larger impact of the fever on the autistic mode, in healthy subjects the TNF-α level increases to a much lower extent, however in both cases the direction of changes

are qualitatively the same. According to [82, 92], TNF-α is pro-inflammatory cytokine and it may be dangerous for the central nervous system, so the results from the simulations are not beneficial for ASD.

INF-γ was not chosen to be elevated in ASD so in non-autistic and autistic cases without the fever observed levels of INF-γ are very similar. The fever induces an increasing amount of INF-γ in both cases, however the impact of fever is bigger in an autistic subject (**Fig. 10e)**. According to [92] the INF-γ is a neuroprotective cytokine and it is beneficial for the central nervous system so changes observed in the simulations are positive.

In **Fig. 10f** the amount of IL-6 (during the humoral response exclusively) is presented. Without the fever in the autistic case the level of IL-6 is significantly higher than in the non-autistic study without the fever. When the effect of fever is present, the level of IL-6 for a non-autistic subject increases, however for the autistic case the level of IL-6 decreases and it is almost the same as for healthy cases. That is a promising result, better than the results from simulations with both immune responses present. The presence of only the humoral response is possible, and it occurs during normal infections and depends on type of the antigen. Observations [86] show that the fever does not always improve the state of autistic children. Perhaps in some cases only a part of IS responses to the inflammation or another pirogen, and thus the change in the behavior is seen. The hypothesis is that the improvement in mental abilities of autistic children is more significant when only the humoral response is present. Verification of such a statement requires further studies and more experimental data.

The general findings in the PN IS model are: the fever changes the amount of cytokines and usually brings it closer to the level observed in healthy children or the changes are qualitatively the same for the healthy and autistic children (for TNF-α). However, we still have problems with keeping a homeostasis in the model.

Towards genetic test for ASD susceptibility

Obviously this chapter is not directly linked to my thesis topics but I decided that it is worth mentioning. During the preparation of this thesis, and my participation in NCN founded project (N519 578138) I have learnt that ASD has a strong genetic component. Together with other colleagues, I have developed a framework for information search on new ASD related genes. We have initiated a project (GENIUS) aimed at finding an inexpensive biochemical, PCR based, test checking selected genes. The project received the approval of the medical bioethics committee (KB 402/2014). The first mutations have been selected and the first blood samples were tested. The GENIUS project is carried out in cooperation with the *Pracownia Genetyki Nowotworów w Toruniu* lab and the Centre for Modern

Interdisciplinary Technologies of Nicolaus Copernicus University in Torun, with the financial support from the kujawsko-pomorskie voivodeship.

2.6 T-invariants analysis

The analyses of t-invariants of the IS model were performed and our model had a CTI property. Forty seven t-invariants were found and biological meanings of all were studied. T-invariants, transitions which are included in the t-invariant and the biological meaning of each t-invariant are presented in **Table 6**.

Table 6. Description of t-invariants of the PN model of the immune system.

T-inv	Transitions	Biological meaning
Inv0	t76, t72,	Tc cell arrives and isn't active.
Inv1	t26, t65, t28, t75, t31, t34, t67, t30,	Th cell recognizes antigen but B cell doesn't
Inv2	t75, t71,	Th cell arrives and dies
Inv3	t26, t75, t27, t108, t29,	Th cell doesn't recognize antigen (only 1 signal) and dies
Inv4	2*t79, 4*t75, 2*t78, 4*t73, 3*t105, t106, t111,	Tc cell kills HIV virus which kills Th cell.
Inv5	t26, t28, t75, t32, t33, t30, t74,	Th cell activates and it is destroyed by immunosuppressant
Inv6	2*t79, 4*t28, 4*t75, 4*t103, 2*t78, 3*t105, t106, t111,	Tc cell kills HIV virus which kills Th cell.
Inv7	2*t96, 2*t63, t97, 2*t68, 2*t57, 2*t110, 2*t58, 2*t77, t84, 2*t88, 2*t93, 2*t66, 2*t83, t85, 2*t89, t25, 2*t60, t69,	B cell recognizes antigen, antigen divides. B cell activated by macrophage's cytokines produces antibodies and dies.
Inv8	2*t61, 2*t96, 2*t63, t97, 2*t68, 2*t57, 2*t110, 2*t59, 2*t77, t84, 2*t88, 2*t93, 2*t66, 2*t83, t85, 2*t89, t25, t69,	B cell recognizes antigen, antigen divides. B cell activated by macrophage's cytokines produces antibodies and dies.
Inv9	2*t107, t26, 2*t99, t28, t32, 2*t102, t33, t30, 4*t94, t84, 2*t46, t92, 2*t43, 2*t41, 2*t83, t85, 2*t40, 2*t39, 2*t35, 2*t36, 10*t56, 2*t3, t75, 2*t76, 2*t1,	Virus activates Th and Tc cell stimulated by cytokines produced by macrophage.
Inv10	t48, 2*t52, 2*t49, t50, t51, 2*t53, t54,	Tc cell kills virus.
Inv11	4*t52, t47, 3*t49, t50, 2*t51, 4*t53, 2*t55,	Tc cell kills virus.
Inv12	2*t107, 2*t99, t26, t28, t32, 2*t102, t33, t30, 4*t94, t84, t92, 2*t46, 2*t44, 2*t83, 2*t42, t85, 2*t40, 2*t39, 2*t35, 2*t36, 10*t56, 2*t3, t75, 2*t76, 2*t1,	Virus activates Th and Tc cell stimulated by cytokines produced by macrophage.
Inv13	4*t64, t87, t90, t25, t69,	Active B cell produces antibodies, their bind to antigen. Antigen divides.
Inv14	4*t82, 4*t86, t25, t69,	Antigen (humoral) is killed by macrophage
Inv15	2*t61, 2*t96, 2*t63, 2*t68, 2*t57, 2*t110, 2*t59, 2*t77, 2*t88, 2*t87, 2*t66, 2*t89, t25, t69,	B cell recognizes antigen, antigen divides. B cell produces antibodies and dies.

Inv16	t81, t102, t95,	Macrophage kills virus.
Inv17	t84, 8*t64, t97, 2*t93, 2*t83, t85, 2*t90, 2*t25, 2*t69,	Antigen divides and it is killed by antibodies. Production of the antibodies was stimulated by macrophage's cytokines.
Inv18	t80, 2*t75, t78, 2*t73, t112, 2*t105, t106,	Tc cell kills HIV virus which kills Th cell.
Inv19	t81, t86, t102,	Macrophage kills virus.
Inv20	2*t107, 2*t99, t26, t28, t32, 2*t102, t33, t30, t84, t92, 2*t46, 2*t93, 2*t44, 2*t83, 2*t42, t85, 2*t40, 2*t39, 2*t35, 2*t36, 10*t56, 2*t3, t75, 2*t76, 2*t1,	Virus activates Th and Tc cell stimulated by cytokines produced by macrophage.
Inv21	2*t107, t26, 2*t99, t28, t32, 2*t102, t33, t30, t84, 2*t46, t92, 2*t43, 2*t93, 2*t41, 2*t83, t85, 2*t40, 2*t39, 2*t35, 2*t36, 10*t56, 2*t3, t75, 2*t76, 2*t1,	Virus activates Th and Tc cell stimulated by cytokines produced by macrophage.
Inv22	2*t70, 4*t64, t25, t69,	Antigen divides and it is killed by antibodies
Inv23	t76, t38, t37, t35,	Tc cell doesn't recognize antigen.
Inv24	2*t107, t26, 2*t99, t28, t32, t33, 2*t101, t30, 4*t94, t84, 2*t46, t92, 2*t43, 2*t41, 2*t83, t85, 2*t40, 2*t39, 2*t35, 2*t36, 10*t56, 2*t3, t75, 2*t76, 2*t1,	HIV activates Th and Tc cell stimulated by cytokines produced by macrophage.
Inv25	t62, t57, t77,	New B cell doesn't recognize antigen.
Inv26	t80, 2*t28, 2*t75, 2*t103, t78, t112, 2*t105, t106,	Tc cell kills HIV virus which kills Th cell.
Inv27	t2, t1,	Antigen dies before recognition.
Inv28	4*t82, 4*t95, t25, t69,	Antigen divides. Macrophage kills antigen. Macrophage becomes inactive.
Inv29	4*t26, 2*t79, 4*t75, 4*t27, 2*t78, 4*t29, 4*t104, 3*t105, t106, t111,	Tc cell kills HIV virus which kills Th cell.
Inv30	2*t96, 2*t63, 2*t68, 2*t57, 2*t110, 2*t58, 2*t77, 2*t88, 2*t87, 2*t66, 2*t89, t25, 2*t60, t69,	B cell recognizes antigen, antigen divides. B cell produces antibodies and dies.
Inv31	2*t107, 2*t99, t26, t28, t32, 2*t101, t33, t30, 4*t94, t84, t92, 2*t46, 2*t44, 2*t83, 2*t42, t85, 2*t40, 2*t39, 2*t35, 2*t36, 10*t56, 2*t3, t75, 2*t76, 2*t1,	Virus activates Th and Tc cell stimulated by cytokines produced by macrophage.
Inv32	4*t4, 4*t107, 4*t5, 4*t1, t25, t69, 4*t100,	DC cell finds antigen (humoral), presents it and dies.
Inv33	2*t96, 2*t63, t97, 2*t68, 2*t57, 2*t110, 2*t58, 4*t94, 2*t77, t84, 2*t88, 2*t66, 2*t83, t85, 2*t89, t25, 2*t60, t69,	B cell recognizes antigen, antigen divides. B cell activated by macrophage's cytokines produces antibodies and dies.
Inv34	2*t61, 2*t96, 2*t63, t97, 2*t68, 2*t57, 2*t110, 2*t59, 4*t94, 2*t77, t84, 2*t88, 2*t66, 2*t83, t85, 2*t89, t25, t69,	B cell recognizes antigen, antigen divides. B cell activated by macrophage's cytokines produces antibodies and dies.
Inv35	t84, 8*t64, t97, 2*t83, t85, 2*t90, 4*t94, 2*t25, 2*t69,	Antigen divides and it is killed by antibodies. Production of the antibodies was stimulated by macrophage's cytokines.
Inv36	t80, 2*t26, 2*t75, 2*t27, t78, 2*t29, 2*t104,	Tc cell kills HIV virus which kills Th cell.

	t112, 2*t105, t106,	
Inv37	t84, 4*t64, 2*t83, 3*t94, t91, t25, t69,	Antigen divides and it is killed by antibodies. Production of the antibodies was stimulated by macrophage's cytokines.
Inv38	2*t84, 8*t64, 3*t93, 4*t83, 2*t91, 2*t25, 2*t69,	Antigen divides and it is killed by antibodies. Production of the antibodies was stimulated by macrophage's cytokines.
Inv39	t81, t101, t95,	Macrophage kills virus.
Inv40	t81, t86, t101,	Macrophage kills virus.
Inv41	t107, t26, t99, t28, t32, t33, t101, t30, 5*t56, t3, t46, t75, t76, t44, t42, t40, t39, t35, t36, t1, t45,	HIV virus activates DC and Th cell, Th cell activates Tc cell. Tc cell dies.
Inv42	t107, t26, t99, t28, t32, t102, t33, t30, 5*t56, t3, t46, t75, t76, t44, t42, t40, t39, t35, t36, t1, t45,	Virus activates Th and Tc cell
Inv43	t107, t26, t99, t28, t32, t33, t101, t30, 5*t56, t3, t46, t75, t76, t43, t41, t40, t39, t35, t36, t1, t45,	HIV activates Th and Tc cell
Inv44	t107, t26, t99, t28, t32, t102, t33, t30, 5*t56, t3, t46, t75, t76, t43, t41, t40, t39, t35, t36, t1, t45,	Virus activates Th cell and Tc cell.
Inv45	2*t107, t26, 2*t99, t28, t32, t33, 2*t101, t30, t84, 2*t46, t92, 2*t43, 2*t93, 2*t41, 2*t83, t85, 2*t40, 2*t39, 2*t35, 2*t36, 10*t56, 2*t3, t75, 2*t76, 2*t1,	HIV activates Th and Tc cell stimulated by cytokines produced by macrophage.
Inv46	2*t107, 2*t99, t26, t28, t32, 2*t101, t33, t30, t84, t92, 2*t46, 2*t93, 2*t44, 2*t83, 2*t42, t85, 2*t40, 2*t39, 2*t35, 2*t36, 10*t56, 2*t3, t75, 2*t76, 2*t1,	HIV activates Th and Tc cell stimulated by cytokines produced by macrophage.

All t-invariants have the biological meaning so no new processes were discovered. It is also strong condition for the validity of the model. T-invariant analysis allowed to find one mistake in the model. The mistake was connected to the HIV introduction to the model and it manifested in the presence of the meaningless t-invariants. The error had been corrected before the simulations were performed.

Many t-invariants are very similar and they differ only in a few transitions, for example, Inv20 and Inv31 or Inv35, Inv37 and Inv38. That similarity is caused by equivalent biochemical mechanisms. A good idea was to merge those similar t-invariants into t-clusters. It will make t-invariants data more clear and easier to analyze. To create t-clusters the UPGMA algorithm was used with a cut parameter of 35%. Thirteen t-clusters were generated, see **Table 7**. All t-clusters have the precise biological meaning.

Some t-clusters contain multiple t-invariants and they specifically describe the main processes during the immune response, for example, Cluster7 or Cluster10. Other t-clusters (i.e. Cluster12 , Cluster11) describe “non-standard” processes

occurring during the immune response, such as a presence of an antigen not leading to the immune response.

Table 7. Description of t-clusters generated from t-invariants of the PN model of the immune system.

Cluster	T-invariants	Biological meaning
Cluster1	Inv11, Inv10	Tc cell kills virus.
Cluster2	Inv40, Inv39, Inv19, Inv16	Macrophage kills virus.
Cluster3	Inv32	DC cell finds antigen (humoral), presents it and dies.
Cluster4	Inv28, Inv14	Antigen (humoral) is killed by macrophage
Cluster5	Inv38, Inv37, Inv35, Inv17, Inv22, Inv13	Antigen divides and it is killed by antibodies.
Cluster6	Inv25	New B cell doesn't recognize antigen.
Cluster7	Inv30, Inv15, Inv34, Inv8, Inv33, Inv7	The humoral response: B cell recognizes antigen, antigen divides. B cell, activated or not by macrophage's cytokines, produces antibodies and dies.
Cluster8	Inv36, Inv29, Inv26, Inv18, Inv6, Inv4	Tc cell kills HIV virus which kills Th cell.
Cluster9	Inv3	Th cell doesn't recognize antigen (only 1 signal) and dies
Cluster10	Inv44, Inv43, Inv42, Inv41, Inv46, Inv31, Inv20, Inv12, Inv45, Inv24, Inv21, Inv9	The cellular response.
Cluster11	Inv5, Inv1	Th cell recognizes antigen but immune response isn't present.
Cluster12	Inv23	Tc cell doesn't recognize antigen.
Cluster13	Inv0	Tc cell arrives and isn't active.

The t-cluster tree is presented in **Fig. 11**. The tree contains three main branches and it consists of Cluster1, a middle tree from Cluster2 to Cluster7 and a top tree from Cluster8 to Cluster13. Cluster1 contains t-invariants represented in the last part of the cellular response: the killing of the antigen by Tc cell. The middle tree can be divided into Cluster2, which describes the killing of viruses by macrophages and into a part which is correlated with a humoral antigen and the humoral response. The top tree contains the t-invariants which are associated with Th cell activation, maturation of the Tc cells and other processes present during the cellular response. The plot shows that the macrophage's response in the model is more similar to the humoral response than to the cellular response. Other finding is that the cellular response is divided into two parts: the first connected with Th cells containing an activation of Th and Tc lymphocytes and the second, which contains only mature Tc cells killing infected body cells.

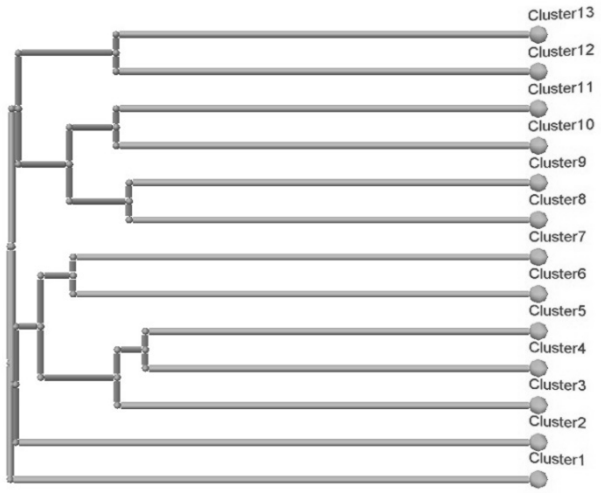


Fig. 11. The cluster tree describes relations between t-clusters.

MCT-sets of the model were also calculated and they are presented in **Table 8**.

Table 8. MCT-sets of the Petri net model of the IS.

Set	Transitions	Set	Transitions
MCT0	t36, t39, t40, t46, t56, t99, t3	MCT10	t54, t48
MCT1	t5, t100, t4	MCT11	t50, t51, t52, t53, t49
MCT2	t69, t25	MCT12	t77, t57
MCT3	t29, t27	MCT13	t60, t58
MCT4	t34, t65, t67, t31	MCT14	t61, t59
MCT5	t33, t32	MCT15	t66, t68, t88, t89, t96, t110, t63
MCT6	t38, t37	MCT16	t105, t106, t78
MCT7	t43, t41	MCT17	t111, t79
MCT8	t44, t42	MCT18	t112, t80
MCT9	t55, t47	MCT19	t84, t83

All MCT-sets were analyzed, any new features of the model were not detected. If the creation of the model is based on the literature, the MCT-sets represents knowledge which is embedded into the model during its construction and it is not so interesting.

2.7 Conclusions

The PN model of the immune system was presented in this chapter. This model confirms that Petri nets are useful and practical tool for modeling complex biological systems. The models created are intuitive and flexible, new elements can be easily added. In the IS model a few features and diseases have been added (effects of fever and ageing, AIDS, AOIS, ASD), and it has been used to study

their impact on the immune system itself. In some cases, for example for ASD, interesting observations have been made correlating fever with cytokines levels. The t-invariant analysis of the IS model was also performed. This method allows for the verification of PN models and the IS model passed the verification.

The model may be used in modeling other, even more complex phenomena, for example, chemical immunosuppression during organ transplants.

Chapter 3. Simulations of Petri nets using GPU

3.1 Introduction

The simulation algorithms presented in the paragraph 1.3 are not very efficient, especially for large networks. For such Petri nets the number of steps of the simulation has to be very large because when the number of transitions is high, the simulation must be very long in order to give all transitions an opportunity to fire at least a few times. As the complexity of the simulation is *O(k n m)*, then for bigger networks the simulation is longer.

In many studies quite small Petri nets are sufficient, as for example in the model of the immune system presented in Chapter 2. However, in Chapter 5 large networks can be easily generated, especially by the OPOA algorithms (paragraph 5.2.1). Simulations of such a big system will be very time-consuming, which is why a new way of the PN simulation was developed.

In this chapter a new algorithm of the simulation of the Petri net, called PINGU (Petri IN Graphical Unit), is described. It is based on the observation that firing of a transition can be parallel if the enabled transitions are not in conflict. Parallelization gives the best effects when the parallel code is executed on many processors or by many threads at the same time. That is the reason why programming on the GPUs (Graphical Processing Units) is used - the graphical cards are quite cheap and provide a great number of cores which can execute instructions in parallel.

This way of research is relatively new and unexplored. There are only a few publications about parallel and distributed Petri net simulations. In 1991 Nicol and Roy [93] analyzed distributed execution of PN in the framework of communicating discrete event simulations. Thomas and Zahorjan [94] in the same year have shown that significant performance improvements over sequential simulation may be achieved by using a selective receive mechanism.

In his paper Ferscha [95] has described the implementation of various adaptations of classical distributed discrete event simulation strategies to the simulation of time transition PN models. The first algorithm of parallel simulation of PN on the GPUs was presented in [96]. The tests performed by the authors show significant performance for a few types of Petri nets. However, their algorithm is written in Cg programming language provided by NVIDIA, which today is outdated and not used.

During the preparation of this algorithm quite a similar work by Chalkidis *et al* was found. They designed an algorithm of simulation of Hybrid Functional Petri nets (HFPNs). Firstly they create lists of independent transitions, after that they are

processed in parallel by a CUDA application. GPU accelerated simulations of the biological HFPNs are observed to run up to 18 times faster than the equivalent CPU procedures. My algorithm contains a new method of decomposition of the Petri net, which is simpler than used by Chalkidis and is (in the majority) implemented on GPU. It is also designed for the classical Petri nets, not HFPNs and can be easily extended to other types of the Petri networks.

For a short description of the CUDA architecture, please see the Appendix A.

3.2 The PINGU algorithm for the parallel simulation of the PN

The PINGU algorithm is divided into two parts: the first – the preprocessing, and the second – the basic simulation. The preprocessing is quite time-consuming, as it is usually longer than the simulation, however, in order to perform the basic simulation it is required, and thanks to the time spent on the preprocessing, the simulation can be performed faster. The first part is prepared only once for the Petri net, at the beginning of the algorithm, and its results can be saved and used for many simulations. An important idea in the algorithm is a conflict-free list.

Def. 33. The conflict-free list L_i is a list of transitions, which contains a transition t_i (as the first one added to the list) and every transition to be added to the list is not in the conflict with any other transition from the list.

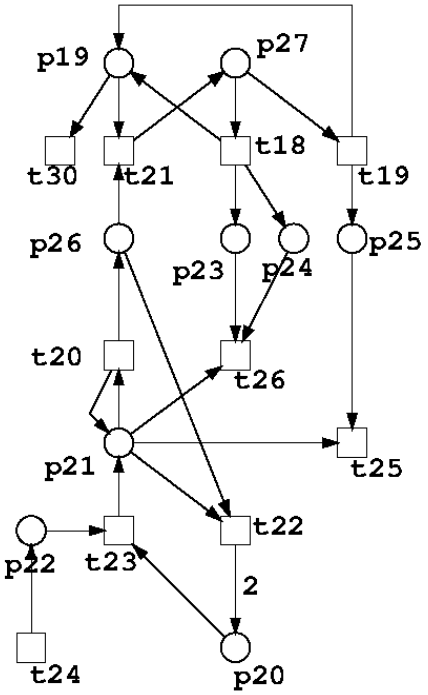


Fig. 12. Examples of conflict-free lists. This network is part of a bigger IS network. One possible conflict-free list of transitions with respect to t21 is {t21; t26, t24} or {t21, t25, t24}, with respect to transition t26: {t26; t19, t24}.

The restriction that every pair of transition from one conflict-free list cannot be in the conflict is very limiting, however, due to that, the transitions from a one conflict-free list may fire fully independently. When the Petri net is large and does not contain many connections those conflict-free lists will be quite long. It is a typical situation, like for example, in the OPOA algorithms (described later). For one transition many conflict-free lists may exist, with the composition of the list depending on the order in which transitions are added to the list (**Fig. 12**).

3.2.1 Preprocessing

The preprocessing is the first part of the PINGU algorithm and consists of a few stages: the creation of the conflict matrix, the creation of the conflict-free lists and the calculation of the maximal number of input and output places together for any transition.

The creation of the conflict matrix is parallelized and implemented on GPU.

Algorithm 3. The creation of the conflict matrix (parallel).

Input: The input and the output matrices of the Petri net in the global memory. The number of blocks is *max(maxBlocks, m^2)*, where *maxBlocks* is the maximum number of blocks, depending on the model of the graphical card and *m* is the number of transitions. The blocks are organized into a two-dimensional grid. The number of threads is *max(maxThreads, n)*, where *maxThreads* is the maximum number of threads and *n* is a number of places in the PN. Value *conflict* which is kept in the shared memory.

Output: The conflict matrix *confMatrix*, which is *m x m* matrix, where *m* is the number of transitions.

Steps:

```
1.begin
2.if (thread.id = 0) then
3.    begin
4.    confMatrix[block.idX, block.idY] := 0;
5.    conflict := 0;
6.    end
7.synchronization();
8.if      (two      of      inputMatix[block.idX,      thread.id],
   inputMatix[block.idY,          thread.id],outputMatix[block.idX,
   thread.id], outputMatix [block.idY, thread.id] are common places
   for transition idX and idY) then
9.    conflict := 1;
10.synchronization();
11.if (thread.id = 0) AND (conflict = 1) then
12.   confMatrix[block.idX, block.idY] := 1;
13.end
```

Every block of threads will perform the same operations consistent with the **Algorithm 3**. The blocks are organized into two-dimensional grid, so each block has its *x* and *y* index, described as *block.idX* and *block.idY*. The block with indexes *block.idX* and *block.idY* will check conflicts between transitions number *block.idX* and *block.idY*. If the number of transitions is bigger than the maximum number of blocks, some blocks will repeat their operations for those remaining (waiting) transitions. Every thread in the block has its own number, which is the number of the place for which the thread will check if it is a common place for transitions *block.idX* and *block.idY*. If the number of places is bigger than the maximum number of threads, some threads will repeat its computations for those remaining places. At the beginning of the algorithm one thread from every block, with an index 0, will set the value of the *confMatrix* and the *conflict* value to 0 (lines 4-5). Other blocks will wait for this operation (line 7). Next every thread will check if its place is the source of the conflict between transitions assigned to the block. It has to read appropriate values from the input and output matrices and check if two of them are not equal 0. If the conflict is found, the common for all threads in the block a flag *conflict* is changed to 1 (lines 8-9). When every thread finishes its operation (line 10) the 0th thread will check the *conflict* flag and according to its value sets the corresponding element of the conflict matrix (lines 11-12).

The pessimistic time complexity of the **Algorithm 3** seems to be constant, however when the number of transitions is higher than the maximum number of blocks or the number of places is higher than the maximum number of threads, then the complexity depends on the *n* and *m*. The maximum numbers of blocks and threads are usually quite big, for example the maximum number of blocks in newer devises is $2^{31} - 1$ [97] and the maximum number of threads is 1024 [97], so the complexity only for very large networks will be not constants. The same algorithm executed sequentially will have pessimistic complexity $O(m^2\, n)$, because conflicts between every pair of the transitions must be checked and every place must be verified as the source of the conflict. Many contacts with the global memory occur in the **Algorithm 3**, but the input and output matrices may be too big to be kept in the shared memory. Only for small Petri nets will it be possible. In such cases, the algorithm will lose its versatility. However, the situation is not so bad, as with appropriate organization of the input and output matrices the succeeding threads will read succeeding cells of data and such a procedure is optimized in the CUDA architecture.

When the conflict matrix is created, the conflict-free lists can be obtained. This part of the preprocessing is not implemented on GPU.

Algorithm 4. The calculation of the conflict-free lists (serial).

Input: The input and output matrices representing the Petri net, both *n x m*, where *n* is the number of places, *m* is the number of transitions. The conflict matrix which is *m x m* matrix.

Output: The conflict-free lists, which is represented as *m x m* matrix. The *i*-th row of the matrix contains the numbers of transitions which are not in a conflict with any other transition from the list and the list is created for a transition *i*, this transition is the first added to the list.

Steps:

```
1.begin
2.for i := 0 to m do
3.    begin
4.    add(list[i], i);
5.    for j := 0 to m do
6.          begin
7.          conflict := 0;
8.          for e in list[i] do
9.                if transition j in conflict with e then
10.                     begin
11.                     conflict := 1;
12.                     break;
13.                     end
14.         if conflict = 0 then
15.               add(list[i], j);
16.         end
17.   end
18.end
```

The algorithm is very simple. For every transition its own conflict-free list is created. Firstly, the transition itself is added to the list (line 4). After that every other transition (loop in line 5) is checked if it is in a conflict with any element from the previously created list (lines 8-13). If it is not in the conflict, the transition is added to the conflict-free list (lines 14-15). After that, the next transitions are checked with that elongated list.

The complexity of **Algorithm 4** is quite big, two loops with *m* iterations (loop stared in line 2 and loop stared in line 5) and one which also in most pessimistic case will be performed *m* times (loop in lines 8-13). Thus, the pessimistic complexity is $O(m^3)$. However, we resigned from the parallelization of this algorithm - this step is too hard to parallelize. In my opinion, the most efficient way of parallelization is an assignment of a transition to the threads - each thread will calculate the conflict-free list for its "own" transition. The thread will keep the constructed list in the memory. If the size is not too large, the shared memory or registers are desirable. The thread will test the remaining transitions whether they are in a conflict with any of the transitions from the list. For that operation, the conflict matrix should be used. If no conflict is found, the transition is added to the list and the list is extended. This is a source of the problem with parallelization: this part cannot be parallel because the next operations depend on the previous results and many branches are possible. In other words, the long and complex loop of calculations for each thread could not be avoided. It is not a good type of

computations for light GPU's threads. Moreover, the reading of the conflict matrix by a thread requires many contacts with the global memory in a random order. This is a serious obstacle in an efficient parallelization.

In the second part of the PINGU algorithm, apart the conflict-free lists, two values are also necessary: the length of the longest conflict-free list and the maximum number of input or output places together for one transition. The first value can be calculated together with calculation of the conflict-free lists. The second value is the maximum number of input or output places among all transitions - the result will be the maximum number of input places if it is bigger than the maximum number of output places or opposite. The calculation of the maximum number of input or output places can be performed in parallel.

Algorithm 5. The calculation of the maximum number of input or output places (parallel).

Input: The input and output matrices of the Petri net. The number of threads (together across all blocks) should be equal to *m*, where the *m* is the number of transitions. So the number of threads per block should be maximum and the number of blocks should be sufficient to obtain at least *m* threads. *n* is the number of places.

Output: The maximum numbers of input or output places calculated by each block, so if the number of blocks is *k*, the output is an array of *k* values.

Steps:

```
1.begin
2.counterInput := 0;
3.counterOutput := 0;
4.for i := 0 to n do
5.    begin
6.    inpPlace := inputMatrix[threadId, i];
7.    outPlace := outputMatrix[threadId, i];
8.    if(inpPlace <> 0) then counterInput := counterInput + 1;
9.    if(outPlace <> 0) then counterOutput := counterOutput + 1;
10.   end
11.if(counterInput > counterOutput) then
12.   maxCounter[thread.id] := counterInput;
13.else maxCounter[thread.id] := counterOutput;
14.synchronization();
15.if(thread.id = 0) then
16.   findMax(maxCounter);
17.end
```

In this algorithm each thread finds the maximum number of input or output places for one transition (that is the reason why the joint number of threads is at least equal to the number of transitions). The returned value is consistent with the formula max(|input places of the transition|, |output places of the transition|). After

resetting the input and output counters each thread reads information about input and output places of the corresponding transition from the input and output matrices – a loop in lines 4-10. Every thread calculates the number of input and output places, and after the loop it chooses the bigger value (lines 11-13). This value is written to the array in the shared memory - each thread has a corresponding cell in the array. When all threads finish its operations (line 14) the 0-th thread will view the whole shared array to find the maximum value from the block, and it will write it in the global, output array.

Calculations of the maximum number of input and output places require n operations. It should be noted that if the maximum number of threads (for whole grid) is smaller than the number of transitions, then some threads will have to repeat its calculations. In practice, however, the maximum number of threads which can be calculated by multiplication of maximum number of blocks, which is $2^{31} - 1$ times number of threads per block, which equals 1024, is enormous and it is hard to imagine such a big network. The 0-th thread also has to iterate across the shared array, which takes as many instructions as the number of threads per block, at most 1024. So, we can assume that the pessimistic complexity of the **Algorithm 5** is O(1024 * n) = O(n). Operations on the shared memory are fast, but we cannot avoid reading from the global memory. An appropriate organization of the input and output matrices will provide the succeeding reading of data by succeeding threads.

3.2.2 Simulations

The simulation is performed on GPU. In this part of the algorithm the loop iterates over conflict-free lists. For every conflict-free list a new kernel is launched. If the number of connections is not big and the number of places and transitions is large, then the conflict-free lists will be long and the parallelization will give better effects. In the kernel each grid will correspond to each conflict-free list and each block will correspond to one transition from the list. Thus, the number of blocks should be at least equal to the length of the longest conflict-free list which was calculated during the preprocessing. Every thread in the block will process two places: one input and one output of the transition assigned to the thread's block, and the number of threads will be at least the maximum number of input or output places. Let us assume that this maximum is k and we consider a transition t_i with n input places and m output places, $m, n \leq k$. Then min(n, m) of initial threads in the block assigned to t_i serve (correspond to) as input and output places of the transition t_i. The next max(n, m) – min(n, m) threads correspond to a single real place (input if n is bigger, output if m is bigger) and one phantom place. The rest of the threads serve two phantom places. The organization of the memory and the grid is presented in **Fig. 13**. Every transition from one conflict-free list is fully independent from the other transitions from the list, so every block can work independently and firing of its transition depends only on the marking of input places of the transition assigned to the block.

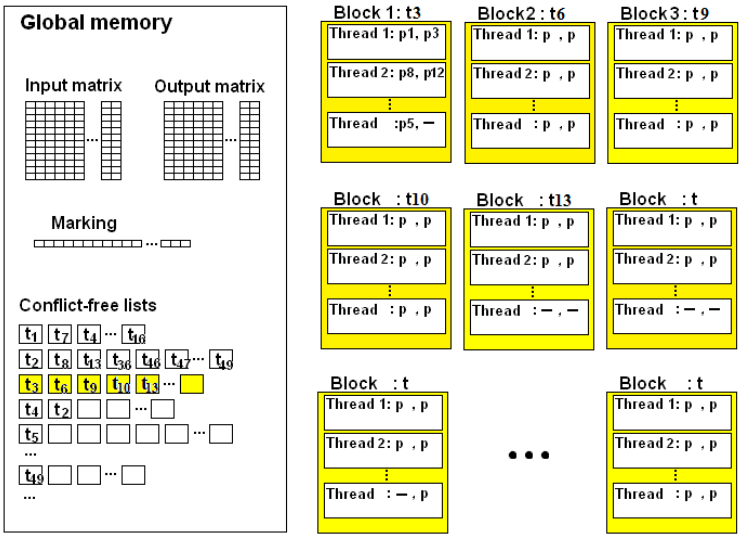


Fig. 13. A scheme of the content of the global memory and blocks' and threads' organization in the second part of the PINGU algorithm. The current conflict-free list and blocks corresponding to its transitions are shown in yellow.

Algorithm 6. The simulation algorithm (parallel).

Input: The conflict free lists, the input and output matrices, the vector of the initial marking of length *n*, when *n* is the number of places. The number of blocks is at least the length of the longest conflict-free list, and the number of threads is at least the maximum number of the input or output places. The value *trials*, which describes how many times the threads will try to fire transition.

Output: The marking vector of length *n*, obtained by modification of the initial marking by firing the transitions.

Steps:

```
1.begin
2.readData(blockId, threadId);
3.for i := 0 to trials do
4.    begin
5.    if(threadId = 0) then canFire := 1;
6.    synchronization();
7.    if(inputWeight > inputMarking) then canFire := 0;
8.    synchronization();
9.    if(canFire = 1) then fireTranstion();
10.   end
11.writeMarking();
12.end
```

The **Algorithm 6** starts with reading data by every thread from the global memory to its registers. Every thread copies: indexes of the places corresponding to the thread - one input and one output place or phantom places, their markings and cells from the input and output matrices corresponding to the served places and the

transition t_i which is represented by the block in which the thread occurs (line 2). The next loop (lines 3-10) depends on the number of trials i.e. how many times the transition will be tested to fire and fired. Firstly, the *0*-th thread will set the shared value *canFire* to 1 (line 5), which will mean that the transition can fire. After synchronization every thread will check its condition of firing, if the marking of the input place designed to the thread is smaller than the weight which connects the place to the transition served by the block, then the transition cannot fire and a flag *canFire* will be changed to 0 (line 7). When all threads finish this operation (line 8) and the result is that the transition can fire - flag *canFire* is still 1 (line 9), so that no thread finds that the firing condition is not satisfied, every thread changes the markings of the corresponding to its places according to the firing rule (line 9). Those operations are repeated *trial* times. At the end of the simulation of the current transition, every thread will copy a marking of assigned places to the global memory (line 11). Those values will be copied by the following threads during the simulation of the next conflict-free list. It is not necessary to copy them earlier, because the only running blocks are those corresponding to the other transitions from the same conflict free list. According to the definition of the list, no transitions have common places (input and output), so other blocks will not need information about marking changed by the block. This information may be only necessary for the next conflict-free list.

The pessimistic complexity of the algorithm is *O*(*trials*), *trials* is usually a small integer. However, a lot of communication with the global memory is required in the algorithm. Every thread has to copy a lot of data necessary to perform the simulation in a non-consecutive way. Such feature is not effectively supported by the CUDA technology. Additional transfers of data are performed when one conflict-free list is processed for the second and more times. The way of avoiding that would be keeping all the blocks corresponding to all transitions running. However, in that case another problem appears, i.e., the system does not know how to control which blocks should process data and which ones should wait. The mechanism of block synchronization would be necessary to resolve this issue, but it is not provided by the CUDA interface. In order to optimize performance of this GPU based computation the *trials* parameter was introduced. A better usage of the once transferred data is possible now. Since all threads will at a certain moment have all necessary information, they may fire transitions many times. The firings will be successful as long as marking allows or trials parameter has not been reached. This activity quickly adds new data for PN simulation, because it does not require expensive contacts with the global memory. For that reason, the parameter *trials* should be large. However, some caution is necessary. Numerous firings of some transitions performed in this way, for example those *t* from a conflict-free list K_i, may in practice prevent firing transitions being in a soft conflict with the transitions from the list K_i. That is why the value of *trials* should be well-balanced. In the tests it was usually set to 10. The correlation of the time of the simulation and the value of *trials* is presented in **Fig. 15**.

After the simulation of one conflict-free list the kernel stops and a new simulation of the next list is started. The conflict-free lists can be simulated in any order, for example, the order may based on the numbers of the transitions for which the list is created. In this case, the result of the simulation of the same network will be always the same. The conflict-free lists can be also chosen randomly, then the results of such simulations will be usually different (it depends also on the structure of the network). The implementation of other orderings of the conflict-free lists is simple in my algorithm.

When all conflict-free lists are preprocessed, the algorithm may stop or the preprocessing of the list may be repeated. Again, the first conflict-free list will be simulated, with the input marking obtained from the previous simulations. It is also possible to repeat the simulations of the all conflict-free lists a few times and this value is called *repetitions*. The dependence of the time of the simulation on the *repetitions* parameter is presented in **Fig. 15**. In the tests performed the value of the *repetitions* was usually 1, so each conflict-free list was simulated once.

3.2.3 Concluding remarks

This parallel algorithm PINGU has some special features. It warrants that each transition has an opportunity to fire (it may happen or not!), because it appears on at least one conflict-free list (its own list). In contrast to previously described algorithms, here we cannot determine the exact number of simulation steps performed without additional variables. The numbers of *trials* and *repetitions* are known, thus an approximate number of steps can be determined. In this algorithm a different philosophy of checking what transitions should fire is used. Previously (**Algorithm 1** and **Algorithm 2**) enabled transitions were identified first and then fired, here a transition is fired if it is enabled.

3.3 The analysis of performance

3.3.1 Testing protocol

The algorithm has been implemented in C language and compiled and launched on two operating systems: Microsoft Windows 7 and Linux Fedora 15. On Windows 7 the Microsoft Visual Studio 2010 was used to compile and run the program. On Fedora source the code was compiled directly by the nvcc compiler from the command line. In both cases CUDA 5.0 was used. Created software is available at http://www-users.mat.umk.pl/~leii/thesis/.

During the tests of GPU the implementation time of execution of the following kernels was measured: the first part of preprocessing, the calculation of the maximum number of input or output places of transitions, and the simulation. The time of preprocessing and the basic simulation were calculated separately. CUDA events were used to measure the time of kernels execution. In CPU implementation only the time of execution of the functions corresponding to the kernels was

measured. The *sys/timeb.h* library was used. Also, the time of preprocessing and the simulation were measured separately.

The algorithm was tested on two CPU and two GPU, in both cases one device was quite efficient and the other was weaker. The CPUs were: AMD Phenom 9650 Quad-Core Processor (lower efficiency, benchmark results: 2,533) and Intel Core i7-2630QM (perhaps not the latest model, but it is a processor from the highest current (2013) Intel family, benchmark results: 5,687). Tested GPU: nVidia GeForce GT 540M (not very good graphical card according to the present standards, 3DMark Ranking: 123) and nVidia GeForce GTX 580 (a powerful graphical card, 3DMark Ranking: 13, as of year 2013).

During the tests a few parameters which have an impact on the computation time were checked. Of course, the most important was the size of the network, described by the number of places and transitions. The other two were the values of *trials* and *repetitions*, and their impact on the simulation time was investigated.

The PINGU algorithm was tested on two types of the networks: artificially automatically generated Petri network and a Petri net describing molecular dynamics (MD) simulations (Chapter 5). The scheme of the artificial Petri net is presented in **Fig. 14**. It consists of many circular parts (one part is presented in black box in **Fig. 14**), which are connected and they create a long chain. The length of the chain, i.e. the number of parts, which is connected to the number of places and transition, is given by a parameter to the software which generate the network. This Petri net was created especially for the tests and is rather meaningless, however, it can adopted to some studies. For example, one part of PN may represent one cell and some metabolic pathway inside the cell, many cells are connected by sharing products and substrates of the pathway. The Petri nets which refer to MD simulations are described in greater detail in Chapter 5. They are generated by the algorithm and can be very large - it was the reason for the creation of the parallel algorithm in the simulation. The size of the MD Petri net depends on the resolution of the grid representing the Cartesian space in the algorithm and of course on the type and number of the trajectories.

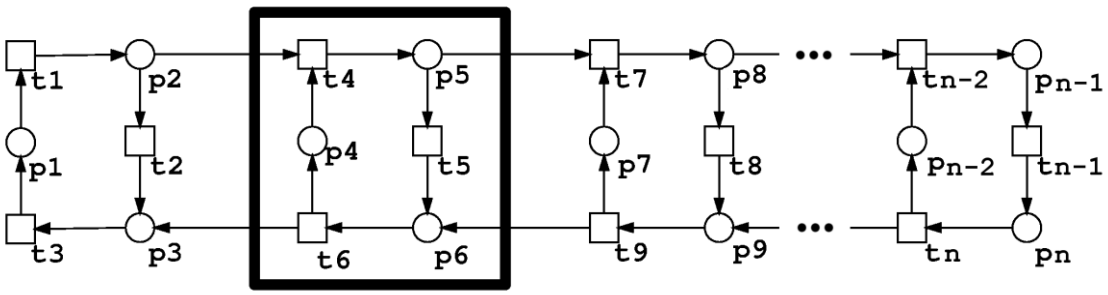


Fig. 14. A scheme of the Petri network constructed for tests. One part of the network, which can be multiplied, is selected in a black box.

3.3.2 Results, discussion and conclusions

The results of the simulations of the artificial Petri net are presented in **Fig. 15**. For all simulations the *repetitions* value was set to one and the *trials* value was ten. On the *y* axes only the numbers of places are shown, but the number of transitions is the same in this type of the nets (**Fig. 14**), so the number of places is directly proportional to the size of the network. In **Fig. 15a** the time required for the preprocessing is presented. One can see that the algorithm works faster while running on GPUs. For the nVidia GeForce GTX 580 card the time is almost constant, while for nVidia GeForce GT 540M it grows, but much slower than for CPUs. For both CPUs clear polynomial growth is visible. Especially the times of the slower process grow very fast and are extraordinarily long.

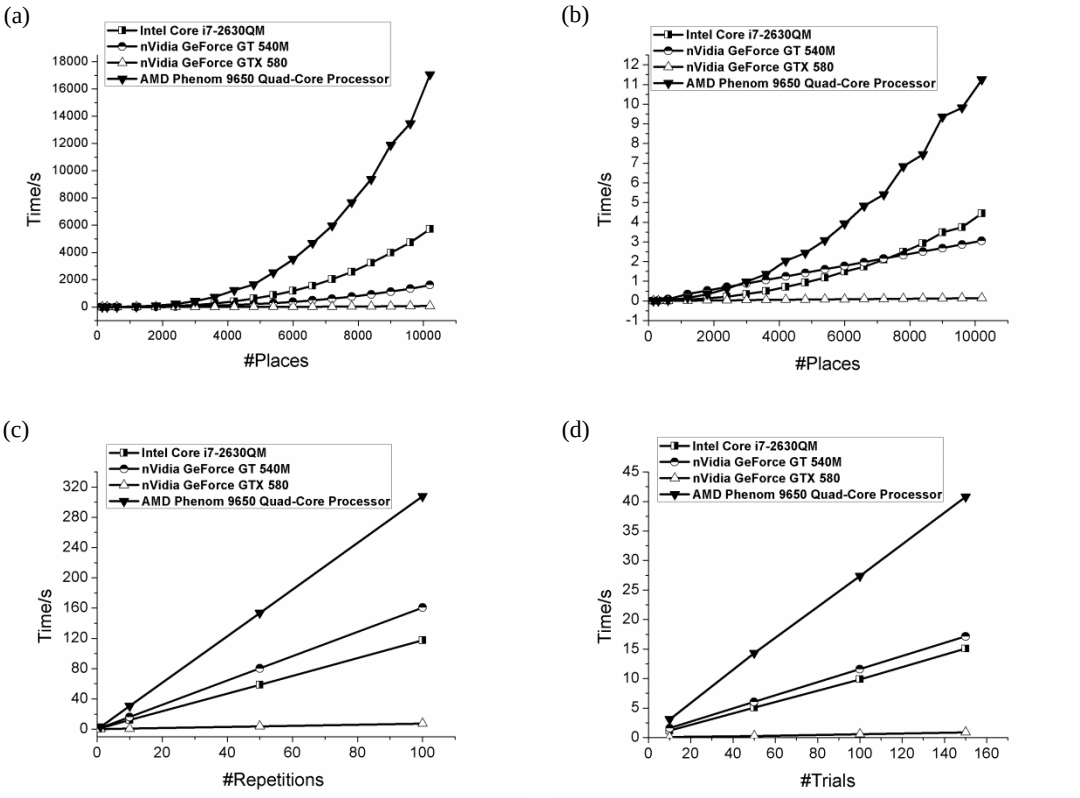


Fig. 15. The results of the simulation of the artificially generated Petri nets. (a) and (b) shows the time of the simulation as the function of the size of the network, in (c) and (d) the dependences of the time of the simulation from the values of *repetitions* and *trials* are presented.

The timings of the basic part of the PINGU algorithm on the GPUs form a linear relationship with the size of the network **Fig. 15b**. For nVidia GeForce GTX 580 the complexity of the calculations is almost constant. For nVidia GeForce GT 540M a growth is observed, but the plot is clearly linear. In both CPUs the time of the simulations grows polynomial. For AMD Phenom 9650 Quad-Core Processor the times of the simulations are the longest. For Intel Core i7-2630QM the initial times of the simulations are lower than the results for weaker graphical card, but

because of the polynomial growth for networks with 7000 places and bigger the times of the simulations for Intel Core i7-2630QM are longer than that for nVidia GeForce GT 540M. For smaller networks the costs of the threads management in GeForce GT 540M are bigger than the profits obtained from the parallelization.

In **Fig. 15c** and **Fig. 15d** the relationships between time of the simulation and the values of *repetitions* and *trials* are presented. One can see that the relation is linear in both cases, so those parameters should not change general findings. Those results were measured for networks consisting of 5400 places and 5400 transitions, but calculations were performed also for the networks of other sizes and the results were the same.

The results of the preprocessing for Petri nets, which represents MD trajectories (**Fig. 16a**) are very similar to those obtained for artificial Petri nets, so for graphical cards the preprocessing is faster and the time for the CPUs grows much faster. The second plot **Fig. 16b** presents the times of the simulation and they are a little bit different from the previous results. For all devices the growth seems to be linear which, however, is in contradiction with the calculations of the time complexity and the results for artificial Petri nets. Probably the characteristics of the networks cause the polynomial growth to be observed only for bigger Petri nets. However, the GPUs still work faster than the corresponding CPUs, and the smallest times are obtained for the nVidia GeForce GTX 580 card.

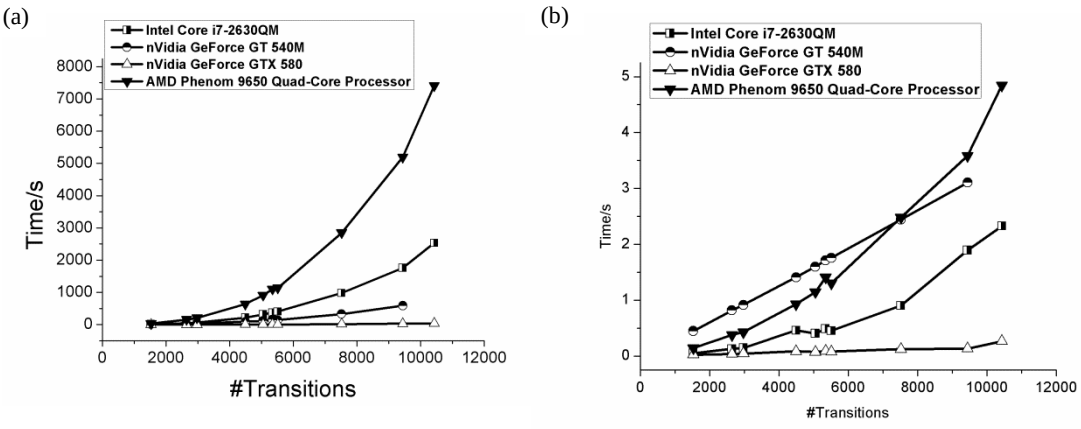


Fig. 16. Times of the simulations of the Petri nets describing MD: (a) time of the preprocessing, (b) time of the simulation.

The preprocessing step is always faster for the parallel version of the algorithm. This is also consistent with my calculations of the computational complexity. For the simulation step the GPU calculations are faster than the calculations performed on the corresponding CPUs. The computations acceleration depends on the size of the network- if the net is bigger, then the time saving is larger. However, the direct profit depends also on structures of the Petri nets - the most important is the length

of the conflict-free lists. For some networks the costs of the threads creation and organization may be higher than the profits from the parallelization. Notably, for other types of the Petri nets the main ideas of the parallel algorithm will be suitable as well. In such cases only a part of the simulation related to the mechanism of the firing should be changed.

In summary, presented results shows that for large networks the CUDA simulation is more efficient than the simulation executed on CPU.

Chapter 4. **MD and SMD computer simulations of antigen-antibody complexes**

4.1 Introduction

During my research project molecular dynamic studies of two complexes were performed in order to obtain a better understanding of molecular interactions. This work was also connected with the immune system, because modeling was focused on antibody-antigen interactions. Two protein complexes important in biology and medicine were studied: pollen from timothy grass Phl p2 and chemokine MCP-1 - both with corresponding antibodies. A series of molecular dynamics simulations were performed, a standard MD as well as Steered MD (SMD) were applied. During the SMD simulations complexes were forcedly dissociated by a force in different directions in order to check the molecular interaction during such experiments. The impact of the force vector direction on the value of the maximum forces was also investigated. Moreover, the role of a random factor associated with the numerical noise in the obtained results was studied. A bioinformatics tool, such as a sequence alignment, was used to better characterize highly specific antigen-antibody interactions.

Monocyte chemoattractant protein-1 (MCP-1), also known as chemokine ligand 2 (CCL2), is a member of the chemokine family [98]. The basic function of those cytokines is to act on chemoattraction in traffic regulation of immune cells. MCP-1 is a chemoattrator to monocytes, memory T cells, and dendritic cells. Its main function is to recruit those immune cells to the place of infection or injury [99-100]. However, recent studies have shown that MCP-1 is also present in the central nervous system and it can modulate the activity of neurons, astrocytes and microglia [98, 101-103]. Because of that, it may play a role in autism spectrum disorders, which is presented in [104-106]. Timothy grass is a very popular plant, so its pollen is very common. Unfortunately, it is a cause of allergic reaction in many people - an allergic reaction is an overly strong reaction of one’s immune system for normally harmless factor. This is the reason why the complex of pollen Phl p2 and antibody was studied.

This chapter is partially based on results presented in *Gogolinska, A. and W. Nowak, "Molecular basis of lateral force spectroscopy nano-diagnostics: computational unbinding of autism related chemokine MCP-1 from IgG antibody", Journal of Molecular Modeling*.

4.2 Methods

In the studies presented in this chapter, the crucial part were molecular dynamics (MD) simulations [107]. They are computer calculations of the trajectory of the motion of every atom from a given input set, in my case the molecules are proteins.

The theoretical basis of the MD simulation is straightforward: the localization of every atom in each time step t_{i+1} is calculated by solving the Newton equation of motion using, for example, the Verlet algorithm:

$$r(t_i + \Delta t) = 2r(t_i) - r(t_i - \Delta t) + \frac{f(t_i)}{m_k}\Delta t_i^2 . \quad (4.1)$$

where Δt is a time step, $r(t)$ is the localization of an atom in time *t*, $f(t)$ is the force in time *t* and *m* is the mass of the atom. In order to obtain the position of atom *k* in time t_{i+1} its positions in two previous steps are necessary - during the algorithm execution they are revoked from the previous steps. The mass of the atom is assigned to the atom type, a time step is defined in the input and the value of the force is calculated on the fly based on the positions of all interacting atoms and predefined potentials V. A set of analytical formulas together with appropriate parameters is called a force field [108].

The potential energy of the system is calculated as the sum of a few components: energy of bonds, angels, dihedral angels, electrostatic and van der Waals forces [108]. The last two are non-bonded terms and their computations require nested loops over all atoms. In this thesis well established CHARMM27b force field has been used [109]. The starting atomic positions are obtained from the file with an experimental protein structure [110].

Advanced software contains some mechanism to reduce a number of loops' executions, but still MD simulations are very time-consuming. One of the modifications of standard MD simulations are Steered Molecular Dynamics (SMD) simulations [111-112]. In such simulations some atoms are fixed in the space and the harmonic potential is added to the others. The spring constant *k* and the pulling velocity are defined as constant. When some relaxation of the system is observed, for example bonds are broken, the value of the external force decreases. During the SMD simulations observation of the value of force and displacement is a crucial part of the computer experiment.

In this study all-atom force field CHARMM was used [113-114] - all-atoms force fields contain parameters for every atom and they are processed separately. The simulations were performed in NAMD [115] software and for the analysis the VMD [116] code and own software were used. The structure of the complexes was obtained from the Protein Data Bank [117] and they are 2BDN with MCP-1 [99] and antibody and 2VXQ with Timothy pollen grass and antibody [118]. The complexes were embedded in an 8 Å thick TIP3P model water shell and after 0.4 ns equilibration at 300 K the 3ns MD simulation of 2BDN and 1ns of 2VXQ were performed. The structures obtained after those simulations were the staring points for the SMD simulations. An external force was applied to the antigens (chain A), which should dissociate the complexes in two perpendicular directions: the „vertical" force (V, almost parallel to the main axis of the antibody, the direction „z" in **Fig. 17**) and the „lateral" one (L, approximately perpendicular to the main

axis of the antibody). During the simulations of stretching all CA atoms of the antibody (chains L and H) were fixed. Structures were pulled for 2 ns at a constant speed of 0.025 Å/ps with a spring constant of 278 pN/Å.

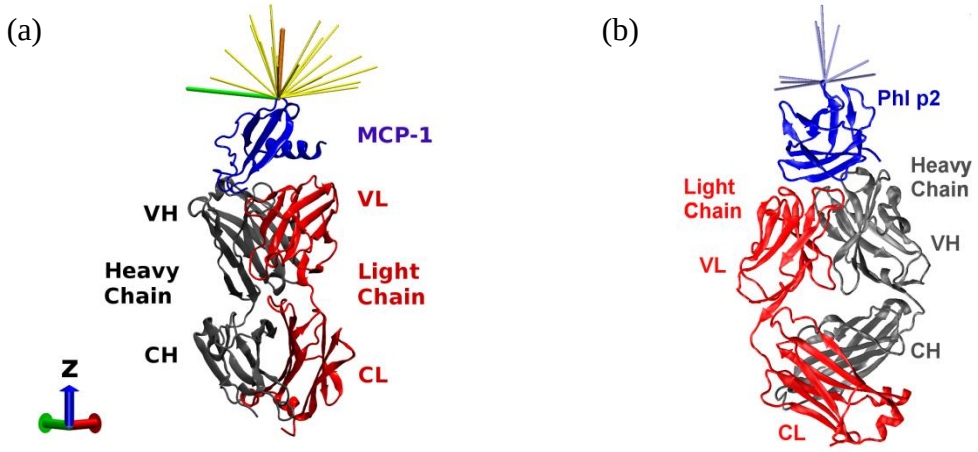


Fig. 17. Overview of studied structures: (a) MCP-1 (blue) with antibody (silver and red) (from [12]), (b) timothy grass pollen Phl p2 (blue) with antibody (silver and red). Dragging directions are also presented as lines: (a) yellow and two chosen in orange and green, (b) silver.

For pollen Phl p2 eleven pulling directions were tested: six vertical and five lateral. In the case of MCP-1 22 pulling directions were tested, 9 vertical and 13 lateral. Also the importance of the disulfide bridges was checked, in every direction 2 simulations without, and one with disulfide bridges were performed. For checking impact of the statistical errors in the maximum force determination, two directions (one V and one L) were chosen and ten 2ns simulations were generated for each direction. For ten directions (five V and five L) simulations with pulling speed ten times slower than before i.e. 0.0025 Å/ps were performed to study the dependence of the calculated forces on the puling speed. All those SMD simulations have practical aspects. They were prepared to test the possibility of using Friction (or Lateral) Force spectroscopy (FFS) method which is type of Atomic Force Microscopy (AFM). In this technique a probe quickly scans the analyzed surface laterally and the "unbinding" is enforced by the lateral forces. Usually, the AFM tip is functionalized by the antibody and a protein is immobilized on a surface. During such an experiment forces are measured but sequence of molecular events cannot be obtained. Also for MCP-1 one longer standard MD simulation of 10ns was generated to study the behavior of the structure not subject to external forces.

4.3 Results

4.3.1 Steered Molecular Dynamics – mechanically enforced dissociation

In SMDs simulations the scenario of events in all studied cases was similar: a steep rise of the force up to a certain maximum value, a gradual decrease of the interaction force and a separation phase characterized by a force close to 1 nN corresponding to the hydrodynamic drag.

Plots of the force curves obtained for timothy grass pollen SMD simulations are presented in **Fig. 18**.

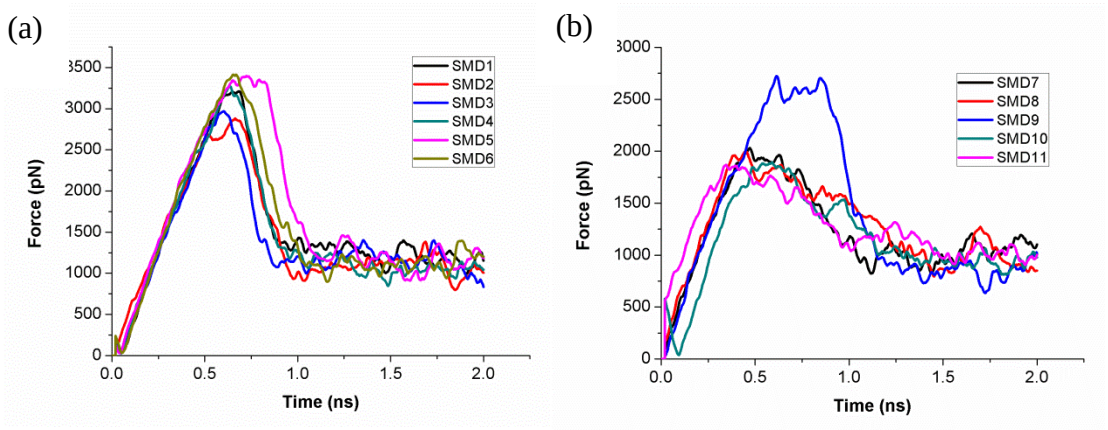


Fig. 18. SMD calculated force spectra for unbinding process obtained during simulations of timothy grass pollen Phl p2. In (a) forces measured during vertical draggig, in (b) drugin lateral dragging.

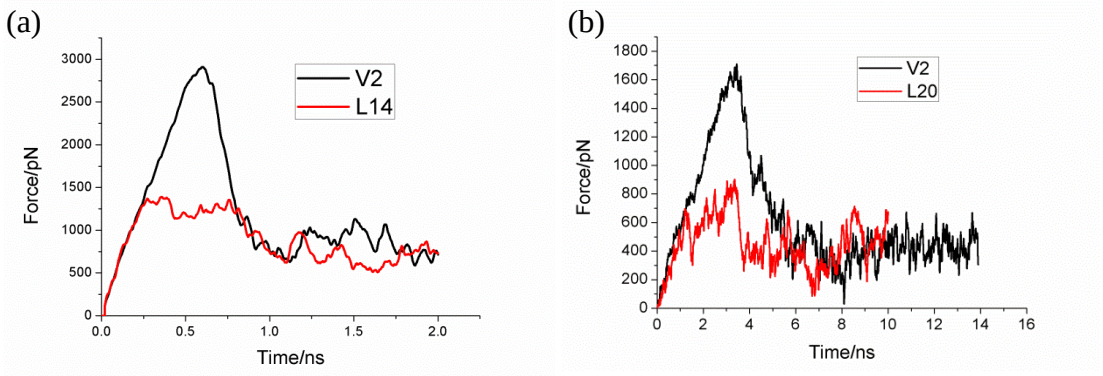


Fig. 19. Examples of SMD calculated force spectra for unbinding process. Typical plots of values of the force in two selected directions: V- vertical, L - lateral for (a) 2ns simulations and (b) 10ns simulations (10x slower pulling speed than in 2ns simulations).

For MCP-1 in total 69 output files were analyzed and values of the forces necessary to dissociate the complex were calculated. It is obvious that they all

cannot be presented in one plot like in **Fig. 18**. Instead the two examples are shown in **Fig. 19**. Other values are presented in **Table 9**.

In the standard forms (S-S bonds present), for 2 ns simulations, when the vertical (V) direction of the pulling force vector was applied, the lowest calculated force value was 1768 pN, while the highest value was 2911 pN. The average value of the maximum force observed during V direction SMD simulation was 2302 ± 366 pN. For laterally oriented pulling vectors the forces were lower: the highest value was 2229 pN, the lowest was 1391 pN, the average 1799 ± 301 pN. Respectively, for the simulations with 10 times slower pulling speed the average for the vertical forces was 1437 ± 254 pN, and for the lateral force simulations the average was 962 ± 169 pN. Thus, the process of mechanical unbinding requires lower forces if it proceeds in the direction L parallel to the MCP-1- antibody contact plane.

Table 9. Maximum values of force obtained in each 2ns SMD simulation (in pN) and 10 simulations with pulling speed 10 time slower (slower10x).

Direction vector V	**SS present**	**First without SS**	**Second without SS**	**L**	**SS present**	**First without SS**	**Second without SS**
V1	2102	1791	2213	L11	1522	1856	2001
V2	2911	2342	1943	L12	2045	1958	1497
V3	2678	1989	2448	L13	1668	1622	1877
V4	2663	2195	2215	L14	1391	1551	1511
V5	2084	2264	2294	L15	1859	1760	1548
V6	1768	1961	1671	L16	1737	2330	2293
V7	2537	2418	2208	L17	2148	1859	2001
V8	2152	2293	2239	L18	1988	2170	1959
V10	2776	2611	2431	L19	2229	2003	2033
V5v2	2161	2423	2323	L20	1606	1798	1604
V5v3	2214	2061	2294	L21	1472	1608	1556
V5v4	1907	2067	2178	L22	2224	2076	2127
V5v5	1974	2345	2259	L23	1493	1856	2255
Average	2302	2212	2209	Average	1799	1880	1866
Standard deviation	366	228	204	Standard deviation	301	227	289
V2 slower10x	1591			L14 slower10x	905		
V7 slower10x	1259			L20 slower10x	829		
V10 slower10x	1515			L21 slower10x	793		
V5 slower10x	1097			L13 slower10x	1159		
V8slower10x	1723			L15 slower10x	1124		
Average	1437			Average	962		
Standard deviation	254			Standard deviation	169		

In order to better study possible correlations between the value of the force necessary to separate an antibody and MPC-1 and the dragging force direction, we transformed force vectors into standard spherical coordinates. The forces with respect to values of the φ angle (only for lateral L cases) and the Θ angle are shown in **Fig. 20**.

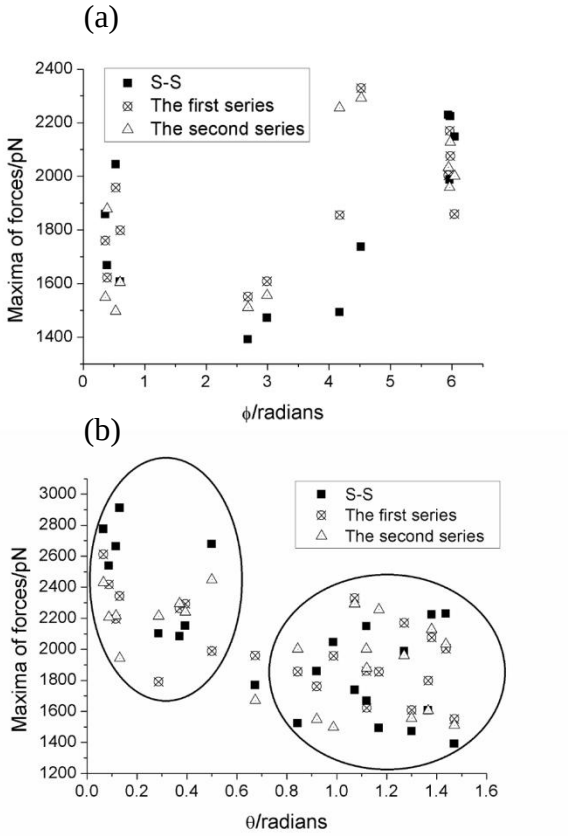


Fig. 20. Plots showing the dependence of maximum values of the forces for simulation of MCP-1 on the pulling force vector orientation - in spherical coordinates φ (a) and Θ (b). Only shorter 2 ns trajectories are presented from [12]).

The data presented in **Fig. 18** and **Fig. 20** clearly showed that in vertical the dragging forces necessary to force dissociation of complexes are significantly bigger. However, in some lateral cases (for example, a blue curve in **Fig. 18a**) the obtained forces values are also significant and more similar to the vertical dragging, but such situations are quite rare, which can be easily deduced from data presented in **Table 9** and averages and standard deviations values calculated on the basis of that data. Probably those high forces observed in the lateral cases are caused by van der Waals interactions between some side groups of the antigen and the antibody ("hooking" during the lateral dragging).

Moreover, the values of forces calculated during SMD simulations with 10 times slower pulling speed are smaller than those observed in simulations with a higher pulling speed. This observation is in accordance with the AFM experiments, where the pulling speed is even slower and the forces measured are also smaller. However, the general finding that vertical dragging requires higher forces is

preserved, and it seems that this correlation between the pulling direction and the values of forces does not depend on the pulling speed.

The maxima of forces are correlated with breaking of hydrogen bonds and salt bridges. Usually the sequence of bonds breaking is almost the same in all simulations. Only in the lateral dragging some changes in events sequence were observed, which was related to the direction of pulling, but even in those cases the general scenario was the same. Also in the vertical pulling usually one big force maximum correlated with the first bonds breaking was observed, in contrast to the lateral cases when very often a few, almost identical, force maxima were present, or the maximum is not well marked.

Impact of numerical noise

In the previous paragraph differences between lateral and vertical pulling scenarios were presented. The main finding was that the forces required to dissociate the complexes in the vertical dragging were higher than in the lateral one. However, the numerical data, presented for example in **Table 9**, can be affected by a numerical noise because to compute those data quite complex calculations were performed. The method of calculations is described in the paragraph 4.1. If the impact of numerical noise is significant, the obtained results will not be valuable and difficult to interpret. In order to clarify this issue additional simulations were preformed: two cases were chosen, one vertical and one lateral. For the vertical pulling case 10 the same, SMD simulations were performed twice - the input file and the starting structure were always the same. For the lateral case, similarly, 10 the same SMD simulations were carried out. The maximum values of the forces obtained in those simulations are presented in **Table 10**.

Table 10. The maximum force values obtained during repeated simulations of two cases vertical V5 and lateral L14. For V5 two times ten simulations and for L14 ten simulation were performed.

V5, first	V5, second	L14
2423	2323	1464
2061	2294	1425
2067	2178	1441
2345	2259	1537
2220	2895	1782
2154	2264	1720
2251	2259	1504
2703	2044	1315
2398	2383	1353
2358	2361	1570

For the first round of V5 simulations the average was: 2297,9 ± 193 pN, for the second it was: 2325,9 ± 222 pN, and for the simulations of the L14 the average was: 1511 ± 148 pN. One can observe that the intervals of two studied cases do not

overlap. Also the differences between the results for vertical and lateral dragging from **Table 9** are bigger than the errors (scatter) caused by the numerical noise.

4.3.2 B-factors analysis and molecular recognition

Long 10 ns classical MD trajectory of MCP-1 was analyzed for compatibility between the results from computer calculations and the data from the PDB structure file. RMSF (Root Mean Square Fluctuations) fluctuations of amino acids from the simulation and temperature B-factors were compared. The correlation between them is very good and plots of it for MCP-1 and heavy chain of the antibody are presented in **Fig. 21**.

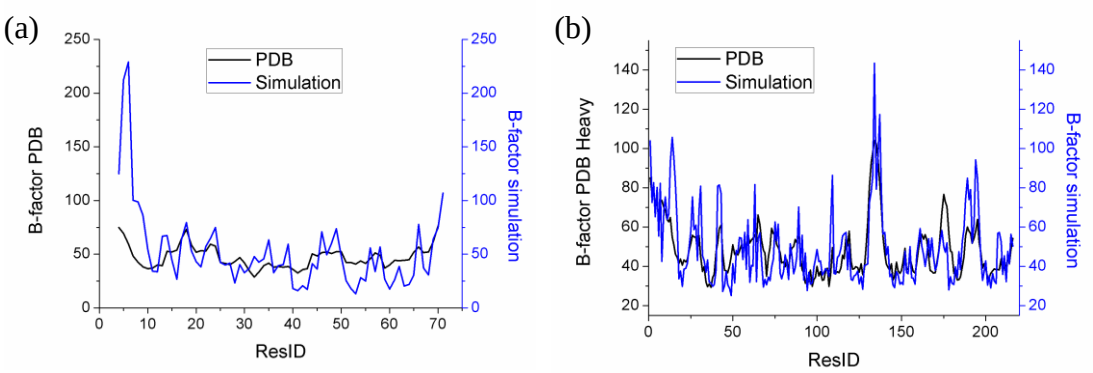


Fig. 21. A comparison of calculated RMSF fluctuations of MCP-1 (a) and heavy chain of Fab IgG antibody fragment (b) with experimental temperature B-factors, from [12]).

The main difference is observed in **Fig. 21a** for the first ten amino acids. This region is a flexible loop, so probably in the crystal this part of the molecule is much more stabilized by packing crystal interactions. Some other regions in the MD simulation are also more flexible, but probably the mismatch with the X-ray data has the same origin. For this analysis my own software was developed and used.

During the analysis some amino acids responsible for the interactions between antigens and antibodies were identified. In the complex with MCP-1 there are 6 the most important hydrogen bonds and in the complex with Phl p2 there are 10 important hydrogen bonds. The MPC-1 complex is stabilized by strong salt bridges (in italic) and hydrogen bonds between: *Glu39A* - *Arg98H*, *Lys56A* - *Asp52H*, *Asp65A* - *Arg32L*, *Asp68A* - *Arg32L*, Thr32A - Glu55L, Gln61A - Tyr33H, where A denotes MCP-1, L - a light chain of IgG Fab fragment and H – the heavy chain of Fab.

We have used the APBS program [119-121] to calculate a map of the molecular electrostatic potential (MEP) of MCP-1 and the Fab fragment. Rigid separated structures extracted from the 2BDN data were used for calculations. Results are presented in **Fig. 22**.

There are at least 3 regions (a, b, c, **Fig. 22**) with higher values of MEP. The regions in MCP-1 have corresponding counterparts in the Fab system of opposite charge. Thus electrostatics contribute to the stability of this complex as well. These calculations help to identify the regions crucial for effective recognition of the important MCP-1 chemokine.

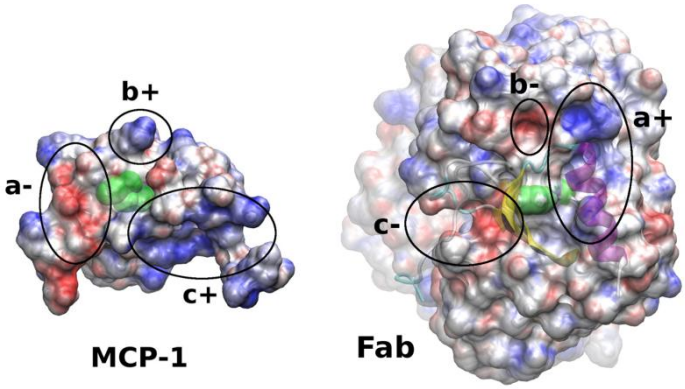


Fig. 22. Maps of electrostatic potential projected on solvent accessible surfaces of MCP-1 and Fab fragment of IgG. Positive regions are colored in blue, negative – in red. Complementary regions a, b and c are schematically indicated. Figures were prepared with VMD software [116] from [12]).

4.3.3 Bioinformatics analysis

The fragment of MCP-1 specifically recognized by an antibody is called epitope. In order to check to what extent amino acids present at the interface in the chemokine are conserved in other proteins, the PSI-BLAST search in standard non-redundant protein sequences database was performed and the ClustalX2 code [122-123] to make alignments of 10 most similar sequences was used. The results processed by Jalview 2.7 program [124-125] are presented in **Fig. 23**.

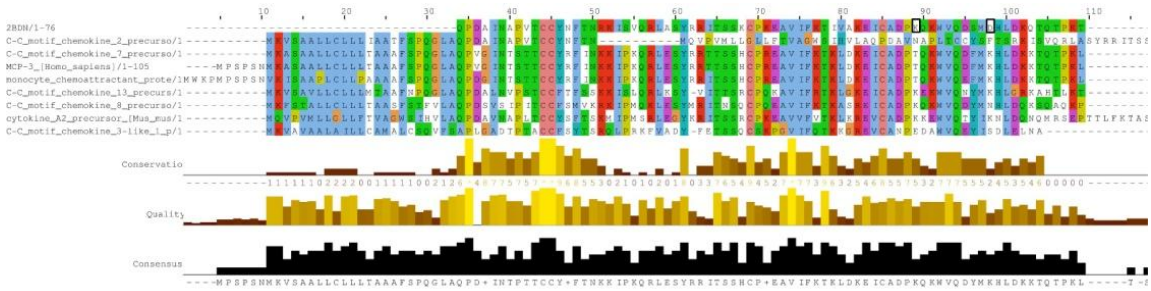


Fig. 23. Alignment of MCP-1 sequence with 10 most similar proteins. The conserved residues are shown. Black rectangles in MCP-1 sequence denote Lys56(A) and Asp65(A) amino acids from [12]).

For proteins in this set over 80% similarity to MCP-1 is observed. Amino acids important for strong interactions identified by the SMD simulations were analyzed

in greater detail. Two distinct groups of polar epitope amino acids are present in MCP-1: conserved set (Thr32, Glu39, Gln61, Asp68) and a specific set: Lys56 and Asp65. In the conserved set the same amino acids are present in nearly all similar proteins. The residues from the specific set are characteristic only for MCP-1 chemokine. This finding corresponds well with the observed 160-fold decrease of IgG antibody affinity to a Lys56Asn MCP-1 mutant described in [99]. This bioinformatics study can show how specific is mechanism of antigen-antibody recognition. The studied antibody can distinguish MPC-1 from other cytokines, even when they are very similar.

4.4 Conclusions

The simulations results presented in this chapter show that the dragging of an antigen from an antibody depends on the direction of the force applied. The force dissociation in the lateral direction requires 30% lower forces than in the vertical direction. My studies also show how strong and specific the interactions between antigens and antibodies are. Those results give a theoretical foundation for Lateral Force Spectroscopy measurements by the Atomic Force Microscope method.

The MD/SMD data were presented in this chapter just to illustrate what useful information may be extracted from computer simulations of biomolecular systems. The detailed discussion of obtained results has been recently published in an international journal [12]. The data related to molecular foundations of allergy or the components of the immune system are valuable in medicine and biology. The methods of research draw heavily on computer science and new ways of MD/SMD data are desirable. The last and the most comprehensive part of this thesis is devoted to paving new ways of MD/SMD results analysis, based on the PN formalism.

Chapter 5. **Petri nets and computer molecular dynamics simulations**

5.1 Introduction

As it was presented above, MD simulations are very popular in computational biology, chemistry and physics. They produce enormous amount of data [126]. The main problem is an effective analysis of this structural and dynamics information. Various methods are used [116, 127], but there is still huge demand for new and original ways of data scrutiny and representation (Big DATA problem). In my opinion, PN has a potential to be used in MD simulations, They were never tried in this field (weak communication between branches of science). The problem is how to connect physical simulations with graphs of PN and simulations of PN dynamics. Numerous ways are possible.

Here new algorithms for MD data analysis are presented, based on our original ideas.

- OPOA – in this algorithm one place is assigned to a position of one atom
- OPOC – one place is assigned to a confirmation of a molecule
- CONTACT – one place represents contact between two amino acids.

The algorithms have been implemented in our lab and used for analysis for real simulations (my own data MCP1 from Chapter 4) and TTR (data obtained from mgr inż. Rafał Jakubowski, IF UMK).

Before we go into details, the following definition has to be presented.

Def. 34. The Petri net which is generated from the trajectories from the Molecular Dynamics simulations will be called MD Petri net.

In our opinion, PNs have good potential to be further used in MD computer simulations field.

5.1.1 General overview

The general scheme of the molecular dynamics simulations should be recalled. Using numerical algorithms, programs like CHARMM, NAMD, GROMOS solve system of thousands of differential equations. The results are spatial structures of the studied system, which present time evolution of the system. Output files from the MD simulations contain frames (snapshots) which describe localization of every atom from the protein studied. Information about physical states of the system is hidden in those output files. They are called trajectories, because they represent a motion of the system in the phase space ($\mathbb{R}^{3N} \times \mathbb{P}^{3N}$).

The scheme of the whole process and location of PN in MD studies in presented in **Fig. 24**. We would like to obtain knowledge of the physical system from the MD trajectories using PN.

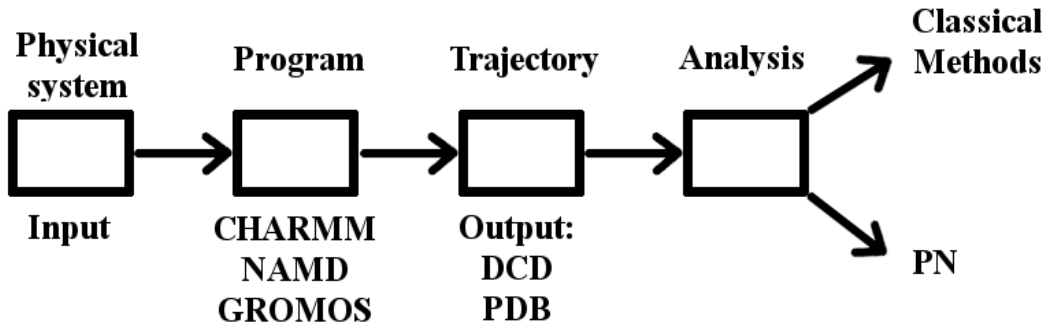


Fig. 24. The diagram which presents a flow of data in the MD studies.

We aim at making some representations of the MD simulation process or MD results (trajectory files) in the form of a PN. During the process of such PN representation, the creation of a few selected aspects was considered as important and was taken into account during the development of our algorithms:

(1) An enhancement of important (or desired) features of MD simulations, for example, a large change of the conformation, a formation of new contacts. Those "events" in the trajectories are hard to notice, but often important during the analysis. Thus, in this aspect the main focus of the Petri net representation is to facilitate the analysis of the MD simulations results.

(2) Petri nets are dynamical structures and this allows to connect Petri nets dynamics with the molecular dynamics simulations. We would like that the MD Petri nets simulations should mimic the MD simulations as well as possible.

(3) The next aspect in MD data reduction is an exploration of PN for clustering of molecular structures. Analysis of a small number of clusters is usually much easier in scientific studies than dwelling with millions of MD frames.

In order to obtain the first goal, i.e. to facilitate an analysis of the MD trajectories, at this moment 3 different algorithms were designed (OPOA, OPOC, CON). In my opinion, the character of MD methodology prevents any design of a fully universal PN algorithm. It is impossible to focus on all possible aspects regarding a MD simulation in one network. That is why specialized Petri nets have to be formed. Here several types of networks, exploiting various MD elements, are presented.

In the One Place One Atom algorithms the behavior of every amino acids in the function of time is monitored. In One Place One Conformation algorithm the time evolution of a whole molecule is represented. In the Contacts algorithm the changes of contacts between amino acids is the main focus. More detailed overviews of the algorithms are presented in the following paragraphs.

Before we address the second goal, it is important to highlight that simulations of the MD Petri net cannot be completely equivalent to the standard MD simulations. In the MD simulations complicated calculations, i.e. based on an advanced physical theory, are performed. Typically, they require a lot of CPU time. In contrast, Petri nets simulations are quite simple, based on simple tokens transfer, and thus they cannot generate any new "physical information" that was not already hidden in the underlying standard MD trajectories. So, the main goal in this part of the Thesis is to make the MD Petri nets simulations as similar as possible to the MD simulations. One can think that this reproduction is not useful because once we have a few (or even one) MD trajectories we do not need their repetition anymore. However, it is not fully true, the strength of PN approach developed here lies in connecting of a few trajectories into a single MD Petri net. Then the simulation of such a network will not give any new structural data but may produce completely new trajectories, which will be called PN trajectories. These will be rational combinations of the MD underlying trajectories used to generate the PN network.

Def. 35. PN conformation is a representation of the molecule conformation as the set of marked places.

Def. 36. The PN trajectory is the sequence of the PN conformations of molecule obtained during simulation of the MD Petri net.

Detailed description of possible representations of molecular conformations as sets of Petri net places will be given below. It depends on the type of algorithm used to generate the MD Petri net.

The reproduction of trajectories can be useful when one has only a few MD trajectories at hand and needs more possible trajectories, or wants to check if any new events, hidden in a classical MD data, can be observed. Such a case is presented in **Fig. 25**. This is a scheme of the content of two MD trajectories Tr1 and Tr2. Up to the point E3 they are the same, but after achieving this conformation the trajectories differentiate, for example, two different conformations are obtained in Tr1 an Tr2, respectively. After this a common conformation for both trajectories are present, and then the next splitting is observed, after which the trajectories again converge. In the first trajectory, outside common conformations, conformations E4.1 and E6.1 are unique for this trajectory, E4.2 and E6.2 are observed only in Tr2. Because E5 is the same in both trajectories, it can evolve into either E6.1 or E6.2. The same situation is for E3. Using the PN approach we can easily obtain two more trajectories: E1E2E3E4.1E5E6.2E7E8 and E1E2E3E4.2E5E6.1E7E8, The advantage is that only two MD simulations were necessary, and to get those two in a classical way at least two additional MD simulations would be required. Those calculations would require much more time than the PN simulations. Of course, in the example presented in **Fig. 25** those new alternative trajectories may be easily found by hand,

but in more complex situations of biomolecular systems usually many possibilities exist.

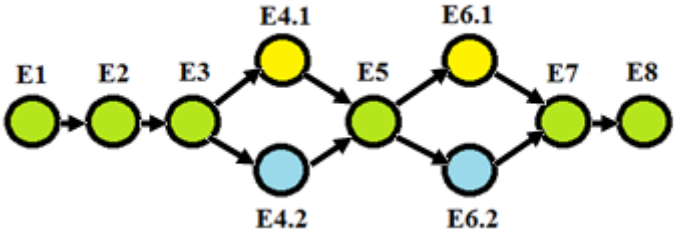


Fig. 25. The schematic representation of two trajectories Tr1 and Tr2, green conformations are common for both trajectories, yellow are conformations observed only in Tr1 and blue ones in Tr2.

Having in mind our purpose, i.e. a good reproduction of main features of standard molecular dynamics trajectories by MD PN simulations, we need to make simulations of the MD Petri nets as realistic as possible.

Realistic in this sense means able to mimic some aspects of the MD trajectories. In order to obtain this goal various types of the Petri nets are proposed. Extensions selected here provide various rules of firing and can add desirable features to the PN simulations. A few types of Petri nets were chosen (classical, timed, priority-based, random priority-based) and were used in the algorithms designed in this thesis. The motivation for this choice is presented in the following section.

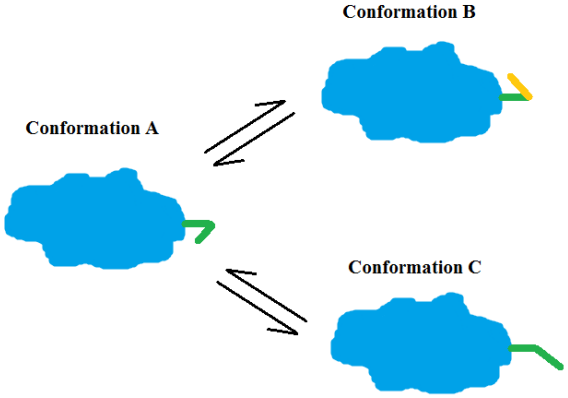


Fig. 26. A scheme of conformational transitions during a MD simulation: a molecule in the conformation A may transform into conformations B or C, both transformations are reversible. Blue - the main part of the molecule, green and yellow – a side chain, which changes its conformation and this is the sole difference in conformations A, B and C.

Typically, in MD simulations a molecule exhibits continuous small changes in the conformation, some structural changes occur more often, others quite rarely. However, even that rare conformation transformations may be the most important,

for example, only in that rarely accessible form a molecule is biologically active. Such a situation is schematically depicted in **Fig. 26**.

In this figure three conformations of one molecule are shown: A, B and C. They differ only in a specific arrangement of a side chain (highlighted in **Fig. 25**). Here we assume further that transitions between A and C occur very often in the course of the MD simulation, but transitions between A and B are rare, for example *fq(A→B)/fq(A→C)=1:9*. A model situation from **Fig. 26** will be our "test example" and will be as illustration of different variants of the Petri nets.

5.1.2 Petri nets types used in MD modeling

Marked Petri nets

The basic types of the networks generated by my algorithms are the classical Petri nets. They are universal, however, they do not provide any control during transitions firing: one of the enabled transition will fire and the probability of the firing is the same for every transition enabled. Thus, for MD simulations outlined in **Fig. 26** both conformational changes will occur with the same frequency and this symmetric statistics will not reproduce the true relationship present in the MD simulations. For example, having ten conformational transitions from A we must expect that five transitions will lead to the conformation B and five to the conformation C (remember please that in our example we have assumed that *fq(A→B)/fq(A→C)=1:9*).

The classical type of PN is also not sufficient for the SMD simulations. Since in such a PN every enabled transition may fire, an unrealistic situations may happen. For example, in the SMD stretched protein, one part of the molecule will be fully expanded, while the other part may remain not affected at all.

Timed Petri nets

The timed Petri were described in paragraph 1.4.4. In those networks the value of firing time function is associated with transitions and the transition with the smallest value of the clock will be selected to fire. The firing time function can be associated with the time registered during the MD simulation. So, in timed PN some supervision mechanism of transition firing order exists. However, performed tests showed that this supervision is event too restricted. One can easily design a PN with a circle created by transitions with small firing times. They would fire alternately and other transitions would not have any opportunity to fire. This problem will be discussed in greater detail in the next paragraphs. However, the idea of some type of time supervisions for SMD simulations is, in my opinion, useful.

One should note that transitions' firing order associated with time will not be suitable in the model situation considered in **Fig. 26**. A particular conformational

transition will occur depending on the particular value of time function (time-step), but this is not necessary in 1:1 correspondence with the frequency of this conformational change. However, the timed PNs were introduced in this study because they can mimic the sequence of events during the SMD simulations. The firing time associated with every transition will help to avoid a situation described in the previous paragraph (un-physical partial unfolding). So, we expect that timed PNs should work fine in SMD trajectories.

Priority-based Petri nets

In priority-based Petri nets (paragraph 1.4.5) the value of the priority function is associated with every transition and the transition with the highest priority from enabled transitions will fire. This gives us large control of the transition firing order and can diametrically change the PN trajectory. If the priority of the transitions is connected with the frequency the PN simulation can reproduce the situation from the **Fig. 26** quite well. The transition, for example t_1, which represents the change between conformation A and C will have a bigger priority than the transition t_2 from A to B, so the transition t_1 will fire always when both t_1 and t_2 are enabled. Thus, in priority-based Petri nets for ten transformation from the conformation A we will obtain ten paths into the conformation C and none into B. It is not the desired perfect 9:1 ratio, but it is a better representation of a "real" situation than the results expected from a simple marked PN. However, when the real proportions between the conformations B and C were more balanced, the simulations of the priority-based PN would generate biased results as well. The bias will have an opposite nature than that in the marked PN:

Another problem is that the conformation B (with a lower priority) will never be reached. This might be an even bigger problem than just wrong proportions between conformations, especially if B represents some unique features of the system studied.

Priority-based PN are quite useful for nets produced by OPOA algorithms. Namely, the places related to one amino acid will lead to separated (or quite separated, depending on the type of the OPOA algorithm) conflict sets. Using the modification of the firing rule introduced in **Def. 27**, every amino acid may change its localization in parallel. It was the main reason why this modification has been introduced, since it describes correctly the MD trajectory. Within one conflict set transitions, a transition with the highest priority will fire according to the firing rule given in **Def. 27**. In contrast to it, in the classical priority-based PN, a completely independent transitions will have to wait for firing for a transition with the highest priority. Usually this transition corresponds to the movement of some another amino acid and such motion is completely unnatural.

For SMD simulations modeling the priority-based Petri nets have the same disadvantage as the marked PN. The features added here will not help to make PN simulations be more similar to SMD simulations.

Random priority-based Petri nets

In random priority-based Petri nets transitions fire with some probability and this probability is proportional to the transition priority (paragraph 1.4.6). The bigger the priority is, the bigger change to fire a transition has. However, even the transition with a very small priority may have an opportunity to fire in this type of PN. The priority may be connected to the frequency showing how many times a given transition occurs in the MD simulations. This feature is very useful in representing situations like the one in **Fig. 26**. In simulations of random priority-based Petri nets both conformation B and C may occur – this is an advantage with respect to the priority-based Petri nets. Moreover, the proportion between conformations observed during the random priority-based PN simulations perhaps will be the same as in the underlying MD simulations. This is a substantial advantage of my networks over the marked Petri nets.

Another advantage is clear when the proportion between occurrences of conformations B and C is balanced. In contrast to a standard PN, within this model similar frequencies of B and C will be kept in the PN dynamics. Thus, the random priority-based Petri nets seem to be the best suited for keeping relations between conformations. It is nothing strange, as those networks were designed especially for such situations and they can emulate the MD simulations accurately. For the SMD simulations random priority-based PN behave in the same way as marked or priority-based Petri nets, so the timed Petri nets will be better choice in that case.

Random priority-based PN should be especially useful for networks generated by OPOC algorithm. For another class, i.e. OPOA algorithms, simpler priority-based PN are quite sufficient.

During simulations of such priority-based, generated by the OPOA, PN all amino acids may change their positions in parallel, like in the MD simulations. However, in the OPOC generated networks only one token is present and therefore the simulation of this simple priority-based PN is not correct. The PN trajectory will be always the same. This PN trajectory will exclusively contain places connected with transitions having the highest priority.

In a more involved random priority-based PN different PN trajectories will be accessible. The probability of their occurrence will depend on the frequency represented in the MD trajectories.

5.2 Algorithms for Petri Nets generation

5.2.1 One Place One Atom algorithms

Introduction

In the One Place One Atom (OPOA) algorithm a single place in the Petri net represents the position of one atom from the system studied by MD simulation. A transition represents the movement of the atom. This transition connects the previous localization of the atom (the input place) and the new localization of the same atom (the output place). Such a representation is very accurate and similar to the MD simulations. However, one can image that such a PN for even small proteins will be enormous and it will contain at least N_p places: $N_p = N_A \, x \, N_r$,where N_A is a number of atoms, and N_r the average number of localizations of the atoms. Such big networks are extremely hard to analyze, so to reduce the number of places and transitions two methods were used: a coarse grain representation and a discretization of the space.

The coarse grain representation is standard and widely used method of reduction complexity of the model. For example in [128-129] special force fields for the coarse grain molecular simulations of biomolecules were developed [130]. This method involves a representation of a group of atoms, usually an amino acid, by a smaller group of atoms, usually by just one atom. There are different types of such reduction, the easiest is a representation of an amino acid by its CA atom [131], in the other a virtual atom is assigned by the calculations. In more complex CG models two atoms may represent one amino acid: they may be "real" atoms, like CA and CB, or they could be artificially added. To avoid such additional calculations, in the OPOA algorithm one amino acid is represented by a single CA atom. However, this reduction in OPOA complexity is not sufficient, as a number of points has to be reduced as well.

Discretization of the space

Molecular dynamic simulations generate the trajectories of atoms (objects) in the three dimensional Cartesian space (see Chapter 4). A numerical representation of this continuous space is discrete. It is up to the user to choose what the recommended “granularity” of such a representation is. If one wants to have a very precise trajectory, the points representing the traces of an atom should be located very closely. However, in numerous situations a very tight representation is not required, moreover, there is a physical limit with respect to the shortest distance (point to point) travelled by an atom in the realistic MD simulations. Therefore, playing with computer representations of the Cartesian space used in MD or PN simulations gives a lot of opportunities for substantial savings in computing time.

In order to reduce the number of points used to represent atoms' position, the three dimensional grid is laid over the space and it divides the space into cubes. The edge of each cube is equal to the resolution of the grid. Each cube represents a new point in the new three dimensional discrete space and every atom which sits in the same cube in the $\mathbb{R}^3$ is localized in one point in the new space.

Def. 37. Let $\mathbb{R}^3$ be three dimensional Cartesian space in which molecular dynamics simulations take please. We define a new three dimensional space with Cartesian coordinates, called C'^3_k, where k is a real number, generated by the mapping function $f: \mathbb{R}^3 \to C'^3_k$, which assigns every point (x_1, x_2, x_3) from $\mathbb{R}^3$ a point (y_1, y_2, y_3) in C'^3_k by the formula:

$$y_i = f(x_i) = \begin{cases} \max\{p \in \mathbb{Z} \mid pk \leq x_i\}, x_i \geq 0 \\ \min\{p \in \mathbb{Z} \mid pk \geq x_i\}, x_i < 0 \end{cases}. \quad (5.1)$$

The k is the resolution of the dividing grid and it can be any real number. The mapping function is a surjection- every point from the $\mathbb{R}^3$ is mapped, but an inverse function does not exist. An opposite projection will assign a cube in the $\mathbb{R}^3$ space and the edge of the cube will be equal to k to every point from the C'^3_k space. The coordinates of the C'^3_k points correspond to the number of assigned cube, counting from the center of the coordinate system . An analogous transformation can be performed between $\mathbb{R}^2$ and C'^2_k, with the exception that a point from C'^2_k will be assigned to a square of points from $\mathbb{R}^2$ and the square edge will be equal k. This situation is presented graphically in **Fig. 27**. One can see that two $\mathbb{R}^2$ points presented in red, correspond to one C'^2_1 point *(1, 1)*. Those two in C'^2_1 will be the same even if the distance between *(1.8, 1.4)* and *(2.1, 1.4)* (in green) is smaller than the distance between the two red points. The blue point will be mapped into *(2, 2)* in C'^2_1.

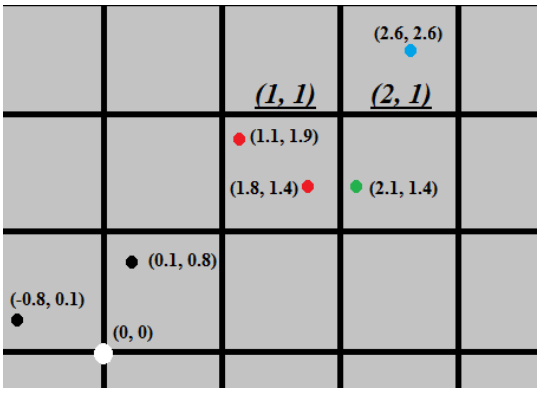


Fig. 27. The discretization of the $\mathbb{R}^2$ space. The space is divided by a squared grid with a resolution equal to *1*, the points from one square in $\mathbb{R}^2$ correspond to one point in the C'^2_1 space. A few points from $\mathbb{R}^2$ space are presented, their colors denote a point in C'^2_1, to which they are assigned (the same color, the same point). Points of C'^2_1 space are labeled in italic and underlined.

According to **Def. 37**, all $\mathbb{R}^3$ points around *(0, 0, 0)* will be mapped into *(0, 0, 0)* in $C_k'^3$ and the area mapped to *(0, 0, 0)* in $C_k'^3$ will correspond to eight cubes, all having one common vertex *(0, 0, 0)*. Let us analyze a simpler 2D case shown in **Fig. 27**. Here four squares around *(0, 0)* in $\mathbb{R}^2$ will be mapped into one point in $C_1'^2$, like two points labeled in black. They are in separate squares, but the mapping function will assign them to *(0, 0)* in $C_k'^2$. The same problem occurs also for the points which are near to one of the coordinate axis and one of their coordinates will be mapped into *0*. This situation is not desirable in the algorithm because movements of atoms around the coordinate axes will not be recognized if the atom is close enough to such a coordinate axis. A few solutions of such a problem were considered, with a modification of the **Def. 37** being finally added to the code.

Def. 38. We define a new three dimensional discrete Euclidean space with Cartesian coordinates, called C_k^3, where k is a real number, generated by the mapping function $f: \mathbb{R}^3 \to C_k^3$, which assigns every point (x_1, x_2, x_3) from $\mathbb{R}^3$ to a point (y_1, y_2, y_3) in C_k^3 by the following formula:

$$y_i = f(x_i) = \begin{cases} \max\{p \in \mathbb{Z} \mid pk \leq x_i\}, x_i \geq k \\ \min\{p \in \mathbb{Z} \mid pk \geq x_i\}, x_i \leq -k \\ \frac{k}{2} * sign(x_i), otherwise \end{cases}. \quad (5.2)$$

Def. 38 is superior with respect to **Def. 37**. According to the **Def. 38**, a point from $\mathbb{R}^3$, will be mapped to a point with *i-th* coordinate equal to *k/2* with the same sign like the original coordinate in the $\mathbb{R}^3$, while according to **Def. 37**, it would be mapped into *0* in $C_k'^3$. When the modification will be applied to the 2D case (see **Fig. 27**) the point *(-0.8, 0.1)* will be mapped into *(-0.5, 0.5)* and the point *(0.1, 0.8)* into *(0.5, 0.5)*.

This mapping will reduce the number of points obtained in the MD simulations by "gluing" a few points from the $\mathbb{R}^3$ into one in the space in which the algorithm is operating. How many points should by mapped into one can be regulated by the value of the grid resolution *k*? Of course, for small systems, or when the user needs such a model, the mapping can be skipped and the OPOA algorithm can work in the same space as used in the MD trajectories.

Overview of the algorithm, atom transition set

The general idea in the OPOA algorithm is that one place corresponds to a position of one CA atom. It is worth mentioning that a place may be created for every atom, not only of the CA type. In such OPOA based PN the transition represents a relocation of an amino acid. A conformational change in the molecule is represented by a transfer of a token from the input place of the transition (i.e. initial localization of the CA atom) to the output place of the transition (which represents the destination point of the CA atom). The motion here is in the discrete, simple C_k^3 space described in the previous section.

During the designing of the OPOA algorithm based PN some issues have to be addressed. In addition to the regular "position dependent" places, other types of places ("atomic presence", "starting") are introduced in OPOA based PNs. Every "position dependent" place has to annotated by the coordinates of the point in the C_k^3 space it is representing. Every transition has to be labeled not only by the initial and final localizations of the corresponding CA atom, but by the description (identity) of this CA atom as well. This information is stored in places of a new type – "atomic presence" places. In the algorithm such places which indicate the presence (or the absence) of every CA atom from the molecule studied are added to the growing MD Petri net. Notably, every transition affecting given atom is connected by the loop with this particular atom's "atomic presence" place. So, due to this loop, the transition cannot consume a token from the "atomic presence" place and at the same time it cannot fire if this place is empty. This mechanism works as a switch and may be very helpful in PN simulations. This new modification of PN gives a user a lot of control over the PN model. In such a way the participation of amino acids in the dynamics can be easily modified.

The concept of "atomic presence" places is not necessary. However, in that case another mechanism showing which transitions correspond to the movements of which CA atoms would be necessary. It has been checked that without connections between transitions and a new type of places the CUDA simulation algorithm works more efficiently because the conflict-free lists is shorter. The third type of places present in the OPOA algorithm is called "starting places". They are solely input places for transitions and represent arrival movements of the CA atoms from hypothetical "former" localizations to the initial ones.. So, in the MD Petri net three types of the places are present: "position dependent" places represent points in the C_k^3 space, places representing CA atoms – "atomic presence" places and artificial "starting" places.

Def. 39. The set of transitions which are connected by a loop with one specific atomic presence place and describe the trajectory of this specific CA atom is called atom transitions set (ATS).

Let us recall that our initial information from MD consists of CA trajectory (DCD or formatted PDB file) and an initial PDB structure of the molecule studied, composed of a number of amino acids (*N* here).

When the PN is created, one token is added to every starting place (One Place One Atom(=One Amino acid)) and the "atomic presence" place. This marking represents that the CA atom is present in the model but its localization is not defined yet. Then, during PN simulations, the token from the "starting" place can be moved by the transitions to position dependent places represented by points in the C_k^3 space. The sequence of such places describes the PN trajectory of each CA atom. For every CA atom one token in the "atomic presence" place and one in the position dependent place are present during the whole PN simulation. Once

initiated, the token from the "atomic presence" place cannot be consumed and the localization of the second token marks the current localization of the CA atom in the reduced space.

Notably, the OPOA algorithm can read data from few files and may use it to create just one MD Petri net, so these few trajectories can be connected and analyzed at the same time. This type of analysis is not possible using standard software, and in our opinion, it gives a new promising analytic tool to computational biology.

The stealing problem

In the OPOA algorithm the presence of token in some place is very important. However, the idea described previously has a disadvantage – the "stealing problem", which is shown in **Fig. 28**. Such a situation occurs when a few CA atoms are allowed to be in the same point in the C_k^3 space. It may happen when the mapping grid constant k is large (large mesh) or when the CA atoms visit the same point in this reduced space, but not at the same time. The places in the PN described so far do not contain any information about the time when the CA atom might enter the place. Such information may be assigned to transitions and in the timed Petri nets only. In other PN cases, the places represent solely the fact of presence of a given CA atom – it may happen anytime during the course of the simulation.

In **Fig. 28** a fragment of PN generated by OPOA is presented. Here two CA atoms: a1 and a2 may happen to be in the same point in the C_k^3 space, represented by the position dependent place p3. Every transition (squares) is connected by a loop (double arrows) to the place representing the atom affected by this transition (these atomic presence places labeled atom1 and atom2 are shown in the plot). A1 can be moved there by the transition t1 from the point p1, and a2 may be moved by t3 from the point p2. When both atoms visit p3 a1 may change its localization to p4 by the transition t2 and a2 may move to the point p5 by firing the transition p4. This is how it should happen in theory, however, nothing prevents (according to **Def. 5**) the transition t4 from firing two times. The transition t4 will consume the token which represents the localization of the atom a2, but it also will "steal" the token which represents the localization of the atom a1 and will put two tokens into p5. This marking would indicate that there are two atoms a2 in the point represented by p5 and a1 will no longer be present in the model. Thus it looks that the token for a1 has been stolen. This is obviously not acceptable.

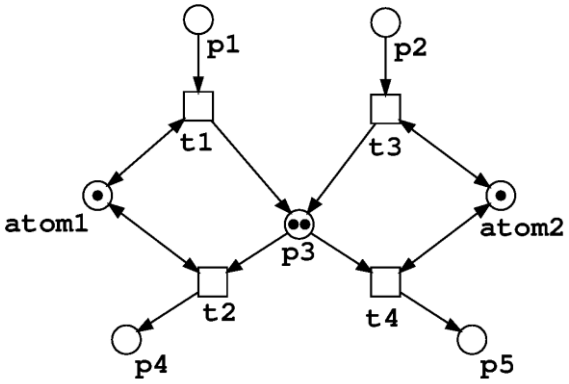


Fig. 28. An example of the stealing problem (for explanations see the text).

Def. 40. The stealing problem occurs when a transition which belongs to *n*-th ATS consumes tokens from a given place which have been delivered there by a transition belonging to the *m*-th ATS ($n \neq m$).

Of course, the stealing problem is not desirable. Two solutions of this problem were developed and they gave rise to two variants of the OPOA algorithm.

One Place One Atom algorithm - 1st variant

In the 1st variant of the OPOA algorithm (OPOAv1) the stealing problem is solved by creating separate places for every atom, so a "stealing situation" cannot occur. In this variant, one point from C_k^3 is represented by a few places in PN, a separate one place for every atom which can visit this point of the C_k^3 space.

Algorithm 7. One Place One Atom algorithm - 1st variant (OPOAv1)

Input: PDB files, which contain MD trajectories of the same molecule, parameter *k* for C_k^3.

Output: MD Petri net described by lists of places, transitions and arcs. Optionally files with description of places and transitions labels.

Steps:

```
1. begin
2. for every PDB file:
3.    begin
4.    frame := readFrame();
5.    for every CA atom in frame:
6.       if position(CA, frame) <> position(CA, previousFrame) then
7.          begin
8.          prevPlace := places.get(CA, position(CA, previousFrame));
9.          if(NOT places.contains(CA, position(CA, frame)) then
10.            begin
11.            newPlace := createPlace(CA, position(CA, frame));
12.            places.add(newPlace);
13.            transition := createTransition(CA);
14.            transitions.add(trans);
```

```
15.            createArcs(newPlace, transition, prevPlace, CA);
16.            end
17.         else
18.            begin
19.            currentPlace := places.get(CA, position(CA, frame));
20.            if(NOT transitions.contains(prevPlace,currentPlace,CA) then
21.               begin
22.               transition := createTransition(CA);
23.               transitons.add(transition);
24.               createArcs(currentPlace, transition, prevPlace, CA);
25.               end
26.            end
27.         end
28.    end
29. end
```

In the algorithm consecutive frames are read from successive PDB files (lines 1-4). The number of files can be any. For every frame each CA atom is processed (line 5). First, the localization of the atom is checked - if it is the same as the localization of this atom in the previous frame (line 6) - if the atom is still in the same point in the C_k^3 space. If yes, this part of the PN will not be changed, if not, we have to check whether the current position of the CA atom was previously obtained, so if there exists a place corresponding to this point in the C_k^3 space and the processed atom CA (line 9). If not, a new place is created and, of course, a new transition is created as well. Arcs are also created according to the general rule in the OPOA algorithm i.e. an arc between the previous place and the new transition, between the new transition and the new place and two links in opposite directions between the new transition and CA atomic presence place (lines 11-15). If there exists the place corresponding to the current localization of the CA atom (line 19-24) the existence of the transition assigned to the CA atom from the previous place and to the current place has to be checked (line 20). If it does not exist, it will be created (line 22) and appropriate arcs analogous to those shown in line 15 will be formed (line 24).

The complexity of the OPOAv1 algorithm depends mostly on the implementation of the structures *places* and *transitions* which keep information about the places and transitions created so far. The loops execute (a number of files multiplied by a number of frames multiplied by a number of CA atoms) times, which is required to read all data on MD trajectories. This number cannot be further reduced, otherwise some frames or atoms will be skipped. If *places* and *transitions* are implemented, for example, as a hash table, operations like *get*() or *contains*() should be performed in constant time, so the complexity depends only on the size of the input data.

Petri nets generated by OPOAv1 algorithm consist of separate networks corresponding to each individual CA atom. This is a natural consequence of the creation of separate places for every atom. This feature can be considered as a disadvantage because in the reality a molecular trajectory is governed by interactions between atoms. However, it can also bring positive effects - created

Petri nets are usually very big, thus if divided into smaller parts, they can be easier analyzed or even displayed.

One Place One Atom algorithm - 2nd variant

The One Place One Atom 2nd variant (OPOAv2) algorithm solves the stealing problem in a different way. In this algorithm the places correspond to the points in the C_k^3 space, and they are not correlated with the CA atoms like before. To avoid the problem an additional mechanism is added. It is presented in **Fig. 29**.

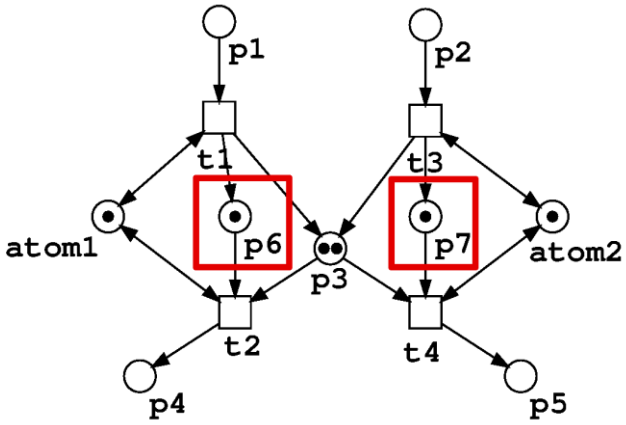


Fig. 29. The illustration showing an additional construction (red boxes) added to avoid the stealing problem in the OPOAv2 algorithm.

In **Fig. 29** one can see the additional construction (red boxes). It involves adding a new place (p6, p7) for every transition from a considered ATS which can put (or take) a token from the common place. This new place becomes an output place for every transition which puts tokens to the common place and an input place for every transition which consumes tokens from the common place. The weights of arcs which connect the new place with the rest of PN are set to one. When the transition puts a token to the common place, like for example the transition t1 in **Fig. 29**, it also simultaneously puts one token to the new place p6. After that, when the transition which consumes tokens from the common place, for example t2, fires, it consumes at the same time a token from this new place. Thus the transition can fire only once, because even if the common place still contains tokens, the new place p6 is empty. Also, if only the transition t1 puts a token into the p3, the transition t4 cannot consume it because its new input place p7 is empty. So, a transition from one ATS which takes tokens from the common place can fire as many times as the transitions that have provided tokens into the common place and are from the same ATS. Such mechanism will fully prevent OPOA based PN from the stealing problem. This modification will also make creation of one Petri net for all CA atoms possible.

Algorithm 8. One Place One Atom algorithm - 2nd variant (OPOAv2)

Input: PDB files, which contain MD trajectories of the same molecule, parameter k for C_k^3.

Output: MD Petri net described by lists of places, transitions and arcs. Optionally files with descriptions of places and transitions names.

Steps:

```
1. begin
2. for every PDB file:
3.    begin
4.    frame := readFrame();
5.    for every CA atom in frame:
6.       if position(CA, frame) <> position(CA, previousFrame) then
7.          begin
8.          prevPlace := places.get(position(CA, previousFrame));
9.          if(NOT places.contains(position(CA, frame)) then
10.            begin
11.            newPlace := createPlace(position(CA, frame));
12.            places.add(newPlace);
13.            transition := createTransition(CA);
14.            transitions.add(transition);
15.            createArcs(newPlace, transiton, prevPlace, CA);
16.            end
17.         else
18.            begin
19.            currentPlace := places.get(position(CA, frame));
20.            if(NOT transitions.contains(prevPlace,currentPlace,CA) then
21.               begin
22.               transition := createTransition(CA);
23.               transitons.add(transition);
24.               createArcs(currentPlace, transition, prevPlace, CA);
25.               end
26.            end
27.         end
28.   end
29.for i := 0 to placesNumber do
30.   begin
31.   counterT := 0;
32.   transitionsAdding.clear();
33.   for j := 0 to transitionsNumber do
34.      begin
35.      if(place[i] ∈ inputPlaces(transitions[j]) then
36.         begin
37.         counter = counter + 1;
38.         transitionsAdding.add(transitions[j]);
39.         end
40.      end
41.   if(counter > 1) then
42.      begin
43.      for k := 0 to transitionsAdding.size() do
44.         begin
45.         transition = transitionsAdding.get(k);
46.         atom = getAtom(transition);
47.         newPlace = createPlace();
```

```
48.          createArc(transition, newPlace);
49.          for l := 0 to transitionsAdding.size() do
50.            begin
51.            otherTransition := transitionsAdding.get(l);
52.            if(l <> k) AND (getAtom(otherTransition) = atom) then
53.               createArc(otherTransition, newPlace);
54.            end
55.          for j := 0 to transitionsNumber do
56.            begin
57.            if(place[i] ∈ outputPlace(transitions[j]) AND
                  (getAtom(transitions[j]) = atom) then
58.               createArc(newPlace, transitions[j]);
59.            end
60.          end
61.       end
62.    end
63. end
```

OPOAv2 algorithm starts almost in the same way as OPOAv1 (lines 1-26). Consecutive frames from successive trajectory files are read and a localization of every CA atom is checked. If this atom has been previously found in this particular point of the C_k^3 space then the place corresponding to it already exists (line 9). Depending on the results of this check, a new place and a new transition (lines 10-16) is added or only a new transition (lines 18-24) is inserted to the MD Petri net. The main difference in this part of the algorithm between the 1st and the 2nd variant of OPOA is that here, in v2, the place is labeled by the space point only, and not by the CA atom and space point as it was required in v1. Thus transitions from many ATSs may be connected to one place. Up to this step the OPOAv2 implements the basic idea of the OPOA algorithm, however, the stealing problem will still be present in the PN. In order to avoid this, the postprocessing of the created Petri net is necessary (lines 29-63). At the begging of this additional process every place is checked if is it an output place for more than one transition (lines 33-40). Every transition which can put token in the place is added to a list *transitionsAdding*. If more than one such transition is found, then the additional construction presented in **Fig. 29** should be added, depending on their ATS origin. For every transition from the *transitionsAdding* list an index of appropriate CA atom is found (line 46) and a new place is created and connected as the output place to the transition considered (lines 47-48). Other transitions from the *transitionsAdding* list are checked if they are in the same ATS, if yes the new place should be also their output place (lines 49-54). After that, the set of all transitions is searched for transitions which can consume tokens from the common place and are connected with the same CA atom (lines 54-57). Every transition found is connected with the new place. The new place becomes the input place for this transition (line 58).

The complexity of the first part of the algorithm is the same as for the OPOAv1 - it equals only to the size of the input data. The postprocessing is computationally more demanding. Every place and every transition has to be checked, but the biggest complexity is caused by the loop in lines 55-59, which will be executed *n*

times *m* times size of the longest *transitionsAdding* list, where the *n* is the number of places and *m* is the number of transitions. It should be considered whether this complexity can be reduced and this case requires further studies. Anyway, the performed tests have shown that my implementation of this algorithm works in acceptable time for real data (usually few minutes on PC).

Case studies and features of the OPOA algorithm

Both versions of the OPOA algorithm were used to analyze sixteen MD simulations of a small chemokine MCP-1 (see Chapter 4). Ten of those simulations were calculated from the same starting frame, and six others were randomly selected from a reference trajectory. Size k of the grid was 2 so C_2^3 was used. The MD Petri net obtained by OPOAv1 had 62250 transitions and 15441 places and that generated by OPOAv2 had 62250 transitions and 4048 places which correspond to positions of all CA atoms from MCP-1 chemokine.

Analyzing an MD Petri net generated by the OPOAv1 algorithm, one may easily find amino acids which are more flexible than others by a simple checking number of places in every ATS. Such an analysis has been performed for MCP-1 and its results are presented in **Fig. 30**. The selected amino acids with relatively large number of places in their ATS are highlighted in red. This feature is related to the bigger flexibility of the region in which those amino acids are located. And indeed, those amino acids are located at the terminal loops of the protein which are the most flexible. The amino acids shown in blue have a rather low number of the places in their ATS. This low value should be correlated with the higher stability. Those "blue" amino acids are located in the β-sheet, which is known to be the most stable part of the protein. Interestingly, two amino acids from this group are located in a long loop and without any additional information they might be expected to be very flexible. However, closer inspection shows that those amino acids form H-bonds with some amino acids from the β-sheet, which probably affects their flexibility. The method is sensitive enough to detect such features.

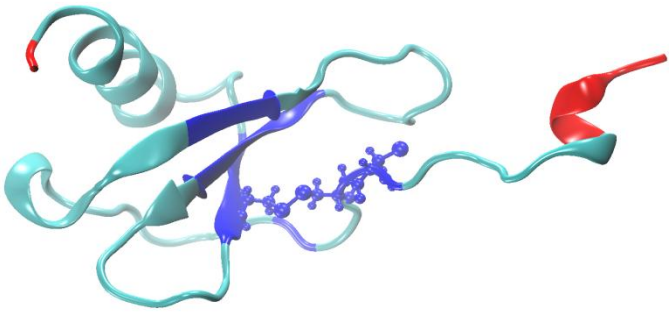


Fig. 30. MCP-1 with highlighted amino acids with a large number of places in their ATS (red) and the small number (blue).

Petri nets created with OPOA algorithms allow to study dynamical properties of every amino acid. Useful information comes from a frequency analysis. Positions which are preferred (or are rarely visited) can be easily detected. Moreover, the transitions allow to characterize amino acids' movements. One can check if each CA atom stays most of the time in one point in the C_k^3 space, or if it moves between a few points. Such frequently visited points/places are easily identified and may be analyzed. One can easily see if they are connected by transitions and create some preferred areas in the conformational space.

Such results are presented in **Table 11**. The most often visited places for two MCP-1 amino acids, no. 15 and no. 52, are presented. Both CA atoms are from the "stable" part (marked in blue in **Fig. 30**), however CA_{52} is from the β-sheet and CA_{15} is from the loop. In the first part of the table the places which are marked in the same color (except green) are connected to each other via transitions, so the CA atom can move between them freely. On the other hand, the transfer of a token is possible between the sets highlighted in different colors, but it has occurred only via the place p5310 (the green one). In the second part of the Table 11 such sets cannot be separated and the CA atom can move between each place freely. Interesting information stems from the difference between the frequencies for CA_{52} and CA_{15} atoms. Amino acid 52 is more stable than 15, and it has visited the points represented by p3882 and p2464 much often than CA_{15} visited its "most popular" places.

Table 11. The places visited most often by two CA atoms: 15 and 52 in MD PN generated by the OPOAv1 algorithm. "Place" describes a number of the place in the network and "Visited" describes how many times the corresponding place was visited by the amino acid during the simulation.

Amino acid 15										
Place	p224	p2169	p6760	p5441	p2174	p5310	p702	p4062	p12354	p7127
Visited	335	261	212	211	193	187	170	164	162	160

Amino acid 52										
Place	p3882	p2464	p196	p6932	p725	p2606	p5626	p7731	p7994	p6250
Visited	522	437	303	263	254	226	225	222	221	219

A similar study of the most commonly visited places was performed for the simulations of transthyretin (TTR) wild type (WT) tetramer and two mutants V30M and L55P. Transthyretin is a tetrameric, 55 kDa, a protein which transports thyroid hormones and retinol binding protein in blood and the cerebral fluid [132]. Amino acids located at positions 30 and 55 were analyzed and results are presented in **Table 12** and **Table 13**. OPOAv1 was used with the size of the grid of 1.5 Å. MD simulations were 50ns long. The first conspicuous observation is that amino acid 30 stays longer in the most frequently visited places than the amino acid 55. It

indicates that CA_{30} region is more stable than CA_{55} neighborhood. Indeed, CA_{30} is in the middle of the β-sheet structure and CA_{55} is at the beginning of the loop so mechanical nature of their localization is reflected in OPOA generated PNs. The next observation is that for both mutated amino acids there are "leader" places/points in the $C^3_{1.5}$ space i.e. which are more often visited than any other place of the same amino acid. Distributions of occupancy of those leader places are also good descriptors of the molecular system studied.

Table 12. The most commonly visited places for amino acid 30 in MD PN generated by OPOA ver. 1 for simulation of V30M TTR mutant and wild type TTR.

Amino acid 30 V30M										
Place	p3327	p409	p3285	p653	p3513	p6192	p5251	p3162	p2182	p1631
Visited	125	110	72	44	44	42	40	39	38	37

Amino acid 30 WT										
Place	p5290	p6553	p5085	p2489	p3270	p5390	p1922	p2704	p5212	p2041
Visited	108	106	101	70	68	39	38	33	31	29

Table 13. The most commonly visited places for amino acid 55 in MD PN generated by OPOA ver. 1 for simulation of L55P TTR mutant and wild type TTR.

Amino acid 55 L55P										
Place	p896	p3312	p4381	p482	p1822	p2199	p5557	p6262	p6553	p3856
Visited	95	74	52	51	49	49	44	40	38	37

Amino acid 55 WT										
Place	p5392	p2773	p6777	p4890	p5622	p2821	p5655	p2199	p5347	p5792
Visited	76	75	58	57	57	48	46	45	45	41

For example, in WT (see Table 12 and Table 13) the best "leader" places occupancy ("Visited") is more or less uniform. In V30M mutant the p3327 dominates and it was visited 53 times more often than the nest to second leader p3285. Such a difference in the number of visits is only 7 for WT. This means that the mutation in position 30 of TTR affects substantially the mobility of this part of the protein. Similar observation has been made for L55P mutant. In our opinion, such a result is difficult to obtain from a classical fluctuations analysis. Moreover, the usage of PN allows for a deeper data mining: one can extract all the paths leading to/from the most frequently visited place. Such a tool is easy to implement in my code.

5.2.2 One Place One Conformation algorithm

General idea

In the One Place One Conformation (OPOC) algorithm one place will represent one whole conformation of the molecule. In Physics or Chemistry the conformation of the molecule in $\mathbb{R}^3$ space can be represented in different ways: by positions (x,y,x) of all atoms, positions of atoms from the backbone only (for proteins) or by positions of CA atoms only. In OPOC algorithm that last representation will be used.

Def. 41. The conformation of the protein (molecule) having N amino acids is a set of positions of CA atoms in $\mathbb{R}^3$ space.

So in the OPOC algorithm one place in PN corresponds to the collective positions of every CA. Such a conformation may also be regarded as a single point in the $\mathbb{R}^{3N}$ space. Transitions represent changes between conformations and a token marks the current conformation. Because a molecule can be at a given time point in one conformation only, one token is present in the PN. The network can be generated from a few trajectories, so usually the Petri net will have a special form: one place will represent a starting conformation and numerous paths starting from there and containing transitions and places. These paths will represent particular trajectories.

MD Petri nets generated by the OPOC algorithm describes "the journey" of the molecule in conformation $\mathbb{R}^{3N}$ space. Obviously, the points in this space correspond to one conformation each, the same as a place in the PN of OPOC type. The networks are thus a kind of recording of movements of the molecule in the conformation space during the MD simulations used to generated the nets. Such a PN is a generalization of a standard MD trajectory.

Structural alignment

One of the problems in the OPOC algorithm is how to determine if two conformations are the same or different. Of course, the easiest solution would be to check if every CA atom is located in the same point in the $\mathbb{R}^3$ space for both conformations. The standard procedure is an introduction of a distance between conformations and a threshold. If a predefined distance (difference) between the conformations is smaller than the threshold then the conformations are considered to be the same. In order to calculate the difference between conformations different measures (distances) were developed, for example: the root-mean-square deviation (RMSD) [133], the global distance test (GDT TS) [134] or the template modeling score (TM-score) [135]. In the OPOC algorithm any measure can be used and it will not have an impact on the form of the algorithm. However, in my implementation the RMSD distance is used, which is perhaps the most popular one in bioinformatics. It expresses the similarity between three-dimensional structures

and it is calculated by the average distance between the atoms from two sets (usually structures) having *n* atoms each, *v* and *w*, given by formula:

$$RMSD(v,w) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\|v_i - w_i\|^2}. \quad (5.3)$$

However, the distances between atoms are not sufficient to determine whether two conformations are the same or not. Any rotations and shifts of a molecule as a whole (so called rigid body motions) should also be ignored. They are not ignored in this simple RMSD measure definition. In order to get a fair comparison of conformations the two structures should be carefully overlaid before RMSD is calculated. Two structures can be overlaid in many different ways and the goal is to obtain an overlap with the smallest differences between the structures analyzed. This whole process is called a structural alignment. It is quite complex and many algorithms were developed to solve this problem [136-138].

In the OPOC algorithm a conformation from every MD frame has to be aligned with the conformations which represent the places created so far (see **Algorithm 11**). It can be done by means of additional software, for example, [116] or [139]. It should be noted that in the OPOC algorithm the structural alignment problem is less complex than the general case. In the general scheme the structural alignment algorithms are often used to align molecules that are similar but not the same. During our MD Petri nets generation it is always the same molecules that are compared. They have the same number and type of amino acids, which makes the alignment easier and faster. Here two straightforward methods were developed and implemented.

Algorithm 9. Structural alignment algorithm with rotations.

Input: The threshold for the maximum rotation angle, the initial angle and the step of the rotation, two structures A and B represented by lists of CA atoms positions in $\mathbb{R}^3$ space - the number of CA atoms must be the same, it is assumed that both structures represent the same protein, but in different conformations.

Output: RMSD calculated between structures A and B.

Steps:

```
1. begin
2. for every CA atom:
3.    begin
4.    center(structureA, CA);
5.    center(structureB, CA);
6.    RMSD := calculateRMSD(structureA, structureB);
7.    ang := initial_angle;
```

```
8.    while((calculateRMSD(structureA, structureB) < RMSD) AND
          (ang < threshold))
9.      begin
10.     rotate(structureB, ang);
11.     ang := ang + step;
12.     end
13.   end
14. end
```

The main part of **Algorithm 9** is a loop which iterates over every CA atom (lines 2-12). Firstly, both structures are translated in such a way that corresponding CA atoms from the current loop are moved to the center of the coordinate system (lines 4-5). Then, the first RMSD between the transformed structures is calculated (line 6). The structure B is rotated by a small angle *ang* (line 10) in the next step. The rotation is continued until the accumulated rotation angle is larger than the given threshold or the RMSD calculated after this rotation is bigger than that already calculated (lines 8-12). It is important that *rotate()* function represents a rotation over every coordinates' system axis and their combinations.

The number of atoms in both structures is the same and we can assume that it is *n*. So the time complexity of the **Algorithm 9** is the following: the main loop (lines 2-12) will be executed *n* times, methods *center*(), *calculateRMSD*() and *rotate*() do some calculations for every CA atom so their complexity is also *n*, the while loop (lines 8-12) will be executed maximally (*threshold - initial_angle*)/*step* times. So, the total complexity of the algorithm is equal to *n* (execution of the main loop) times (*threshold - initial_angle*)/*step* (execution of the while loop) times *n* (execution of rotate() and calculateRMSD() functions), together it is O(n^2 * *(threshold - initial_angle)/step*). The value of *(threshold - initial_angle)/step* may be small and good results may be obtained. In my implementation this parameter is equal to 4 or 5, and my tests have shown that a bigger value of the rotation did not result in a smaller RMSD. So, in the total complexity the main factor is $O(n^2)$.

Presented **Algorithm 9** was the first one used to analyze MD trajectories (see Cases studied). Later, after doing numerous computer experiments, it has been improved and **Algorithm 10** presented below has been introduced. In this algorithm "the center of mass" concept was used.

Def. 42. The center of mass $\overrightarrow{r_0}$ of the molecule is the unique point at the center of a distribution of mass in space given by the formula: $\overrightarrow{r_0} = \frac{\sum_k m_k \overrightarrow{r_k}}{\sum_k m_k}$ where *k* is index of atoms in the molecule, m_k is mass of *k*-th atom, $\overrightarrow{r_k}$ is position of *k*-th atom.

Algorithm 10. Structural alignment algorithm with rotations using centers of mass.

Input: Two structures A and B represented by lists of CA atoms positions in $\mathbb{R}^3$ space - the number of CA atoms must be the same, as it is assumed that both

structures represents the same protein, but in different conformations. A threshold for the maximum rotation angle, the initial angle and the step of the rotation.

Output: RMSD calculated between structures A and B.

Steps:

```
1. begin
2. massCenterA := calculateMassCenter(structureA);
3. massCenterB := calculateMassCenter(structureB);
4. center(structureA, massCenterA);
5. center(structureB, massCenterB);
6. RMSD := calculateRMSD(structureA, structureB);
7. ang := initial_angle;
8. while((calculateRMSD(structureA, structureB) < RMSD) AND (ang <
   threshold))
9.    begin
10.   rotate(structureB, ang);
11.   ang := ang + step;
12.   end
13. end
```

The **Algorithm 10** is very similar to the **Algorithm 9**, the main difference is that calculations are not performed for every CA atom, but only for the center of mass for every structure. This allows to avoid a long loop in lines 2-13 from the **Algorithm 9**. At the begging both centers of masses are calculated for both structures (lines 2-3), being simply points in space. Then the structures are centered in such a way that both centers of masses are at the beginning of the coordinate system. After that, RMSD between the structures is calculated (line 6) and a rotation by a small angle is performed, similar to the one from the **Algorithm 9**.

The **Algorithm 10** has a smaller time computational complexity than the **Algorithm 9** because the long loop in lines 2-13 is avoided. For calculation of the center of mass a position of every atom has to be checked. Here only CA atoms are used, so the complexity of *calculateMassCenter*() method is *n* (number of CA atoms), the same as *center*() and *rotate*() methods. The *while* loop in lines 8-12 will be executed similarly as before (*threshold* - *initial_angle*)/*step* times. Thus, the total complexity of the algorithm is equal to (*threshold* - *initial_angle*)/*step* (execution of the *while* loop) times *n* (execution of *center*(), *rotate*() and *calculateMassCenter*() functions), together it is O(*n* * *(threshold* - *initial_angle)/step*), where like before *(threshold* - *initial_angle)/step* may be small integer.

The presented algorithms are approximate, however they are quite fast - especially **Algorithm 10**. For the same input data tests, the following structural alignment algorithms were applied: **Algorithm 9**, **Algorithm 10**, Combinatorial Extension Algorithm (CE) [138] and FATCAT algorithm [140]. CE and FATCAT are implemented in BioJava package [139]. Thirty nine structures of one protein from

the consecutive frames were aligned to a structure taken from the first frame. Only CA atoms in all algorithms were used for the alignment. The time of the execution and the smallest obtained RMSD between the structures were calculated.

Table 14. Execution times of structural alignment algorithms for the same data, in ms.

Algorithm 10	**Algorithm 9**	CA algorithm	FATCAT algorithm
43	433	1613	6059
15	308	1178	5541
5	316	1008	5288
4	372	976	5347
5	309	1084	5346
4	372	965	5399
4	345	991	5370
5	323	1044	5311
2	308	971	5397
2	308	967	5331
1	293	1059	5377
2	292	974	5318
2	293	970	5351
2	299	983	5307
2	293	969	5360
2	306	1248	5319
2	299	1024	5359
2	307	1042	5294
2	305	974	5343
3	307	952	5274
2	289	959	5417
2	295	973	5437
2	302	1045	5411
2	298	1270	5434
1	292	1553	5509
2	293	1504	5347
1	289	1227	5315
2	288	1007	5404
1	283	941	5389
0	290	959	5486
1	290	961	5429
1	293	968	5381
1	287	974	5481
1	294	963	5411
1	276	981	5412
1	280	969	5456
0	289	936	5383
2	300	957	5373
2	317	981	5384
1	313	986	5367

Table 15. The smallest RMSD which have been obtained between structures using different algorithms, in Å - first four columns. Then selected differences in results between the algorithms. Averages values in the last row.

Algorithm 10	**Algorithm 9**	CA algorithm	FATCAT algorithm	**Algorithm 9 - Algorithm 10**	CA - **Algorithm 9**	CA - **Algorithm 10**	CA - FATCAT
0.673	0.679	0.666	0.666	0.006	-0.013	-0.007	0.000
0.716	0.719	0.702	0.702	0.003	-0.018	-0.014	0.000
0.793	0.797	0.770	0.770	0.004	-0.027	-0.023	0.000
0.938	0.943	0.898	0.898	0.005	-0.046	-0.041	0.000
0.893	0.895	0.856	0.856	0.002	-0.038	-0.037	0.000
0.966	0.970	0.906	0.906	0.003	-0.064	-0.060	0.000
0.969	0.979	0.920	0.920	0.009	-0.059	-0.050	0.000
1.074	1.078	1.015	1.015	0.004	-0.063	-0.059	0.000
1.040	1.050	1.001	1.001	0.010	-0.050	-0.039	0.000
1.076	1.077	0.984	0.984	0.001	-0.093	-0.092	0.000
1.097	1.107	1.049	1.049	0.010	-0.058	-0.048	0.000
1.219	1.237	1.132	1.132	0.018	-0.105	-0.087	0.000
1.265	1.279	1.151	1.151	0.015	-0.128	-0.114	0.000
1.212	1.215	1.117	1.117	0.003	-0.098	-0.094	0.000
1.181	1.185	1.118	1.118	0.004	-0.066	-0.062	0.000
1.276	1.281	1.177	1.177	0.005	-0.104	-0.099	0.000
1.121	1.126	1.091	1.091	0.005	-0.035	-0.030	0.000
1.105	1.107	1.068	1.068	0.001	-0.039	-0.037	0.000
1.131	1.132	1.082	1.082	0.001	-0.050	-0.049	0.000
1.195	1.205	1.138	1.138	0.010	-0.066	-0.057	0.000
1.183	1.184	1.171	1.171	0.001	-0.013	-0.012	0.000
1.201	1.203	1.143	1.143	0.002	-0.060	-0.058	0.000
1.384	1.389	1.257	1.257	0.005	-0.133	-0.127	0.000
1.377	1.388	1.238	1.238	0.011	-0.150	-0.139	0.000
1.392	1.396	1.265	1.265	0.004	-0.130	-0.126	0.000
1.421	1.426	1.278	1.278	0.005	-0.148	-0.143	0.000
1.389	1.393	1.257	1.257	0.004	-0.135	-0.132	0.000
1.325	1.329	1.243	1.243	0.004	-0.086	-0.082	0.000
1.326	1.331	1.213	1.213	0.005	-0.118	-0.113	0.000
1.427	1.435	1.260	1.260	0.008	-0.176	-0.167	0.000
1.434	1.437	1.230	1.230	0.003	-0.207	-0.204	0.000
1.393	1.398	1.097	1.227	0.005	-0.301	-0.296	0.130
1.344	1.345	1.079	1.195	0.001	-0.265	-0.264	0.116
1.289	1.291	1.056	1.183	0.002	-0.235	-0.232	0.127
1.332	1.335	1.095	1.238	0.003	-0.239	-0.236	0.143
1.462	1.465	1.221	1.360	0.002	-0.244	-0.241	0.139
1.381	1.384	1.130	1.262	0.003	-0.254	-0.251	0.132
1.487	1.496	1.149	1.290	0.009	-0.347	-0.338	0.141
1.376	1.376	1.114	1.247	0.001	-0.262	-0.262	0.133
1.518	1.527	1.209	1.337	0.009	-0.317	-0.309	0.127
Avg: 1.210	**1.215**	**1.089**	**1.118**	**0.005**	**-0.126**	**-0.121**	**0.030**

According to data presented in **Table 15**, **Algorithm 9** and **Algorithm 10** give very similar results, the differences being usually at the level of thousandths of the Å. Both algorithms are a little bit worse than the CE algorithm, about 0.1-0.4 Å in the obtained RMSD (here a smaller result is better). At the same time, we see that for a few last frames the FATCAT algorithm is worse than CA. Those results suggest that the CA algorithm is the best for the data considered here, however **Algorithm 9** is about 3-4 times faster, and **Algorithm 10** is about 500 times faster, than CA (see **Table 15**). Because the structural alignment is performed hundreds or thousands of times during OPOC execution (see **Algorithm 11**), the shortest possible calculations time is very important and such small errors in the obtained RMSD are acceptable. In our case I do not need the best solution of the alignment problem, but any reasonably good approximation is enough. One should note that even well establish methods, like CA and FATCAT, may give different results, (see **Table 15**).

Algorithm

The general idea of the OPOC algorithm is not very complicated, however some issues have to be considered during the MD Petri net generation. In particular, we need to check if the current conformation was previously obtained, or if there exists a transition between the current and the previous conformation.

Algorithm 11. One Place One Conformation algorithm

Input: PDB files which contains MD trajectories, a parameter *threshold*, which is necessary to distinguish two conformations.

Output: MD Petri net described by lists of places, transitions and arcs. Optionally - files with the description of places and transitions.

Steps:

```
1. begin
2. for every PDB file:
3.    begin
4.    frame := readFrame();
5.    prevPlace := places.get(previousFrame);
6.    if(NOT places.find(frame, threshold)) then
7.      begin
8.      place := createPlace(frame);
9.      places.add(place);
10.     transition := createTransition(prevPlace, place);
11.     transitions.add(transition);
12.     createArcs(prevPlace, transition, place);
13.     end
14.   else
15.     begin
16.     place := places.get(frame, threshold);
17.     if(NOT transitions.contains(prevPlace, place) then
```

```
18.         begin
19.         transition := createTransition(prevPlace, place);
20.         transitions.add(transition);
21.         createArcs(prevPlace, transition, place);
22.         end
23.       end
24.    end
25. end
```

The algorithm reads the data from every input file frame by frame (lines 1-4). When the next frame is read, it is checked if this conformation has been previously obtained (line 6). Here, to compare the current conformation (from the current frame) and the previous conformations (i.e. obtained from the previous frames represented by appropriate and unique places) the RMSD measure and fast structural alignment **Algorithm 9** or **Algorithm 10** can be used. The current frame is aligned to the conformation encoded in every existing place and the smallest calculated RMSD is examined to check if it smaller than the threshold. If such a place is not found (lines 6-13), it means that we encountered a new unique conformation, therefore a new place is created (lines 8-9) in the PN. Of course, if the place also does not exist, then no transition connected to this place is present in the PN, and such a transition must be created (lines 10-11). After that, the place which represents the previous frame is connected as the input place to the newly created transition and the new place just generated from the current frame is connected as the output place (line 12). If the current conformation has been previously obtained (lines 15-23) it must be checked whether there is also a transition from the previous place/frame to the place which corresponds to the present frame (lines 17). If it exists, there is nothing to do, if it does not exist (lines 17-22) we need to create a new transition (lines 19-20). Thus, a connection from the place which corresponds to that previous frame to the new transition and a connection from the new transition to the place of the current frame are created (line 21).

The computational complexity seems to depend only on the size of the input data, however additional calculations are hidden inside the places.*find*() method, which compares places created so far with the current conformation. This comparison also involves an important step: a structural alignment. So, this method will have complexity equals at least number of places multiplied by complexity of the structural alignment algorithm.

Cases studied

The MD Petri net obtained from the OPOC algorithm is usually quite small - much smaller than the net obtained from the OPOA algorithms. It is a natural observation because OPOC generated network describes the movement of the whole molecule and in OPOA algorithms it describes the motions of every amino acid or every atom. The size of the network depends on the value of the threshold parameter, see

Fig. 31. In this figure four MD Petri nets are presented, all generated by the OPOC algorithm with the same structural alignment and for the same trajectory, but for different threshold parameters. An increasing number of places and transitions can be observed for the increasing threshold, so the network is more precise. The size of PN depends also on the length of the simulation, a protein for which the simulation has been calculated and on a number of the trajectories used to generate the network.

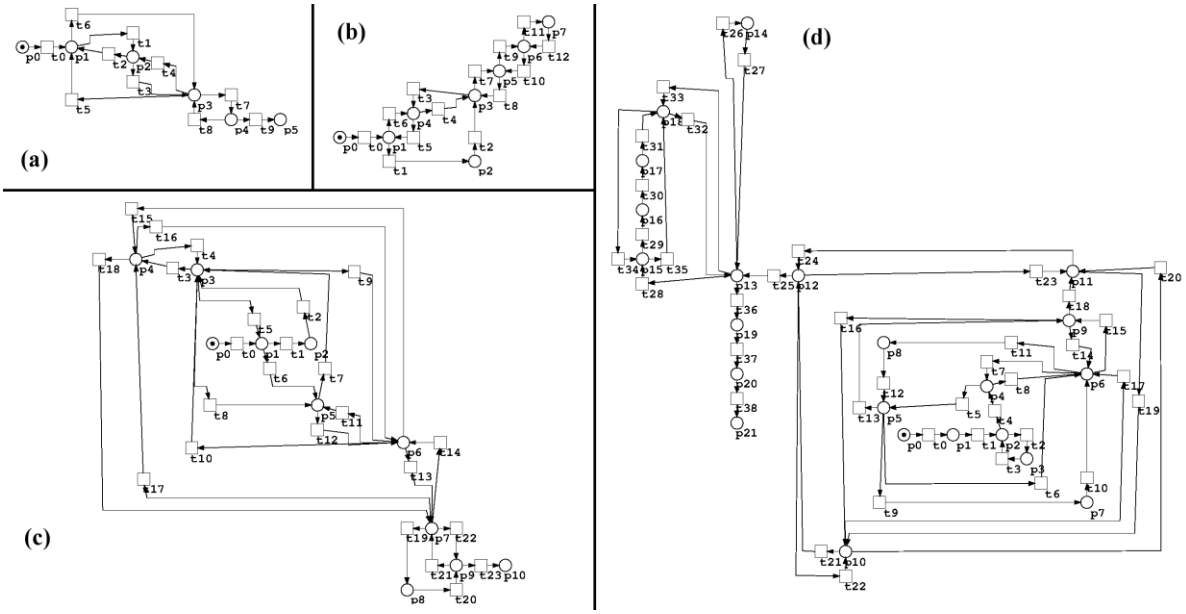


Fig. 31. An example of MD Petri nets generated by the OPOC algorithm with **Algorithm 10** used as the structural alignment tool for the same trajectory, but with different thresholds: (a) 1.6Å, (b) 1.5Å, (c) 1.4Å, (d) 1.2Å.

The goal of the OPOC algorithm, like of others presented in this chapter, is to facilitate the analysis of the MD trajectories. In order to achieve this goal the generated MD Petri net has to represent the main features and events hidden in the trajectory used to generate the network. Only then conclusions drawn from PNs will have a chance to be reliable, especially when those conclusions refer to features which are difficult to pick up by human inspection of the trajectory. From a large pool of trajectories and numerous Petri nets generated during this project here only three MD Petri nets were selected and are presented, just to show the reliability and utility of the OPOC generated networks. Following "case studies" were generated using **Algorithm 9**, however Petri nets created with **Algorithm 10** are very similar and reproduce the same features. These results has been submitted for publication in *Rairo-Operations Research* journal [141].

Let us analyze at the beginning the MD Petri net generated from one SMD simulation of transthyretin (TTR, wild type) tetramer dissociation into two dimers. CA atoms from segments A and B were fixed and the virtual force was added to all CA atoms from chains C and D (see **Fig. 32**). The TTR initial structure has been obtained from the PDB record 1ICT [142]. The analyzed trajectory was 10ns long.

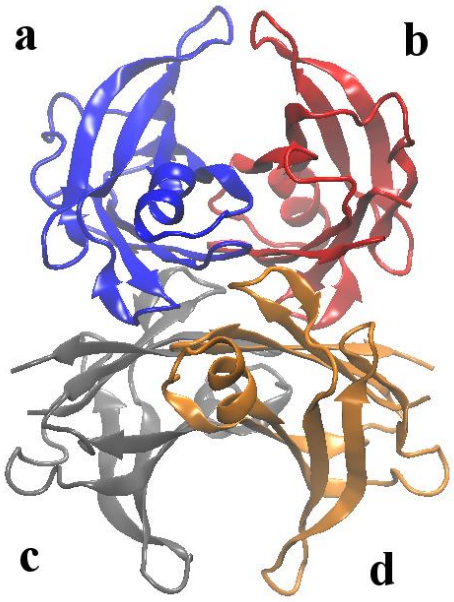


Fig. 32. Structure of TTR with marked segments (a, b, c, d).

In a typical SMD simulation protocol the structure of a protein analyzed is gradually deformed in order to examine its mechanical properties and to observe possible hydrogen bond breaks. During the simulation the pulled and fixed segments separate from each other and the RMSD of the whole system obviously rises. Since adding new places to the PN in the OPOC algorithm depends on RMS distance between the structures from consecutive frames then a general shape of the generated network can be easily predicted. Due to these systematic deformations of the protein, new places are created in PN when the RMSD value between the current frame and the previously created places is bigger than the set-up threshold. As system modeled changes and new arising conformations are not usually present in the already generated part, the created network will be only a straight chain of places and transitions. Indeed, a MD Petri net generated for this TTR trajectory has that form. In **Fig. 33** the net generated for a threshold of 1.3 Å is presented. However, networks for different thresholds were also generated and they had the same general structure.

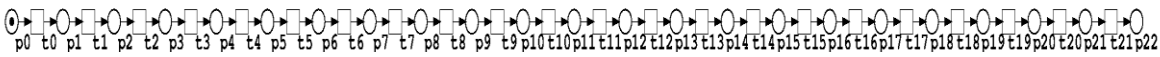


Fig. 33. A Petri net generated for TTR SMD simulation with the threshold of 1.3Å.

The second case studied is SMD dissociation of the complex chemokine MCP-1 from its antibody (see Chapter 4). The initial structure was obtained from the PDB data bank (code 2BDN) [99]. MCP-1 was used here because this molecule is very small and the changes of its conformations can be easily monitored by watching the trajectory, and then by comparing frames to the OPOC generated MD Petri net. In the paper [12] and in Chapter 4 SMD studies of this complex were presented and many pulling directions were tested. For the generation of the MD Petri net a

direction was randomly selected (V3 in **Table 9**). If the protocol of the OPOC algorithm was the same as in the first example, the MPC-1simulation would have the same scenario and the resulting Petri net would have the same i.e. trivial form of a straight graph. However, in this case the MD Petri net was generated for the MCP-1 fragment only. The antibody, the second partner in the complex, was not affected.

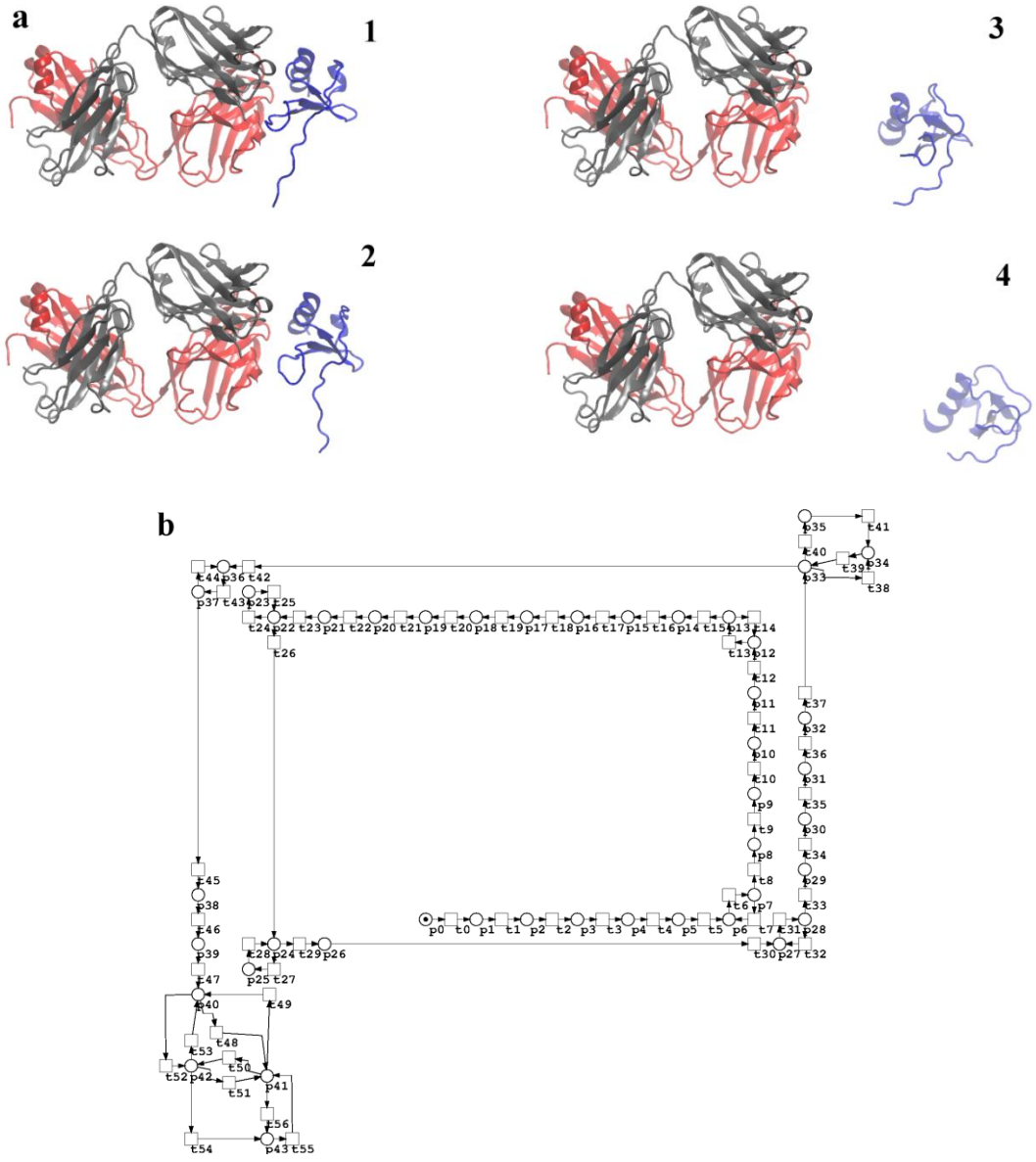


Fig. 34. (a) Four frames from SMD simulation of MCP-1 (blue) dissociation from the antibody (black, red). 1 – the starting frame, 2 – the structure when bonds between the antibody and the antigen are broken, 3 – conformation changes during dragging, 4 – conformation obtained at the end of the simulation. (b) Petri net generated by the OPOC algorithm for the same SMD trajectory with threshold of 1.1 Å.

An advantage of the OPOC algorithm is its versatility. The input to the algorithm may be a list of CA atoms from the whole complex but only a chosen part of a

system is also acceptable. In order to analyze if any new information may be obtained by such methodology, in the second case only MPC-1 atoms were used to generate the new PN (see **Fig. 34**). One can see that the form of this new PN is different than that in **Fig. 33**.

A visual inspection of the trajectory shows that during the first part of the SMD simulation the MCP-1 is gradually deformed. Two opposite forces - the pulling force and the attraction forces between the antigen and the antibody modify the structure. Finally the bonds are broken and the chemokine dissociates. However, during this dissociation the structure of MCP-1 is still changing, which is probably the effect of relaxation (because the antigen is no longer in the complex), and it may adopt a new conformation - more suitable for the unbound state. During this part of the simulation the MCP-1 is dragged in the water and the long loop at the end of the molecule moves closer to the main part of the protein. Also other changes may be observed (see **Fig. 34a**). At the end of the simulation the molecule adopts a more stable structure and large changes in the conformation cannot be observed anymore. Now, one may try to indicate these conformational transitions in the more compact MD PN representation.

In **Fig. 34b** the MD Petri net generated by OPOC algorithm with threshold 1.1 Å is presented. It is similar to the network presented in **Fig. 33**, however some modifications can be observed. One can see that the majority of the PN presents a sequence of places and transitions which corresponds to conformational changes typical for the SMD simulations. This feature of MD PN strongly corresponds to the observations made during the visual analysis of the trajectory. In this part of the net the protein goes back a few times to a previous conformation or adopts new ones, and this phenomenon affects the PN chain of places and transitions. That feature, present in the PN graph, suggests that MCP-1 molecule has some opportunity to relax and to sample new regions of the conformational space. It may happen because the structure of MCP-1 is not deformed in a steady way during the simulation, not like TTR in the previous example. The last part of the network (places 40-43, transitions 48-55) is different. It shows that MCP-1 has already finished quick changes of conformations, and at the end of the SMD simulation it probes the limited region of the conformational space. These easy to obtain observations are in one to one correspondence to the tedious visual inspection of the trajectory.

The two cases presented above show that PNs generated by the OPOC algorithm grasp main events present in the MD trajectories. They provide a simple, compact and effective tool for massive analysis of SMD data. The last example will show further advantages of the OPOC in the analysis of several long MD simulations.

The last Petri net presented in this section has been generated from two classical, MD simulations of TTR mutant L66P, both trajectories were 100 ns long. The original structure was obtained from PDB data bank (code: 1ICT) [142]. Both

trajectory files were split into two files of 50ns. Such long trajectories are very difficult to analyze, "manual" analysis n by watching whole files is almost impossible due to the length of the simulations and the size of the protein - if one does not know when and where to watch, it is not possible to recognize interesting events or features. Thus a simplified MD PN approach is particularly desirable in this case. The OPOC threshold was 1.5Å and the resulting PN is presented in **Fig. 35**. This network has 22 places and 105 transitions. The PN generated in the same conditions for a wild type TTR has only 19 places and 57 transitions. The PN generated for the second TTR mutant V30M has 25 places and 82 transitions. Physically, all three systems are quite similar, as they have almost the same structure and numbers of atoms, but a clear disproportion in number of places and transitions in PN (**Fig. 35**) is visible. Transitions describe changes of conformation in the OPOC algorithm so those different numbers suggest that during the simulation L55P very changes its conformations more often than the other two systems. More specific conclusions may be drawn from closer inspection of the network.

In both simulations the first place is p_0,which is artificially added starting place and does not have any real physical meaning. Transitions t_{29} and t_{91} are also artificial, and were added to the PN because of the division of the MD trajectories into two files. So, the first conformation obtained in both simulations is the one described by p_1. Only four conformations are common in both trajectories (p_1, p_2, p_4, p_7) and all of them are observed in the first part of the simulations. Such a small number of common conformations observed at the beginning (four for MD PN generated for WT TTR and only two for TTR mutant V55P) is typical for other MD PN generated by the OPOC algorithm. The low value of common places number shows how rich a given conformational space is and indicates that just a few simulations are not sufficient to scan it all.

The second observation from the MD PN from **Fig. 35** refers to the first trajectory. One can see that in this part of the network (yellow and blue places) a lot of transitions occur, and some conformations can be transformed into any other, with backward transitions into initial conformations being also possible. It suggest that in this simulation the protein cannot find any stable area in the conformational space, from every conformation can go back to one of the initial conformations and the L55P is sampling different regions the conformational space all time. Such observations would be very difficult to make by watching the MD trajectory only.

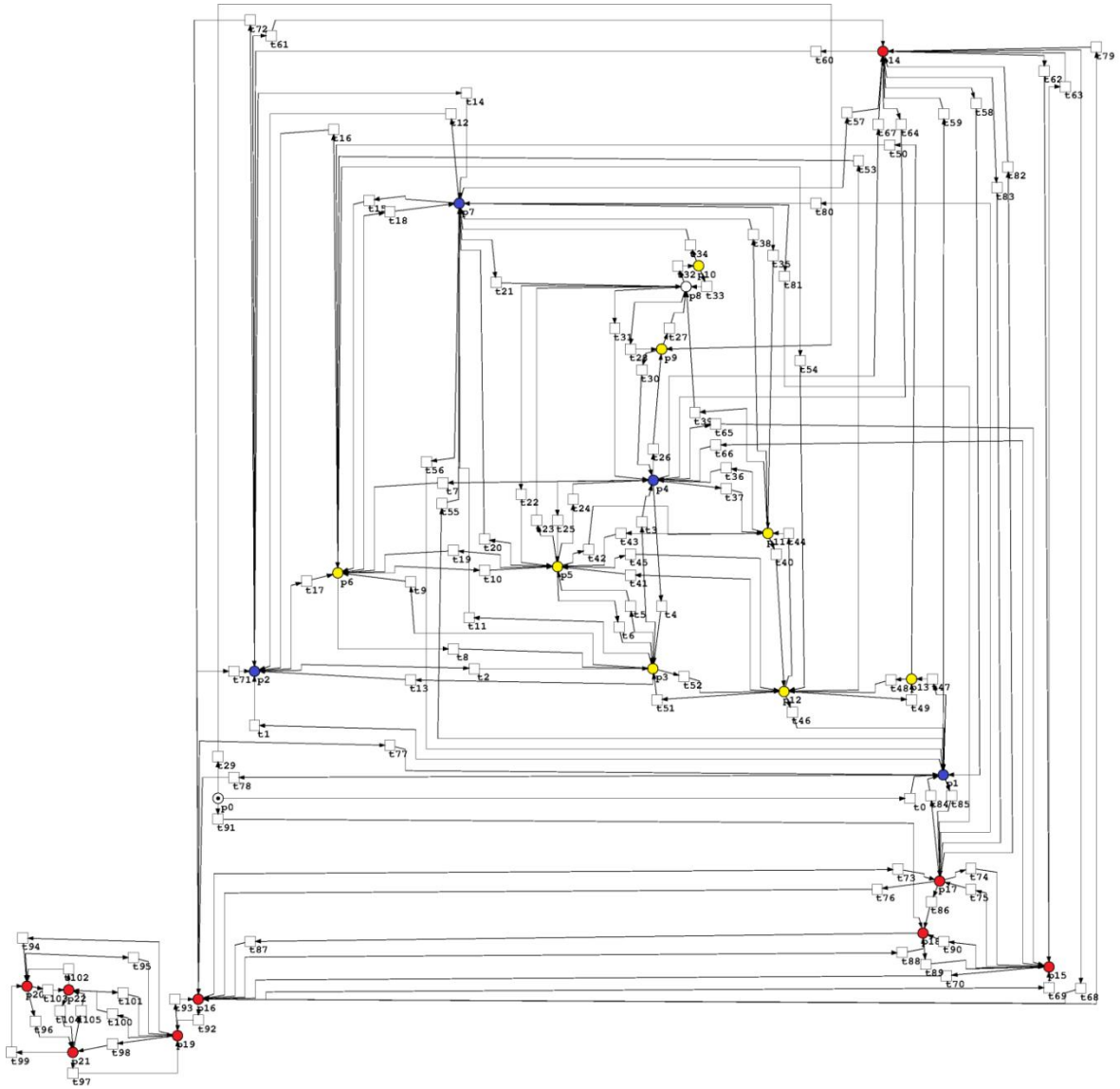


Fig. 35. MD Petri net generated by OPOC algorithm for two 100ns MD simulations of TTR mutant L55P (1.5Å threshold). Places common in both simulations are in blue, places observed only in the first simulations are in yellow and in second only are in red.

The behavior of L55P in the second trajectory is different. From the starting set of four common conformations the protein transforms into conformations represented by places: p_{14}, p_{16}, p_{17} and further on p_{15}, p_{18}. We see from the PN graph (**Fig. 35**) that many transitions exist between those conformations. On the other hand, from the conformation p_{16} the system can also transform into conformation p_{19}, and even though there is an inverse transition $p_{19} \rightarrow p_{16}$, the L55P protein "prefers to stay" in this separate part of the PN throughout the rest of the simulation. This separated "peninsula" is formed by places p_{19}-p_{22} (see **Fig. 35** lower left corner). This suggest that at the beginning of the simulation the protein also scans the conformational space to find a more stable conformation and it finally finds some close-by local minima in the limited conformation space. This observation would be extremely difficult to make by simply watching the trajectory. The created PN shows that

both analyzed simulations were quite different dynamical processes. Thus, in my opinion, the MD PN graphs developed here are new useful computational tools in theoretical biology/structural bioinformatics studies.

Features

Our case studies have shown that PNs generated by the OPOC algorithm properly reflect events and features present in MD trajectories, therefore the proposed approach can be used in routine studies. In this section some general features of the new algorithm are discussed.. The OPOC algorithm is one of the clustering algorithms - places may be treated as clusters, many frames may be assigned to one place/cluster. The degree of the clustering depends on the threshold parameter. Such analysis is useful in MD research and may reduce the number of frames which have to be inspected. However, it is "per se" nothing new, such clustering algorithms are often used in simulation analysis [127]. The new feature of the OPOC algorithm is description of the relationship between clusters. Relationships are coded by transitions. Looking at transitions one can see from which conformation (or a cluster) the protein can transform into another one and what is those transformations path. Such a type of the study is also possible using other methods like, for example, Principal Component Analysis (PCA, [143]), where for every cluster a list of assigned frames is generated. However, in PCA such an analysis requires additional computations and in the OPOC algorithm it is a natural part of the algorithm.

Another advantage of MD PN is that the relationships between conformations and their changes are generated as a PN graph. Graph-based representations of data have known merits [144]. The PN graph describes a journey of a protein through the conformational space and can be easily displayed. One can quickly (after inspection of a few nets) get practice in analyzing of the MD PN generated by OPOC algorithm only by looking at the network graph. One can quickly find if in the analyzed MD trajectory a protein quickly finds any stable conformations or rather constantly probes new regions of the conformational space. Such inspections of PN graphs help to decide what the character of the studied dynamical process is. A similar complex analysis is quite difficult and cannot be performed without sophisticated algorithms (i.e. on computer graphics only).

My own analysis of MD Petri nets generated by the OPOC algorithm was focused on the features of the conformational spaces. Almost 2000 ns of simulations of TTR wild type and its two mutants were examined at 300 K, and graphs were generated (PN graphs not shown, data from mgr inż. Rafał Jakubowski). I have found that despite a huge size of 2 microsecond trajectories conformational space new MD simulations still added new places to my PNs, so the conformational space was not fully probed in 2 microseconds. I noticed that only a few structures were reproduced when new 50 ns trajectories were added to the dataset. In another case studied twenty 2 ns long simulations of MCP-1 were used to generate a single

MD Petri net by the OPOC algorithm. One should note that MCP-1 is a very small protein. The results were similar to the TTR example. Only about 20% of the places were common for two or more trajectories, the rest of places (representative conformations) were unique for every simulation. So, the OPOC algorithm gives an insight into the character of the conformational space, helps to set up a proper simulation protocol (how long a simulation should be to get converged results) and facilitates finding the links between separated regions of the conformational space.

5.2.3 Contacts algorithm

Overview

In the contacts algorithm (CON) different approach to the PN generation is used. In the OPOA and OPOC algorithms the MD Petri net obtained refers to coordinates of atoms in the $\mathbb{R}^3$ space. In both cases the places are linked to points in the $\mathbb{R}^{3NCA}$ space (NCA is a number of CA atom, OPOC) or the $\mathbb{R}^3$ space (OPOA). In contrast to that, in the CON algorithm places represent contacts between two amino acids. The analysis of contacts is a very important part of any MD trajectory research process. Contacts are widely used to study interactions between molecules or even for protein folding methods [145].

Transitions usually represent some actions or events. In the CON algorithm they represent the changes between contacts calculated for every MD frame. In order to organize those changes in some systematic way I assumed that one transition corresponds to one frame. The input places for the transition are all the contacts from the previous frame and the output places are the contacts found in the current frame. In such a representation the PN presents the evolution of contacts over the MD trajectory. Many tokens are present in the MD Petri net generated by the CON algorithm. Tokens located in appropriate places mark the existence of contacts in a given frame. During the MD trajectory number of contacts obviously changes, so the number of tokens in this PN is not constant. It is equal to the number of pairs of amino acids which are in contact in the current frame.

The idea of CON algorithm is quite straightforward, however, a calculation of the contacts is not so simple. Generally, different definitions of contacts between amino acids have been proposed in the literature [146]. In my work I have used the following definition contacts between amino acids:

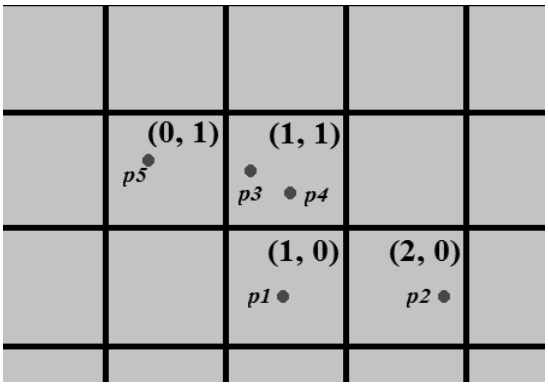
Def. 43. Two amino acids *a1* and *a2* are in contact if the distance between their CA atoms is smaller than a given constant *c*:

$$\|CA_{a1} - CA_{a2}\| \leq c. \tag{5.4}$$

The constant c is given by the user, however, by default it is 8Å, like in [116].

Calculation of contacts

There are many tools to calculate the contacts between amino acids [116, 147-148]. In my implementation the following algorithm, called Contact Calculating Algorithm (CCA), was used. In the CCA algorithm contacts are only calculated, in the CON algorithm they are used to create a PN. In the CCA algorithm, similarly to the OPOA algorithm, the $\mathbb{R}^3$ space is transformed into another three dimensional discrete space (a grid) like in **Def. 38**. One of the important features of the mapping function used is the following: when the coordinates of one point from C_k^3 are given, it is easy to obtain the coordinates of its neighboring points. It is done simply by adding or subtracting 1 to each coordinate. Any other mapping function can be here applied as well, however, it should have two properties: it should be a surjection and not an injection, and the coordinates of the neighboring points should be obtainable from the coordinates of every point.



AtomsPositions:
PossibleNeighbors:

(0, 1): p5	(0, 1): p1, p3, p4
(1, 0): p1	(1, 0): p2, p3, p4, p5
(1, 1): p3, p4	(1, 1): p1, p2, p5
(2, 0): p2	(2, 0): p1, p3, p4

Fig. 36. An example of *AtomsPositions* and *PossibleNeighbors* arrays in the $\mathbb{R}^2$ space. Points in the space are marked by red dots, points in the C_k^2 space are marked by black boxes and their coordinates are given in brackets.

In the CCA algorithm two arrays are used. In both indexes should be C_k^3 points coordinates. During the implementation it will not be comfortable to define such data structure, so it is better to create arrays indexed by natural numbers and additional arrays to translate indexes into C_k^3 points coordinates. However, in this chapter such a problem of arrays indexes will not be considered. The first array will be called *AtomsPositions* and an element of index *p* of the array will contain list of

atoms which are in the point *p* in the C_k^3 space. The length of this array will be maximally equal to the number of CA atoms *n*, the same as the number of elements in the whole array because only *n* points can be occupied. The second array is called *PossibleNeighbors* and an element of index *p* of this array contains a list of atoms which possibly are in contact with an atom in the point *p*. The length of this array is at most 26*n* - the number of CA atoms times the number of the points neighboring to every point in the C_k^3 space. An example of the arrays for two dimensional case is presented in **Fig. 36**. The $\mathbb{R}^2$ can be mapped into C_k^2 in an analogical way like the $\mathbb{R}^3$ space, similarly like in **Fig. 27**.

Now, the contact calculating algorithm may be introduced. It consists of two loops – within the first one atoms are added to the arrays, during the second one the arrays are analyzed and the contacts are found.

Algorithm 12. Contact calculating algorithm (CCA).

Input: A list of *n* points, between which the contacts have to be calculated. The value of a threshold *c* which defines the distance between two atoms being in the contact. Parameter *c* is also a resolution of the grid dividing the $\mathbb{R}^3$ used to obtain the C_{c-1}^3 space, *c* is an integer from the interval [6, 12] Å.

Output: The list *contacts* of pairs of points which are in the contact.

Steps:

```
1. begin
2. AtomsPositions := NULL;
3. PossibleNeighbors := NULL;
4. contacts := NULL;
5. mapping := NULL;
6. for i := 0 to n do
7.    begin
8.    point := getCoordinates(atom[i]);
9.    AtomsPositions[point].add(atom[i]);
10.   mapping[i] := point;
11.   neighbors := getNeighbors(point);
12.   for j = 0 to 26 do
13.     PossibleNeighbors[neighbors[j]].add(atom[i]);
14.   end
15. for i := 0 to min(n, AtomsPositions.length()) do
16.   begin
17.   list := AtomsPositions[mapping[i]];
18.   for j := 0 to list.length() do
19.     begin
20.     for k := j + 1 to list.length() do
21.       if(j <> k) then contacts.add(list.get(j), list.get(k));
22.     listNeighbors := PossibleNeighbors[mapping[i]];
23.     for k := 0 to listNeighbors.length() do
24.         if(distance(list.get(j),listNeighbors.get(k))<c) then
25.           contacts.add(list.get(j), listNeighbors.get(k));
```

```
26.      end
27.    end
28. end
```

The CCA algorithm consists of two loops. During the first one (lines 6-14) every point *P* from the input list is processed: coordinates (l_P,m_P,n_P) in the C_k^3 space of the point are returned (line 8), the point is added to the *AtomsPositions* array to the list located at the position indexed by (l_P,m_P,n_P) (line 9). The auxiliary mapping array is necessary to obtain pointers which are indices to freely iterate over the *AtomsPositions* array. Then the coordinates of every point neighboring to the currently processed atom's point are calculated (line 11). Every point in the C_{c-1}^3 has 26 neighboring points so the next *for* loop iterate 26 times (lines 12-13). In the loop the currently processed atom's point is added to the every *PossibleNeighbors* array to the lists corresponding to every neighbor (line 13). In the second loop (lines 15-27) every list for the *AtomsPositions* array is processed (lines 17-18). The distance between two atoms which are added to the same list is smaller than the *c* value, because when two atoms are on the same list in *AtomsPositions* array it means that they are in the same point in the C_{c-1}^3 space, which is equivalent to the fact that in the $\mathbb{R}^3$ space they are located in the same cube with the edge size of *c - 1*. So, every pair of atoms from *AtomsPositions* array is added to the final list of pairs in contact (lines 20-21). For every atomic position *i* in the *PossibleNeighbors* array at position *i* a list of atoms which can possibly be in contact with the atom/atoms located at position *i* is saved. So the distance between the currently processed atom and every atom from appropriate *PossibleNeighbors* array element is calculated. If the distance is smaller than *c* atoms are in contact and they are added to the final contacts list (lines 22-26).

The calculation complexity of the first loop is *26n* = O(*n*). The estimation of complexity of the second loop is rather complicated. The loops from lines 15 and 17 will execute in total *n* times because during the execution of those loops every atom will be processed, and it will be processed only once. The total length of all lists from the *AtomsPositions* array is equal *n*. So, the time complexity of the second loop will be: $n * ap_{max} + n * pn_{max}$, where the ap_{max} is the length of the longest list from the *AtomsPositions* array and the pn_{max} is the length of the longest list from the *PossibleNeighbors* array. Of course, in general these lists may contain *n* elements at most, so then the complexity will be O(n^2). But the algorithm is designed to calculate contacts across real molecular structures, and such situations are not possible. Atoms in real systems are scattered in numerous points in the reduced space. Furthermore, when the *c* is between 6 Å and 12 Å large it is not realistic that all molecule's atoms would be in the same point in the C_{c-1}^3 space. Otherwise these atoms would all occupy a cube with the edge smaller than 12 Å in $\mathbb{R}^3$ space. Such a situation cannot occur in nature. In reality Van der Waals forces will not allow to have such small distances between atoms. So, the length of the longest *AtomsPositions*' and *PossibleNeighbors*' lists will be equal to a rather small constant *v* and the complexity of the second loop will be O(*vn*).

The CON algorithm

Having a method to calculate contacts between amino acids, the CON algorithm can be introduced. The algorithm follows the general idea presented above in the Overview. In the CON algorithm PN places represent unique pairs of CA atoms being in contact. Every place corresponds to some real contact. All contacts are somehow represented in places, but multiple MD contacts between atoms from the same pair have only one the same place in the PN net. Dynamics in a real space $\mathbb{R}^3$ results in changes of contacts. These changes are reflected in transitions.

Algorithm 13. The contact algorithm CON

Input: Value *c* which describes the maximum distance between CA atom to recognize pair of amino acids as a pair in contact. The set of files with MD trajectories.

Output: MD Petri net described by lists of places, transitions and arcs. Optionally - files with descriptions of places and transitions names.

Steps:

```
1. begin
2. for every PDB file:
3.    begin
4.    frame := readFrame();
5.    if(frame.getNr() = 1) then
6.          oldContacts := NULL;
7.    else
8.          oldContacts := contacts;
9.    contacts := findContacts(frame);
10.   if(NOT transitions.contains(oldContacts, contacts) then
11.         transition := createTransition();
12.   else
13.         transition := transitions.get(oldContacts, contacts);
14.   for i := 0 to contacts.length() do
15.         begin
16.         pair := contacts.get(i);
17.         if(NOT places.contains(pair)) then
18.               place := createPlace(pair);
19.         else
20.               place := places.get(pair);
21.         createArc(transition, arc);
22.         end
23.   createArcs(oldContacts, transition);
24.   end
25. end
```

In the algorithm every frame from every MD trajectory file is read (lines 2-4). At the beginning of every main loop the set of contacts from the previous frame is remembered in *oldContacts* variable (line 8) - of course for the first frame the previous set of contacts cannot exist and the *oldContacts* will be NULL (line 6).

After that the contacts from the current frames are calculated (line 9). It can be done by any algorithm, for example by **Algorithm 12**. It is checked if the transition corresponding to the current frame already exists. It has to have a set of input places equal the set of contacts from the *oldContacts* and a set of output places equals contacts from the current frame. If it does not exist (line 10) it is created. Then every pair of CA atoms in contact is processed (loop in lines 14-22). If the corresponding place does not exist it is created (line 17-18). The place representing the pair in contact is connected as the output place to the transition corresponding to the current frame (line 21). At the end every place assigned to the previous contacts from the *oldContacts* is connected to the transition as the input places (line 23).

The complexity of the CON algorithm depends on the main loop, which reads input data. It will be executed for every frame from each MD file. This will be multiplied by the length of the list of pairs in contact. This length cannot be predicted and it depends on the molecule. However, one can expect that it will be greater than the number of CA atoms because every standard amino acid is in a contact with at least two neighboring amino acids.

The CON algorithm will lead to PN representing the evolution of contacts present in the molecule during the simulation. It worth noting that many versions of CON inspired algorithms can be designed to represent better relationships between amino acids, for example, an algorithm in which only new contacts are presented. Other possible modification of the algorithm might be based on omission of neighboring amino acids. Such a simplified CON algorithm can be very useful and might present the time evolution of contacts in a simpler way, moreover, the main idea and the scheme would be the same as in **Algorithm 13**. Sometimes even a simple changing of the internal procedure which calculates contacts should be enough to obtain a new, useful variant of the CON algorithm developed here.

5.3 Simulation of the MD Petri nets

The algorithms described above can generate only basic, classical Petri nets. In the present section the modifications of algorithms required to generate extended types of PN are described (5.3.1). Appropriate algorithms for simulations of those extended PN are also presented, as well as a few examples of the simulations. A simulation means here that the distribution of tokens within PN may be monitored, and basing on it, new data on real molecular systems represented by PN are gathered. The marking obtained during such simulations can be used to generate a PDB file – the algorithm of PDB file creation and a number of examples are also presented. The extended algorithms have been applied in simulations of real MD Petri nets. The results, sometimes unexpected, are critically analyzed and possible improvements and lines of developments are suggested.

5.3.1 Generation of the extended types of PNs

Every algorithm described in section 5.2 (OPOAv1, OPOAv2, OPOC, CON) can be used to generate marked Petri nets. However, to create timed, priority based and random priority based PN additional information is necessary. In the paragraph 5.1.2 it was required that priorities should be associated with the number of transitions occurrences. This condition will guarantee correct representation of the relations between conformations. So, every time when a transition occurs during the PN generation, its frequency (i.e. total number of occurrences) will be increased.

In timed Petri nets a representation of time assigned to transitions is required. Luckily, time is present in the MD trajectories as frame numbers, so a natural choice is to use those numbers as the time and assign them to transitions. However, transitions may occur in many frames, especially when a few MD trajectory files can be used and in different files one transition may be present in different frames. So, it should be chosen which time will be used. Petri nets with time are introduced mostly to avoid an unnatural protein unfolding during the SMD simulations. Time in this case is some kind of a guard which secures roughly the correct order of events. The exact order of events is less desirable because the MD Petri net simulation is performed to obtain new features and observe new phenomena during the simulation. In order to ensure such requirements, the time when transitions occur for the first time will be associated to them.

Calculation of time and frequency

In simulations the time and the number of occurrences of transitions (frequency, later designed by NOC) have to be calculated. In both cases the necessary modifications of the algorithms will be insignificant. They are presented below. For the OPOA algorithms only the first variant is shown, because exactly the same operations have to be performed for the second variant. The main difference between the variants occurs after the basic generation of the PN and it will not be affected by calculation of times and frequencies. There is only one important difference. In OPOAv1 places are distinguishable by numbers of CA atoms and coordinates in the C_k^3 space, in OPOAv2 only coordinates differentiate the places. Therefore, the input and output places of transitions will be different in v1 and v2 but it will not impact directly the issue discussed in this paragraph.

Algorithm 14. OPOAv1 with time and NOC calculation.

Input: PDB files, which contain MD trajectories of the same molecule. Parameter *k* for C_k^3, which describes the resolution of the grid dividing the $\mathbb{R}^3$ space.

Output: MD Petri net described by lists of places, transitions and arcs. Files with description of places and transitions names. Information assigned to transitions about time and frequency.

Steps:

```
1. begin
2. times := NULL;
3. frequencies := NULL;
4. for every PDB file:
5.    begin
6.    frame := readFrame();
7.    for every CA atom in frame:
8.       if position(CA, frame) <> position(CA, previousFrame) then
9.          begin
10.         prevPlace := places.get(CA, position(CA, previousFrame));
11.         if(NOT places.contains(CA, position(CA, frame)) then
12.            begin
13.            newPlace := createPlace(CA, position(CA, frame));
14.            places.add(newPlace);
15.            transition := createTransition(CA);
16.            times[transition] := frame.getNr();
17.            frequencies[transition] := 1;
18.            transitions.add(transition);
19.            createArcs(newPlace, transition, prevPlace, CA);
20.            end
21.         else
22.            begin
23.            currentPlace := places.get(CA, position(CA, frame));
24.            if(NOT transitions.contains(prevPlace,currentPlace,CA) then
25.               begin
26.               transition := createTransition(CA);
27.               times[transition] := frame.getNr();
28.               frequencies[transition] := 1;
29.               transitions.add(transition);
30.               createArcs(currentPlace, transition, prevPlace, CA);
31.               end
32.            else
33.               begin
34.               transition:=transitions.get(currentPlace, prevPlace, CA);
35.               time := frame.getNr();
36.               if(time < times[transition]) then
37.                  times[transition] := time;
38.               frequencies[transition] := frequencies[transition] + 1;
39.               end
40.            end
41.         end
42.   end
43. end
```

Lines corresponding to time calculation are marked in red, lines of the variant corresponding to calculations of NOC parameter (i.e. frequency) are presented in blue. For every new transition the time is set as the number of current frame and the frequency is set to one. When a transition occurs once again the frequency is increased. Simultaneously, the number of the current frame is checked. If it is smaller than the time currently assigned to the transition, then new time is set to this smaller number. The same modifications are added to other two algorithms: **Algorithm 15** and **Algorithm 16**.

Algorithm 15. One Place One Conformation algorithm with time and NOC calculation.

Input: PDB files which contain MD trajectories, parameter *threshold*, which is necessary to distinguish two conformations.

Output: MD Petri net described by lists of places, transitions and arcs. Files with descriptions of places and transitions names. Information assigned to transitions about time and frequency.

Steps:

```
1. begin
2. times := NULL;
3. frequencies := NULL;
4. for every PDB file:
5.    begin
6.    frame := readFrame();
7.    prevPlace := places.get(previousFrame);
8.    if(NOT places.find(frame, threshold)) then
9.      begin
10.     place := createPlace(frame);
11.     places.add(place);
12.     transition := createTransition(prevPlace, place);
13.     times[transition] := frame.getNr();
14.     frequencies[transition] := 1;
15.     transitions.add(transition);
16.     createArcs(prevPlace, transition, place);
17.     end
18.   else
19.     begin
20.     place := places.get(frame, threshold);
21.     if(NOT transitions.contains(prevPlace, place) then
22.       begin
23.       transition := createTransition(prevPlace, place);
24.       times[transition] := frame.getNr();
25.       frequencies[transition] := 1;
26.       transitions.add(transition);
27.       createArcs(prevPlace, transition, place);
28.       end
29.     else
30.       begin
31.       transition := transitions.get(prevPlace, place);
32.       time := frame.getNr();
33.       if(time < times[transition]) then
34.         times[transition] := time;
35.       frequencies[transition] := frequencies[transition] + 1;
36.       end
37.     end
38.   end
39. end
```

Algorithm 16. The CON algorithm with time and NOC calculation.

Input: Value *c* which describes the maximum distance between CA atoms used to recognize a pair of the amino acids as a pair in contact. The set of files with MD trajectory.

Output: MD Petri net described by lists of places, transitions and arcs. Files with description of places and transitions names. Information assigned to transitions about time and frequency.

Steps:

```
1. begin
2. times := NULL;
3. frequencies := NULL;
4. for every PDB file:
5.    begin
6.    frame := readFrame();
7.    if(frame.getNr() = 1) then
8.          oldContacts := NULL;
9.    else
10.         oldContacts := contacts;
11.   contacts := findContacts(frame);
12.   if(NOT transitions.contains(oldContacts, contacts) then
13.         begin
14.         transition := createTransition();
15.         times[transition] := frame.getNr();
16.         frequencies[transition] := 1;
17.         end;
18.   else
19.         begin
20.         transition := transitions.get(oldContacts, contacts);
21.         time := frame.getNr();
22.         if(time < times[transition]) then
23.               times[transition] := time;
24.         frequencies[transition] := frequencies[transition]+ 1;
25.         end;
26.   for i := 0 to contacts.length() do
27.         begin
28.         pair := contacts.get(i);
29.         if(NOT places.contains(pair)) then
30.               place := createPlace(pair);
31.         else
32.               place := places.get(pair);
33.         createArc(transition, arc);
34.         end
35.   createArcs(oldContacts, transition);
36.   end
37. end
```

Towards MD PN applications

When the time and NOC of every transition is calculated it can be used to generate extended types of Petri nets. Once they are stored in files they can also be used for other purposes. In particular, new information of molecular system properties may be obtained.

In order to generate timed Petri nets, the firing time function *f* for every transition is necessary (other elements of the PN are the same as in marked Petri nets, see **Def. 20**). The simplest approach is to adopt the value of time calculated during the PN generation as firing time. The same refers to priority-based and random priority-based Petri nets - here the value of the priority function *prio* is necessary (see **Def. 24** and **Def. 28**.). The frequency NOC calculated for every transition will be used to define this function.

It is worth noting that calculations of transitions' times and frequencies are costless in terms of computational complexity of the modified algorithm. This calculation might be performed during normal Petri net generation. One may calculate frequencies for places as well. It will be performed in the same manner like described previously for transitions. An example of an OPOC algorithm capable of NOC for places calculations is shown below. Here the new data are obtained almost without additional costs. The same modifications may be applied to OPOA algorithms and the CON algorithm.

Algorithm 17. One Place One Conformation algorithm with places frequency calculation.

Input: PDB files which contain MD trajectories, a parameter *threshold* necessary to distinguish two conformations.

Output: MD Petri net described by lists of places, transitions and arcs. Files with description of places and transitions names. Information about places frequency.

Steps:

```
1. begin
2. frequenciesP := NULL;
3. for every PDB file:
4.    begin
5.    frame := readFrame();
6.    prevPlace := places.get(previousFrame);
7.    if(NOT places.find(frame, threshold)) then
8.      begin
9.      place := createPlace(frame);
10.     frequenciesP[place] := 1;
11.     places.add(place);
12.     transition := createTransition(prevPlace, place);
13.     transitions.add(transition);
14.     createArcs(prevPlace, transition, place);
```

```
15.      end
16.   else
17.      begin
18.      place := places.get(frame, threshold);
19.      frequenciesP[place] := frequenciesP[place] + 1;
20.      if(NOT transitions.contains(prevPlace, place) then
21.        begin
22.        transition := createTransition(prevPlace, place);
23.        transitions.add(transition);
24.        createArcs(prevPlace, transition, place);
25.        end
26.   end
27. end
```

As mentioned above, the analysis of places and transitions frequencies gives supplementary information about the MD trajectories and the molecules. Firstly, in the OPOC algorithm one may get the preferred conformations and conformation changes "for free". Specific changes in structures can be easily spotted. Similarly, in the OPOA algorithm the preferred positions and movements of each amino acids can be easily found. In the CON algorithm the most persistent contacts can be identified. NOC values may be collected into vectors and such objects calculated for separate MD trajectories may be compared. For example, by a simple subtraction changes between trajectories will be immediately obtained and registered. This possibility is especially useful when the MD trajectories were performed in different conditions. The role of those conditions on dynamical phenomena may be quickly determined.

MD trajectories are usually calculated by using different force fields. For the algorithms presented here it does not matter. A simple parser will solve this problem. Interestingly, vectors of NOC (frequencies) for places and/or transitions for different force fields can be compared and this would give critical information about specific aspects of force fields applied in a study. I believe that NOC analysis of places and transitions may have many more applications, here only a few examples have been mentioned.

5.3.2 Simulation of generated PN

Four algorithms of PN generation were presented, every algorithm can be modified to create marked PN, timed PN, priority-based PN and random priority-based PN. It is quite a big number of combinations, however, when the PN is created the basic simulation algorithm will be the same for all these algorithms. Only a few additional, depending on the type of an algorithm, operations are necessary, for example, to determine the number of places and transitions or the number of places marked in the M_0 marking. In this section, algorithms for MD PN simulations are presented. As an input for a simulation algorithm the network files created during the PN generation step (see **Fig. 42**) are used. PN once created may be used for simulations many times. In my implementation all necessary information is included in those files, so a user only has to set files directory and does not have to

worry about which PN generation algorithm had been used. The output of the simulation algorithm will be a marking stored in the file.

The MD classical Petri nets, created by any of the generation algorithms, can be simulated by **Algorithm 2**. Only a few additional elements are necessary, like the input and output matrices. The generation of the matrices is very simple, so this algorithm will be omitted. Also MD marked PN may be directly simulated by CUDA **Algorithm 6**. For other types of the Petri nets the CUDA algorithm should be modified. In particular, the firing rule should be changed but the main idea is the same.

Simulations of timed PN

The simulation of timed PN requires: (i) basic information about the list of places, transitions and arcs, (ii) data on transitions times. Information (ii) can be generated by **Algorithm 14**, **Algorithm 15** or **Algorithm 16**. In my implementation this information is stored in dedicated files. The transitions times may be taken from other sources as well. The data on times will be used to set a firing time assigned by the *f* firing time function (**Def. 20**). In the simulation algorithm it is assumed that the *f* function is already defined. The simulation is run following the description given in **Def. 23** and the formula (1.7).

Algorithm 18. The simulation algorithm of timed Petri net.

Input: List of places, transitions and arcs (may be located in files). List of firing times associated to transitions. Parameter *steps* describing how many steps of simulations is to be executed.

Output: The marking *M* after *'steps'* steps of the simulation.

Steps:

```
1. begin
2. oldEnabled := NULL; enabled := NULL; times := NULL;
3. previousTime := 0; previousNr := -1;
4. for i := 0 to steps do
5.    begin
6.    enabled := findEnabledTransitions();
7.    for j := 0 to enabled.length() do
8.          begin
9.          nr := enabled.get(j);
10.         if((NOT oldEnabled.contains(nr))OR(previousNr=j)) then
11.               times[nr] := firingTimes[nr];
12.         else
13.               times[nr] := times[nr] - previousTime;
14.         end
15.   fireNr := findMin(enables, times);
16.   previousNr := fireNr;
17.   previousTime := times[fireNr];
18.   oldEnabled := enabled;
```

```
19.    fire(fireNr);
20.    end
21. writeMarking();
22. end
```

The main part of the algorithm is the loop in lines 4-20. Before the loop variables are initialized. *Enabled* is the list of transitions enabled in a current step, *oldEnabled* is a saved list of the enabled transitions from the previous loop, *previousNr* is the number of previously fired transition and *previusTime* is the value of that transition's clock. Those two last values are not necessary and may be obtained by using other variables, however, to clarity the algorithm they were highlighted. The list *times* contains the clock states for enabled transitions. In fact, it may also contain "historical" clock states of transitions that are not enabled currently, however only the values of currently enabled transitions will be used. In every step a new list of the enabled transitions is found (line 6). For every enabled transition its clock state has to be determined - for newly enabled or previously fired it is equal to the value of the firing time function (lines 10-11). For those which were enabled in the previous step, its value of the clock function is decreased by the clock state of the previously fired transition (lines 12-13). After that the transition with the smallest clock state is found (line 15) and fired (line 19). In the next step, the number of this transition will be the number of transition previously fired – *previousNr*, and its time will be *previousTime* (lines 16-17). The current list of the enabled transitions is saved in *oldEnabled* (line 18). At the end, the marking is written to a file.

The complexity of **Algorithm 18** equals to the number of steps multiplied by the number of transitions multiplied by the number of places. It is hard to reduce this number. At first glance, it seems that the complexity can be reduced, however, the list of all enabled transitions may contain every transition. Finding all enabled transitions may require checking all transitions. When the list of the enabled transitions is created for the first time, it is necessary to verify every transition. In the next step only those transitions which have common places with the transition already fired may change their state, so only those have to be checked. This might reduce the complexity. However, in some PNs transitions may have many connections, and eventually every transition must be inspected. During such checking which transitions are enabled or during the firing, a loop over all places is necessary. Thus every place has to be processed and the complexity remains high.

The performed tests have shown that timed Petri nets, as described in paragraph 1.4.4, are not suitable in my study for PN generated by the OPOA algorithms – more details in paragraph 5.2.1. This problem had not been predicted earlier and it was revealed during the tests. However, as it was said above, the time control of a transitions firing is necessary in simulations of MD PNs generated from SMD simulations. Therefore, a new type of PN with time was invented and introduced here. Those new PNs are called the guard time Petri nets.

Def. 44. A guard time Petri net is a seven-tuple: *(P, T, F, W, M_0, f, V)*, where *P, T, F, W, M_0* are like in **Def. 3**, *f:* $T \rightarrow \mathbb{R}^+$ is a firing time function, which assigns a positive real number, called firing time, to each transition, and $V \in \mathbb{N}$ is a clock state.

In the guard time PNs we are focused on a preservation of the general sequence of the transitions firings. Transitions with the highest value of the time function should not be fired before transitions with a small value of the time function. The clock state is some type of a guard in this type of PN. Transitions with a value of the time function higher than the current clock state cannot fire. This control is not as restricted as in timed Petri nets, transitions with values of the time function smaller than the current clock state can fire freely in random order. The clock state must be increased during the simulation, however I do not define how and the rate of the increase may be set arbitrarily, depending on the studied network.

Def. 45. The transition *t* is firable in a guard time Petri net *P = (P, T, F, W, M_0, f, V)* if $t \in enb(M) \wedge f(t) < V$.

The following definitions **Def. 44** and **Def. 45** of the simulation algorithm of guard time PNs will be very similar to one presented for classical, marked PNs (**Algorithm 1**, **Algorithm 2**).

Algorithm 19. The simulation algorithm of guard time Petri nets.

Input: List of places, transitions and arcs (may be located in files). List of firing times associated to transitions. Parameter *steps* describing how many steps of simulations is to be executed.

Output: The marking *M* after *'steps'* steps of the simulation.

Steps:

```
1. begin
2. clockState := 0;
3. i := 0;
4. while(i < steps)
5.    begin
6.    t := findEnabledTransitions();
7.    if(times[t] < clockState) then
8.          begin
9.          fire(t);
10.         i := i + 1;
11.         increase(clockState);
12.         end
13.   end
14. end
```

The main part of the **Algorithm 19** is a while loop (lines 4-13). In this loop one of the enabled transitions is found (line 6). Then, if its value of the time function is smaller than the clock state (line 7), the transition fires (line 9), the number of steps

increases (line 10) and the clock state may be also increased (line 11). In my opinion, it is not recommended that the clock state is the same as the number of a current step of the simulation. It should be more similar to a number of frames from an MD trajectory used to generate the PN, so to the real time. Therefore, in my implementation of **Algorithm 19** the clock state increases every *N* steps of the simulation, when *N* is the number of CA atoms in the studied protein. If the value of the clock function of the selected transition (line 7) is bigger than the current clock state, a next, enabled transition is found (again line 6).

The time complexity of the algorithm will be the same as **Algorithm 1**. The main loop will be executed *steps* times. Finding the enabled transition with a proper value of the time function may require checking of every transition, so its complexity will be *n* * *m*, when *n* is the number of places and *m* is the number of transitions.

Using **Algorithm 19** simulations of PN generated from SMD simulations can be performed and proper order of events will be protected.

Simulation of priority-based PN

For a simulation of priority-based PN the information about transitions priorities is required. It may be obtained, for example, from **Algorithm 14**, **Algorithm 15**, **Algorithm 16** or in the other way. Data can be also provided by an external source. As it was mentioned above, in my implementation priorities are equal to the numbers showing how many times an event represented by transitions occurs in MD trajectories. To prevent totally deterministic simulations of the PN, the **Def. 27** is used to determine which transition should be fired. The following algorithm is an exact implementation of this definition.

Algorithm 20. The simulation algorithm of a priority-based Petri net.

Input: List of places, transitions and arcs (may be located in files). List of *priorities* associated to transitions. Value *steps* describing how many steps of simulations should be executed.

Output: The marking *M* after *'steps'* steps of a simulation.

Steps:

```
1. begin
2. for i := 0 to steps do
3.    begin
4.    enabled := findEnabledTransitions();
5.    fireNr := enabled.getAny();
6.    ConfSet := findConflicts(fireNr);
7.    maxNr := findMax(ConfSet, priorities);
8.    if(fireNr <> maxNr) then
9.          fireNr := maxNr;
```

```
10.    fire(fireNr);
11.    end
12. writeMarking();
13. end
```

The algorithm is very short and rather simple. Additional variables to mark some objects are not necessary, so the algorithm starts with a loop which will be executed *steps* times. In every step, a list of enabled transitions is found and one transition from the list is chosen (line 5). The method *getAny*() was used in the pseudo code to distinguish it from the classical *get*() method. The *getAny*() method may be implemented in any way, for example, it may return one element randomly, or calculated according to a pre-defined formula. It is not recommended to always return to the first transition, because the priority-based PN already has a problem with the deterministic sequence of transitions firings. Our goal is to obtain simulations as non-deterministic as possible. If now *fireNr* transition is fired without any additional operations, a simulation of the marked Petri net will be obtained. According to **Def. 26**, the conflict set of *fireNr* transition is found (line 6) and the transition with the highest priority from the set is found as well. If this transition with the highest priority is different from that *fireNr* transition, *fireNr* will be replaced by one with the highest priority (lines 8-9) and it will be fired (line 10).

The complexity of the algorithm is *O(steps * m * n)*, where *m* is the number of transitions and *n* is the number of places. The main loop will be executed *steps* times. Inside the loop *findEnabledTransitions()*, *findConflicts()* and *findMax()*, depending on the situation, may be executed *m* times. Even when some smart algorithm may be used to reduce the complexity of the mentioned methods for some "evil" Petri nets which can be generated, for example, by the CON algorithm, the complexity will be still at least *O(m)*. The same situation is with places - in *findEnables()*, *findConflicts()* or *fire()* loops over all places may be unavoidable.

In this way, using **Algorithm 20**, one can perform simulations of priority-based PN.

Simulation of random priority-based Petri nets

In random priority-based Petri nets information about priorities is necessary. The only difference with respect to previously discussed nets is that now an additional random variable *X* will be used. The simulation is performed following **Def. 32**.

Algorithm 21. The simulation algorithm of random priority-based Petri net

Input: List of places, transitions and arcs (may be located in files). List of *priorities* associated to transitions. Value *steps* which describes how many steps of simulations should be executed.

Output: The marking *M* after *'steps'* steps of simulation.

Steps:

```
1. begin
2. for i := 0 to steps do
3.    begin
4.    enabled := findEnabledTransitions();
5.    fireNr := enabled.getAny();
6.    ConfSet := findConflicts(fireNr);
7.    firePrio := X.getNumber();
8.    fireNr := returnTransitonWithPriority(firePrio, ConfSet);
9.    fire(fireNr);
10.   end
11. writeMarking();
12. end
```

The beginning of the algorithm is the same as in **Algorithm 20**, till line 7. Here the value returned by the random variable X is obtained and the transition whose priority matches this value is found (line 8) according to **Def. 32**. In the simplest approach, the method *returnTransitonWithPriority()* can be exact implementation of this definition, but some modifications to avoid dividing may be added as well. The transition whose priority was identified by a value returned by X is fired in every step (line 9). At the end the marking is written.

The complexity of **Algorithm 21** is the same as in **Algorithm 20**. Operations in the new method *returnTransitonWithPriority()* will be executed at most NCT times, where NCT is the largest size of a conflict-set, the NCT number is same as in *findMax()* from the previous algorithm, so the complexity of **Algorithm 21** will not be changed with respect to **Algorithm 20**.

5.3.3 Generation of PDB file

The Protein Data Bank (PDB) file format [149], often used in this thesis, is a text file format developed by the worldwide famous data base storing structural data on proteins and nucleic acids [110]. A file in this format contains the information about coordinates of atoms, bonds, secondary structure, annotations, details about the data collection and structure solution, and bibliographic citations. However, the most important is a part related to the description of the structure, where all atoms are listed with given number, chain, coordinates in $\mathbb{R}^3$ space and other data. The file in the PDB format may contain just one structure or a list of numerous molecular structures (for example a trajectory). The PDB data format is a type of the standard, every software designed to work with structures of proteins can read/write such files. One should note that, from time to time, this PDB standard is updated.

For biological community it is desirable to have data presented in the PDB format. Therefore, the last step during Petri nets analysis of a MD trajectory should be a generation of a PDB file. In this easily readable file molecules' structures after the MD Petri net simulation are stored. It can be done for the OPOA and OPOC

algorithms because those algorithms operate on coordinates. For the CON algorithm, due to its very nature, such a format is not available. Having PN results in PDB format, one may compare a MD trajectory and structures with the corresponding results obtained from the MD Petri nets simulations. This is of great importance for practical applications of the methodology developed here.

The PDB file generation is straightforward: we need to find and translate the information about coordinates associated with marked places into a structure file. This task is trivial in the OPOC algorithm, since here every place has the data on the coordinates of every CA atom. The generation of the PDB file from a PN created by the OPOAv1 algorithm is a little bit more involving. In this case, an additional operation has to be performed: we need to find the localizations of every CA atom which is described by a marked place. For every localized atom a new line can be added to a PDB file, which will contain the number of that CA atom and its coordinates linked to the place. In the OPOAv1 algorithm the information about the CA atoms are indeed linked in some way to places, so this operation should be quite easy.

The situation is different for the OPOAv2 algorithm. A unique feature of this algorithm is that one place represents only one point in the C_k^3 space without association to any specific CA atom. So, we only have a marking describing which places are marked, but we cannot determine which particular CA atom is in concerned place. This problem has to be somehow solved during the OPOAv2 simulation. An additional data structure, for example a list, should be added and this structure might store information where in PN all CA atoms are located in the current step. An example of such a modification is presented below for a priority-based PN and analogous changes can be added to other simulation algorithms presented so far. Once this information is available, the generation process of the PDB file is simple. It is only one loop over the marking adding data represented by every marked place to one row in the PDB file. This file will store in this common format the results of MD simulation of PN and it may be used for further studies.

Algorithm 22. The simulation algorithm of priority-based Petri net with CA atom localization memory.

Input: List of places, transitions and arcs (may be located in files). List of *priorities* associated to transitions. Value *steps* which describes how many steps of simulations should be executed.

Output: The marking *M* after *'steps'* steps of simulation and *localization* list with information in which place every CA atom is located.

Steps:

```
1. begin
2. localization := NULL;
3. for i = 0 to steps do
```

```
4.    begin
5.    enabled := findEnabledTransitions();
6.    fireNr := enabled.getAny();
7.    ConfSet := findConflicts(fireNr);
8.    maxNr := findMax(ConfSet, priorities);
9.    if(fireNr <> maxNr) then
10.         fireNr := maxNr;
11.   fire(fireNr);
12.   atom := getAtom(fireNr);
13.   place := getOutputPlace(fireNr);
14.   localization[atom] := place;
15.   end
16. writeMarking();
17. end
```

The algorithm is very close to **Algorithm 20**, the new lines are marked in blue. The method *getAtom*() is the same as in **Algorithm 8** - it returns the number of CA atom of ATS to which a transition *fireNr* belongs. The method *getOutputPlace*() returns the number of a place to which the transition *fireNr* moves a token. Both information should be obtained from the description of the transition in a constant time, so the complexity of this algorithm will be the same as that of the **Algorithm 20**. When the atom number and the output place are obtained, they are stored in *localization* list (line 14). Since during every firing the data in the *localization* list are updated, they are up-to-date and when the simulation is done this list contains the information necessary for the PDB file construction.

There are also minor technical issues. In the procedure described above one obtains: the number of the CA atom, the type of this atom (of course it is CA) and its coordinates in the space. Since coarse-grain representation of amino acids is used, protein residues will be represented by one CA atom only. Thus the number of the amino acid considered will be the same as the number of its CA atom. However, in the classical PDB file also other data are usually necessary, for example, the type of the amino acid or the chain to which an amino acid belongs. The chain of every atom may be set to identical arbitrary value or imported from the MD trajectory file. Other information required by the PDB format can be set freely by a user, such flexibility will not affect significance of MD PN simulations. This is in particularly true for lines "CONNECT", where the original molecular topology may be added by a user or not. In the test cases consecutive CA atoms were connected in the created PDB files.

5.3.4 Examples

I have performed numerous PN simulations (>100). Some of them are presented here to show utility of those methods. The main feature of the generated MD Petri nets is that they may gather data from many MD simulations, and the generated PN trajectories reflect the properties from several different MD simulations combined into one net.

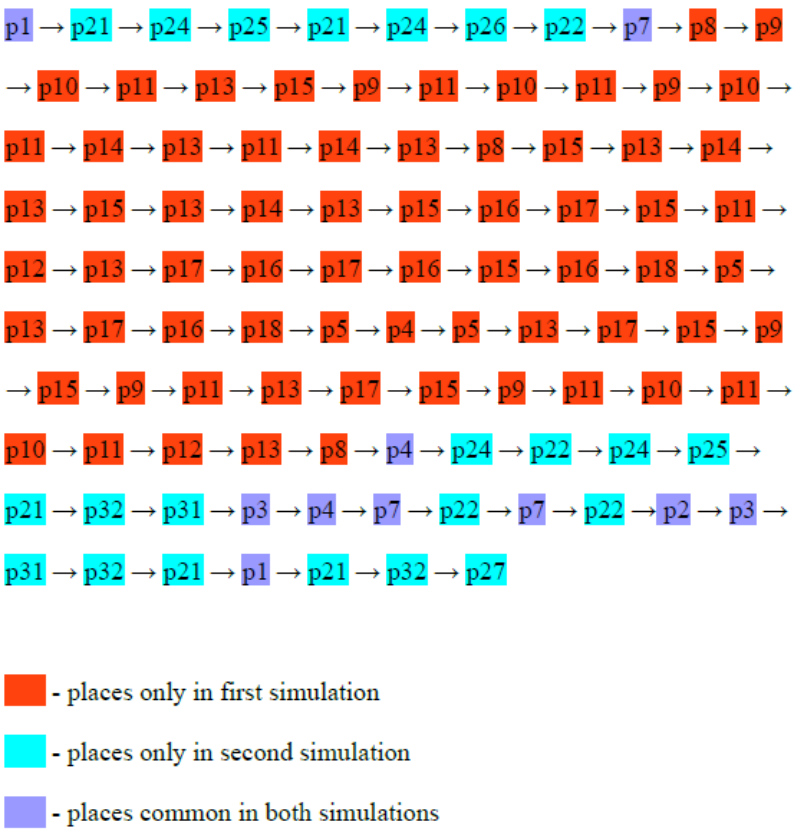


Fig. 37. Places present in a trajectory obtained from the simulation of a random-priority Petri net generated from two MD simulations of the TTR mutant V30M. The places common for both simulations are highlighted in blue, the places present in the first simulation only are in red, and those present in the second one are in cyan.

In **Fig. 37** a sequence of places from a simulation of MD PN is presented. The Petri net was generated from two MD simulations of TTR mutant V30M using the OPOC algorithm with the threshold of 1.4 Å. The PN was treated as a random-priority Petri net. The places common for both MD simulations or observed only in one of them are denoted by colors (red, cyan, blue). One can see that in the PN trajectory the molecule changes its conformations between those observed only in the first or only in the second MD simulation. Clearly, such a trajectory was not directly obtained from the MD calculations. However, changes between the conformations revealed in the PN simulation are highly probable because always when the molecule transforms from a conformation encountered in one MD simulation to a conformation characteristic in another one, it goes via a conformation common for both MD simulations. In the case studied those changes between conformations from different MD simulations were observed exactly twice in the PN simulation.

When the MD Petri net is generated from a bigger set of MD trajectories, during the simulation changes between conformations derived from distinct MD simulations may be observed even more often than in the previous example. Such a situation is presented in **Fig. 38**. Here the MD Petri net has been generated by OPOC from 20 MD simulations of MCP-1, all were 2 ns long. The threshold was 1.5 Å. During this simulation the PN was treated as a regular, marked Petri net. The generated PN trajectory is a mix of conformations from different MD simulations. However, as I mentioned above, such conformational transformations are possible via conformations common in a few MD simulations. Those changes were indeed observed in the performed MD simulations.

The last example is a sequence of places generated from a simulation of MD PN created by the OPOC algorithm. It refers to SMD simulations. Fourteen SMD simulations of the forced dissociation of MCP-1 from its antibody, all 2 ns long, were used to generate this comprehensive MD Petri net. The threshold was 1.3 Å. The Petri net was simulated as a timed Petri net. The PN trajectory converted to a PDB file (see next section) looks very similar to individual trajectories generated by the SMD simulations. In **Fig. 39** the obtained sequence of places is shown. It is a fragment of a longer simulation. In the last part of the Petri net simulation (not shown) the molecule switched between two places, corresponding to conformations of the MCP-1 being fully separated from the antibody. Despite the fact that in SMD simulations conformations typically are not very similar, the token describing a current conformation has appeared in common places in many simulations.

Simulations of PN generated by OPOAv1 and OPOAv2 were performed as well. The obtained PN trajectories were, similarly as for the OPOC algorithm case, combinations of the trajectories collected from MD simulations. Due to a huge number of places, such PN simulations have to be much longer than those performed in OPOC, and their graphical presentation as a sequence of places/transitions is not possible. However, they were analyzed as PBD files generated using the method described in the paragraph 5.3.3. Several interesting examples are shown below.

Place	Simulation	Place	Simulation	Place	Simulation
p157	s16, s17, s18	p34	s20, s18, s19	p90	s13
↓		↓		↓	
p114	s14, s15, s16, s17	p231	s19	p91	s13
↓		↓		↓	
p156	s16	p117	s14, s18, s19	p93	s13
↓		↓		↓	
p159	s16	p231	s19	p94	s13
↓		↓		↓	
p156	s16	p117	s14, s18, s19	p95	s13
↓		↓		↓	
p159	s16	p231	s19	p99	s13
↓		↓		↓	
p156	s16	p117	s14, s18, s19	p99	s13
↓		↓		↓	
p159	s16	p34	s20, s18, s19	p99	s13
↓		↓			
p156	s16	p35	s20		
↓		↓			
p157	s16, s17, s18	p34	s20, s18, s19		
↓		↓			
p88	s13, s14, s15, s18	p36	s20		
↓		↓			
p140	s15, s18	p34	s20, s18, s19		
↓		↓			
p143	s15	p36	s20		
↓		↓			
p142	s15	p34	s20, s18, s19		
↓		↓			
p143	s15	p36	s20		
↓		↓			
p140	s15, s18	p34	s20, s18, s19		
↓		↓			
p197	s18, s19	p231	s19		
↓		↓			
p198	s18	p117	s14, s18, s19		
↓		↓			
p140	s15, s18	p34	s20, s18, s19		
↓		↓			
p34	s20, s18, s19	p199	s18, s19		
↓		↓			
p37	s20	p22	s20, s2, s4, s8, s18, s9		
↓		↓			
p34	s20, s18, s19	p200	s18		
↓		↓			
p36	s20	p144	s14, s15, s16, s17		
↓		↓			
p34	s20, s18, s19	p88	s13, s14, s15, s18		
↓		↓			
p36	s20	p89	s13		
↓		↓			

Fig. 38. Sequence of places observed in a simulation of PN generated from 20 MD simulations of MCP-1. The beginning of the sequence is in the first column, then it is continued in the second column and the end of the sequence is in the third column. Next to the places are the denotations of the MD simulation files, in which a conformation described by the place was observed. For example, the place p200 was observed only in the MD simulation s18, whereas the place p34 in simulations s20, s18 and s19.

p1 s11, s1, s13, s3, s14, s4, s15, s5, s16, s6,s17, s7, s18, s8
↓
p2 s11, s1, s13, s3, s14, s4, s15, s5, s16, s6,s17, s7, s18, s8
↓
p3 s11, s1, s13, s14, s4, s5, s7, s18
↓
p76 s15, s5
↓
p77 s15, s5, s17
↓
p78 s15, s5
↓
p79 s15, s5
↓
p91 s5
↓
p80 s15, s5
↓
p91 s5
↓
p80 s15, s5
↓
p91 s5
↓
p80 s15, s5
↓
p91 s5

Fig. 39. The sequence of places obtained from the simulation of MD PN generated from the 14 SMD trajectories for MCP-1 chemokine.

For two MD simulations of MCP-1, both 2 ns long, an MD Petri net was generated by the OPOAv1 algorithm. The size of the grid used in discretization was 1. Then, one thousands steps simulation was performed and from the obtained markings the PDB file has been generated. The file contains one thousand frames, which describe movements of the protein in the space. During the simulation transitions from different MD trajectory files alternately occurred. For example, the first transitions to be fired in the PN simulations were t14440 and t14431 from the 2nd MD simulation, the next transition was t46 which occurred in both simulations, after that t2957 from the first MD simulation, and t14610 from the second simulation fired. This sequence of events led to a new PN trajectory, which obviously is a combination of parts based on the underlying MD trajectories. A selected screen shot from the PN trajectory transformed into the PDB file compared with a structure from the MD trajectory is presented in **Fig. 40**.

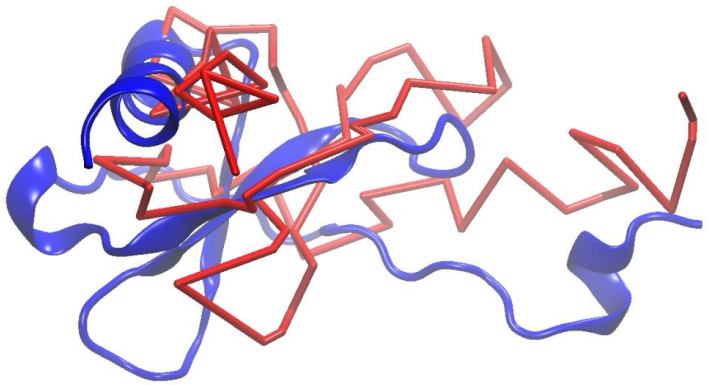


Fig. 40. An example of a structure from the PN (red) and MD (blue) simulations. The PN structure was generated using OPOAv1 algorithm with size of the grid points of 1 Å so the structure has coarse grained representation.

Simulations of MD PNs generated from SMD simulations, using OPOAv1 and OPOAv2, were also performed. Different types of Petri nets were checked, mostly timed PN, guard time PN and classical, marked PN. Contrary to the expectations, timed Petri nets are not suitable for the SMD simulations. The generated PN trajectories show that in the PN simulation the protein stays all the time in a few starting conformations. The performed analyses of this situation show that this is caused by some type of transitions starvation. Some of the transitions, which occur at the beginnings of the SMD simulations, have very small firing times, and they fire alternately. If they create a cycle, other transitions will not have any opportunity to fire. Such a situation can be observed in MD PN generated for a SMD unfolding simulation of MPC-1 using OPOAv1 with the size of the points grid equal to 2. The simulation of the PN has two thousands of steps. In this simulation, for example, only transitions t195, t85 and t167 from ATS of amino acid 51 were firing. The values of the time function of those transitions are smaller than 5. The same was for other amino acids and other transitions. In the guard time PNs such transitions starvation is not present. The performed tests have shown that the guard time Petri nets are the most sufficient for simulations based on SMD trajectories. Anomalies are smaller than in classical, marked PN, and simulations progress normally, in contrast to timed PNs.

The performed tests did not show glaring anomalies in PN simulations of classical Petri nets dynamics if these PNs were generated from the SMD simulations. The tests were performed for dissociation of a small protein MCP-1 from its antibody, as well as for artificial unfolding of this chemokine. I have compared the trajectories from all-atom SMD (“classical” SMD simulations, NAMD code) and novel, PN generated trajectories (OPOA algorithms, new codes). The general scenarios of both phenomena were the same in both methods of simulations. Some irregularities were observed in the PN based trajectory, especially in the initial phase of the unfolding, but the sequence of structural event was conserved.

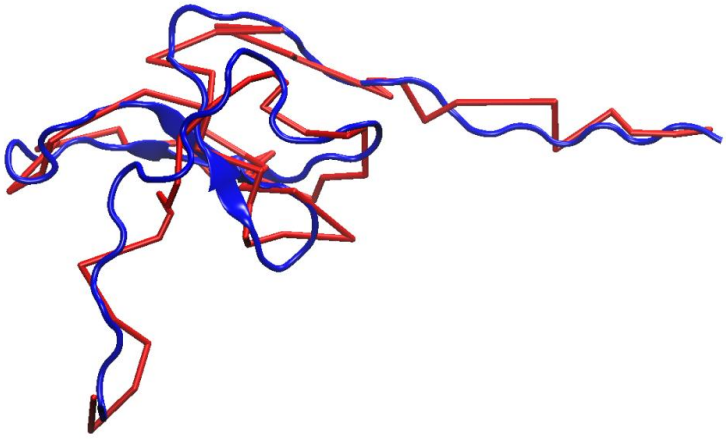


Fig. 41. A comparison of structures from the all-atom SMD (blue) and PN simulations (red). Petri net was obtained by OPOAv1 and the simulation was performed within a classical PN formalism.

The quality of structures from PN based SMD simulations may be estimated basing on **Fig. 41**. This, of course, does not mean that all-atom and PN based SMD will always give such a good agreement, but the examples illustrate the potential of the PN approach developed here. In our opinion, for more complex biological systems, timed PN might be the only option for assuring proper unfolding/unbinding SMD scenarios. Thus, OPOA studies using the classical PN of SMD induced processes should be performed carefully.

It is worth noting in this concluding section that PN based simulations are much faster than the usual MD modeling of molecules' dynamics. Generation of long MD trajectories (>100 ns) for large systems (>200 000) may take CPU weeks even on 100 core clusters. Here the generation of any PN is much faster, performed only once, and the simulations of PN are at least an order of magnitude faster (a few hours) than standard MD simulations. Interestingly, the standard MD data were collected from large, multi-core clusters, but PN results came from a single 4-core notebook. Since there is no one-to one correspondence between physical times of standard MD simulations and time parameters used in the PN formalism, I did not try to make any detailed comparisons of the performance of these two methodologies.

5.4 Chapter 5 summary and conclusions

In this chapter a novel application of Petri nets to representation and analysis of biological structural data have been presented. In the literature such ideas were absent so far. I have tried to solve quite a challenging task – how to provide a new, net based tool for a field of computer simulations of biomolecular dynamics. The main objects considered in this chapter were protein structures represented by sets of CA atoms distributed in $\mathbb{R}^3$ Cartesian space. The time evolution of these

structures explains many biological phenomena and is intensively studied using advanced computer algorithms and physical methods [108]. The simulations are time consuming and any new data analysis method is always welcome.

I have proposed new algorithms for generation of PN inspired by MD trajectories (OPOA OPOC and CON, paragraph 5.2). These nets code the properties of conformational spaces. The analysis of the shapes provides a new tool of scientific scrutiny. Once PN nets are generated, the dynamics of such nets may be studied. New algorithms which help doing such dynamics are presented. The method to map a PN marking into PDB files was developed. The time complexity of algorithms is analyzed for each method. There is no single formula for all the algorithms, but the OPOC algorithm is the most effective. I conclude that PN coded information is readable, useful and in all cases studied (MCP-1, TTR) PNs reflect the main features of conformational transition observed in classical computer models. Hopefully, MD PN based models correspond also to real biological phenomena, but this is a problem of fidelity of a physical model ☺ and not of computer science.

The algorithms are effective and the codes are quite fast. I estimate that the MD based PN simulations are an order of magnitude faster than the classical approach. Of course, the data from PN are approximate and not so exact, but another advantage is that in the same network numerous MD trajectories may be mapped at the same time. I plan that my codes will be in the public domain and will be provided to scientific community.

Conclusions

In modern science computers and algorithms play a fundamental role. There is a constant demand for new, original and better methods of modeling of real phenomena and extraction of information. The computer science may contribute a lot to this end.

The main goal of the present thesis was the development of new application areas of powerful, graph-based techniques in biological and biophysical studies. Petri nets have been chosen as the main formalism since they have been already successfully used in numerous biological disciplines [4, 8-9, 39].

Presented studies gave new results on modeling of the human immune system and provided new research tools suitable for Petri net graph-based analysis of massive molecular dynamics data sets. New algorithms have been introduced. No such results have been previously reported in the literature.

In the first chapter a general overview of the Petri nets and their features were critically presented. Selected extended types of PNs were introduced, inter alia the priority-based PN. I have proposed several modifications of these PNs. The standard features of those known networks were not sufficient for an effective MD data analysis. Therefore, a new type of the PN was introduced: a random-priority based PN. In this new sub-type of the network a probability of a transition firing is proportional to the priority of the given transition. I have found that this property is very suitable for the simulation of the PN generated from the MD data. The conclusion is that the simulations performed with a random priority-based PN are more similar to the real MD simulations than any other type of PN investigated here.

In the second chapter of the thesis the novel PN model of the immune system was described. The new model presents a part of the immune response: the adaptive immune system and the role of the macrophages. The following features were added to the model and tested: an impact of fever, an effect of ageing, changes in IS caused by AIDS, AOIS and ASD diseases. Adding such pathological conditions was not a trivial task. Those elements were introduced to the model in different, carefully scrutinized ways, depending on the required type of the IS model modification. The impact of all pathology induced modifications was tested during the simulations of the Petri nets and compared to the results calculated for a regular, unperturbed model. Moreover, a t-invariant analysis of the model was performed. This method allows for the verification of PN models. For the IS model all t-invariants were found and their biological meanings were recognized. Also t-clusters were calculated and analyzed, as well as MCT-sets. The model passed the verification. This part of the study shows a great potential of PN IS model. My efforts and tests have shown that modeling diverse biological process using this tool is quite possible.

These days the GPU technology is very promising, giving a chance for a good performance/cost ratio in scientific modeling. In the third chapter the GPU simulation algorithm of the PN was presented. The new algorithm allows simulations of Petri nets using the CUDA technology. The simulation process was important, for example, in studying the IS model. The PINGU algorithm consists of two parts: the first is preprocessing, the second is the proper simulation. Both were described in details. The tests of the performance were conducted using different graphical cards and different Petri nets sizes. The presented results show that for large networks the CUDA simulation is better and faster than a CPU based one. Such big Petri nets can be generated, for example, by my OPOA algorithm described in the fifth chapter. The IS model is too small to observe any GPU acceleration.

In the fourth chapter the MD and SMD study of the antigen-antibody protein complexes were presented. Two complexes were studied: pollen from timothy grass Phl p2 and its antibody, and chemokine MCP-1 with its antibody. Those studies gave interesting results themselves, published in paper [12], however they were performed mainly to generate test data sets for analysis done in the last chapter. The first complex is connected with allergies, in the second one the MCP-1 chemokine is a protein which perhaps plays a role in autism spectrum disorder. The SMD method was used to enforce a dissociation of the antigens from the antibodies. About 70 SMD simulations were performed and different directions of the unbinding force vector were tested. We have found that the forced dissociation of the complex in the lateral direction (approximately perpendicular to the main axis of the antibody) requires forces being about 30% lower than that in the vertical direction. These results support feasibility of new, faster, AFM based medical nanodiagnostic procedures.

In the last chapter Petri nets based algorithms were applied to analyze MD data. The idea of this novel approach is schematically presented in **Fig. 42**.

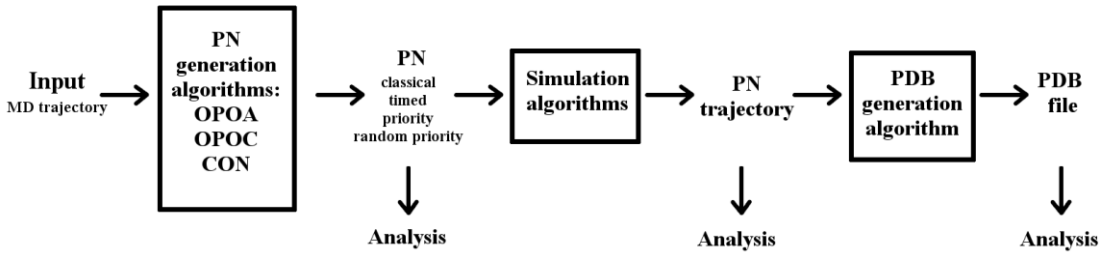


Fig. 42. The workflow scheme of the MD simulations analysis using PN formalism.

At the beginning of the new analysis process the MD input data, i.e. MD trajectory, is used to generate an appropriate PN. Three new, dedicated algorithms of PN

generation were developed and described in the Chapter 5: OPOA (one PN place corresponds to a position of one atom), OPOC (one place corresponds to one conformation of the protein) and CON (one place corresponds to a contact between two amino acids). In every algorithm various aspects of the MD simulations are highlighted. The PN generated by one of these algorithms contains more than merely information on places, transitions and arcs. Such a PN represents additional data which allows to generate extended types of the PN: timed, priority-based and random priority-based. The features of PN stemming from MD can be analyzed statically and examples of useful information extraction were described in the chapter. However, the new PN can be a basis of new types of dynamical simulations. The protocol for simulation of the extended types of the PN was designed and presented as the pseudo-code. From this protocol one obtains a PN trajectory. A PN trajectory is a new and compact representation of selected MD data, suggested for the first time in this chapter. It is usually simpler and can be studied in numerous ways. Moreover, this PN trajectory may be used to generate a PDB format file. In some cases this file is easier to analyze than the original massive MD data. Examples of PN-generated trajectories and PDB files were presented in this chapter as well. In conclusion, I have shown in the Chapter 5 a new way of representation and retrieval of important structural biological data.

Summary

The topics presented is the thesis demonstrate that PNs are useful tools in biological studies. I have tried to show that they are suitable for investigations of the immune system. Our newly created model of IS has been successfully used in analysis of numerous phenomena. Further elements can be added to the model relatively easily. A simulation of the PN executed on the GPU using the current CUDA technology is possible and it is demonstrated that for large networks PN GPU is faster than executed on the CPU. The presented SMD simulations show that the dragging of an antigen from an antibody depends on the direction of the force applied: a lateral process requires lower forces than a vertical one. This simulation result gives theoretical foundation for Lateral Force Spectroscopy measurements by the AFM method. Perhaps the most innovative aspect of this thesis is paving a way to applications of the PN formalism in MD simulations. Such attempts do not exist in the literature. Three algorithms of the PN generation from the MD trajectories were designed. Using created PN changes of molecular conformations, changes in locations and time evolution of amino acid – amino acid contacts can be tracked. It is shown that the PN generated by the OPOC algorithm can be used for clustering of molecules conformations. An advantage of our PNs algorithms over other methods of MD data analysis is that PNs represent not only conformations (or positions) of the elements, but also the connections and relations of those objects as well.

Future prospects

The studies presented in this thesis are the first results in their areas and do not fully exhaust the topic. Many new ideas may be tested, like for example the utility of other types of Petri nets in the analysis of MD trajectories. Especially, a new type or a modification of timed PN is required to accurately present the features of SMD trajectories during the PN simulation. An idea of new PN generation algorithm from a MD trajectory has been invented. This new algorithm may be called OPOV (one place one velocity), and it will describe not the positions of the amino acids, but their velocities. However, first the results of the CON algorithm should be better investigated. The CON algorithm is the least studied. It will be useful for generating PNs using the OPOC algorithm from MD trajectories calculated in lower temperatures. The existing Petri nets generated algorithms can be also enriched. I would like to generate some "macro" PN, whose every place would represent a smaller, more detailed Petri net. New features can be added and studied in the PN model of the IS, like for example, a better representation of memory cells.

I hope that my computational studies will help scientists to make new exciting discoveries in biology and medicine. Networks and graphs based techniques are powerful tools in science and hopefully my efforts contributed to better understanding these branches of the computer science.

Supplementary materials

Appendix A – CUDA architecture

CUDA is a parallel computing platform and programming model created by NVIDIA and implemented by the graphics processing units (GPUs) [97, 150]. It enabled programmers to use GPU computational capacity, which is considerable, for example nVidia GeForce GTX 580 has 512 cores [151], a standard PC computer has 4 or 8 cores. However, the GPU and CPU cores are not the same. Cores on graphical cards are oriented on data processing, not flow controlling or caching. The NVIDIA GPU architecture is built around a scalable array of multithreaded Streaming Multiprocessors (SMs) [97]. The SM creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps. When a multiprocessor is given some threads to execute, it partitions them into warps and execute in sequence.

CUDA comes with a software environment that allows developers to use C as a high-level programming language [97]. The program code, which will be executed on NVIDIA GPU contains of successive blocks: code executed on CPU and code executed on GPU. The parts of code which will be executed on a graphical card are called kernels **Fig. 43**.

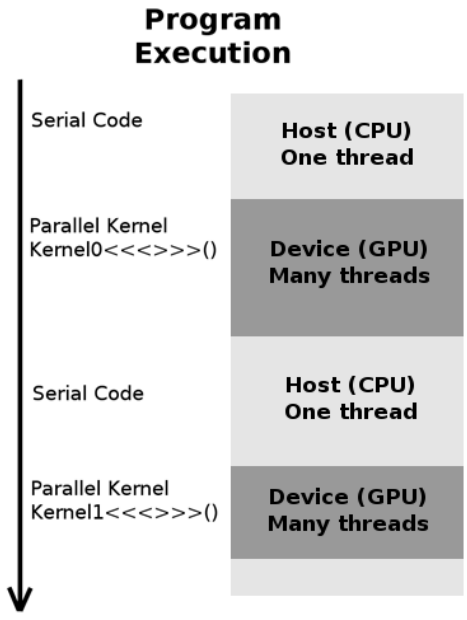


Fig. 43. The structure of the CUDA program execution. On the left blocks of code are presented, successive serial code and parallel code. On the right the device, on which the code is executed is shown.

In the CUDA architecture the most important is the logical structure of processes and access to the memory - both are connected. The basic processes are threads - the smallest logical individual. They execute code instructions. Each thread has its

own part of memory – thread's registers, but the amount of this memory is small, measured in kB (it depends on the model of the graphical card and the number of threads). However, the access to the registers is very fast. Each thread has its individual number (unique in the block). Threads are organized into blocks as one, two or three dimensional structures, so they have the numbers, which may contain one, two, or three coordinates. All blocks execute the same code and every block is scheduled on one SM. All threads from one block have the access to the same part of the memory, which is called the shared memory. The shared memory is assigned to the block. The access to this memory is fast, but the size of the memory is small - measured in kB (similarly to registers, the amount of the shared memory depends on the model of the card and the number of blocks per one SM). Blocks are grouped into a grid and can be organized into one, two or three dimensional structures, therefore blocks have numbers, which may contain one, two, or three coordinates. The grid is created when the kernel is executed, usually one grid is performed at once on the graphic card. All threads from one grid have the access to the global memory, so the data from this memory is shared for the grid. The amount of the global memory is usually sufficient (for nVidia GeForce GTX 580 it is 1536 MB [151]), however the access is slow, roughly 100 times slower than the access to the shared memory and registers [97, 150]. If it is necessary to use the global memory, it is good to read entire blocks of data – succeeding threads should read succeeding pieces of data, because this operation is simultaneous and the loss of the time for the access to the memory is reduced. The logical organization of the processes and memory are presented in **Fig. 44**.

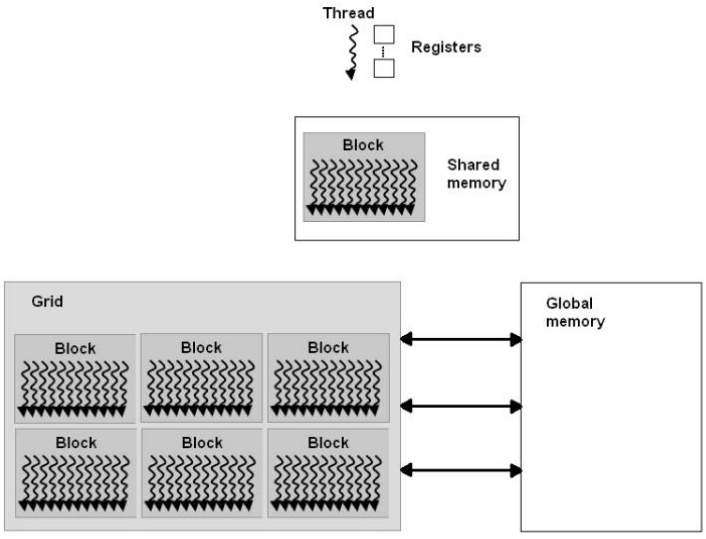


Fig. 44. The logical organization of the processes and the memory in the CUDA framework.

In the CUDA programming not only the algorithm alone is very important, but also the adjustment of the number of threads and blocks to the input and output data of the algorithm, and appropriate distribution of the data to the memory. Of course,

the best situation is when all the data are in registers or shared memory, but often it is not possible. When the global memory is necessary, the succeeding reading should be used. The best performance is obtained when all SMs are busy, so every number describing the amount of thread, blocks etc., whenever it is possible, should be always rounded up to the power of two. It should be assumed even if it is not mentioned in the description of the algorithm.

Appendix B – My implementation of MD Petri nets algorithms

All algorithms presented above were implemented by me in my software Petri Efficient Analysis (PEAN). It is available at http://www-users.mat.umk.pl/~leii/thesis/. The Java [152] language was used. PEAN consists of three tabs - each corresponds to following steps: generation of MD Petri net, simulation of PN and generation of PDB file based on PN and marking after the simulation. The first tab in presented in **Fig. 45**.

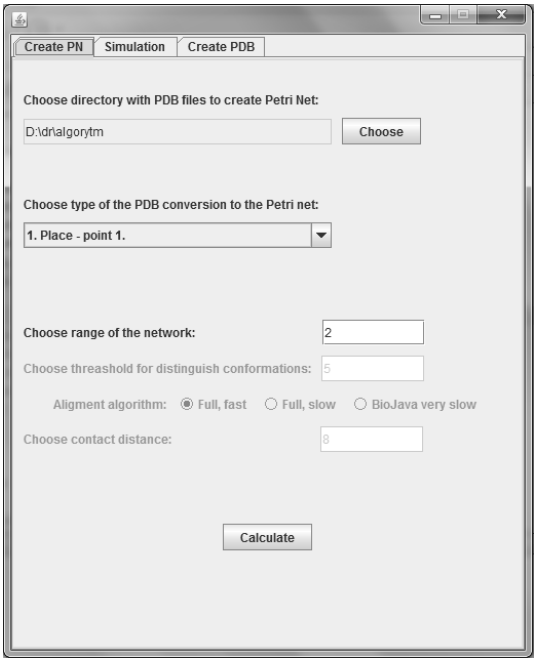


Fig. 45. The first tab of MD Petri nets algorithms implementation PEAN program.

By using "Choose" button a user can choose a directory with MD trajectory files converted to PDB type files. Below the type of the generation algorithm (OPOAv1, OPOAv2, OPOC, CON) can be chosen. Then, in three text fields, the user can change parameters of the selected algorithm. When the algorithm is chosen the corresponding text filed becomes enabled. Additionally for the OPOC algorithm one can chose type of the structural alignment. **Algorithm 10** is described as "Full, fast", **Algorithm 11** corresponds to "Full, slow", "BioJava very slow" is CA algorithm. After the "Calculate" button is pressed, the algorithm is launched and MD Petri net's files are created and written to a new directory created in the folder with PDB trajectory files.

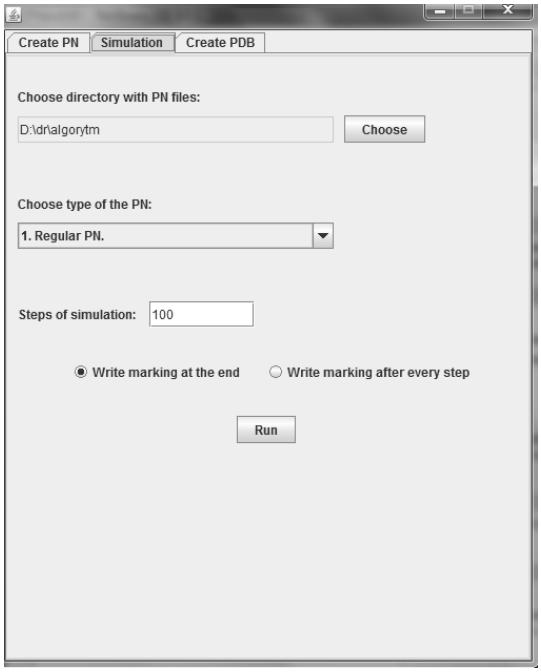


Fig. 46. The second tab of MD Petri nets algorithms implementation PEAN.

The second tab of the PEAN program is presented in **Fig. 46** and it is as intuitive as the first tab. Starting from the top by using "Choose" button, the user can point a directory containing MD Petri net's files (created during the previous step). Because all necessary information is written to those files, the user does not need to choose a type of an algorithm used to generate the PN. Also data required for extended types of Petri nets like, for example, times are kept in those files, so the directory contains all information required to simulate every type of the network. Next, the user has to choose which type of the PN they want to simulate - marked, timed, priority-based, random priority-based or guard time. The text field contains a number of simulation steps. At the end, the user can chose if they want to write the obtained marking at the end of the simulation (only one marking will be saved in the file) or after every step of the simulation (the number of markings will be equal to number of simulation steps). After pressing the "Run" button, the simulation starts and when it is completed the obtained marking is written to the PN directory.

The last tab is presented in **Fig. 47**. Initially, the user can chose if the marking of the network is inside the directory with the PN files or indicate (fill in) the numbers of those places which should have a token. If the second option is chosen, only a place having the indicated number will be marked - it will have one token, and other places will be empty. This option is useful for the PN generated by the OPOC algorithm. Next, the tab contains a button for selection of a directory with PN files, and a text field which contains a path of the selected directory. Only MD Petri nets generated by algorithms OPOA (both variants) and OPOC can be used because the networks created by the CON algorithm do not contain information about coordinates. If the user does not provide the number of marked place, then the

directory has to include a marking file, generated during the simulation or created in other way. After pressing the "Create" button, the PDB file is created and written to the directory with PN files. In my implementation the PDB file name is "generated.pdb". If the user has a marking file created with previous step with option "Write marking after every step", for every marking stored in the file one protein structure will be generated and saved to a PDB file as a frame so the created file will be recording of MD PN simulation.

Fig. 47. The third tab of MD Petri nets algorithms implementation PEAN program.

Appendix C – List of publications and conferences

Publications:

a) part A of the list published by Ministry of Science and Higher Education:

- Gogolinska, A. and W. Nowak, "Petri Nets Formalism Facilitates Analysis of Complex Biomolecular Structural Data" RAIRO-OPERATIONS RESEARCH, 2015. **under review**
- Gogolinska, A., Nowak, W., "Molecular basis of lateral force spectroscopy nano-diagnostics: computational unbinding of autism related chemokine MCP-1 from IgG antibody", Journal of Molecular Modeling (Springer), 19(11):4773-80 (2013)

b) other reviewed publications:

- Jakubowski, R., Gogolinska, A., Peplowski, L., Skrzyniarz, P., Nowak, W., „*Computational studies of TTR related amyloidosis: exploration of conformational space through Petri net-based algorithm*", TASK Quarterly. 10/2014; 18(3):267.
- Gogolinska, A., Nowak, W. „*Petri Nets Approach to Modeling of Immune System and Autism"*, Artificial Immune Systems, Lecture Notes in Computer Science. In: Coello Coello, C., Greensmith, J., Krasnogor, N., Liò, P., Nicosia, G., Pavone, M. (eds.), vol. 7597, pp. 86-99. Springer Berlin / Heidelberg (2012)
- Gogolińska A., Ochmański E., Nowak W., „*Petri Nets in Immunological System Modeling*". In: Ochmański, O., Penczek, W. (eds.) Matematyczne metody modelowania i analizy systemów współbieżnych MASYW 2010, pp. 35-46 (2012).

c) published and indexed (ISI) conference abstracts:

- Mikulska, K., Jakubowski, R., Peplowski, L., Dabrowski, M., Gogolinska, A., Duch, W., Nowak, W., „On applications of virtual atomic force micro-scope in studies of brain proteins", European Biophysics Journal, Volume 42, Issue 1 Supplement (2013)
- Gogolińska, A., Nowak, W. „Molecular recognition at atomic level: Interaction of autism related protein MCP-1 with an antibody", European Biophysics Journal, Volume 40, Supplement 1, p. S106 (2011)

Conferences (22):

- IV 2015 RECOMB 2015, the 19th Annual International Conference on Research in Computational Molecular Biology, 12-15 April 2015, **Poster**: „Petri Nets as a Novel Representation of Biomolecular Simulations Data", A. Gogolińska, W. Nowak

  BioInformatics in Torun – BIT15, 16-18.06.2015, Toruń. **Poster**: "On the possibility of using GPU in Petri nets simulations.", A. Gogolińska, W. Nowak

- VIII 2014 The 2014 International Conference on Brain Informatics and Health 11–14 August 2014, Warsaw, Poland, **Talk**: „Panomics contributions to understanding Autism Spectrum Disorders", A. Gogolińska, W. Nowak
- VI 2014 BioInformatics in Torun – BIT14, 12-14.06.2014, Toruń. **Poster**: "HOPE = HOw to keeP Eye on Autism – a new bioinformatics tool for data mining.", A. Gogolińska, W. Nowak

IV EURO WG Conference on Operational Research in Computational Biology, Bioinformatics and Medicine, Poznań - Biedrusko (Poland) 26-28 June, 2014, **Talk**: „Petri Nets Formalism Facilitates Analysis of Complex Structural Data", A. Gogolińska, W. Nowak, (travel grant)

- II 2014 The Immunology of Ageing, Monday, 24 February 2014, Londyn, Wielka Brytania, **Poster**: „Petri Nets Computer Model of Immune System Opens Opportunity to Study Effects of Ageing, Autisms and Fever", A. Gogolińska, W. Nowak, (travel grant)
- IX 2013 11th Workshop on Bioinformatics and 6th Symposium of the Polish Bioinformatics Society, Wrocław, Polska, 27 - 29 September 2013, **Talk**: „Mathematical networks as a tool in MD computer simulations data analysis", A. Gogolińska, W. Nowak
- VI 2013 Bioinformatics 2013 and BioInformatics in Torun – BIT13, 26-29.06.2013, Toruń. **Poster**: "New ideas on using Petri nets in molecular dynamics simulations", A. Gogolińska, W. Nowak

VII Kopernikańskie Seminarium Doktoranckie, 19-21 czerwca 2013, Toruń. **Talk**: „Jak rozpoznawanie molekularne wspomagana diagnostykę medyczną? - aspekty komputerowe", A. Gogolińska, W. Nowak

- IV 2013 XXVII Forum Informatyki Teoretycznej, Toruń, Polska, **Talk**: „Metody analizy modeli biologicznych stworzonych przy użyciu sieci Petriego", A. Gogolińska, W. Nowak
- IX 2012 BioInformatics in Torun – BIT12, 27-29 września 2012, Toruń, **Poster**: „Analysis of t-invariants in Immune System Petri Net Model", A. Gogolińska, W. Nowak
- VI 2012 VI Kopernikańskie Seminarium Doktoranckie, Toruń, Polska, **Talk**: „O możliwościach wykorzystania kart graficznych w algorytmach sieciowych", A. Gogolińska, W. Nowak
- III 2012 26th Molecular Modelling Workshop, Erlangen, Niemcy, **Talk**: „Interactions of antibodies with selecting antigens – computer MD modeling", A. Gogolińska, W. Nowak, (travel grant, IIIrd award for the best presentation)
- XI 2011 Multi-Pole Approach to Structural Biology, Warszawa, Polska, **Poster**: „Direction specific molecular recognition: A computational study of autism related protein MCP-1 interactions with an antibody", A. Gogolińska, W. Nowak
- X 2011 9th Workshop on Bioinformatics and 4th Convention of the Polish Bioinformatics Society, Kraków, Polska, **Talk**: „Monocyte chemoattractant proteins - bioinformatical and SMD studies of

molecular recognition in the immune system", A. Gogolińska, W. Nowak

- VIII 2011 8th EBSA European Biophysics Congress, Budapeszt, Węgry, **Poster**: „Molecular recognition at atomic level: interaction of autism related protein MCP-1 with an antibody", A. Gogolińska, W. Nowak, (travel grant)
- VI 2011 V Kopernikańskie Seminarium Doktoranckie, Toruń, Polska, **Talk:** „Genetyczne podstawy autyzmu i możliwości modelowania procesów chorobowych", A. Gogolińska, K. Mikulska W. Nowak

  BioInformatics in Torun - BIT11, Toruń, Polska, **Poster**: "Petri nets in biotechnology, medicine and bioinformatics", A. Gogolińska, W. Nowak
- X 2010 IIIrd Convention of the Polish Bioinformatics Society in conjunction with 8th Workshop on Bioinformatics, Ustroń, Polska, **Talk**: "Petri Nets in Immune System's Homeostasis Modeling", A. Gogolińska, W. Nowak
- VII 2010 Mathematical Modeling and Analysis Methods of Concurrent Systems, MASYW 2010, Tleń, Polska, **Talk**: „Petri Nets in Immunological System Modeling", A. Gogolińska, W. Nowak, E. Ochmański
- VI 2010 BioInformatics in Torun - BIT10, Toruń, Polska, **Poster**: „Molecular Recognition in Major Respiratory Allergen Phl p2 - A Computational Study", A. Gogolińska, L. Nowakowska, W. Nowak
- V 2010 Advanced Bioinformatics Tools, Warszawa, Polska, **Poster**: „Application of Petri Nets in Immune System's Modeling", A. Gogolińska, W. Nowak, E. Ochmański

Index of abbreviations

| - | - cardinality of a set

AFM – atomic force microscopy

AIDS – Acquired Immunodeficiency Syndrome

AOIS – Adult-Onset Immunodeficiency Syndrome

APC – antigen presenting cell

ASD – autism spectrum disorder

ATS – atom transition set (Chapter 5)

C – incidence matrix

C_k^n – n-dimensional Euclidean space with Cartesian coordinates, obtained by the mapping of $\mathbb{R}^n$ space, using dividing grid with resolution k.

CA – C alpha, carbon alpha, the first carbon atom that attaches to a functional group, in amino acids or proteins it is the backbone carbon before the carbonyl carbon.

CB – C beta, carbon beta, the second carbon atom that attaches to a functional group

CCA – Contact Calculating Algorithm

CPN – continuous Petri net

CPU – central processing unit

CTI – converter by t-invariants, property of a PN

CUDA – Compute Unified Device Architecture

c – constant which describes if two amino acids are in contact (paragraph 5.2.3)

DC – dendritic cell

d_{ij} – distance between *i*-th and *j*-th t-invariants

F – set of arcs

f – firing time function

GPU – graphical processing unit

HIV – human immunodeficiency virus

IL – interleukin

IS – immune system

inv – t-invariant

INF-γ – interferon γ

K_j – set of all enabled transitions in soft conflict with t_j (including t_j)

k – in Chapter 5 resolution of dividing grid in C_k^n space

L – lateral direction

L_i – conflict-free list generated for transition t_i

LPS – Lipopolysaccharides

M – marking

MCP1, MCP-1 – Monocyte Chemoattractant Protein-1

MCT – maximal common transition set

MHC – major histocompatibility complex

MTPN - Merlin Time Petri net

m – usually number of transitions

N – incidence matrix (Chapter 1)

NOC – number of occurrences of transitions

n – usually number of places

ns – nanosecond

OPOA – one place one atom algorithm

OPOC – one place one conformation algorithm

P – set of places

PDB – Protein Data Bank, also format of the files in this data base

PINGU – Petri IN Graphical Unit – algortimh of PN simulation on GPU

PN – Petri net

p – place; p_i, p*i* – *i*-th place

prio – priority function

RMSD – root-mean-square deviation

RMSF – root mean square fluctuations

RTPN – Ranchamdani's Timed Petri nets

S_j – sum of values of priority function of all transitions from the conflict set K_j

$S_{j,k}$ – sum of values of priority function of k first elements of the conflict set K_j

S-S, SS – disulfide bond

SMD – steered molecular dynamic

SPN – stochastic Petri net

T – set of transitions

Tc – cytotoxic T lymphocytes

Th – helper T lymphocytes

TTR – Transthyretin

t – transition; t_i, t*i* – *i*-th transition

W – weight function

WT – wild type, native form (without mutations) of the protein or gene

w – weight of the arc

V – clock valuation function (0), vertical direction (Chapter 4)

X – random variable

x – t-invariant

List of figures

References

1. Kursa, M.B., A. Jankowski, and W.R. Rudnicki, *Boruta–a system for feature selection.* Fundamenta Informaticae, 2010. **101**(4): p. 271-285.
2. Dojer, N., et al., *Applying dynamic Bayesian networks to perturbed gene expression data.* Bmc Bioinformatics, 2006. **7**(1): p. 249.
3. Gambin, A., S. Lasota, and M. Rutkowski, *Analyzing stationary states of gene regulatory network using Petri nets.* In silico biology, 2006. **6**(1): p. 93-109.
4. Sackmann, A., et al., *An analysis of the Petri net based model of the human body iron homeostasis process.* Computational Biology and Chemistry, 2007. **31**(1): p. 1-10.
5. Reisig, W., *Petri nets: an introduction*. 1985: Springer-Verlag New York, Inc. 161.
6. Diaz, M., *Petri nets: fundamental models, verification and applications*. 2013: John Wiley & Sons.
7. Goss, P.J. and J. Peccoud, *Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets.* Proceedings of the National Academy of Sciences, 1998. **95**(12): p. 6750-6755.
8. Chaouiya, C., *Petri net modelling of biological networks.* Briefings in Bioinformatics, 2007. **8**(4): p. 210-219.
9. Koch, I., W. Reisig, and F. Schreiber, *Modeling in systems biology the petri net approach*. 2011, London: Springer.
10. Waterman, M.S., *Introduction to computational biology: maps, sequences and genomes*. 1995: CRC Press.
11. Sarzynska, J., L. Nilsson, and T. Kulinski, *Effects of base substitutions in an RNA hairpin from molecular dynamics and free energy simulations.* Biophysical journal, 2003. **85**(6): p. 3445-3459.
12. Gogolinska, A. and W. Nowak, *Molecular basis of lateral force spectroscopy nano-diagnostics: computational unbinding of autism related chemokine MCP-1 from IgG antibody.* Journal of Molecular Modeling, 2013. **19**(11): p. 4773-4780.
13. Murata, T., *Petri nets: Properties, analysis and applications.* Proceedings of the IEEE, 1989. **77**(4): p. 541-580.
14. Petri, C.A., *Kommunikation mit automaten.* 1962.
15. May, R.M., *Simple mathematical models with very complicated dynamics.* Nature, 1976. **261**(5560): p. 459-467.
16. Sangiorgi, D., *Bisimulation for higher-order process calculi.* Information and Computation, 1996. **131**(2): p. 141-178.
17. Shmulevich, I., et al., *Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks.* Bioinformatics, 2002. **18**(2): p. 261-274.
18. Jensen, F.V., *An introduction to Bayesian networks*. Vol. 210. 1996: UCL press London.
19. Arnold, L., *Stochastic differential equations: theory and applications.* New York, 1974.
20. Wolfram, S., *Cellular automata as models of complexity.* Nature, 1984. **311**(5985): p. 419-424.
21. Reisig, W., *Understanding Petri Nets*. 2013: Springer.
22. Tuncel, G. and G.M. Bayhan, *Applications of Petri nets in production scheduling: a review.* The International Journal of Advanced Manufacturing Technology, 2007. **34**(7-8): p. 762-773.
23. Lee, D.Y. and F. DiCesare, *Scheduling flexible manufacturing systems using Petri nets and heuristic search.* Robotics and Automation, IEEE Transactions on, 1994. **10**(2): p. 123-132.
24. Viswanadham, N. and Y. Narahari, *Performance modeling of automated manufacturing systems*. 1992: Prentice Hall Englewood Cliffs, NJ.

25. Lopez-Grao, J.-P., J.-M. Colom, and F. Tricas. *The deadlock problem in the control of Flexible Manufacturing Systems: An overview of the Petri net approach*. in *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*. 2014: IEEE.
26. Di Febbraro, A., D. Giglio, and N. Sacco, *Urban traffic control structure based on hybrid Petri nets.* Intelligent Transportation Systems, IEEE Transactions on, 2004. **5**(4): p. 224-237.
27. DiCesare, F., et al., *The application of Petri nets to the modeling, analysis and control of intelligent urban traffic networks*, in *Application and Theory of Petri Nets 1994*. 1994, Springer. p. 2-15.
28. Pura, M.L. and D. Buchs. *Model checking ARAN ad hoc secure routing protocol with algebraic Petri nets*. in *Communications (COMM), 2014 10th International Conference on*. 2014: IEEE.
29. Heindl, A. and R. German, *Performance modeling of IEEE 802.11 wireless LANs with stochastic Petri nets.* Performance Evaluation, 2001. **44**(1): p. 139-164.
30. Little, T.D.C. and A. Ghafoor, *Synchronization and storage models for multimedia objects.* Selected Areas in Communications, IEEE Journal on, 1990. **8**(3): p. 413-427.
31. Bo, C., C. Junliang, and D. Min, *Petri net based formal analysis for multimedia conferencing services orchestration.* Expert Systems with Applications, 2012. **39**(1): p. 696-705.
32. Zouaghi, L., et al., *Mission-based online generation of probabilistic monitoring models for mobile robot navigation using Petri nets.* Robotics and Autonomous Systems, 2014. **62**(1): p. 61-67.
33. Gao, M., M. Zhou, and Y. Tang, *Intelligent decision making in disassembly process based on fuzzy reasoning Petri nets.* Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 2004. **34**(5): p. 2029-2034.
34. Reddy, V.N., M.L. Mavrovouniotis, and M.N. Liebman. *Petri net representations in metabolic pathways*. in *ISMB*. 1993.
35. Reddy, V.N., *Modeling biological pathways: a discrete event systems approach*. 1994.
36. Hofestädt, R., *A Petri net application to model metabolic processes.* Systems Analysis Modelling Simulation, 1994. **16**(2): p. 113-122.
37. Heiner, M., I. Koch, and J. Will, *Model validation of biological pathways using Petri nets—demonstrated for apoptosis.* Biosystems, 2004. **75**(1): p. 15-28.
38. Kielbassa, J., et al., *Modeling of the U1 snRNP assembly pathway in alternative splicing in human cells using Petri nets.* Computational Biology and Chemistry, 2009. **33**(1): p. 46-61.
39. Matsuno, H., et al. *Hybrid Petri net representation of gene regulatory network*. in *Pacific Symposium on Biocomputing*. 2000: World Scientific Press Singapore.
40. Yang, J., et al. *Modeling the genetic information transmission based on Colored Petri Nets*. in *Information and Automation (ICIA), 2014 IEEE International Conference on*. 2014: IEEE.
41. Rogers, D.J. and T.T. Tanimoto, *A computer program for classifying plants.* Science, 1960. **132**(3434): p. 1115-1118.
42. Xu, R. and D. Wunsch, *Survey of clustering algorithms.* Neural Networks, IEEE Transactions on, 2005. **16**(3): p. 645-678.
43. Gronau, I. and S. Moran, *Optimal implementations of UPGMA and other common clustering algorithms.* Information Processing Letters, 2007. **104**(6): p. 205-210.
44. David, R. and H. Alla, *Petri nets for modeling of dynamic systems: A survey.* Automatica, 1994. **30**(2): p. 175-202.
45. Natkin, S.O., *Les reseaux de Petri stochastiques et leur application a l'evaluation des systemes informatiques*. 1980: Conservatoire National des Arts et Metiers.
46. Molloy, M.K., *On the integration of delay and throughput measures in distributed processing models.* 1981.

47. Marsan, M.A., *Stochastic Petri nets: an elementary introduction*, in *Advances in Petri Nets 1989*. 1990, Springer. p. 1-29.
48. Jensen, K., *Coloured petri nets*, in *Petri nets: central models and their properties*. 1987, Springer. p. 248-299.
49. Jensen, K., *Coloured Petri nets and the invariant-method.* Theoretical Computer Science, 1981. **14**(3): p. 317-336.
50. Popova-Zeugmann, L., *Time Petri Nets*. 2013: Springer.
51. Silva, J.R. and P.M. del Foyo, *Timed Petri Nets.* 2012.
52. Fortier, P.J. and H.E. Michel, *Computer systems performance evaluation and prediction*. 2003: Access Online via Elsevier.
53. Parkin, J. and B. Cohen, *An overview of the immune system.* The Lancet, 2001. **357**(9270): p. 1777-1789.
54. Male, D., *Immunology: an illustrated outline*. 2004: Mosby.
55. Na, D., et al., *Integration of Immune Models Using Petri Nets*, in *Artificial Immune Systems*, G. Nicosia, et al., Editors. 2004, Springer Berlin / Heidelberg. p. 205-216.
56. Park, I., et al., *Fuzzy Continuous Petri Net-Based Approach for Modeling Helper T Cell Differentiation*, in *Artificial Immune Systems*, C. Jacob, et al., Editors. 2005, Springer Berlin / Heidelberg. p. 331-338.
57. Park, I., et al., *Fuzzy Continuous Petri Net-Based Approach for Modeling Immune Systems*, in *Neural Nets*, B. Apolloni, et al., Editors. 2006, Springer Berlin / Heidelberg. p. 278-285.
58. Gogolinska, A., E. Ochmanski, and W. Nowak, *Petri Nets in Immunological System Modeling*, in *Matematyczne metody modelowania i analizy systemow wspolbieznych MASYW 2010*, O. Ochmanski and W. Penczek, Editors. 2012. p. 35-46.
59. Gogolinska, A. and W. Nowak, *Petri Nets Approach to Modeling of Immune System and Autism*, in *Artificial Immune Systems*, C. Coello Coello, et al., Editors. 2012, Springer Berlin / Heidelberg. p. 86-99.
60. Iwasaki, A. and R. Medzhitov, *Regulation of Adaptive Immunity by the Innate Immune System.* Science, 2010. **327**(5963): p. 291-295.
61. Gołąb Jakub, et al., *Immunologia*. 2008: Wydawnictwo Naukowe PWN.
62. Britannica, E. *http://www.britannica.com/EBchecked/topic/205674/fever*. 2012.
63. Hasday, J.D., K.D. Fairchild, and C. Shanholtz, *The role of fever in the infected host.* Microbes Infect, 2000. **2**(15): p. 1891-904.
64. Ostberg, J.R., et al., *Regulatory potential of fever-range whole body hyperthermia on Langerhans cells and lymphocytes in an antigen-dependent cellular immune response.* J Immunol, 2001. **167**(5): p. 2666-70.
65. Jampel, H.D., et al., *Fever and immunoregulation. III. Hyperthermia augments the primary in vitro humoral immune response.* J Exp Med, 1983. **157**(4): p. 1229-38.
66. Mullbacher, A., *Hyperthermia and the generation and activity of murine influenza-immune cytotoxic T cells in vitro.* J Virol, 1984. **52**(3): p. 928-31.
67. Duff, G.W. and S.K. Durum, *Fever and immunoregulation: hyperthermia, interleukins 1 and 2, and T-cell proliferation.* Yale J Biol Med, 1982. **55**(5-6): p. 437-42.
68. Kluger, M.J., *Is fever beneficial?* Yale J Biol Med, 1986. **59**(2): p. 89-95.
69. Meinander, A., et al., *Fever-like hyperthermia controls T Lymphocyte persistence by inducing degradation of cellular FLIPshort.* J Immunol, 2007. **178**(6): p. 3944-53.
70. Evans, S.S., et al., *Fever-range hyperthermia dynamically regulates lymphocyte delivery to high endothelial venules.* Blood, 2001. **97**(9): p. 2727-33.
71. Huang, Y.H., A. Haegerstrand, and J. Frostegard, *Effects of in vitro hyperthermia on proliferative responses and lymphocyte activity.* Clinical & Experimental Immunology, 1996. **103**(1): p. 61-66.
72. Larbi, A., et al., *Aging of the immune system as a prognostic factor for human longevity.* Physiology, 2008. **23**(2): p. 64-74.

73. Weiskopf, D., B. Weinberger, and B. Grubeck-Loebenstein, *The aging of the immune system.* Transplant international, 2009. **22**(11): p. 1041-1050.
74. Browne, S.K., et al., *Adult-Onset Immunodeficiency in Thailand and Taiwan.* New England Journal of Medicine, 2012. **367**(8): p. 725-734.
75. Newschaffer, C.J., et al., *The epidemiology of autism spectrum disorders*.* Annu. Rev. Public Health, 2007. **28**: p. 235-258.
76. Freitag, C.M., *The genetics of autistic disorders and its clinical relevance: a review of the literature.* Molecular Psychiatry, 2006. **12**(1): p. 2-22.
77. Szatmari, P., *Heterogeneity and the genetics of autism.* Journal of Psychiatry and Neuroscience, 1999. **24**(2): p. 159.
78. Cook Jr, E.H., *Genetics of autism.* Mental Retardation and Developmental Disabilities Research Reviews, 1998. **4**(2): p. 113-120.
79. Ozonoff, S., et al., *Recurrence risk for autism spectrum disorders: a Baby Siblings Research Consortium study.* Pediatrics, 2011. **128**(3): p. e488-e495.
80. Ashwood, P., et al., *In Search of Cellular Immunophenotypes in the Blood of Children with Autism.* PloS one, 2011. **6**(5): p. e19299.
81. Ashwood, P., S. Wills, and J. Van de Water, *The immune response in autism: a new frontier for autism research.* Journal of leukocyte biology, 2006. **80**(1): p. 1.
82. Goines, P.E. and P. Ashwood, *Cytokine dysregulation in autism spectrum disorders (ASD): Possible role of the environment.* Neurotoxicology and Teratology, 2012.
83. Depino, A.M., *Peripheral and central inflammation in autism spectrum disorders.* Molecular and Cellular Neuroscience, 2012.
84. Patterson, P.H., *Maternal infection and immune involvement in autism.* Trends in molecular medicine, 2011.
85. Onore, C., M. Careaga, and P. Ashwood, *The role of immune dysfunction in the pathophysiology of autism.* Brain, behavior, and immunity, 2012. **26**(3): p. 383-392.
86. Curran, L.K., et al., *Behaviors associated with fever in children with autism spectrum disorders.* Pediatrics, 2007. **120**(6): p. e1386.
87. Mehler, M.F. and D.P. Purpura, *Autism, fever, epigenetics and the locus coeruleus.* Brain research reviews, 2009. **59**(2): p. 388-392.
88. Szelényi, J., *Cytokines and the central nervous system.* Brain research bulletin, 2001. **54**(4): p. 329-338.
89. Quan, N. and W.A. Banks, *Brain-immune communication pathways.* Brain, behavior, and immunity, 2007. **21**(6): p. 727-735.
90. Rohr, C., W. Marwan, and M. Heiner, *Snoopy - a unifying Petri net framework to investigate biomolecular networks.* Bioinformatics, 2010. **26**(7): p. 974.
91. Einloft, J., et al., *MonaLisa—visualization and analysis of functional modules in biochemical networks.* Bioinformatics, 2013: p. btt165.
92. McAfoose, J. and B. Baune, *Evidence for a cytokine model of cognitive function.* Neuroscience & Biobehavioral Reviews, 2009. **33**(3): p. 355-366.
93. Nicol, D.M. and S. Roy. *Parallel simulation of timed Petri-nets*. in *Proceedings of the 23rd conference on Winter simulation*. 1991: IEEE Computer Society.
94. Thomas, G.S. and J. Zahorjan. *Parallel simulation of performance Petri nets: extending the domain of parallel simulation*. in *Simulation Conference, 1991. Proceedings., Winter*. 1991: IEEE.
95. Ferscha, A. *Concurrent execution of timed Petri nets*. in *Simulation Conference Proceedings, 1994. Winter*. 1994: IEEE.
96. Geist, R., et al. *Parallel simulation of Petri nets on desktop PC hardware*. in *Simulation Conference, 2005 Proceedings of the Winter*. 2005.
97. Nvidia, C., *Programming guide*. 2013.
98. Comerford, I. and S.R. McColl, *Mini-review series: focus on chemokines.* Immunology and Cell Biology, 2011. **89**(2): p. 183-184.

99. Reid, C., et al., *Structure activity relationships of monocyte chemoattractant proteins in complex with a blocking antibody.* Protein Engineering Design and Selection, 2006. **19**(7): p. 317-324.
100. Carr, M.W., et al., *Monocyte chemoattractant protein 1 acts as a T-lymphocyte chemoattractant.* Proceedings of the National Academy of Sciences, 1994. **91**(9): p. 3652-3656.
101. Garay, P.A. and A.K. McAllister, *Novel roles for immune molecules in neural development: implications for neurodevelopmental disorders.* Frontiers in synaptic neuroscience, 2010. **2**.
102. De Haas, A., et al., *Neuronal chemokines: versatile messengers in central nervous system cell interaction.* Molecular neurobiology, 2007. **36**(2): p. 137-151.
103. Banisadr, G., et al., *Highly regionalized neuronal expression of monocyte chemoattractant protein-1 (MCP-1/CCL2) in rat brain: Evidence for its colocalization with neurotransmitters and neuropeptides.* Journal of Comparative Neurology, 2005. **489**(3): p. 275-292.
104. Vargas, D., et al., *Neuroglial activation and neuroinflammation in the brain of patients with autism.* Annals of Neurology, 2005. **57**(1): p. 67-81.
105. Ashwood, P., et al., *Associations of impaired behaviors with elevated plasma chemokines in autism spectrum disorders.* Journal of neuroimmunology, 2011. **232**(1): p. 196-199.
106. Bauman, M.L. and T.L. Kemper, *The neurobiology of autism*. 2005: JHU Press.
107. Berendsen, H.J.C., et al., *Molecular dynamics with coupling to an external bath.* The Journal of Chemical Physics, 1984. **81**(8): p. 3684-3690.
108. Nowak, W., *Applications of computational methods to simulations of proteins dynamics*, in *Handbook of Computational Chemistry*. 2012, Springer. p. 1127-1153.
109. Foloppe, N. and A.D. MacKerell Jr, *All-atom empirical force field for nucleic acids: I. Parameter optimization based on small molecule and condensed phase macromolecular target data.* Journal of computational chemistry, 2000. **21**(2): p. 86-104.
110. Berman, H.M., et al., *The protein data bank.* Nucleic acids research, 2000. **28**(1): p. 235-242.
111. Grubmüller, H., B. Heymann, and P. Tavan, *Ligand Binding: Molecular Mechanics Calculation of the Streptavidin-Biotin Rupture Force.* Science, 1996. **271**(5251): p. 997-999.
112. Marszalek, P.E., et al., *Mechanical unfolding intermediates in titin modules.* Nature, 1999. **402**(6757): p. 100-103.
113. Brooks, B.R., et al., *CHARMM: A program for macromolecular energy, minimization, and dynamics calculations.* Journal of computational chemistry, 1983. **4**(2): p. 187-217.
114. Brooks, B.R., et al., *CHARMM: the biomolecular simulation program.* Journal of computational chemistry, 2009. **30**(10): p. 1545-1614.
115. Phillips, J.C., et al., *Scalable molecular dynamics with NAMD.* Journal of computational chemistry, 2005. **26**(16): p. 1781-1802.
116. Humphrey, W., A. Dalke, and K. Schulten, *VMD: visual molecular dynamics.* Journal of molecular graphics, 1996. **14**(1): p. 33-38.
117. Bernstein, F.C., et al., *The Protein Data Bank: a computer-based archival file for macromolecular structures.* Journal of Molecular Biology, 1977. **112**(3): p. 535-542.
118. Padavattan, S., et al., *High-affinity IgE recognition of a conformational epitope of the major respiratory allergen Phl p 2 as revealed by X-ray crystallography.* The Journal of Immunology, 2009. **182**(4): p. 2141-2151.
119. Baker, N.A., et al., *Electrostatics of nanosystems: application to microtubules and the ribosome.* Proceedings of the National Academy of Sciences, 2001. **98**(18): p. 10037–10041.

120. Dolinsky, T.J., et al., *PDB2PQR: expanding and upgrading automated preparation of biomolecular structures for molecular simulations.* Nucleic acids research, 2007. **35**(suppl 2): p. W522-W525.
121. Dolinsky, T.J., et al., *PDB2PQR: an automated pipeline for the setup of Poisson–Boltzmann electrostatics calculations.* Nucleic acids research, 2004. **32**(suppl 2): p. W665-W667.
122. Altschul, S.F., et al., *Gapped BLAST and PSI-BLAST: a new generation of protein database search programs.* Nucleic acids research, 1997. **25**(17): p. 3389-3402.
123. Larkin, M., et al., *Clustal W and Clustal X version 2.0.* Bioinformatics, 2007. **23**(21): p. 2947-2948.
124. Waterhouse, A.M., et al., *Jalview Version 2—a multiple sequence alignment editor and analysis workbench.* Bioinformatics, 2009. **25**(9): p. 1189-1191.
125. Clamp, M., et al., *The jalview java alignment editor.* Bioinformatics, 2004. **20**(3): p. 426-427.
126. Mayor, U., et al., *The complete folding pathway of a protein from nanoseconds to microseconds.* Nature, 2003. **421**(6925): p. 863-867.
127. Shao, J., et al., *Clustering molecular dynamics trajectories: 1. Characterizing the performance of different clustering algorithms.* Journal of Chemical Theory and Computation, 2007. **3**(6): p. 2312-2334.
128. Shelley, J.C., et al., *A coarse grain model for phospholipid simulations.* The Journal of Physical Chemistry B, 2001. **105**(19): p. 4464-4470.
129. Srinivas*, G. and M.L. Klein, *Coarse-grain molecular dynamics simulations of diblock copolymer surfactants interacting with a lipid bilayer.* Molecular Physics, 2004. **102**(9-10): p. 883-889.
130. Marrink, S.J., et al., *The MARTINI force field: coarse grained model for biomolecular simulations.* The Journal of Physical Chemistry B, 2007. **111**(27): p. 7812-7824.
131. Tozzini, V., *Coarse-grained models for proteins.* Current opinion in structural biology, 2005. **15**(2): p. 144-150.
132. Jakubowski, R., et al., *Computational Studies of TTR Related Amyloidosis: Exploration of Conformational Space through a Petri Net-Based Algorithm.* TASK Quarterly, 2014(10/2014): p. 18(3):267.
133. Cohen, F.E. and M.J. Sternberg, *On the prediction of protein structure: the significance of the root-mean-square deviation.* Journal of Molecular Biology, 1980. **138**(2): p. 321-333.
134. Zemla, A., et al., *Processing and analysis of CASP3 protein structure predictions.* Proteins: Structure, Function, and Bioinformatics, 1999. **37**(S3): p. 22-29.
135. Zhang, Y. and J. Skolnick, *Scoring function for automated assessment of protein structure template quality.* Proteins: Structure, Function, and Bioinformatics, 2004. **57**(4): p. 702-710.
136. Godzik, A., *The structural alignment between two proteins: Is there a unique answer?* Protein science, 1996. **5**(7): p. 1325-1338.
137. Holm, L. and C. Sander, *Protein structure comparison by alignment of distance matrices.* Journal of Molecular Biology, 1993. **233**(1): p. 123-138.
138. Shindyalov, I.N. and P.E. Bourne, *Protein structure alignment by incremental combinatorial extension (CE) of the optimal path.* Protein engineering, 1998. **11**(9): p. 739-747.
139. Holland, R.C., et al., *BioJava: an open-source framework for bioinformatics.* Bioinformatics, 2008. **24**(18): p. 2096-2097.
140. Ye, Y. and A. Godzik, *Flexible structure alignment by chaining aligned fragment pairs allowing twists.* Bioinformatics, 2003. **19**(suppl 2): p. ii246-ii255.
141. Gogolinska, A. and W. Nowak, *Petri Nets Formalism Facilitates Analysis of Complex Biomolecular Structural Data.* RAIRO-OPERATIONS RESEARCH, 2015. **under review**.

142. Wojtczak, A., P. Neumann, and V. Cody, *Structure of a new polymorphic monoclinic form of human transthyretin at 3 A resolution reveals a mixed complex between unliganded and T4-bound tetramers of TTR.* Acta Crystallographica Section D: Biological Crystallography, 2001. **57**(7): p. 957-967.
143. Jolliffe, I., *Principal component analysis*. 2005: Wiley Online Library.
144. Angles, R. *A comparison of current graph database models*. in *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*. 2012: IEEE.
145. Vendruscolo, M. and E. Domany, *Protein folding using contact maps.* Vitamins & Hormones, 2000. **58**: p. 171-212.
146. Faure, G., A. Bornot, and A.G. de Brevern, *Protein contacts, inter-residue interactions and side-chain modelling.* Biochimie, 2008. **90**(4): p. 626-639.
147. Cock, P.J., et al., *Biopython: freely available Python tools for computational molecular biology and bioinformatics.* Bioinformatics, 2009. **25**(11): p. 1422-1423.
148. Vehlow, C., et al., *CMView: interactive contact map visualization and analysis.* Bioinformatics, 2011. **27**(11): p. 1573-1574.
149. Henrick, K., et al., *Remediation of the protein data bank archive.* Nucleic acids research, 2008. **36**(suppl 1): p. D426-D433.
150. Nvidia, C., *CUDA C Best Practices Guide*. 2013.
151. NVIDIA. *http://www.nvidia.pl/object/product-geforce-gtx-580-pl.html*.
152. Gosling, J., et al., *Java (TM) Language Specification, The (Java (Addison-Wesley))*. 2005: Addison-Wesley Professional.