

## BACHELOR PROJECT ASSIGNMENT

**Student:** Michal Frdlík  
**Study programme:** Software Engineering and Management  
**Specialisation:** Intelligent Systems  
**Title of Bachelor Project:** The Use of Artificial Immune Systems in Biomedical Information Retrieval

### Guidelines:

1. Perform study of the problematic using relevant literature.
2. Implement at least two AIS algorithms inspired by immune system. Target the problem of free text pattern mining (classification/clustering).
3. Evaluate performance of the algorithm using the data provided.
4. Compare the performance with one non-AIS algorithm.
5. The implementation should be multithreaded (intended for grid computing).

### Bibliography/Sources:

- [1] De Castro, L. N.: Fundamentals of Natural Computing; Chapman & Hall/CRC New York, 2006.
- [2] De Castro, L.N and Von Zuben, F. (2001). "aiNET: An Artificial Immune Network for Data Analysis", in Data Mining: A Heuristic Approach. Abbas, H.; Sarker, R. and Newton, C. (Eds). Idea Group Publishing.

**Bachelor Project Supervisor:** Ing. Miroslav Burša

**Valid until:** the end of the winter semester of academic year 2012/2013

  
prof. Ing. Vladimír Mařík, DrSc.  
**Head of Department**



  
prof. Ing. Pavel Ripka, CSc.  
**Dean**

## ZADÁNÍ BAKALÁŘSKÉ PRÁCE

**Student:** Michal Frdlík

**Studijní program:** Softwarové technologie a management

**Obor:** Inteligentní systémy

**Název tématu:** Využití umělých imunních systémů pro získávání medicínských informací

### Pokyny pro vypracování:

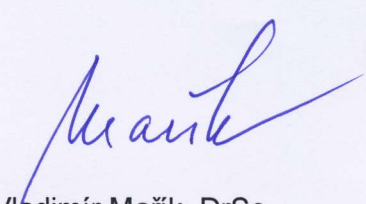
1. Provedte studium odborné literatury týkající se zmiňovaného problému.
2. Naimplementujte alespoň dva vybrané algoritmy inspirované imunitním systémem. Cílem je klasifikace a shlukování elementů ve volném textu.
3. Vyhodnoťte úspěšnost algoritmu na zadaných datech.
4. Dle časových možností srovnajte s jedním algoritmem, který nepatří do skupiny imunitních.
5. Implementace bude vícevláknová (spouštění na gridu).

### Seznam odborné literatury:

- [1] De Castro, L. N.: Fundamentals of Natural Computing; Chapman & Hall/CRC New York, 2006.
- [2] De Castro, L.N and Von Zuben, F. (2001). "aiNET: An Artificial Immune Network for Data Analysis", in Data Mining: A Heuristic Approach. Abbas, H.; Sarker, R. and Newton, C. (Eds). Idea Group Publishing.

**Vedoucí bakalářské práce:** Ing. Miroslav Burša

**Platnost zadání:** do konce zimního semestru 2012/2013

  
prof. Ing. Vladimír Mařík, DrSc.  
vedoucí katedry

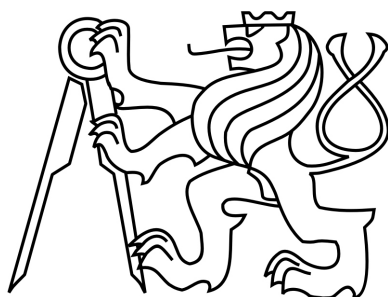


  
prof. Ing. Pavel Ripka, CSc.  
děkan

V Praze dne 30. 11. 2011

Bachelor's Thesis

# The use of Artificial Immune Systems in Biomedical Information Retrieval



Czech Technical University in Prague  
Faculty of Electrical Engineering  
Department of Cybernetics

Michal Frdlík

Supervisor: Ing. Miroslav Burša

Field of study: Software Technologies and Management, Intelligent Systems

May 2012



## Acknowledgements


I would like to express my gratitude to my supervisor, Ing. Miroslav Burša, for his support, professional guidance and for the time he spent reading and correcting this thesis when it was far from being finished and in the times thereafter.



## Statement

I hereby state, that I have completed this thesis on my own and I have properly specified all the information sources used, according to the Guideline about observance of the ethic principles concerning university theses.

In Prague, 9<sup>th</sup> May, 2012



.....  
Michal Frdlík





# Abstract

The aim of this thesis is to evaluate performance and usability of selected Artificial Immune Systems (AIS) on the problem of classification and automatic processing of loosely structured free-text medical records. After a review on the state-of-the-art AIS algorithms, we have selected and implemented Artificial Immune Recognition System (AIRS) and Negative Selection Algorithm (NSA) algorithms as representatives. After preliminary testing and behaviour study we have altered the algorithms to fit provided datasets using (among others) modified distance metric based on the Damerau-Levenshtein distance. On the datasets sized 22 000 and 1 500 000 words, we have obtained the following best classification accuracy: 78.17 %, 65.80 % respectively for the AIRS and 81.22 %, 64.49 % respectively for the NSA.

# Abstrakt

Cílem této práce je užití vybraných algoritmů skupiny Umělých imunitních systémů (AIS) ke klasifikaci volně strukturovaného textu z oboru biomedicíny. Po posouzení a zhodnocení algoritmů skupiny AIS byly jako reprezentativní algoritmy vybrány Umělý imunitní rozpoznávací systém (AIRS) a Algoritmus negativní selekce (NSA). Po předběžném testování těchto algoritmů na jednoduchých reálných a umělých datech byly původní algoritmy pozměněny tak, aby byly schopny klasifikace volně strukturovaného textu, mimo jiné za použití vzdálenostní metriky založené na Damerauově Levenshteinově vzdálenosti. Na množinách dat o velikostech 22 000 slov a 1 500 000 slov dosáhl algoritmus AIRS nejlepší klasifikační přesnosti 78,17 % a 65,80 %, algoritmus NSA 81,22 % a 64,49 %.



# Table of contents

1. Introduction . . . . .	13
1. <i>Aim of this work</i> . . . . .	13
<i>Goal specification</i> . . . . .	13
2. <i>What is an Artificial Immune System?</i> . . . . .	13
3. <i>Brief history of AIS[12]</i> . . . . .	14
2. Understanding the Immune System . . . . .	15
1. <i>Adaptive Immune System</i> . . . . .	15
<i>Antibody → antigen matching</i> . . . . .	15
<i>Somatic Hypermutation</i> . . . . .	16
<i>Clonal Selection Mechanism</i> . . . . .	16
3. Artificial Immune Systems . . . . .	17
1. <i>Artificial Immune Recognition System (AIRS)</i> . . . . .	17
<i>Pseudocode</i> . . . . .	19
2. <i>AIRS Implementation Test</i> . . . . .	20
3. <i>Negative Selection Algorithm</i> . . . . .	22
<i>Pseudocode</i> . . . . .	22
4. <i>NSA Implementation Test</i> . . . . .	23
5. <i>Parallelisation capabilities</i> . . . . .	24
4. Preliminary testing . . . . .	25
1. <i>Datasets</i> . . . . .	25
<i>Gaussian dataset</i> . . . . .	25
<i>Iris dataset</i> . . . . .	25
<i>Test methodology</i> . . . . .	26
2. <i>Testing AIRS</i> . . . . .	26
3. <i>Testing NSA</i> . . . . .	28
<i>Test methodology</i> . . . . .	28
<i>Antibody count</i> . . . . .	28
<i>Matching factor</i> . . . . .	28
4. <i>Optimising parameters for given datasets</i> . . . . .	29
<i>Results</i> . . . . .	31
5. <i>Comparison</i> . . . . .	32
5. Biomedical data . . . . .	33
1. <i>Data character</i> . . . . .	33
2. <i>Classification strategy</i> . . . . .	33
<i>Cluster analysis</i> . . . . .	33
<i>Word frequency analysis</i> . . . . .	33
3. <i>Distance measurement</i> . . . . .	34
<i>Hamming distance</i> . . . . .	35
<i>Levenshtein distance</i> . . . . .	35
<i>Damerau-Levenshtein distance</i> . . . . .	35

<i>The cosine similarity</i>	36
<i>The product distance</i>	36
4. <i>Word modifiers encoding</i>	37
5. <i>Similar word identification and frequency merging</i>	38
6. <i>Unique wordlist</i>	39
7. <i>Unimportant data removal</i>	39
8. <i>Class generation</i>	39
9. <i>Class identifier object</i>	40
6. AIRS for biomedical data	41
1. <i>Classification strategy</i>	41
<i>The binary counter and sentence based distance approach</i>	41
<i>The 1-of-k counter and word based distance approach</i>	41
2. <i>Algorithm variation</i>	42
3. <i>Testing</i>	43
<i>Test methodology</i>	43
<i>Test results</i>	43
<i>Real classification performance</i>	48
<i>Optimisation of input parameter values</i>	49
<i>Results</i>	49
7. NSA for biomedical data	51
1. <i>Definitions</i>	51
<i>Information source</i>	51
<i>Markov chain</i>	51
<i>Markov information source</i>	51
2. <i>Classification strategy</i>	51
3. <i>Testing</i>	52
<i>Test results</i>	56
<i>Real classification performance</i>	56
<i>Optimisation of input parameter values</i>	57
<i>Results</i>	57
8. Final results (on both datasets)	59
1. <i>Less complex dataset (22 000 words)</i>	59
<i>Settings</i>	59
<i>Algorithm settings</i>	59
<i>Results</i>	59
2. <i>More complex dataset (1 500 000 words)</i>	61
<i>Preprocess</i>	61
<i>Results</i>	61
<i>Parameters</i>	62
9. Conclusion	63
1. <i>Goals to achievements mapping</i>	63
2. <i>Work not declared in goals</i>	64
<i>Algorithm alteration</i>	64
<i>Product distance</i>	64
<i>Genetic Algorithm Parameter Selector</i>	64
3. <i>Final conclusion</i>	64

A. Statistics . . . . .	66
1. <i>Mean value</i> . . . . .	66
2. <i>Median</i> . . . . .	66
3. <i>Variance</i> . . . . .	66
4. <i>Standard deviation</i> . . . . .	66
5. <i>String Markov information source example</i> . . . . .	67
6. <i>Boxplot</i> . . . . .	68
7. <i>Multiple data series plot</i> . . . . .	68
B. Pseudocode syntax . . . . .	69
C. Machine specification . . . . .	69
D. DVD directory and file structure . . . . .	70
References . . . . .	72



# 1 Introduction

## 1.1 Aim of this work

We have divided the work into the following goals which have to be achieved.

### Goal specification

1) **Literature study**

We need to study relevant and useful publications concerning AIS and free-text pattern mining approaches, as well as string distance metrics.

2) **Selection of 2 algorithms**

Based on 1), we need to select two representative algorithms, which will be implemented, analysed and used.

3) **Implementation**

We need to design and implement modular and well-arranged framework in order to make algorithm diagnostic, testing and text analysis possible.

4) **Preliminary testing**

We need to make preliminary tests in order to see, how these algorithms behave on simple datasets. These tests need to be made on both real and artificially created datasets.

5) **Biomedical data preprocessing and class assignment**

We need to preprocess the biomedical data in order to normalize them and in order to create and assign artificially created classes to them, as we are provided with text records only, with no annotation whatsoever.

6) **Algorithm performance evaluation**

We need to evaluate the performance of the adapted algorithms on a real biomedical data in order to study their behaviour and to find optimal parameters.

7) **Large scale testing and comparison**

In the end, we need to compare performance of the algorithms on very large real biomedical datasets and compare it with performance of a non-AIS algorithm.

## 1.2 What is an Artificial Immune System?

An *Artificial Immune System* is generally a biologically inspired problem-solving algorithm. Its inspiration comes from the mammalian immune system, strictly speaking from its generalisation abilities (see chapter 2). These algorithms usually solve classification and optimisation problems, but there are also several algorithms from this

branch, which are used for intrusion detection. Optimisation algorithms search for optimal states in functions (for instance, travelling salesman problem, SAT problem, knapsack problem etc.), whereas classification algorithms classify their inputs, based on example inputs presented to them before (for instance, several shapes are presented to hypothetical algorithm along with their respective labels (classes) and then an unknown shape is presented to this algorithm. It's job is to say, of what class that shape most likely is).

### 1.3 Brief history of AIS<sup>[12]</sup>

The origins of AIS has its roots in the early theoretical immunology work of J. Doyne Farmer, Alan Perelson and Francisco Varela, with a key work being by Farmer, Packard and Perelson<sup>[11]</sup>. These works investigated a number of theoretical immune network models proposed to describe the maintenance of immune memory. Whilst controversial from an immunological perspective, these models began to give rise to an interest from the computing community. The most influential people at crossing the divide between computing and immunology in the early days were Hugues Bersini and Stephanie Forrest. It is fair to say that some of the early work by Bersini was very well rooted in immunology, and this is also true of the early work by Forrest. It was these works that formed the basis of a solid foundation for the area of AIS. In the case of Bersini, he concentrated on the immune network theory, examining how the immune system maintained its memory and how one might build models and algorithms mimicing that property. With regards to Forrest, her work was focussed on computer security (in particular network intrusion detection) paying attention to the ability of the immune systems to discriminate between self and non-self. These works formed the basis of a great deal of further research by the community on the application of immune inspired techniques to computer security. Due to a growing amount of work conducted on AIS, the International Conference on Artificial Immune Systems (ICARIS) conference series has been started in 2002.



## 2 Understanding the Immune System

In [7], Dasgupta and Nino thoroughly describe how human immune system works, how cleverly and almost perfectly it has been designed by the nature and finally, how computer scientists can utilize the ideas, on which it is based. In the following paragraphs, the basic principles are mentioned, as they are crucial for understanding, how AIS work.

According to [7], living organisms, such as human bodies, need to resist harmful effects of a biological environment, they live in. The resistivity to biological entities is provided by an *immune system*, which is three-layered. *Physical barriers* (such as mucous membrane) represent the first layer of defence. The second layer, called an *innate* (also non-specific) *immune system*, is supposed to destroy antigens (antigen means entity harmful to the body, such as bacteria, viruses etc.), which shows certain molecular structure, known to the body. Finally, the third layer of immunity is called an *adaptive* (also specific) immune system, which is supposed to destroy antigens, which are recognised from past attacks. This layer (third) clearly disposes of certain recognition abilities, thus it becomes particularly important for this work.

### 2.1 Adaptive Immune System

Adaptive immune system of human body shows two major abilities—memory and adaptivity. It is capable to remember the *pattern*, which detected an *antigen* (we say an antibody *matched* an antigen), improve it and reuse it in a later exposure to the same or similar antigen. Cells called *lymphocytes* represent antibodies in a human immune system. These lymphocytes are of several kinds—generally two—*T cells* and *B cells*.

#### Antibody → antigen matching

B-lymphocytes have protein called *BCR* (B-Cell Receptor—immunoglobulin) on their surface, that can bind to another cell on a molecular basis. When the binding between a B-lymphocyte and an antigen is tight enough (we say, the *affinity* is high), the B-lymphocyte is said to be stimulated. When stimulated, the matched cell is probably an antigen, thus an immune reaction is started. Firstly it's *somatic hypermutation*.

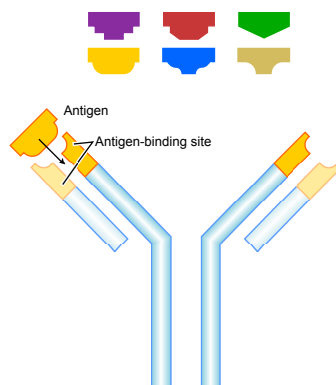


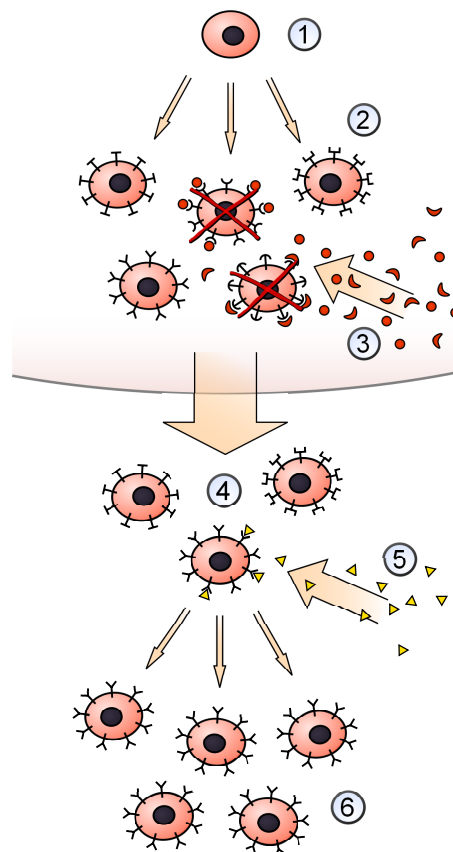
fig. 1 — B-Cell Receptors and molecular binding<sup>[[wikimedia commons](#)]</sup>

## Somatic Hypermutation

When an antigen is bound by an antibody's BCR, the antibody starts to *clone* itself (clone rate depends on the rate of *stimulation*), whereas the clones undergo a process of mutation, where their BCR's are slightly altered. After this, they test their affinity to the antigen and then they clone and mutate again (with various rates depending on their affinity). Products of this process, which show low affinity or they're not stimulated at all, are disposed. On the other side, cells, that have the highest affinity at all, are cloned and kept for future encounters.

## Clonal Selection Mechanism

While B-lymphocytes mature in spleen, using mechanics, that are unimportant for this work, T-lymphocytes mature in Thymus, where they undergo (among others) the process called *Negative selection*. In this process, various self-cells (body cells) are presented to T-lymphocytes. When a T-lymphocyte matches any of the self-cells, it is disposed, otherwise, it is kept, cloned and mutated. By the end of this process, only 2 % of T-lymphocytes will have satisfied the criteria.



*fig. 2 — Depiction of the Clonal Selection and the Somatic Hypermutation*<sup>[[wikimedia commons](#)]</sup>

# 3 Artificial Immune Systems

There are many algorithms derived from the original works by Farmer, Packard and Perelson<sup>[11]</sup>, but generally they all belong to four groups, each being inspired by a specific immunological theory:

- Clonal Selection Algorithms (namely AIRS<sup>[4]</sup>, Immunos, CSA<sup>[19]</sup>)
- Negative Selection Algorithms (namely NSA<sup>[1]</sup>)
- Dendritic Cell Algorithms (namely DCA<sup>[20]</sup>)
- Immune Network Algorithms (namely AINE<sup>[21]</sup>, optAInet<sup>[22]</sup>)

The other classification is:

- B-Cell Inspired Algorithms (namely AIRS)
- T-Cell Inspired Algorithms (namely NSA)

Because B-Cell inspired algorithms and T-Cell inspired algorithms share very little in their operation principles, the AIRS and the NSA were chosen to be the two algorithms, which are studied in this thesis. Namely AIRS and NSA are completely different—AIRS can work with multiple classes, whereas NSA is purely a binary classifier. AIRS uses the k-nearest-neighbour algorithm as its matching function, whereas NSA uses radius threshold matching. AIRS tries to cover the self-space, whereas NSA tries to do the exact opposite—cover the non-self space.

## 3.1 Artificial Immune Recognition System (AIRS)

AIRS uses a training set to build a pool of *memory cells*, which should properly match a cell, which is unknown to the system during training.

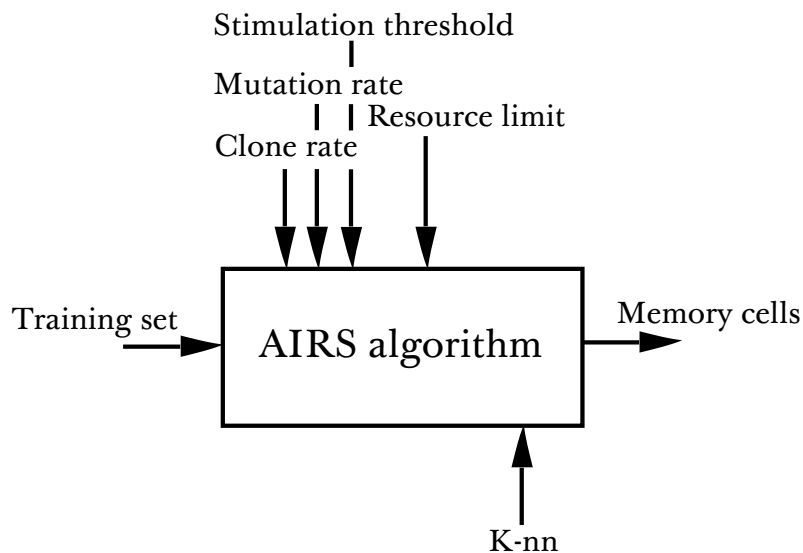


fig. 3 – Basic AIRS algorithm scheme (input, output, parameters)

The artificial cell is an object (or generally a data structure), which contains a data carrier, that carries the data it should represent, a class label (with no information in case of an unknown cell). Every artificial cell must also store and provide the information about its stimulation and about the amount of resources (see (5)) it claims.

In figure 4, a basic scheme of the AIRS is shown.

Firstly, let us define the domain  $d$  on which the AIRS will classify (in  $d$ , there are data separated to  $n$  classes). Let the *memory cell pool*  $MCP$  be the list of cells, that will be the output of AIRS and let the distance function be the mapping  $f : (a, b) \rightarrow d$ , where  $a$  and  $b$  are cells and  $d \in \mathbb{R}$ . Let  $clonerate \in \mathbb{R}$ ,  $mutrate \in \mathbb{R}$  be the input parameters of AIRS, that control rates of cloning and mutation and let  $maxres \in \mathbb{N}$  ( $\mathbb{N} = \{1, 2, \dots\}$ ) be the maximal amount of resources, that all cells can possess (explained below, see (5)).

Firstly, the system is initialised with one sample cell from every class. Then a random antigen cell  $c$  is generated over a domain  $d$ . Then every cell  $c_i$  in a memory cell pool  $MCP$  is stimulated by  $c$ —that means, for every  $i \in MCP$  a distance  $distance = f(c, i)$  is measured, where  $f$  is a cell distance function (in two-dimensional domain of real numbers it may be euclidean distance). Then the affinity is calculated as a relative distance

$$affinity = distance / max\_distance, \quad (1)$$

where  $max\_distance$  is maximal possible distance of two cells in the current domain. Then the rate of stimulation is calculated logically as

$$stimulation = 1 - affinity. \quad (2)$$

Then, the most stimulated cell  $c_{best}$  is compared with presented antigen  $c$ —if there's no match in class, the antigen cell is added straight into the memory cell pool. If there is a match in class and if the antigen  $c$  is not equal to antibody  $c_{best}$  (i.e. the stimulation is not exactly 1), then *Artificial Recognition Ball* (ARB) pool is created. ARB pool is initialised by cloning  $k$  clones of  $c_{best}$ , where  $k$  is

$$k = stimulation(c_{best}) \cdot clonerate \cdot mutrate \quad (3)$$

In addition, every clone undergoes a *mutation* before its entry to the ARB pool. This procedure mimics somatic hypermutation mentioned in chapter 2. The mutation procedure itself is exactly the same as it is in genetic algorithms—a random alteration to prevent the system deadlock at local extreme. The more stimulated is the cell, the more substantial the mutation is (to maintain convergency).

After initialising, the ARB pool needs to be refined, because it contains too many inviable cells. Entire pool is stimulated and then the mean stimulation is calculated for every loop of refining procedure:

$$meanstim = \frac{\sum_{c \in ARB} (stimulation(c))}{size(ARB)} \quad (4)$$

If a mean *stimulation threshold* (one of input parameters) is met, then the best cell  $c_{cand}$  from ARB pool is selected and claimed a candidate cell.

If it is not met, cells in ARB pool must mutate again to reach the threshold criterion. To maintain convergency, each cell  $c$  in ARB pool has an amount of resources given as:

$$resources(c) = stimulation(c) \cdot clonerate \quad (5)$$

where *clonerate* is a system input parameter. Another input parameter is a maximal amount of resources *maxres* 'taken' by the system. Each loop, the weakest cells (i.e. cells 'taking' the lowest amounts of resources) are being disposed until the threshold criterion is met. When the threshold criterion is met,  $c_{cand}$  of ARB pool is selected as a candidate and the rest of ARB pool is disposed.

Then  $c_{cand}$  is compared with  $c_{best}$ . If  $c_{cand}$  has better stimulation, it is added to the memory cell pool. This whole training procedure repeats for  $numpatterns \in \mathbb{N}$  steps, where *numpatterns* is an input parameter.

The classification itself is then stimulating the memory cell pool with the given antigen. After the memory cell pool is stimulated, then the class-carrying cell is selected using k-nearest-neighbour algorithm.

## Pseudocode

The following pseudocode (syntax explained in Appendix II) describes the most important parts of the algorithm.

```
Procedure: Train System
Input: stimthresh, mutrate, clonerate, maxres, TRS, knn
Output: memcells

memcells := initmemcells(TRS)
for(Cell c in TRS)
  stimulate(memcells, c)
  bestmatch := getMostStimulated(memcells)
  if(bestmatch=c)
    continue
  fi
  if(not label(bestmatch)=label(c))
    add(memcells, newcell(data(c), label(c)))
  else if(stimulation(bestmatch)<1)
    pool := createARBpool(c, bestmatch, clonerate, mutrate)
    candidate := refineARBpool(pool, c, stimthresh, clonerate,...
      ...maxres)
    addifbetter(memcells, candidate, bestmatch)
  fi
end

Procedure: createARBpool
Input: c, bestmatch, clonerate, mutrate
Output: pool
```

---

```
add(pool, newcell(data(bestmatch), label(bestmatch)))
clonecount := round(stimulation(bestmatch)*clonerate*mutrate)

for(i:=0, i<clonecount, i++)
  cell := newcell(data(bestmatch), label(bestmatch))
  add(pool, mutateCell(cell, bestmatch))
end

Procedure refineARBpool
Input: pool, p, stimthresh, clonerate, mutrate
Output: candidate

meanstim:=0
do
  stimulate(pool, p)
  competition(pool, clonerate, maxres)
  candidate := biggeststim(pool)
  sumstim := 0
  for(o in pool)
    sumstim += stimulation(o)
  end
  meanstim := sumstim/size(pool)
  if(meanstim>stimthresh)
    actpoolsize:=size(pool)
    for(i:=0, i<actpoolsize, i++)
      cell := newcell(data(get(pool, i)), label(get(pool, i)))
      setStimulation(cell, stimulation(get(pool, i)))
      cell2 := mutateStimulatedCell(cell)
      if(cell2 sameas cell)
        decrement(i), continue
      fi
      add(pool, cell2);
    end
  fi
while(meanstim < stimthresh)
```

## 3.2 AIRS Implementation Test

In order to test correctness, the AIRS was first run on a domain of two dimensional vectors (X and Y coordinates), where classes represent certain space (area). Random points are generated using discrete uniform distribution over the area and the goal is to determine which points belong to their respective spaces. The only information about the problem provided to the algorithm is the training set.

Graphic capabilities of Java were used to graphically illustrate the results. In fig.5, the red circles are the test instances (antigens), the blue ones are the memory cells (antibod-

---

ies) and the lines between them are the results of 1-nn. The green line denotes wrong classification.

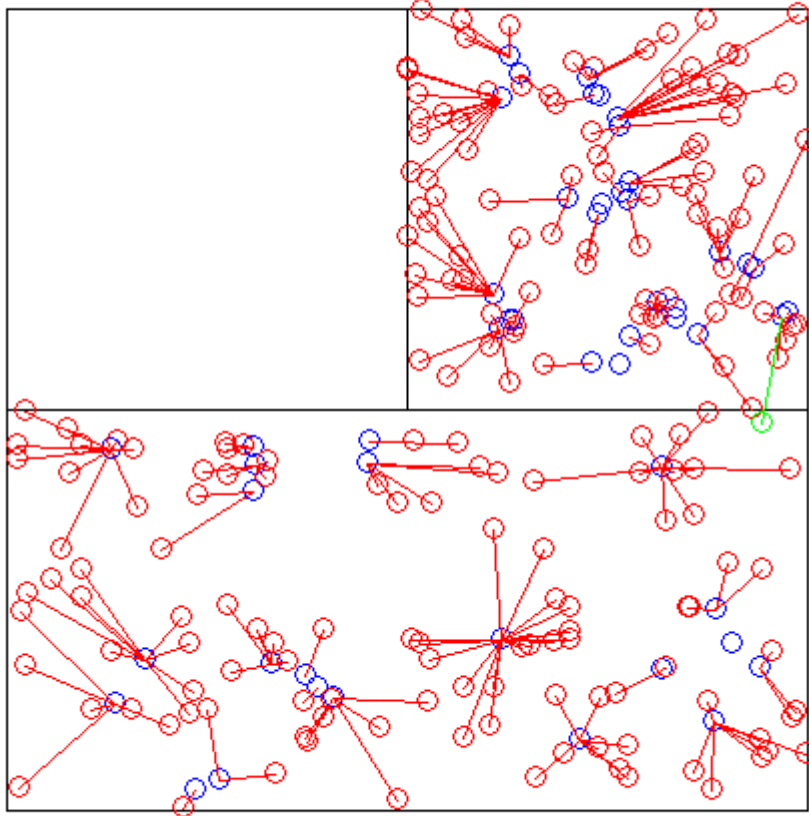


fig. 4 – AIRS sample run on domain of 2D vectors

The upper right quadrant represents one class and the lower two quadrants represent the second.

The time complexity of AIRS (training phase) on this problem can be approximated by the function  $(1/(20 \cdot 10^3)) \cdot n^2$ , whereas the time complexity of testing phase is linear. During this test, AIRS parameter were set as follows: threshold=0.9, antibody count=200, mutrate=2, clonerate=10, maxres=50, knn=1. Machine specification is described in Appendix C.

tab. 1 – AIRS time complexity sample table

Dataset size(n)	10	100	1000	2000	5000	10000	25000	50000
Training [ms]	0	5	60	235	1305	5082	32053	126397
Testing 200 [ms]	3	5	20	40	97	195	553	1136
$(1/(20 \cdot 10^3)) \cdot n^2$	0	0	50	200	1250	5000	31250	125000

### 3.3 Negative Selection Algorithm

The Negative Selection Algorithm is a T-Cell inspired algorithm, designed for binary classification. The principle of NSA is different from that of AIRS. Firstly, the NSA iterates the training set and groups the instances labeled SELF together (i.e. creates a self-set). Then it generates more or less random cells (antibodies) and matches them against the self-set. The matching process itself is different from that of AIRS. Instead of using k-nn, the matching radius is used (i.e. only cells that are less or equally distant from the cell than a given factor are matched). If the antibody matches any cell in the self-set, it fails its mission (because it is supposed not to match SELF cells) and it is disposed. If it does not match anything, it is kept as an antibody, because there is a chance, that it would match an unknown (and therefore NON SELF) cell.

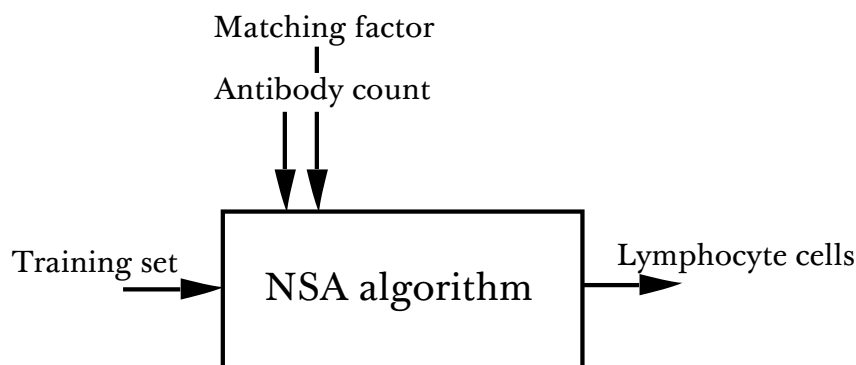


fig. 5 – Basic NSA algorithm scheme (input, output, parameters)

NSA takes two parameters: *Matching factor*  $\in < 0, 1 >$ , which is the above mentioned maximal matching distance and *Antibody count*  $\in \mathbb{N}$ , which denotes, how many antibodies should be created.

#### Pseudocode

The following pseudocode (syntax explained in Appendix II) describes the most important parts of the algorithm.

```
Procedure: Train System  
Input: TRS, size, factor  
Output: antibodies
```

```
selfset := createSelfSet(TRS)  
antibodies := generateAntibodies(selfset, size, factor)
```

```
Procedure: createSelfSet  
Input: TRS  
Output: selfset
```



```
for(cell in TRS)
  if(isSelf(cell))
    add(selfset, cell)
  fi
end

Procedure: generateAntibodies
Input: selfset, size, factor
Output: antibodies

while(1)
  if(immuneReaction(selfset, p = randomAntibody(domain)...
  ..., factor)!=NULL)
    count++
    add(antibodies, p)
    if(count=size)
      break;
    fi
  fi
end

Procedure: immuneReaction
Input: set, stimulus, factor
Output: matched_antibody

matched_antibody:=null

for(cell in set)
  if(match(cell, stimulus, factor))
    matched_antibody:=cell
    break
  fi
end
```

### 3.4 NSA Implementation Test

In order to test correctness, the NSA was first run on a domain of two dimensional vectors (X and Y coordinates), where classes represent certain space (area). Random points are generated using the discrete uniform distribution over the area and the goal is to determine which points belong to their respective spaces. The only information about the problem provided to the algorithm is the training set.

Again, the Java graphics were used to illustrate the results (fig. 6). The red circles are the test instances (antigens), the blue ones are the antibodies and the blue circles around the antibodies represent their matching factor.

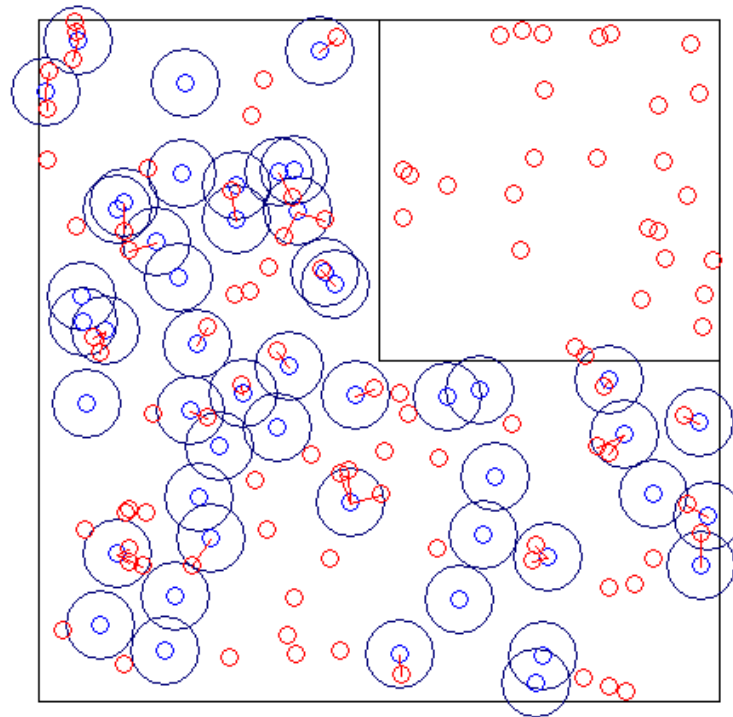


fig. 6 – NSA sample run on domain of 2D vectors

The settings for this test were following: factor=0.05, count=50.

### 3.5 Parallelisation capabilities

In [10], Andrew Watkins and Jon Timmis, the creators of AIRS, explore parallelisation capabilities of AIRS. They had inspired themselves in human immune system and it's natural parallelisation, which is clearly it's dominant feature. They think, that if the AIRS itself is inspired in human immune system, it should contain parallelisation capabilities somehow naturally.

They propose dividing training phase into several processes (each process should own a part of the training set), running AIRS on them independently and then merge final memory cell pools. The question is raised, that if the training set is divided (therefore cell interaction during training is disturbed), shall the results remain the same?

The answer is no. They show, that classification accuracy drops a little with every additional processor, as the cell interaction rate lowers. Also, there is a significant increase in the final memory cell pool size. They propose solving this problem by using *affinity-based merging* (practically it is well known resource competition).

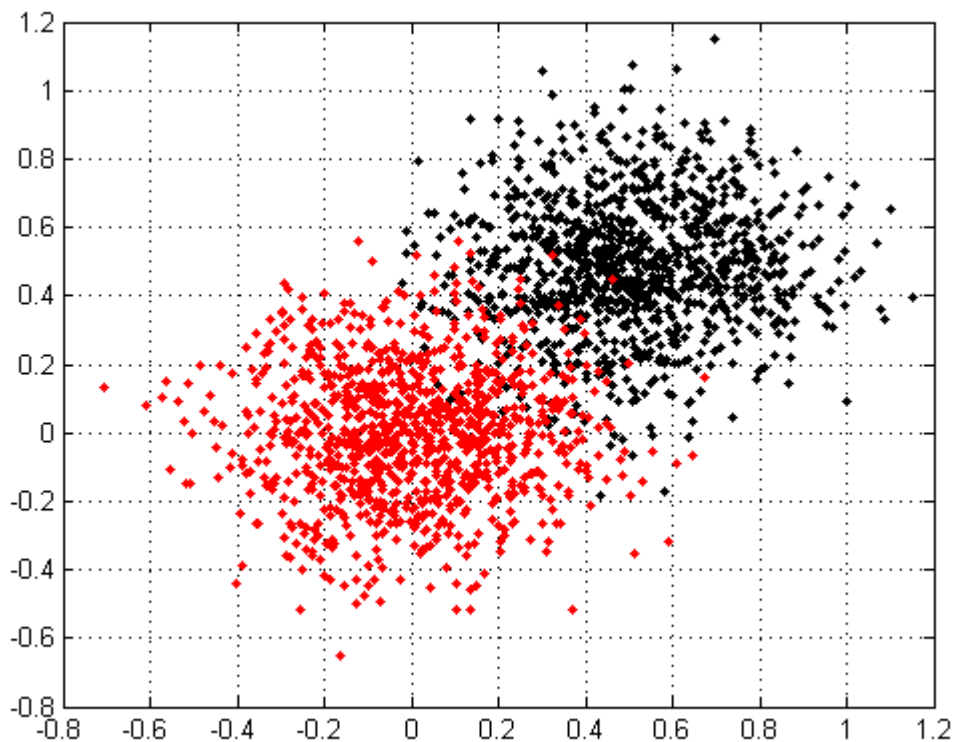
When applying the affinity-based merging, the classification accuracy remains a bit lower (95.86 %/62 cells on 1 processor, 94.86 %/88 cells on 24 processors), but the run-time is significantly decreased.

## 4 Preliminary testing

### 4.1 Datasets

#### Gaussian dataset

For the purpose of thorough testing, a simple two dimensional dataset was defined. In this dataset, points are generated by a random number generator with the normal probability distribution (see Appendix A), where class A has distribution  $N(0, 0.2)$  and class B  $N(0.5, 0.2)$ . Figure 8 visualises the dataset in the Cartesian plane.



*fig. 7 – Gaussian dataset visualisation in the Cartesian plane*

#### Iris dataset

The famous Iris dataset (introduced by Sir Ronald Aylmer Fisher in 1936) contains 150 samples from three species of the Iris flowers (Setosa, Virginica and Versicolor, fig. 8), which grow in Gaspé Peninsula, Canada. The dataset has four dimensions—each sample carries information about petal length, petal width, sepal length and sepal width in centimeters.

Following picture (fig. 8) shows a photo of an Iris Versicolor flower in bloom with distinction of its sepal and petal.

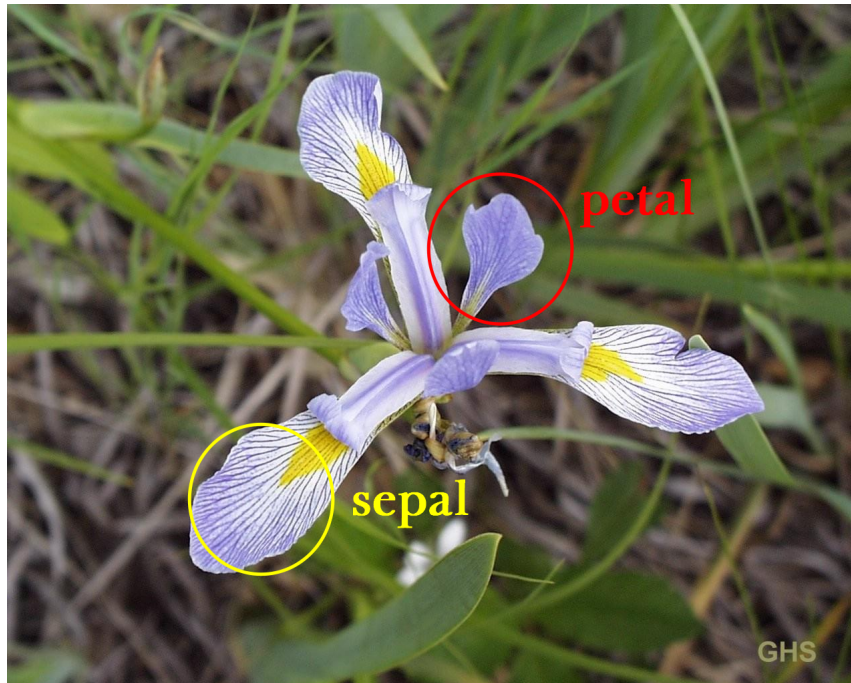


fig. 8 — *Iris Versicolor flower*<sup>[wikimedia commons]</sup>

## Test methodology

In following tests, a dependency of classification accuracy on algorithm parameters is observed. Assuming parameters  $a$ ,  $b$  and  $c$ , when testing  $a$ , then  $b$  and  $c$  are fixed on given value.

The dataset is randomized and split in half. One half is declared the train set and the second one the test set, and the algorithm is run on these. This process is repeated 100 times and the final result is declared an average of the 100 values.

## 4.2 Testing AIRS

When testing AIRS, the values of affinity threshold, clonerate, mutrate, maxres and knn were changed, and the results were observed. On both Gaussian and Iris dataset, there were no significant differences in the classification accuracy for different values of affinity threshold, clonerate, mutrate and maxres, but there were significant differences for different values of knn.

When the parameter values are fixed, they are following: stimthresh=0.9, knn=3, mutation rate=2, clone rate=10.

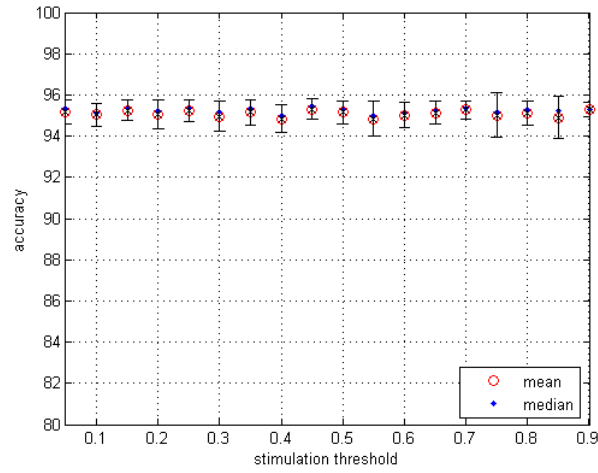


fig. 9 – Dependency of classification accuracy (gauss) on stimulation threshold

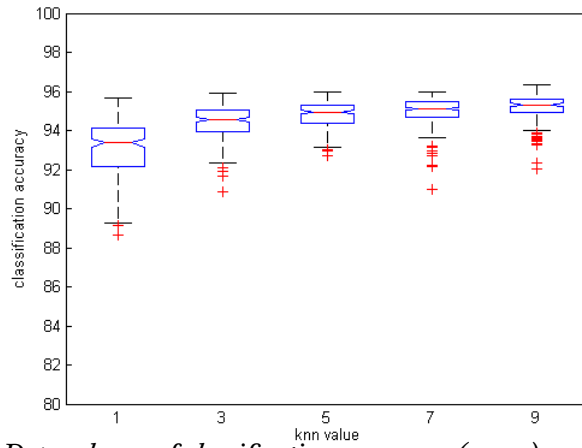


fig. 10 – Dependency of classification accuracy (gauss) on knn (boxplot)

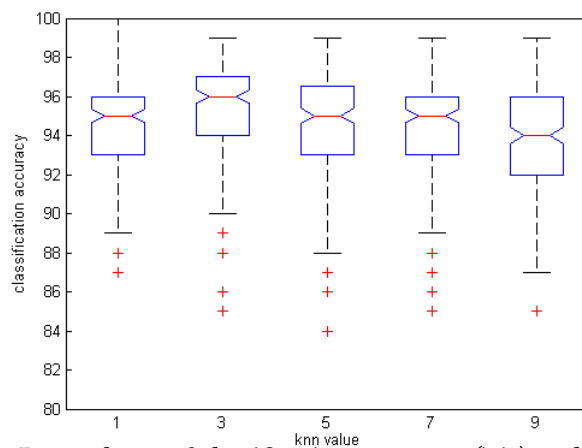


fig. 11 – Dependency of classification accuracy (iris) on knn (boxplot)

Knn parameter seems to be very dataset-specific. On the Gaussian dataset it seems, the bigger knn the better accuracy, but on the Iris dataset, the best value seems to be 3. This preference was observed also when these datasets were tested in Weka (software available online: <http://www.cs.waikato.ac.nz/ml/weka/>) implementation of AIRS.

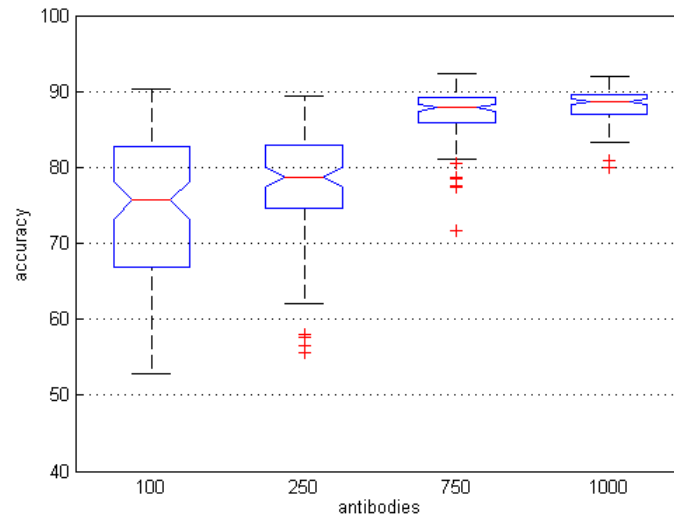
## 4.3 Testing NSA

When the NSA was tested, all of its parameters seemed to matter essentially. When the parameter values are fixed, they are following: antibodies=250, matching factor=1.

### Test methodology

The methodology is the same as in 4.2 (AIRS).

### Antibody count



*fig. 12 — Dependency of classification accuracy (gauss) on antibody count (boxplot)*

The bigger is antibody count, the bigger accuracy. This result makes sense, because more antibodies cover more space and therefore match more non-self cells. On the other side, antibody count is also a parameter of algorithm's time complexity function (which is linear), so bigger count of antibodies will cause worse performance.

### Matching factor

Testing dependency of classification accuracy on NSA matching factor yielded very obvious results. When matching factor is too small or too big, classification is inaccurate, because nothing or everything is matched, respectively. This factor also seems to be dataset-specific (for the Gaussian dataset, bigger values of this parameter made algorithm perform better).

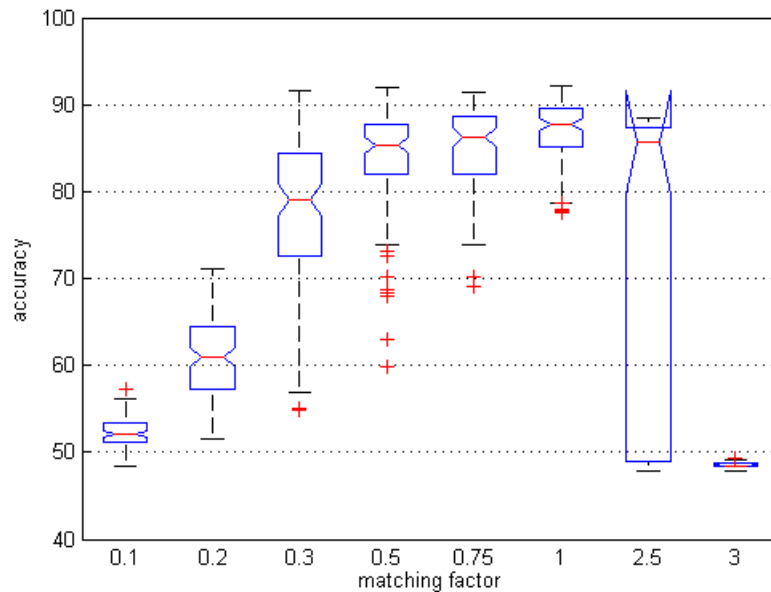


fig. 13 — Dependency of classification accuracy (gauss) on matching factor (boxplot)

Boxplot at  $x=2.5$  is not an error (notches are not bound by quartiles, see Appendix A). :note

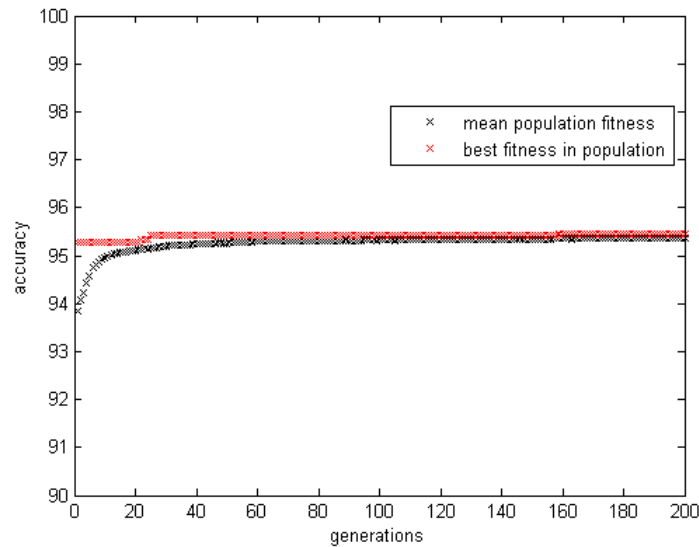
## 4.4 Optimising parameters for given datasets

A simple genetic algorithm (GA<sup>[24]</sup>) was used to determine suboptimal parameter values of AIRS and NSA. A genetic algorithm is a heuristic algorithm used for state-space search inspired in the evolution theory and the theory of natural selection. Basic genetic algorithm has several phases:

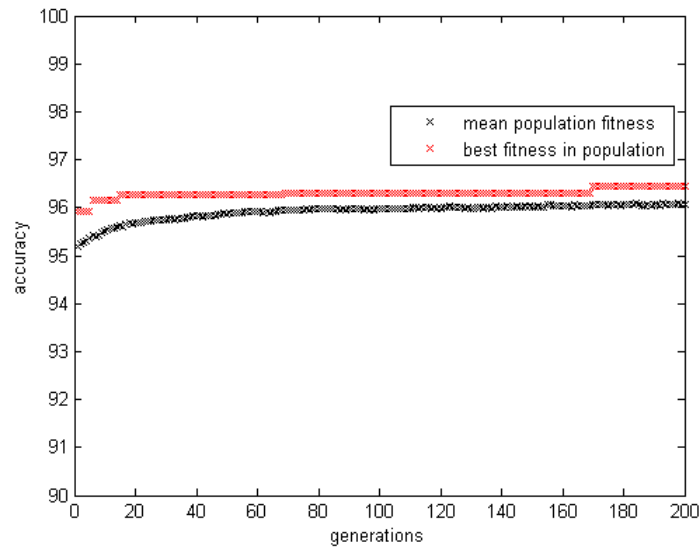
- 1) Initial population generation
- 2) Fitness evaluation
- 3) Selection
- 4) Recombination (also called crossover)
- 5) Mutation
- 6) Altering/Renewing current population

During (1), an initial population is created randomly or semi-randomly (there are situations, when a GA needs to be directed via adding viable individuals to the initial population). During (2), a fitness function is evaluated for every individual in a current population. A fitness function is a function, which computes viability of an individual (its proximity to an optimal state). During (3), usually two individuals are selected based on a given rule (for instance, the roulette selection—every individual takes an area in a virtual roulette wheel relative to its fitness). The selection rule must not be deterministic and must have non-zero probability of choosing non-viable individual. The two selected individuals are then recombined (4). During recombination, genotypes (state representations) of the two selected individuals are mixed based on a recombination rule (usually one-way or two-way crossover). The result of this process are two or more new

individuals, which then undergo a process of mutation (5). A following process of altering/renewing the current population may vary along with different evolution strategies. In one of these strategies, the entire new population is created using the selection, recombination and mutation on the old population. Another strategy is to generate only  $n$  new individuals using the selection, recombination and mutation and replace them with  $n$  least viable individuals in the current population. The second method is used in this thesis. Following diagrams show (fig. 14, 15, 16), how mean population fitness and best fitness varied with number of generations simulated.



*fig. 14 — Dependency of fitness on number of generations (AIRS, gauss)*



*fig. 15 — Dependency of fitness on number of generations (AIRS, iris)*



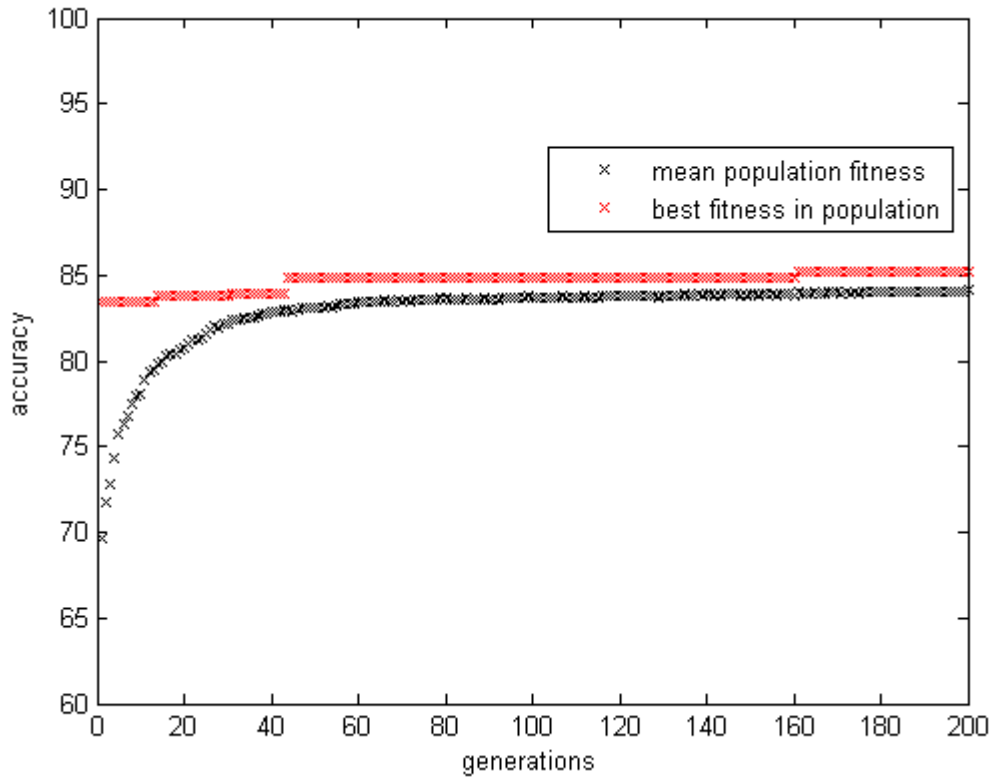


fig. 16 — Dependency of fitness on number of generations (NSA, gauss)

## Results

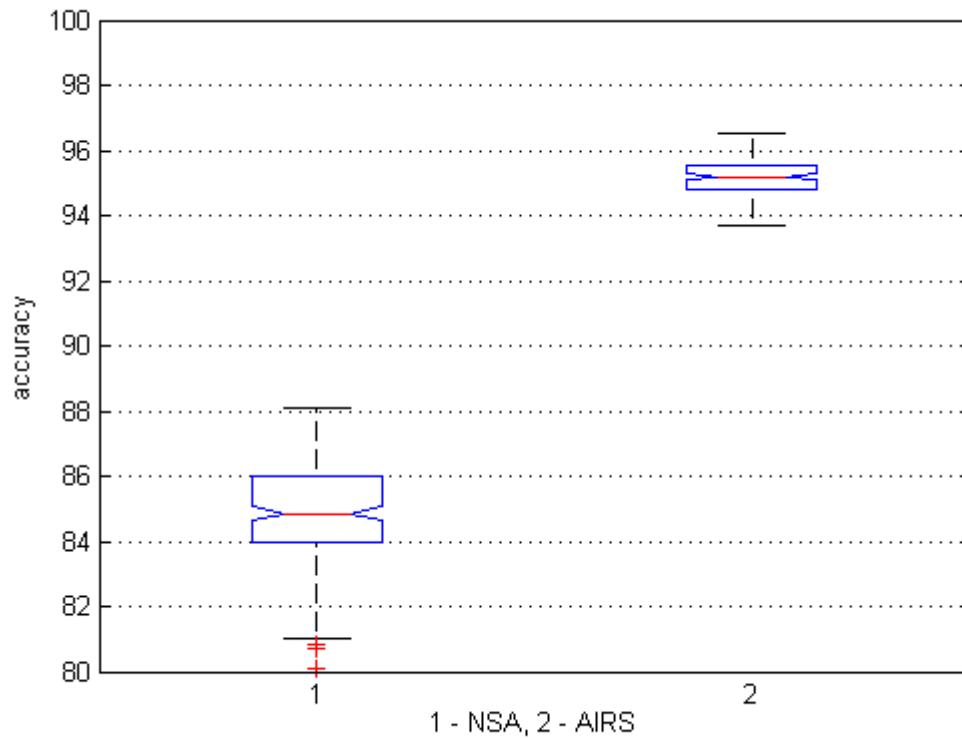
The following table (tab. 2) shows the results of testing AIRS and NSA with parameters, which were optimised by a genetic algorithm

tab. 2 — Result of testing AIRS and NSA with optimised parameters

Algorithm/Dataset	mean acc. [%]	stimthresh	knn	maxres	m.factor	antibodies
AIRS/Iris	96.453	0.357	6	62	—	—
AIRS/Gauss	95.456	0.435	7	160	—	—
NSA/Gauss	85.15	—	—	—	1.481	254

## 4.5 Comparison

Parameter settings obtained from GA (4.4) were used for final comparison (fig. 17)



*fig. 17 — Comparison between AIRS and NSA (gauss)*

In any test performed on both AIRS and NSA, the worst results of AIRS were always better than the best results of NSA.

## 5 Biomedical data

In medical facilities, there are databases, which are maintained by facility personnel, such as doctors and nurses. These databases contain various texts, which may be medicine prescriptions, patient diagnosis, birth progress, applied treatment etc. These texts contain vast amount of typos, because they are often typed in hurry, and for sure, they are not in the form, that would make data mining easy, although there is a need of extracting certain information from these sources.

### 5.1 Data character

Provided datasets contain large amount of *natural language strings*, where most of the words are medical terms, not commonly used in general verbal communication, such as specification of diagnose or applied treatment.

The dataset records have no classes attached to them, so there is no chance to apply conventional classification methods on the raw data. In addition, there are many artefacts in the strings, such as multiple times repeated space, case inhomogeneity and diacritic.

### 5.2 Classification strategy

For there are no classes corresponding to the strings in our dataset, we must construct them artificially, so they will show a direct linkage to the data. We would like to analyse the text and separate the words, that carry the biggest information value. There are several methods to extract these:

#### Cluster analysis

Cluster analysis separates the data to several groups, in which instances form *clusters* (they are near by one another). Illustrative method of cluster analysis is the *minimum spanning tree method*. This method creates a minimum spanning tree in the data graph (spanning tree is such factor of a graph, that is a tree, and minimum spanning tree is a spanning tree with minimal cost among all possible spanning trees) and removes  $n$  longest edges from it, thereby separating the data graph into a forest consisting of  $n + 1$  components. Every component then represents a data cluster, that could be used to assign a class to the instances in it. There are several algorithms designed to solve this problem—namely Jarník-Prim's algorithm<sup>[15]</sup>, Kruskal's algorithm<sup>[16]</sup> and Borůvka's algorithm<sup>[17]</sup>. Basic version of Jarník-Prim's algorithm, which was used for minimum spanning tree generation in this thesis, uses *vertex adjacency matrix* on a complete graph. Time complexity of this method is  $O(n^2)$ , where  $n$  is edge count. This method was rejected—its complexity makes it unusable for very large datasets.

#### Word frequency analysis

Frequency analysis computes a frequency of every word in a dataset and returns a complete dataset *histogram*. Based on the histogram, we can select several words with highest

frequencies and declare them important (of course we must exclude high frequency natural language words, which are unimportant, such as prepositions). This can be done in a linear time by using a *hash table*. A hash table is a one-dimensional associative array that uses a hash function to match a key to its corresponding value.

This method was accepted, because of its low complexity and good performance.

tab. 3 — Time complexity of MST and FA sample table

Samples	100	200	300	400	500	600	700	800	900	1000
MST Time[ms]	459	1716	3775	7044	10781	17796	20853	27295	34752	42794
FA Time[ms]	~ 0	~ 0	~ 0	1	1	2	2	2	3	3

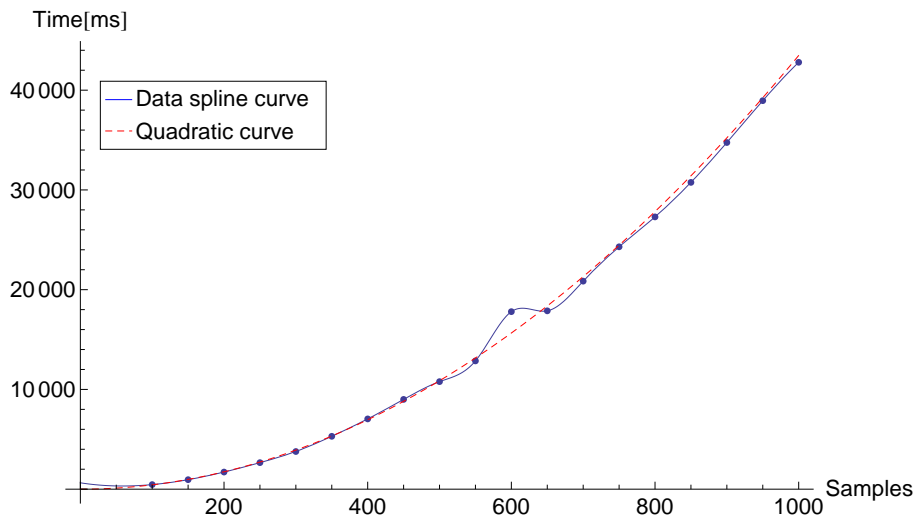


fig. 18 — MST time complexity

### 5.3 Distance measurement

There are many algorithms, that measure distance between strings. Each is suitable for a different field of use. To choose a metric, that satisfies our needs in this problem, we need to specify requirements for that metric. These are:

- Distance must be low between correctly typed and mistyped words
- Distance must be big between completely different words (in terms of their meaning)

Most widely known group of string metrics are *edit distances*. Edit distance of strings A and B is a number of edit operations needed to transform A into B. List of allowable edit operations differs with particular algorithms. Three most used edit distance algorithms are:

- Hamming distance

- Levenshtein distance<sup>[18]</sup>
- Damerau-Levenshtein distance<sup>[18]</sup>

## Hamming distance

Let  $A$  and  $B$  be strings of equal length, and let  $charAt(S, x)$  be a function mapping character indexes  $x$  of string  $S$  to their respective characters. Also let  $same(x, y)$  be a function that is 1 when characters  $x$  and  $y$  are not the same and 0 when they are the same. Then Hamming distance  $d_{ham} = \sum_{x=0}^{length(A)} same(charAt(A, x), charAt(B, x))$ . Simply, it counts indexes, at which two strings are different. Algorithm has linear runtime.

## Levenshtein distance

Levenshtein distance of string  $A$  and  $B$  is a number of insertions, deletions and substitutions used to transform  $A$  into  $B$ . Algorithm has time complexity  $O(n * m)$ , where  $n$  and  $m$  are lengths of compared strings.

## Damerau-Levenshtein distance

Damerau-Levenshtein distance of string  $A$  and  $B$  is a number of insertions deletions, substitutions and transpositions used to transform  $A$  into  $B$ . Algorithm has time complexity  $O(n * m)$ , where  $n$  and  $m$  are lengths of compared strings.

Two most common words of the first of the provided dataset are “mesocain” and “epiduralni”. Dataset also contains these words with various typos, such as “mesocian”, “mescain” or “peiduralni”. Selected distance metrics show following results:

tab. 4 – Hamming distance test chart

	mesocain	mesocian	mescain	epiduralni	peiduralni
mesocain	0	2	5	10	9
mesocian		0	4	9	8
mescain			0	10	9
epiduralni				0	2
peiduralni					0

tab. 5 – Levenshtein distance test chart

	mesocain	mesocian	mescain	epiduralni	peiduralni
mesocain	0	2	1	8	7
mesocian		0	3	8	7
mescain			0	8	7
epiduralni				0	2
peiduralni					0

tab. 6 — Damerau-Levenshtein distance test chart

	mesocain	mesocian	mescain	epiduralni	peiduralni
mesocain	0	1	1	8	7
mesocian		0	2	8	7
mescain			0	8	7
epiduralni				0	1
peiduralni					0

Hamming distance seems to penalise totally different words very much, but it is completely insensitive to words, that are completely the same, but shifted. For that reason, it is inaccurate and unusable for our problem.

On the other side, Levenshtein distance takes shift as only one operation and therefore it is sensitive to words, that are the same, but shifted. It penalises completely different words less than Hamming distance, but the difference is not considerably big.

Without a question, the Damerau-Levenshtein distance performs the best among the metrics mentioned above. It shares the sensitivity of the Levenshtein distance and in addition, it can detect the most common kind of typos—character transpositions. Therefore the Damerau-Levenshtein is a potent candidate distance metric.

The distance, that these algorithms compute, is absolute and therefore unnormalised. For our purposes, we need normalised distance on range  $<0,1>$ . To scale an absolute distance, we need to determine maximal possible distance between two strings (which is called normalised affinity denominator in this thesis, because an absolute distance is divided by this number). If  $A$  is a string of length  $len(A)$  and  $B$  is a string of length  $len(B)$ , then we declare strings  $Ad$  and  $Bd$  of lengths  $len(A)$  and  $len(B)$ , respectively. We call these strings the dummy strings, when a string metric considers them 100% different. Normalised affinity denominator is then equal to the distance between them.

## The cosine similarity

The cosine similarity (see (6)) is a vector distance metric, that computes the cosine of an angle between two vectors. If the vectors point the same direction, angle between them is 0, therefore their cosine similarity is 1. In algebra, an angle between two vectors is their dot product divided by the product of their magnitudes. And because all strings are character arrays and therefore vectors, the cosine similarity can be also applied on them.

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n A_i^2} \times \sqrt{\sum_{i=1}^n B_i^2}} \quad (6)$$

## The product distance

The product distance is a distance, that was designed especially for the purposes of this thesis, because there was a need for a new metric, that would take into account several

factors described in the following paragraphs. It has so far the best performance when used for similar word identification during preprocessing and when used as a distance metric in AIRS and NSA algorithms. It is basically a Damerau-Levenshtein distance corrected by a cosine similarity coefficient and a cosine correction coefficient. Pseudocode follows.

```

Procedure Proddist
Input: String A, String B
Output: product_distance

List X = SPLIT(A," "), List Y = SPLIT(B, " ");
For(i:=0,i<Length(Y), i++)
  If(Y[i] is element of X) ...
  and If(Y[i] is a part of another word)
    x_length+=Length(Y[i])
For(i:=0, i<Length(X), i++)
  If(X[i] is element of Y) ...
  and If(X[i] is a part of another word)
    y_length+=Length(X[i])
occurrence_ratio_x = x_length/length(A);
occurrence_ratio_y = y_length/length(B);
occurrence_ratio = max(occurrence_ratio_x,...
  occurrence_ratio_y) or 0 if (0,0);
cosine_correction = 1 - occurrence_ratio;
cs = cosineSimilarity(X,Y);
product_distance = cs * cosine_correction * normDLS(A,B);

```

The goal of this distance metric was primarily to detect typos which originated from not typing a space between words, like “some word” and “someword”. According to Damerau-Levenshtein metric, the distance between these words is 0.11, but according to product distance, it is 0. This is a good property when comparing sentences, but it also proved to be good when comparing words—it detects prefixes and suffixes and provides a distance bonus in the form of cosine correction coefficient. For example “mesocain” and “mesocainu” are 0.11 units distant according to Dam.-Lev., but only 0.012 units distant according to product distance, because “mesocain” (length 8) is present as a separate word in the first string and as a part of a word “mesocainu” (length 8) in the second string, thus the correction is  $(1-(8/9)) = 0.11$  (11 % of the original distance).

## 5.4 Word modifiers encoding

Because there are many quantifiers and qualifiers in the dataset, that bind to certain words, it would be an error to do a frequency analysis before connecting modifiers to the words, they are bound to. We read a data from a configuration file to determine, which words are units, which words are negative modifiers and of which type they are (prefix—appearing before a word, or postfix—appearing after a word). The configuration file contains the following information:

- List of units (for instance: ml, mg, %)
- List of words bound to units (for instance: drugs)
- List of prefix negations (for instance: bez<sub>[CZE]</sub> (no<sub>[GBE]</sub>))
- List of postfix negations (for instance: nebyla<sub>[CZE]</sub> (unused<sub>[GBE]</sub>))
- Default binding

When a unit is found in a string, the algorithm searches for a word bound to it one step to the left and one step to the right of the word. If such word (or a very similar word, based on Damerau-Levenshtein distance) exists, the unit is connected to it using this convention: “boundword(!10mg)”. Normalised distance is optimised to penalise words containing “(!”, so “boundword(!10mg)” and “boundword” will be considered distant. This significantly increases sensitivity. If such word does not exist, quantity is bound to the first left word when default binding is set to left and to the first right word when default binding is set to right. When there are no left words, the quantity is bound to the first right word and vice versa.

When a negative modifier is found, it is bound to the left or to the right according to its type and connected to it using this convention: “boundword(!NEG)”. Normalised distance is again optimised to penalise words containing “(!NEG)”. Example follows.

Before:

```
celk anestezie 1% mesocain
bez xylocainu
```

After:

```
celk anestezie mesocain(!1%)
xylocainu(!NEG)
```

## 5.5 Similar word identification and frequency merging

A result of a frequency analysis is a list of String-Double pairs. Strings in these pairs represent words and doubles represent their frequencies. Properly typed words and their typos are counted separately, so we need to merge them to get their real frequencies.

In this process, the list is iterated from its start. Every word is then compared to all unprocessed words using the product distance and if they are similar enough (i.e. distance is lower than a preset threshold), their frequencies are added and they are declared one word. Example follows.

Before:

```
kratkodoba celkova anestezie u man lyze xylocain
mesocain(!1%) xylocain epiduralni analgezie
xyloxain spray
epiduralni analgezie pri porodu meoscain(!1%)
mezocain(!1%) epidural
```

After:

```
epiduralni, 3
```



```
mesocain(!1%), 3
analgezie, 2
xylocain, 2
spray, 1
...
u, 1
```

## 5.6 Unique wordlist

Because there are many duplicate words in the String-Double pair list (result of a frequency analysis), they are removed by converting the list into a set and back to the list. The overridden method `equals()` of String-Double pair compares the equality of the string part and therefore the inner Java framework routines causes duplicate records to be disposed when converting list to set.

## 5.7 Unimportant data removal

The configuration file contains list of words, which are unimportant for the classification (such as prepositions). In this process they are all removed. In addition, words, which have lower normalised frequency than the dispose rate (a preset parameter) are removed too.

Before:

```
epiduralni, 3
mesocain(!1%), 3
analgezie, 2
xylocain, 2
spray, 1
...
u, 1
```

After:

```
epiduralni, 3
mesocain(!1%), 3
analgezie, 2
xylocain, 2
//– for example, dispose rate creates dividing line here
spray, 1 // <- removed for having lower rate than the dispose rate
u, 1 // <- removed for being a preposition
```

## 5.8 Class generation

After all the previous processes has taken place, the list of important words contains words, which will be declared important. A string of the same length as the length of the wordlist is created. Every character of this class string will be either 0 or 1 if a word from wordlist, that is assigned to this character's index, is absent or present in the dataset

---

record, respectively. For instance, if the words are A B C D, then following records will have following class assignment:

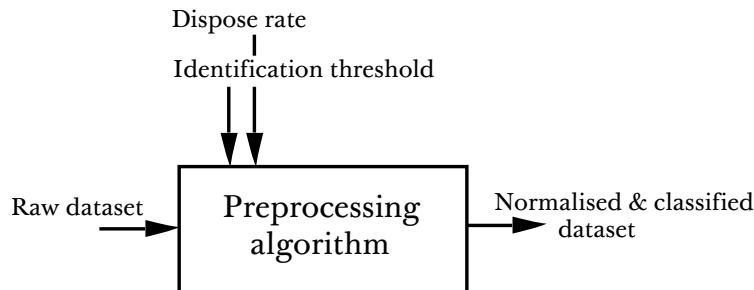
(A B C D) -> (1 1 1 1)  
(A H H C) -> (1 0 1 0)  
(F G) -> (0 0 0 0)

Then every record in a dataset is processed based on absence and presence of important words (with a given dissimilarity tolerance) and it is assigned a class.

The Class Identifier object, which is able to translate class strings to their meanings and vice versa is then created and stored, because it will serve as a necessary class information for classifiers.

## 5.9 Class identifier object

By preprocessing and identifying the records, we gain a class identifier object, which is provided to the classification algorithm. It contains the list of all the important words and a method, which assigns a bit string to corresponding words, therefore the class identifier object creates a link between the class and the data and it is also able to translate a class string to an appropriate data string and vice versa.



*fig. 19 – Preprocess algorithm scheme*

The algorithm takes  $Dispose\ rate \in \mathbb{R}$  (see 5.7) and  $Identification\ threshold \in \mathbb{R}$  (see 5.5) as its parameters.

# 6 AIRS for biomedical data

## 6.1 Classification strategy

### The binary counter and sentence based distance approach

The first tested approach was naïve. It assumed, that if there is a class string, that consists of  $k$  independent values, each having  $n$  possible values, there is  $n^k$  total possible classes in the system, although, the vast majority of them is unused. Therefore, there was an exponential relation between count of the important words and the train/test runtime.

In this approach, the memory cell pool was initialised with one cell for every class. For instance, with 14 important words in a dataset of 1000 records, the cell pool was initialised with 16384 cells which apparently resulted in poor data reduction rate. Exponential runtime also made algorithm impossible to finish (for instance, even 70 detected important words would make the algorithm run ten times longer than the age of the universe).

Beside this, the classification method was inaccurate. The longer the record was, the worse was the distance resolution and the worse was the ability to detect minor dissimilarities in the strings.

### The 1-of-k counter and word based distance approach

The second tested (and actual) approach significantly improves the results. It assumes, that if there is a class string, that consists of  $k$  independent values, each having  $n$  possible values, then we can think of the value as of a vector. For instance, the string 0001000 is a vector in  $\mathbb{Z}_2^7 : (0, 0, 0, 1, 0, 0, 0)$ . We can define a linear space  $V$  of binary vectors of length  $k$  and search for a basis of that space, which gives us a set of vectors, whose linear hull is again a vector space  $V$ , thus any possible vector of  $V$  can be created by a linear combination of the basis vectors<sup>[23]</sup>. The basis of such space is:

$$\begin{aligned} & \dots 1\ 0\ 0 \dots \\ & \dots 0\ 1\ 0 \dots \\ & \dots 0\ 0\ 1 \dots \end{aligned}$$

Due to this assumption, the classifier is only able to identify single words, but from the words it is able to construct a sentence. It can split the test sentence into words and then classify them separately. Using a linear combination it can construct the final class in following manner:

$$V_R = c_1 \odot (V_1) \oplus c_2 \odot (V_2) \oplus \dots \oplus c_k \odot (V_k) \quad (7)$$

where  $c_i$  is 0 if word  $i$  is classified as not present and 1 if classified as present, operator  $\odot$  is a logic multiplication and operator  $\oplus$  is a logic addition (in other domains, these operators will have equivalent functions (multiplication and addition)). For instance,

let important words be A, B, C and D and the test sentence “A E F C G”. First of all, we split the sentence and classify each word separately:

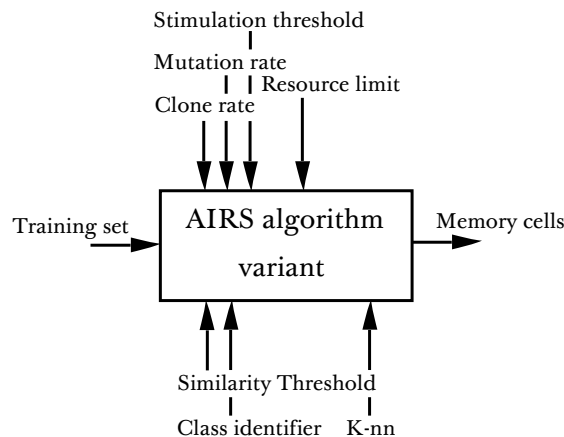
- A - matched word 1,  $c_1=1$
- E - no match
- F - no match
- C - matched word 3,  $c_3=1$
- G - no match

$$CLASS = 1 \odot (1000) \oplus 0 \odot (0100) \oplus 1 \odot (0010) \oplus 0 \odot (0001) = 1010 \quad (8)$$

Hence, AIRS only needs  $k$  cells to initialise pool. This is a proof, that this approach is far more efficient than the sentence based one (which needs  $2^k$ ).

## 6.2 Algorithm variation

The algorithm itself remains basically the same as the original one except for some minor changes. Unlike the original algorithm, this variation needs two additional parameres—similarity threshold and class identifier object. During initialisation, the algorithm fills the memory pool with the initial antibodies specified in 2.1.1 and then makes a copy of the memory pool.



*fig. 20 – AIRS algorithm variant scheme*

During the training, every record is split into words and then every word is compared with the words in the initial pool (the copy, which I mentioned in the paragraph above). If the lowest measured distance is lower than or equal to the similarity threshold, then the particular word is processed by the training algorithm of AIRS (described in 3.1).

After the training phase is complete, the memory cell pool is filled with the important words and their variations, which were deemed important by the algorithm. This process significantly increases the probability of matching mistyped words to their respective classes.

During the testing, every record is split into words and then every word is classified

separately. If the lowest measured distance is greater than the similarity threshold, then the particular word is disposed. If the distance is lower than or equal to the similarity threshold, then the class label of the best-matching antibody is added with the final class using a logic OR function.

## 6.3 Testing

During the testing, several dependencies were observed between stimulation threshold, knn, similarity threshold (independent variables) and accuracy, convergency fault rate, memory cell count and time complexity (dependent variables). The *Convergency fault rate* is a new parameter, that indicates how many ARB pools weren't able to mutate and achieve given mean stimulation after 100 mutation cycles. It happens because the mutation function is not parametric. *Memory cell count* indicates, how many memory cells were trained and used and it is de facto the memory complexity.

### Test methodology

Following tests were run on the first 1000 samples of the simpler of the two datasets. Every time a value of some parameter was tested, the dataset was randomized and split in half. One half was declared the train set and the second one the test set, and the algorithm was run on these. This process was repeated 10 times and the final result was declared an average of the 10 values. Preprocess parameters were: *id.threshold*=0.330, *dispose rate*=0.025, profile used: *lesscplx.profile* (see Appendix D).

### Test results

During testing an individual parameter, other parameters were fixed at one value. In fig. 21, we can see relative steadiness of accuracy throughout all measured stimulation threshold values. While at values greater than 0.7, the convergency fault rate starts to grow, indicating that with given mutation function, reaching given stimulation threshold is getting more difficult.

In fig. 22, we can see the actual memory complexity function with stimulation threshold as its parameter. The higher the stimulation threshold is, the more memory cells will be created.

In fig. 23, we can see the actual time complexity function with stimulation threshold as its parameter. When there is more iterations of mutation needed to reach the stimulation threshold, the more time the algorithm will consume.

In figs. 24–26, we can see, that the algorithm could be the most accurate somewhere between 0.3 and 0.35 (of similarity threshold), but it cannot be said for sure, because we do not know, if there are or are not multiple relations with the arguments, which were fixed during the test. Nevertheless, that higher values of similarity threshold result in bigger convergency fault rate and excessive memory and time complexity.

In fig. 27 and 28, we can see, that the most accurate value of knn could be 1. Values bigger than 1 could lead to enormous decrease of performance.

The suboptimal parameters of this algorithm for this dataset are computed using a genetic algorithm in further sections of this chapter (Optimisation of input parameter values.)

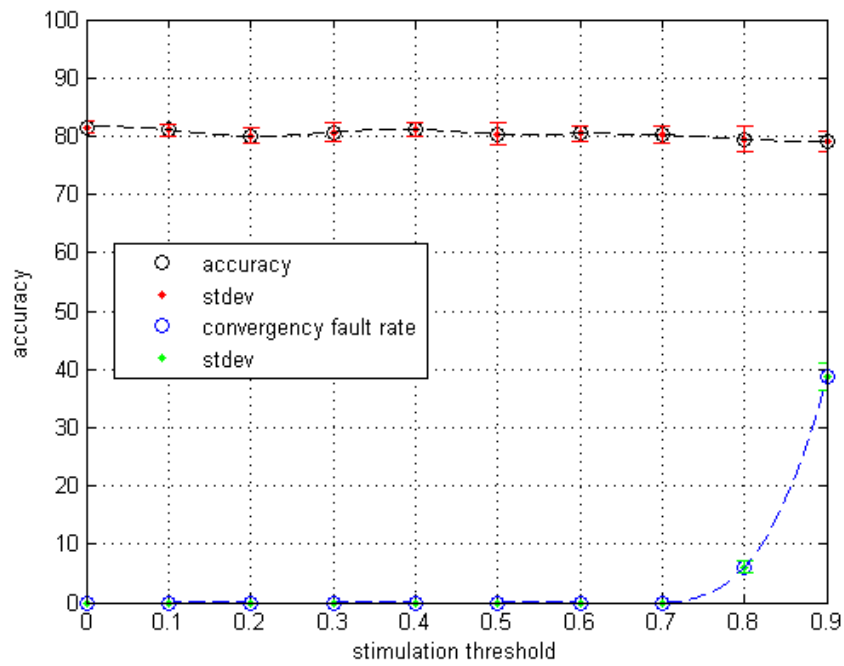


fig. 21 — Dependency of accuracy on stimulation threshold

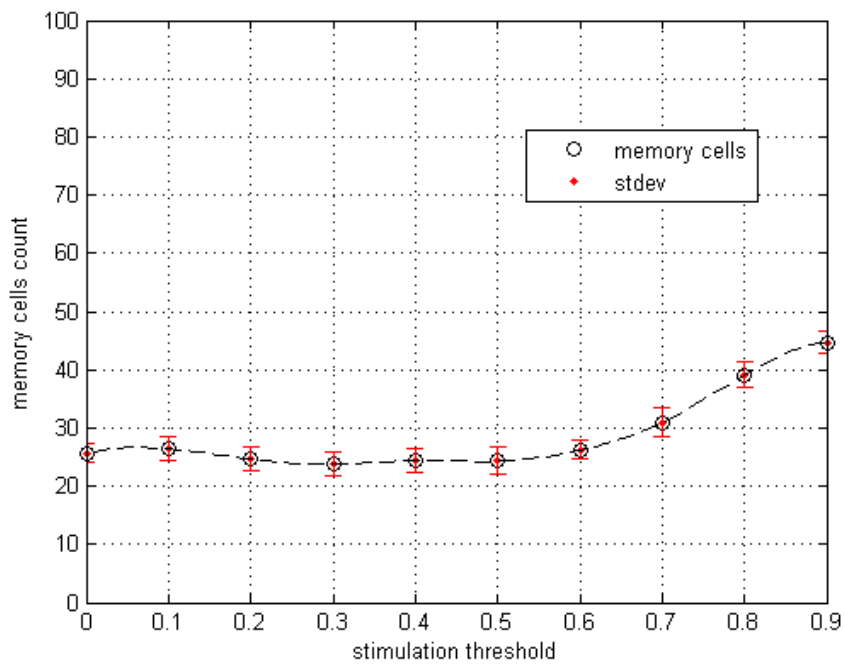


fig. 22 — Dependency of memory cell count on stimulation threshold

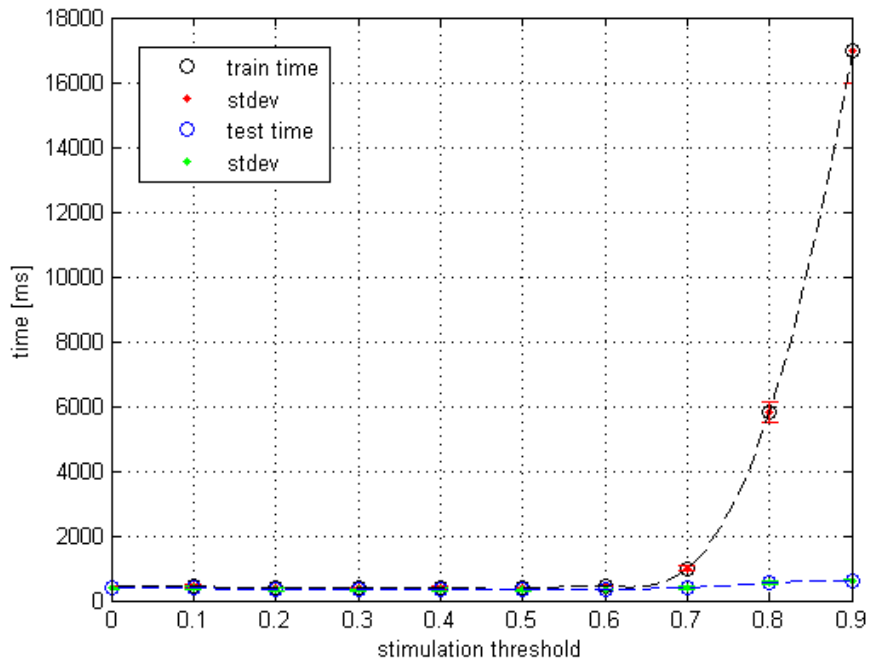


fig. 23 — Dependency of runtime on stimulation threshold

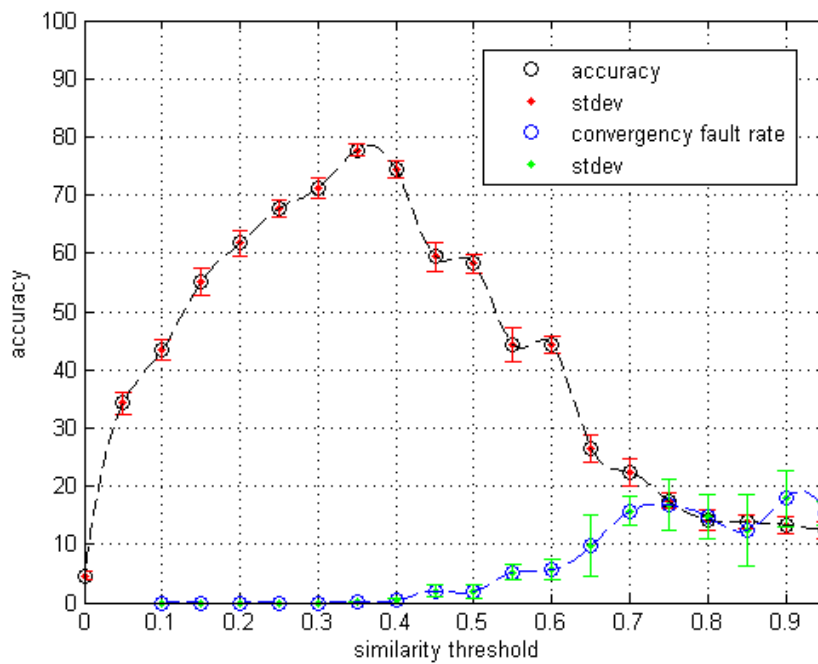


fig. 24 — Dependency of accuracy on similarity threshold

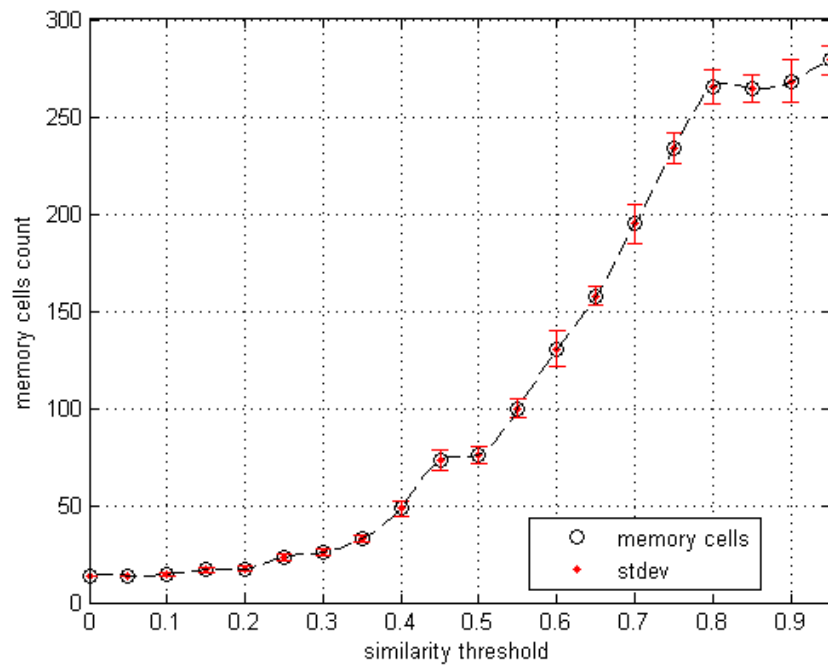


fig. 25 — Dependency of memory cell count on similarity threshold

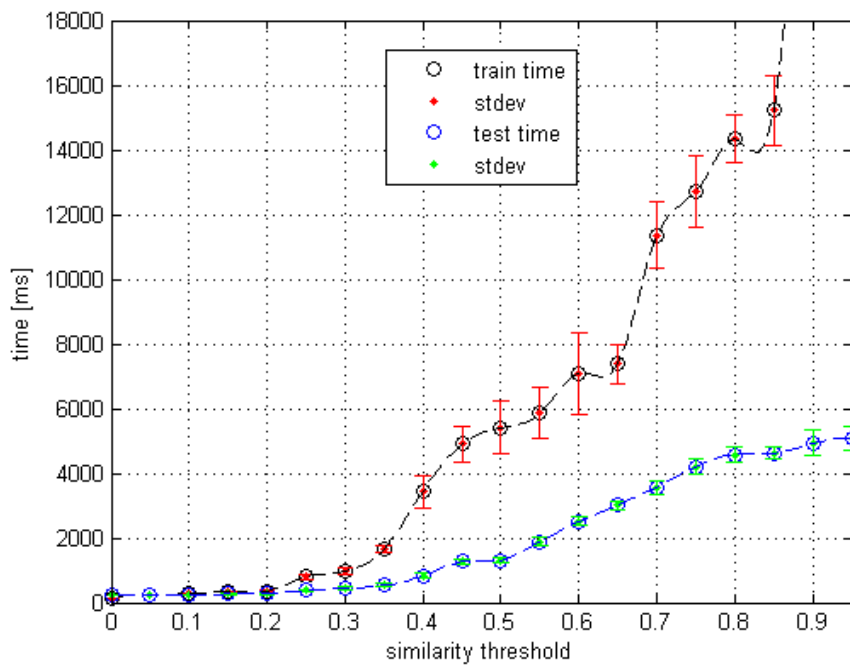


fig. 26 — Dependency of runtime on similarity threshold



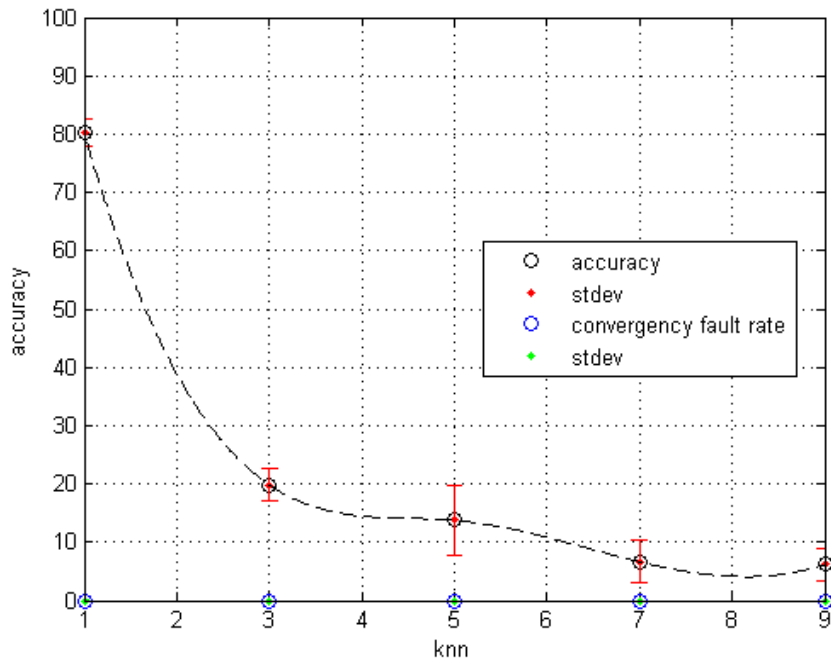


fig. 27 — Dependency of accuracy on knn

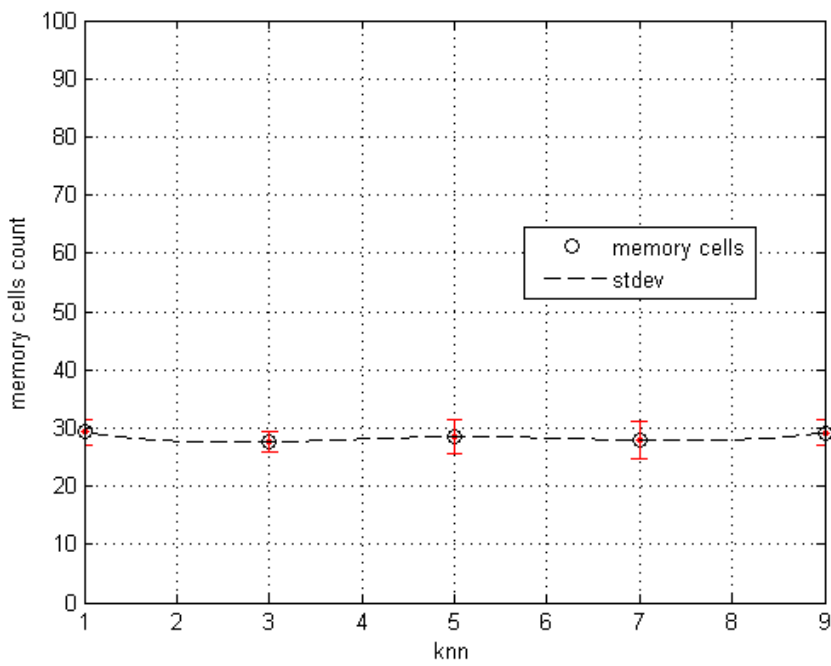


fig. 28 — Dependency of memory cell count on knn

## Real classification performance

The real classification performance is better than the accuracy shown in the above diagrams could imply, because the classes of the strings are artificially created and are not precise nor real. As it can be seen in the following listing, the Preprocessing algorithm made many mistakes, which were repaired by AIRS and therefore the classes don't match theoretically, but it is de facto a good classification.

```
mesocain(!1%): 530.0
epiduralni: 441.0
lok: 278.0
amp: 236.0
analgezie: 221.0
celkova: 134.0
xylocain: 113.0
mesocainu: 110.0
anestezie: 99.0
l: 68.0
loc: 66.0
amp(!1%): 62.0
spray: 50.0
1(!10ml): 34.0
There are 14 significant words.

AIRS
DATASET: C:\biodata\mensi.data, 1 dimensions.
TRAINING SET is 1/2 of DATASET
INCORRECT: 1 amp mesocain != amp mesocainu SHOULD BE mesocain(!1%) amp mesocainu
CORRECT: epifuralni analgezie = epiduralni analgezie
CORRECT: 1 amp mesocain(!1%) = mesocain(!1%) amp
CORRECT: mesocain(!1%) 1 amp 2x = mesocain(!1%) amp
INCORRECT: mesocain(!1%) epiduralni anealgezie != mesocain(!1%) epiduralni analgezie SHOULD BE mesocain(!1%) epiduralni analgezie anestezie
CORRECT: mezokain(!1%) = mesocain(!1%)
CORRECT: epiduralni analgezie dr robotkova = epiduralni analgezie
INCORRECT: 1 % mesocain 1 amp lok != lok amp mesocainu 1 SHOULD BE mesocain(!1%) lok amp mesocainu 1
CORRECT: epid analg mesocain(!1%) = mesocain(!1%) analgezie
CORRECT: remifentanyl mesocain(!1%) = mesocain(!1%)
CORRECT: amp(!1%) msocain lokalne = lok mesocainu amp(!1%)
CORRECT: mesocain(!1%) amp = mesocain(!1%) amp
CORRECT: epiduralni analgezie xylocain = epiduralni analgezie xylocain
INCORRECT: xylocai spray != xylocain spray SHOULD BE xylocain loc spray
INCORRECT: 1amp mesocainu(!1%) != mesocain(!1%) amp SHOULD BE mesocain(!1%) amp mesocainu
CORRECT: epiduralni lokalni mesocaine = epiduralni lok mesocainu
CORRECT: u porodu epiduralni celkova u s c = epiduralni celkova
CORRECT: 1 amp mem i v = amp
CORRECT: 1 amp(!1%) mesoain = mesocainu amp(!1%)
CORRECT: lok mesokain = lok mesocainu
INCORRECT: lokalni lo % xylocain != lok xylocain loc SHOULD BE lok xylocain 1 loc
CORRECT: mezocain(!1%) epiduralni = mesocain(!1%) epiduralni
INCORRECT: epiduralni analgezie mesocain 1 amp != epiduralni amp analgezie mesocainu SHOULD BE mesocain(!1%) epiduralni amp analgezie
mesocainu
CORRECT: 1amp(!1%) mesokain = mesocainu amp(!1%)
CORRECT: celkova revize dd po porodu = celkova
INCORRECT: epiduralni analgesie 1 % mesocain != epiduralni analgezie mesocainu 1 SHOULD BE mesocain(!1%) epiduralni analgezie mesocainu 1
CORRECT: celkova manualni lyze a revize = celkova
CORRECT: celkova anestezie = celkova anestezie
CORRECT: revize hrdla delozniho v zrcadlech sine vulnere =
CORRECT: lokalni 1 % mesocain = lok mesocainu 1
INCORRECT: mesocain lok(!1%) != mesocainu SHOULD BE mesocain(!1%) mesocainu
CORRECT: epiduralni analgezie 1 amp mesocain(!1%) = mesocain(!1%) epiduralni amp analgezie
CORRECT: % mezokain 1 amp = amp
INCORRECT: epiduralni xylocain(!1%) mesocain != mesocain(!1%) epiduralni mesocainu SHOULD BE mesocain(!1%) epiduralni xylocain mesocainu
CORRECT: lokalni mesocain(!1%) epiduralni = mesocain(!1%) epiduralni lok
CORRECT: 1(!10ml) % mesocainu xylokain = xylocain mesocainu 1(!10ml)
CORRECT: epiduralni loc(!10ml) (!1%) mesocainu loc = epiduralni mesocainu loc amp(!1%) 1(!10ml)
CORRECT: traumacel pulv =
CORRECT: mesocain(!1%) epiduralni analgesie dr mala = mesocain(!1%) epiduralni analgezie
CORRECT: mesocain(!1%) 2 amp i v = mesocain(!1%) amp
CORRECT: mesocain(!1%) dolsin(!50mg) i v = mesocain(!1%)
CORRECT: 1 amp(!10%) mesocainu = mesocainu amp(!1%)
CORRECT: epiduralni alalgezie = epiduralni analgezie
INCORRECT: 1(!10ml) % mesocain lokalne != lok mesocainu 1(!10ml) SHOULD BE mesocain(!1%) lok mesocainu 1(!10ml)
INCORRECT: epiduralni a lokalni 1 % mesocain != epiduralni lok mesocainu 1 SHOULD BE mesocain(!1%) epiduralni lok mesocainu 1
CORRECT: mesocain(!1%) 1 amp lokalne = mesocain(!1%) lok amp
```

---

```

CORRECT: mezocain(!1%) loc = mesocain(!1%) loc
CORRECT: epiduralni xylocaine spray = epiduralni xylocain spray
CORRECT: epiduralni analgezie = epiduralni analgezie
CORRECT: eda lokalne xylocaine = lok xylocain
CORRECT: lokalni(!1%) mesocain 1amp = amp mesocainu
CORRECT: 1 amp mesocain(!1%) xylocain sprej = mesocain(!1%) amp xylocain
CORRECT: lokalni mesocaine xylocaine spray = lok xylocain mesocainu spray
CORRECT: analgesia epiduralis ca = epiduralni analgezie
CORRECT: 1 amp mesocaun(!1%) = mesocain(!1%) amp
CORRECT: 2 amp mesocain(!1%) = mesocain(!1%) amp
CORRECT: mesocain(!1%) loc = mesocain(!1%) loc
CORRECT: epiduralni dr stoudek = epiduralni
CORRECT: epiduralni analgezie lokalne mesocain(!1%) = mesocain(!1%) epiduralni lok analgezie
INCORRECT: mesocain 1 % lok != lok mesocainu 1 SHOULD BE mesocain(!1%) lok mesocainu 1
CORRECT: mesocain(!1%) xylocaine = mesocain(!1%) xylocain
CORRECT: mesocain(!1%) epiduralni analg dr slezak = mesocain(!1%) epiduralni analgezie
INCORRECT: xylocain(!1%) != mesocain(!1%) SHOULD BE mesocain(!1%) xylocain
CORRECT: epidur analgesie celkova = epiduralni analgezie celkova
CORRECT: celkova dr krikava = celkova
CORRECT: epiduralni celkova = epiduralni celkova
INCORRECT: mesocain(!1%) 1 amp xylocain(!10%) != mesocain(!1%) amp SHOULD BE mesocain(!1%) amp xylocain
CORRECT: epiduralni u poodu dr gbelcova mesocain(!1%) = mesocain(!1%) epiduralni
CORRECT: epiduralni analgezie kratkodoba celkova pri ml = epiduralni analgezie celkova 1
CORRECT: epiduralni an(!10ml) mesocain(!1%) local = mesocain(!1%) epiduralni loc 1(!10ml)
INCORRECT: 1 % mesocain != mesocainu 1 SHOULD BE mesocain(!1%) mesocainu 1
INCORRECT: mesocain amp local != amp mesocainu loc SHOULD BE mesocain(!1%) amp mesocainu loc
INCORRECT: lokalne 1 % mesocain != lok mesocainu 1 SHOULD BE mesocain(!1%) lok mesocainu 1
CORRECT: epiduralni u s c v celkove = epiduralni celkova
INCORRECT: epiduralni analgezie 1 amp mesocain lok != epiduralni lok amp analgezie mesocainu SHOULD BE mesocain(!1%) epiduralni lok amp analgezie mesocainu
CORRECT: epiduralni analgesie dr zborilova mesocain(!1%) = mesocain(!1%) epiduralni analgezie
CORRECT: eiduralni analgezie i anestezie = epiduralni analgezie anestezie
CORRECT: mesocain(!1%)(!30ml) = mesocain(!1%)
INCORRECT: mesocain amp != amp mesocainu SHOULD BE mesocain(!1%) amp mesocainu
CORRECT: epiduralni analg a cekova anest = epiduralni analgezie celkova anestezie
CORRECT: epiduralni analg % mezocain lok = epiduralni lok analgezie
CORRECT: epiduralni anaalgesie = epiduralni analgezie
INCORRECT: xyloxain(!10%) != mesocain(!1%) SHOULD BE
CORRECT: epidural celkova pri man lyze revizi = epiduralni celkova
INCORRECT: lokalni mesocain != lok mesocainu SHOULD BE mesocain(!1%) lok mesocainu
CORRECT: mezocain(!1%) epidural = mesocain(!1%) epiduralni
CORRECT: epiduralni lokalne mesocaine = epiduralni lok mesocainu
CORRECT: mesocain(!1%) = mesocain(!1%)
INCORRECT: epiduralni analgezie 1amp mesocain loc != epiduralni amp analgezie mesocainu loc SHOULD BE mesocain(!1%) epiduralni amp analgezie mesocainu loc
CORRECT: mesocain(!1%) za porodu epiduralni dr gbelcova = mesocain(!1%) epiduralni
CORRECT: xylocain = xylocain
CORRECT: spinalni an =
CORRECT: episiotomie suttura chirilac =
CORRECT: mesocain(!10%) 1 amp = mesocain(!1%) amp

```

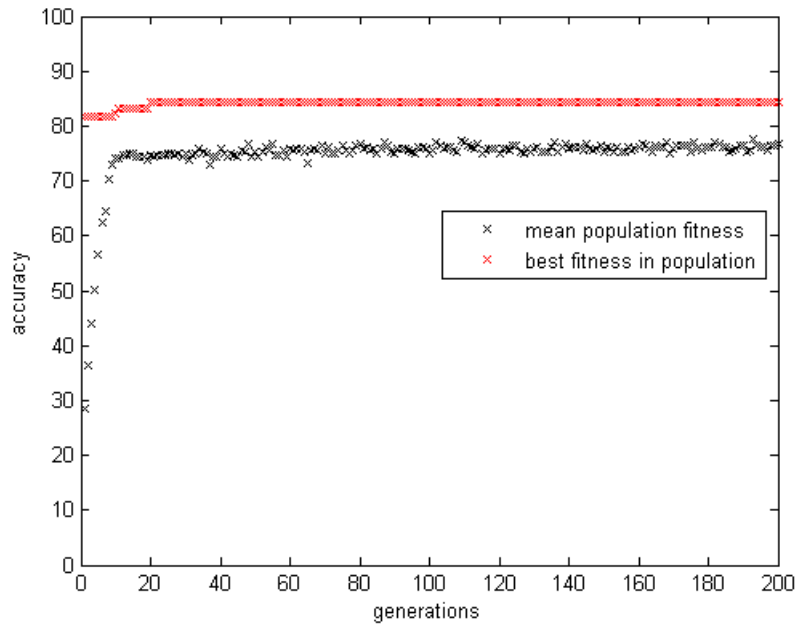
(In this listing, CORRECT: A=B means that, dataset record A and AIRS determined class B are in match. INCORRECT: A!=B SHOULD BE C means that dataset record A has artificially created “real” class C (translated to natural language using class identifier object) and not AIRS determined class B).

## Optimisation of input parameter values

As in 4.4, a simple genetic algorithm was used to determine suboptimal parameter values for this algorithm and dataset. Diagram in fig. 29 shows how mean population fitness and best fitness varied with number of generations simulated.

## Results

- AIRS on Less complex biomedical data: stimulation threshold=0.330, knn=1  
– Accuracy: mean **84.486 %** on 100 randomly selected 1:1 percentage splits



*fig. 29 — Dependency of fitness on number of generations (AIRS variant)*

## 7 NSA for biomedical data

While changes made in AIRS in order to make it able to classify biomedical data were more-or-less simple, such changes made in NSA were severe and they resulted in a new algorithm, that uses several instances of original NSA. All changes are described in the following chapters.

### 7.1 Definitions

#### Information source

The Information source<sup>[14]</sup> is a probability model of a device, which produces messages consisting of characters of a finite alphabet  $\Gamma$ . There are 3 basic types of information sources:

- a random variable (one character long message)
- a random vector ( $n$  characters long message)
- a random process (infinite character count)

#### Markov chain

The Markov chain<sup>[14]</sup> is a mathematical model of a finite automaton with vertex transition probabilities on its edges. Transition from state  $A$  to state  $B$  has a probability  $EDGE(A, B)$  where  $EDGE(x, y)$  is a function that maps vertex pairs to the value of their mutual edge.

#### Markov information source

The Markov information source<sup>[14]</sup> is a stationary Markov chain (a stochastic process). In this thesis, the Markov chain is defined on a finite alphabet, that consist of all pairs of english letters. The transition matrix is built by frequency analysing a *corpus text*, where empirical frequencies of transitions between letter pairs are converted to probabilities. (Thorough example is provided in A.5.)

### 7.2 Classification strategy

The NSA is a binary classifier, therefore it cannot be used directly. However,  $n$  classes can be simplified to just two classes—the self class in the first group and all the other classes in the second group. Then the algorithm can distinguish between one class and the others.

In order to make the algorithm be able to classify all the  $n$  classes, we must create separate antibody pools, each for one class. Each antibody pool must contain a *base set* of antibodies (one instance from every non-self class). If they had not have these base sets, the algorithm would not yield satisfying results no matter how good the antibody generation process would be. Each of the pools matches all classes but one. There are

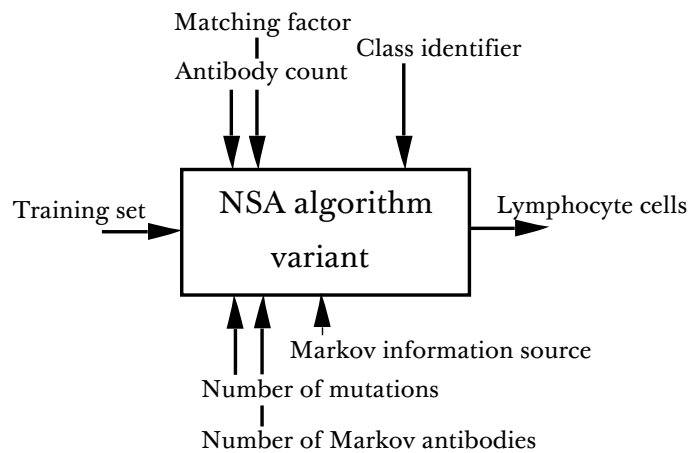
now several possible result of classification:

- 1) exactly one of  $n$  pools does not match an antigen
- 2)  $k$  of  $n$  pools does not match an antigen,  $k \neq n$
- 3)  $n$  of  $n$  pools does not match an antigen

If exactly one pool  $p_0$  does not match an antigen  $c_a$  (1), then it means, that every other pools  $p_k, k \neq 0$  had antibodies for the antigen  $c_a$ , so it must be an important word and if only  $p_0$  does not have antibodies for it, it means, that most probably the self class of  $p_0$  is the class of  $c_a$ .

If  $k$  of  $n$  pools do not match an antigen  $c_a$  (2), then it means, that there are  $n - k$  other pools, that matched the antigen  $c_a$ , so it probably is an important word, but it cannot be decided which one of  $k$  (it is similar to both).

If no pool matches an antigen  $c_a$  (3), then it means, that it is not an important word. These specialized antibody pools may (or may not) be improved by addition of more antibodies, generated by various mechanisms. One of the mechanisms is mutation, i.e. antibodies mutate and they are being checked against their self class (when they are too similar to their self class, they are disposed). The other mechanism which may improve performance is the addition of antibodies generated by a Markov information source. The advantage of such antibodies is that Markov chain generates antibodies, that have nearly the same probability distribution as the corpus text, so they will cover the important areas and not the unimportant areas, rather than pure random antibody generator, which tries to cover the whole non-self space.



*fig. 30 – NSA algorithm variant scheme*

As it can be seen in the above schematic (fig. 30), the algorithm takes several parameters, all being explained in the above paragraphs (chapter 7) and in chapter 3.3.

## 7.3 Testing

During testing, several dependencies were observed between NSA input parameters, accuracy and time complexity. The method used is the same as in 6.3.

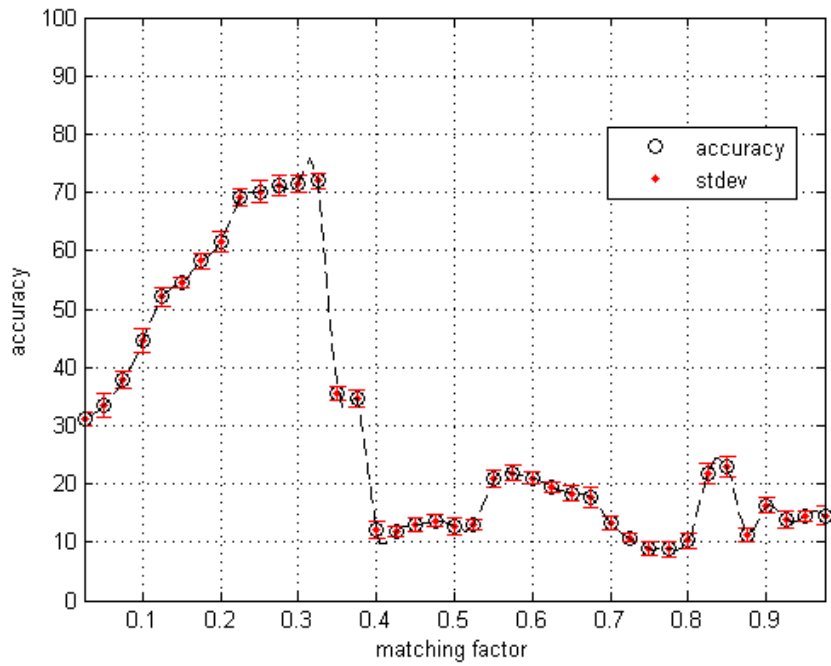


fig. 31 — Dependency of accuracy on matching factor (NSA variant)

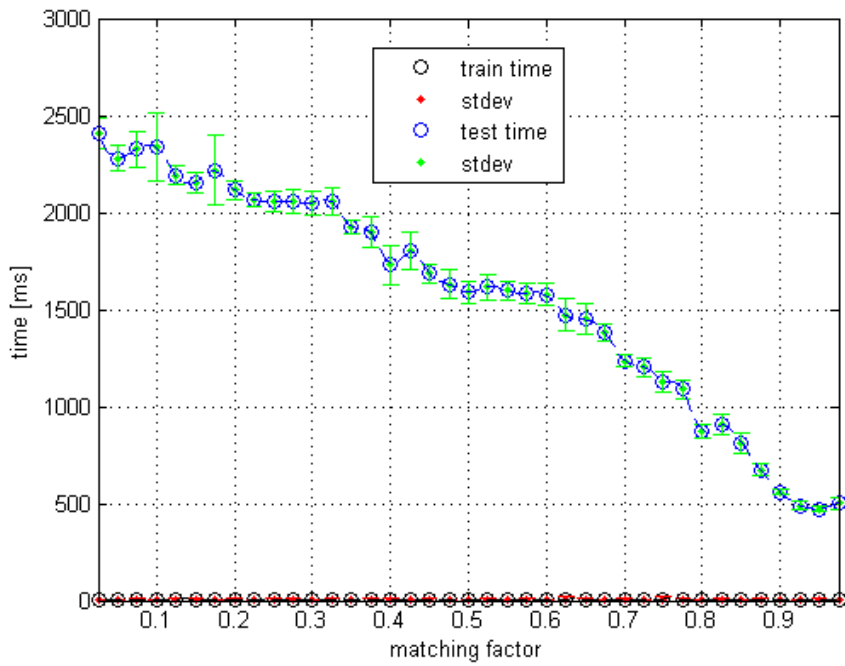


fig. 32 — Dependency of time complexity on matching factor (NSA variant)

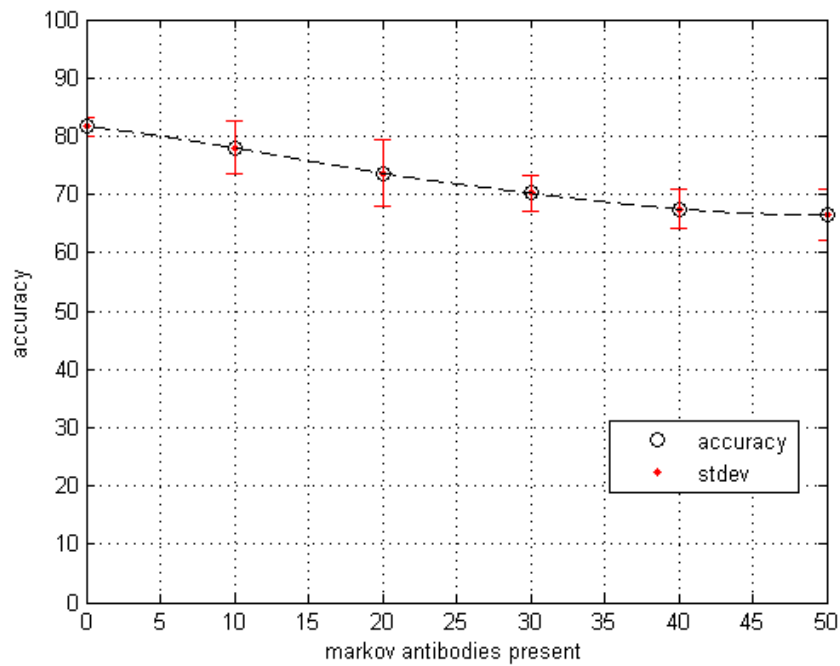


fig. 33 — Dependency of accuracy on Markov antibody count (NSA variant)

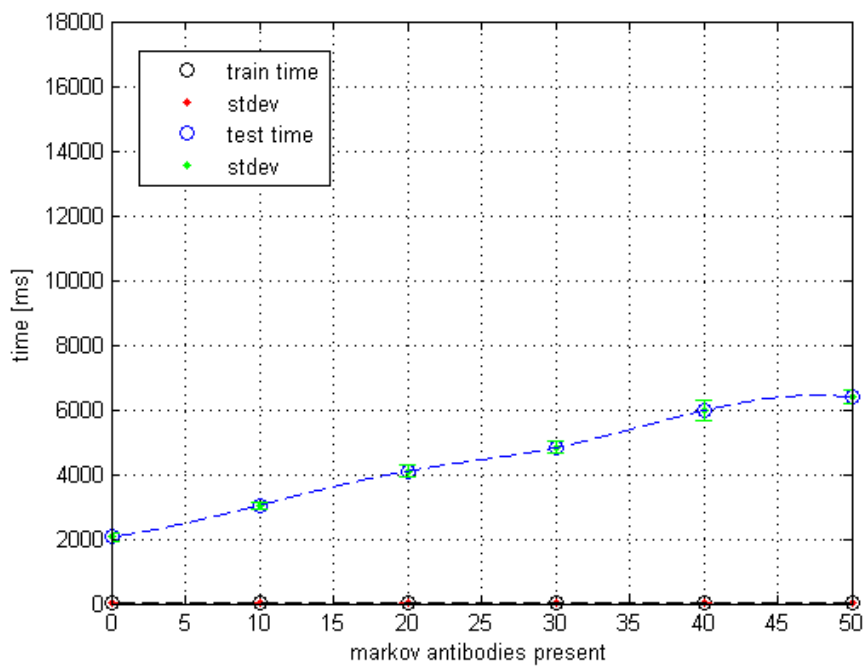


fig. 34 — Dependency of time complexity on Markov antibody count (NSA variant)



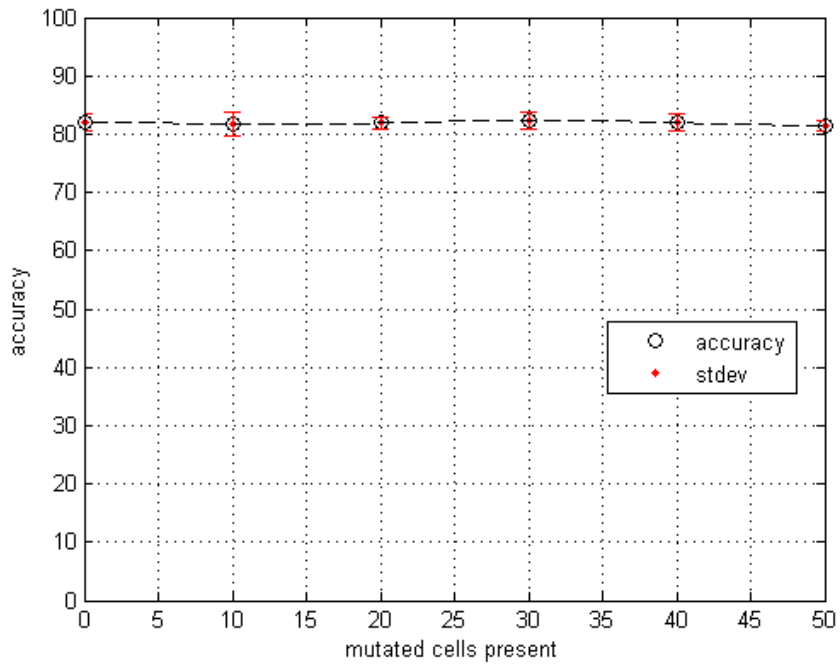


fig. 35 — Dependency of accuracy on mutated antibodies (NSA variant)

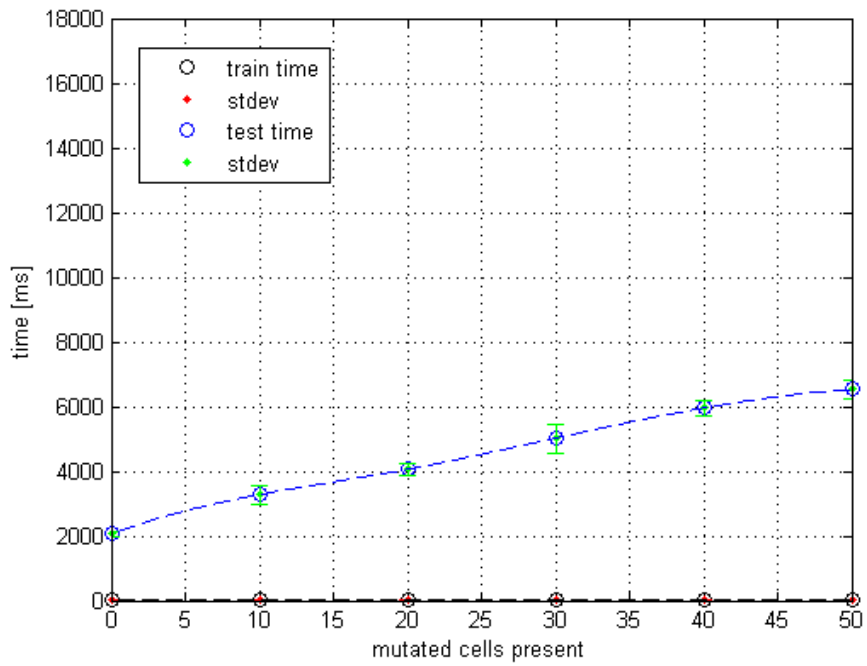


fig. 36 — Dependency of time complexity on mutated antibodies (NSA variant)

## Test results

In fig. 31, we can see how classification accuracy changes with matching factor. We can see a point around 0.3 (of matching factor), which could make algorithm yield the best results. In fig. 32 we can see, that grater matching factor causes shorter runtime.

In fig. 33 and 34, we can see, that addition of antibodies created by a Markov chain causes decrease in accuracy an increase in runtime, thus it should always stay at zero when used with this dataset.

In fig. 35 and 36, we can see, that addition of antibodies created by the process of mutation does not change accuracy, but only increases runtime, thus it should also stay at zero.

## Real classification performance

As in 6.3, the real classification performance is better than the accuracy shown above could imply for the same reason (artificially created classes are not precise nor real).

```
530.0
epiduralni: 441.0
lok: 278.0
amp: 236.0
analgezie: 221.0
celkova: 134.0
xylocain: 113.0
mesocainu: 110.0
anestezie: 99.0
l: 68.0
loc: 66.0
amp(!1%): 62.0
spray: 50.0
l(!10ml): 34.0
There are 14 significant words.

CORRECT: epiduralni analgezie i anestezie = epiduralni analgezie anestezie
CORRECT: mezokain(!1%) epiduralni analgezie = mesocain(!1%) epiduralni analgezie
INCORRECT: epiduralni analgezie mesocain 1 amp != epiduralni amp analgezie SHOULD BE mesocain(!1%) epiduralni amp analgezie mesocainu
CORRECT: mesocain(!10ml)(!1%) 2 amp = amp
CORRECT: epidural celkova u vykonu = epiduralni celkova
CORRECT: epiduralni analgesie anestezie = epiduralni analgezie anestezie
CORRECT: celkova kratkodoba anestezie manualni lyze = celkova anestezie
CORRECT: epidural analg(!1%) mezokain lok = epiduralni lok
CORRECT: xylocaine(!10%) lokalne = lok xylocain
CORRECT: mesocain(!1%) lokalne = mesocain(!1%) lok
CORRECT: epiduralni analgesie dr stoudek mesocain(!1%) = mesocain(!1%) epiduralni analgezie
CORRECT: celkova narkóza = celkova
CORRECT: epiduralni anagesie mesocain(!1%) = mesocain(!1%) epiduralni analgezie
CORRECT: epiduralni analg meosain(!1%) = mesocain(!1%) epiduralni analgezie
CORRECT: remifentanyl za porodu celkova anestezie u sc = celkova anestezie
CORRECT: epiduralni analgezie(!10ml) (!1%) mesocainu = epiduralni analgezie mesocainu amp(!1%)
CORRECT: %(!10ml) mesocainu i v = mesocainu 1(!10ml)
CORRECT: kratkodoba celkova anestezie u man lyze = celkova anestezie
CORRECT: xyloxain spray = xylocain spray
INCORRECT: epidural mesocain l % != epiduralni l SHOULD BE mesocain(!1%) epiduralni mesocainu l
CORRECT: 1 amp(!1%) mesocainu lokalne = lok mesocainu amp(!1%)
CORRECT: celkova anestezie rcui = celkova anestezie
CORRECT: epiduralni 1 meosain = epiduralni
CORRECT: mesocain(!1%) 1amp xylocaine = mesocain(!1%) amp xylocain
CORRECT: epiduralni anagezie = epiduralni analgezie
CORRECT: celkova u vykonu = celkova
CORRECT: xylocaine spray lok = lok xylocain spray
CORRECT: epidural(!10ml) (!1%) mesocainu xylocain spray = xylocain mesocainu amp(!1%) spray
CORRECT: lokalni 2 amp mesocain(!1%) = mesocain(!1%) lok amp
CORRECT: 1(!10ml) % mesocainu xylokain = xylocain mesocainu 1(!10ml)
INCORRECT: xylocai spray != spray SHOULD BE xylocain loc spray
CORRECT: epiduralni dr frncikova = epiduralni
CORRECT: mesocain(!1%) lokalne(!10ml) = mesocain(!1%)
INCORRECT: lokalni mesocain 1amp(!1%) != lok amp(!1%) SHOULD BE mesocain(!1%) lok mesocainu amp(!1%)
```

```

INCORRECT: epiduralni mesocain 1 amp(!1%) != epiduralni amp(!1%) SHOULD BE mesocain(!1%) epiduralni mesocainu amp(!1%)
INCORRECT: mesocain(!1%) epiduralni anealgezie != mesocain(!1%) epiduralni SHOULD BE mesocain(!1%) epiduralni analgezie anestezie
CORRECT: anestezie(!INV) =
CORRECT: xylocaine lok = lok xylocain
CORRECT: mecocain(!1%) 2 amp i v = mesocain(!1%) amp
CORRECT: mesokain(!1%) epiduralni = mesocain(!1%) epiduralni
INCORRECT: mesocain 2 amp(!1%) analgesia epiduralis != epiduralni analgezie amp(!1%) SHOULD BE mesocain(!1%) epiduralni analgezie mesocainu amp(!1%)
INCORRECT: m(!1%) mesocain != amp(!1%) SHOULD BE mesocain(!1%) mesocainu amp(!1%)
CORRECT: epiduralni anesteie = epiduralni anestezie
CORRECT: 1 amp mesocain(!1%) = mesocain(!1%) amp
CORRECT: spinalni an =
CORRECT: mesocian(!1%) 1 amp lok = mesocain(!1%) lok amp
CORRECT: epiduralni analg anestezie = epiduralni analgezie anestezie
CORRECT: xylocaine 1 amp mesocain(!1%) = mesocain(!1%) amp xylocain

```

(In this listing, CORRECT: A=B means that, dataset record A and NSA determined class B are in match. INCORRECT: A!=B SHOULD BE C means that dataset record A has artificially created “real” class C (translated to natural language using class identifier object) and not AIRS determined class B).

## Optimisation of input parameter values

As in 4.4, a simple genetic algorithm was used to determine suboptimal parameter values for this algorithm and dataset. Diagram in fig. 37 shows how mean population fitness and best fitness varied with number of generations simulated.

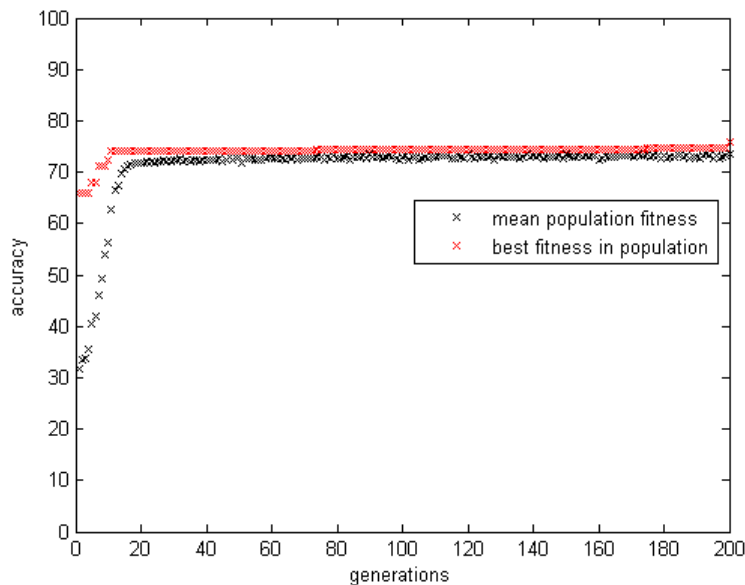


fig. 37 — Dependency of fitness on number of generations (NSA variant)

## Results

- NSA on Less complex biomedical data: matching factor=0.303
  - Accuracy: mean **75.894 %** on 100 randomly selected 1:1 percentage splits



## 8 Final results (on both datasets)

In this chapter, both AIRS and NSA are used for classification of the entire less complex dataset (which was also used in testing) and of part of a more complex dataset, which is significantly greater and has significantly greater word-stock. Also, two variants of KNN algorithm were used in order to compare AIS to a member of non-AIS classification algorithms group. The first variant is the classic KNN (only 1-nn was used, because more-nn was proved not to yield good results in 6.3) with no data reduction. The second variant of KNN does not use training at all and it might not be called KNN at all, because a Class identifier object is provided to it in order to create a final pool of words. Classification itself, in this second variant, follows that of the variant AIRS (section 6, i.e. 1-of-n approach).

### 8.1 Less complex dataset (22 000 words)

Optimised parameters used for this test are specified in 6.3.

#### Settings

Preprocess parameters were set manually to the value, at which the algorithm yielded satisfactory important words (parameter optimisation cannot be done in this situation).

- Preprocess parameters: Similarity threshold: **0.33**, Dispose rate: **0.005**
- Detected important words: **13**

#### Algorithm settings

- AIRS Parameters  
similarity threshold=0.330, stimulation threshold=0.7, knn=1, mutate=2, clonerate=5.
- NSA Parameters  
matching factor=0.303, Markov antibodies=0, mutated antibodies=0
- KNN (original) has no parameters (except  $k$ , which is stated above to be 1).
- KNN' (modified) has no parameters (except  $k$ , which is stated above to be 1).

#### Results

In the results and in fig. 38, it can be seen, that the results of NSA are actually better than the results of AIRS on the whole dataset. Such result was not expected, because on the first 1000 records, AIRS always performed better. In the end, the variant NSA algorithm seems more flexible for this problem than the variant AIRS. It can also be seen, that the modified KNN algorithm outperformed the other ones.

tab. 7 – Results on less complex biomedical dataset

	AIRS	NSA	KNN	KNN'
Average train time [s]	5.000	0.005	0.001	<0.001
Average test time [s]	2.800	6.500	127.8	1.127
Mean accuracy [%]	<b>78.17</b>	<b>81.22</b>	<b>60.64</b>	<b>82.48</b>
Median of accuracy [%]	77.88	81.27	60.59	82.52
Best-so-far accuracy [%]	81.68	83.60	63.51	83.54
Standard deviation [%]	1.59	0.68	1.23	0.50
0.05 quantile [%]	75.92	80.14	58.53	81.61
0.25 quantile [%]	76.85	80.72	59.73	82.12
0.75 quantile [%]	79.56	81.61	61.59	82.83
0.95 quantile [%]	80.82	82.23	62.62	83.25

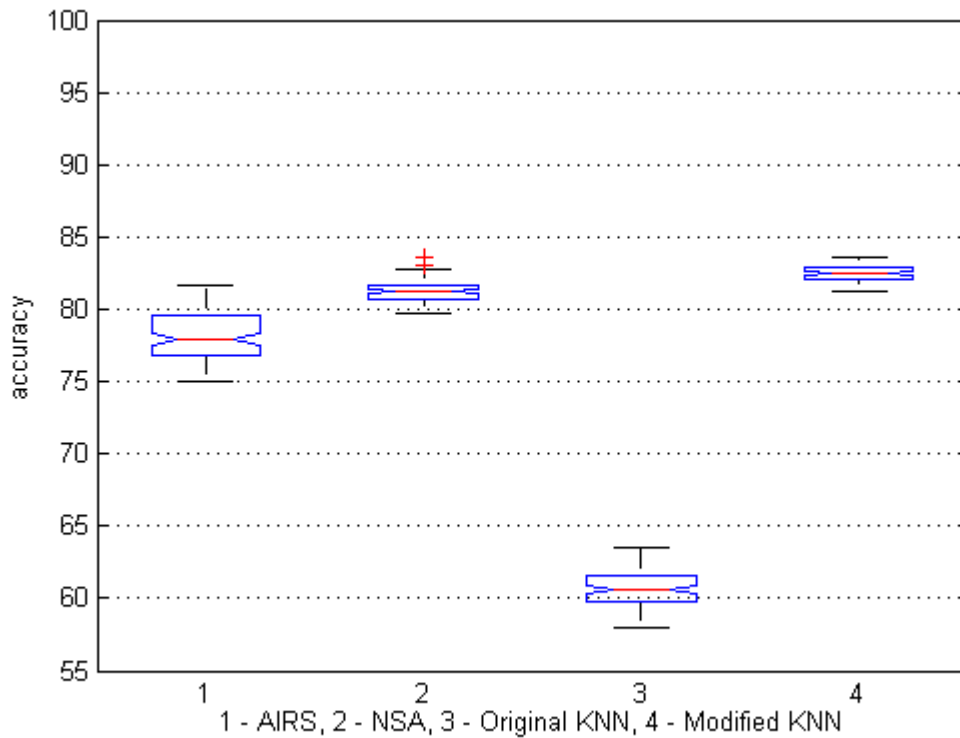


fig. 38 – Comparison of algorithms on entire less complex biomedical dataset

## 8.2 More complex dataset (1 500 000 words)

For instance, single dataset record looks like:

```
partus inductus in grav hebdomada 38 liquorrhoea amnialis praecox praesentatio occipitis funiculus umbilicalis circum collum fetus semel diabetes mellitus gestationis matris th prostin 0 5x2 mg ea amp iv analgesia epiduralis oxytocin i v ifpo episiotomia mediolateralis sut chiriac mem 1 amp iv dr kurecova sps dr huser hsps, 1111101011001001010110111010
```

Because of the complexity of the dataset and low optimisation of the code, it was not possible to process the whole dataset (with 500 MB heap space). The problem was in abundant usage of java Strings which are extremely slow for very large strings. Because of the same thing, it was not possible to test the original KNN algorithm (it has no data reduction).

### Preprocess

Preprocess parameters were set manually to the value, at which the algorithm yielded satisfactory important words (parameter optimisation cannot be done in this situation).

- Preprocess parameters: Similarity threshold: **0.33**, Dispose rate: **0.25**
- Detected important words: **28**

### Results

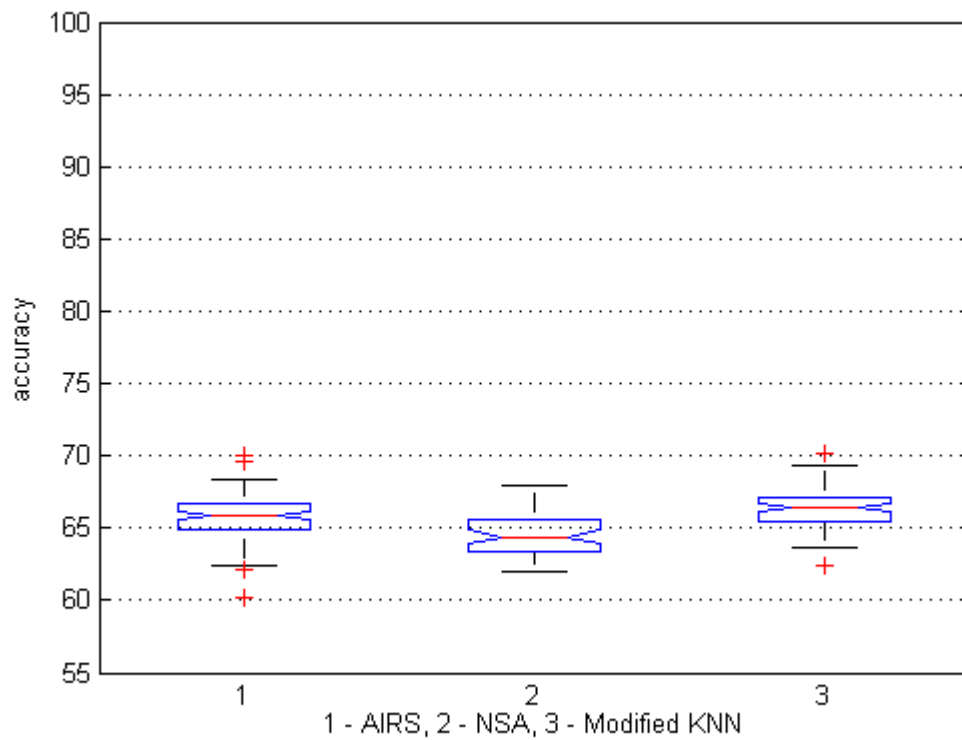
In the results and in fig. 39, it can be seen, that AIRS performs better than NSA and KNN is slightly better than both of them.

*tab. 8 – Results on more complex biomedical dataset*

	AIRS	NSA	KNN'
Average train time [s]	7.940	0.018	<0.001
Average test time [s]	9.210	107.0	4.300
Mean accuracy [%]	<b>65.80</b>	<b>64.49</b>	<b>66.28</b>
Median of accuracy [%]	65.90	64.40	66.40
Best-so-far accuracy [%]	70.00	68.00	70.20
Standard deviation [%]	1.55	1.44	1.32
0.05 quantile [%]	63.40	62.20	64.20
0.25 quantile [%]	64.90	63.40	65.50
0.75 quantile [%]	66.70	65.60	67.10
0.95 quantile [%]	68.30	67.40	68.30

## Parameters

- AIRS Parameters  
    similarity threshold=0.330, stimulation threshold=0.7, knn=1, mu-  
    trate=2, clonerate=5.
- NSA Parameters  
    matching factor=0.303, Markov antibodies=0, mutated antibodies=0
- KNN' (modified) has no parameters (except  $k$ , which is stated above to be 1).



*fig. 39 — Comparison of algorithms on more complex biomedical dataset*



# 9 Conclusion

The main aim of this work was to apply Artificial Immune Systems on large and loosely structured biomedical text datasets (i.e. biomedical datasets thereafter) and evaluate their performance. In this chapter, the goals declared in chapter 1.1 are mapped to respective achievements.

## 9.1 Goals to achievements mapping

### 1) **Literature study**

Relevant and useful publications concerning AIS were studied (chapters 2, 3).

### 2) **Selection of 2 algorithms**

Based on 1), two algorithms were selected, namely AIRS and NSA. They were selected, because they are different in the way they operate—AIRS can work with multiple classes, whereas NSA is purely a binary classifier. AIRS uses the KNN algorithms as it's matching function, whereas NSA uses radius threshold matching. ARIS tries to cover the self-space, whereas NSA tries to do the exact opposite—cover the non-self space.

### 3) **Implementation**

In order to make thorough testing and diagnostics possible, an appropriate software framework was built in the Java programming language. Thanks to modular structure and object oriented approach, the system is designed to be able to easily support new algorithms and diagnostic features (detailed description of the API is a part of the Javadoc documentation of the source code). The current implementation supports 8 algorithms (NSA, variant NSA (adapted to biomedical data), AIRS, variant AIRS (adapted to biomedical data), KNN, variant KNN (adapted to biomedical data), Immunos-1 and Clonalg), Microsoft SQL Database Manipulation Frontend, Preprocessing Frontend, Genetic Algorithm Parameter Selection Frontend, CSV Manipulation Frontend, Profile Manipulation Frontend and Testing Frontend. The Testing Frontend generates MATLAB compliant code and is currently able to automatically output boxplots and multiple data series plots. Resulting source code has 6983 lines of code in 49 files.

### 4) **Preliminary testing**

In order to see, how AIRS and NSA behave on simple datasets, preliminary tests were made (chapter 4). Both AIRS and NSA algorithms were tested on both real (Iris) and artificially created (Gaussian) datasets and were compared to one another. On these datasets, AIRS outperformed NSA with rather promising results (AIRS reached 96.45 % accuracy on the Iris dataset and 95.45 % on the Gaussian dataset, while NSA reached 85.15 % accuracy on the Gaussian dataset).

### 5) **Biomedical data preprocessing and class assignment**

In order to normalize biomedical data and in order to create and assign artificially created classes to them, a text preprocessing algorithm was designed and

implemented (chapter 5). This algorithm automatically selects important words based on their frequencies using a new distance metric (Product distance, proposed in 5.3).

6) **Algorithm performance evaluation**

In order to study behaviour of the modified algorithms, both of them were tested on a sample subset of the less complex of the two biomedical datasets (chapters 6.3 and 7.3).

7) **Large scale testing and comparison**

In the end, a performance comparison of the algorithms on very large datasets was made and it was compared with performance of the KNN algorithm (chap. 8).

*tab. 9 – Results revision table*

	AIRS	NSA	KNN	KNN'
Mean accuracy on less complex dataset [%]	<b>78.17</b>	<b>81.22</b>	<b>60.64</b>	<b>82.48</b>
Mean accuracy on more complex dataset [%]	<b>65.80</b>	<b>64.49</b>	—	<b>66.28</b>

## 9.2 Work not declared in goals

In addition to the specified goals (1.1), an additional work had to be done.

### Algorithm alteration

In order to adapt AIRS and NSA to the biomedical datasets, we have non-trivially altered the original algorithms to fit them.

### Product distance

In order to achieve better resolution in string comparison, a new distance metric was proposed and implemented in chapter 5.3.

### Genetic Algorithm Parameter Selector

In order to optimize parameter values of algorithms, a Genetic Algorithm Parameter Selector was designed and implemented. This genetic algorithm uses metaheuristic (described in 4.4) to provide suboptimal set of parameter values and it is able to output MATLAB-encoded diagrams of mean and maximal fitness.

## 9.3 Final conclusion

The main result of this work is a modular diagnostic and preprocessing framework, which supports easy implementation of new algorithms and functionality. This framework can and will also be used for other datasets of similar kind.

Both AIRS and NSA were tested on real biomedical data and were proved usable for free text pattern mining problem, but their performance is rather average when compared to the KNN algorithm. Nevertheless, their principle and the idea behind them makes them viable for further study and applications.

## Appendix A: Statistics

In this chapter, statistic terms used in this thesis are defined and explained. Also, structure of various diagram types is described. Every information in this chapter is taken from [13] unless stated otherwise.

### A.1 Mean value

In this thesis, the mean value is equal to the unweighted average:

$$\bar{a} = \frac{1}{n} \sum_{i=1}^n a_i \quad (6)$$

The most significant property of so defined mean is that if there are outliers in the selection, they will affect the result greatly.

### A.2 Median

The median of the sorted selection S is the very middle element of this selection. If the selection has even length, then this selection's median is the mean of the two middle elements.

### A.3 Variance

The variance is a mean quadratic deviation from a mean value of a selection:

$$S_A = \frac{1}{n-1} \sum_{i=1}^n (a_i - \bar{a}_n)^2 \quad (7)$$

### A.4 Standard deviation

The standard deviation is a mean deviation from a mean value of a selection:

$$s_A = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (a_i - \bar{a}_n)^2} \quad (8)$$

## A.5 String Markov information source example

Let the alphabet  $\Gamma$  be defined as  $\Gamma = \{a, b, c, \dots, x, y, z\}$  and let the corpus text be `abcd-eaxbycza`. There are following transitions in the corpus text: `a->b`, `b->c`, `c->d`, `d->e`, `e->a`, `a->x`, `x->b`, `b->y`, `y->c`, `c->z`, `z->a`.

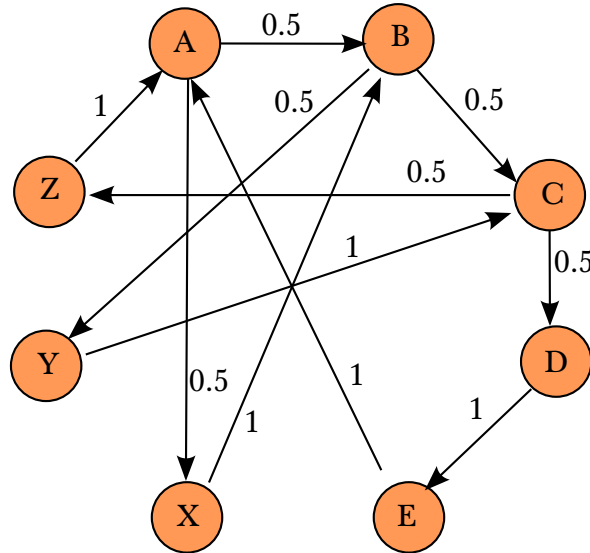


fig. 40 — Graph of a sample Markov information source

We can represent the transitions as a transition matrix or a graph (fig. 40). The information source can have a starting condition or may not. In case it has no starting condition, any vertex can be picked. If a vertex is selected, a letter represented by it is attached to an output. Then a transition is made according to probabilities on edges and a new vertex is selected and so on. For instance, assuming the source described above (fig. 40), valid generated words include: `abczabzabcz`, `xbycdexbcdea`, `abcdeaxbc` and so on.

Markov information source used in this thesis is not based on single letter but on letter pairs (digrams), because they are more likely to mimic natural language when given natural language corpus text. In english language, there are several very common digrams, like `th`, `er`, `is`, `st`, `ct`, `of`, `at` and there are common transition rules, for instance `th` is most likely followed by `is`, `at` or `en`, but not by `of`, although `ho` is also fairly common (i.e. when using single letters as an alphabet, probability of getting `thof` is a product of transition probabilities between `t->h`, `h->o` and `o->f`, which is nonzero, whereas when using digrams as an alphabet, probability of getting `thof` is a product of transition probabilities between `th->of` and because such transition does not exist in english language at all, it is zero).

When using trigrams as an alphabet, results are even better, but this variant needs very large corpus text. One can also use whole dictionary as an alphabet and a very large text source as a corpus text... then the Markov information source gives sensible sentences as its output.

## A.6 Boxplot

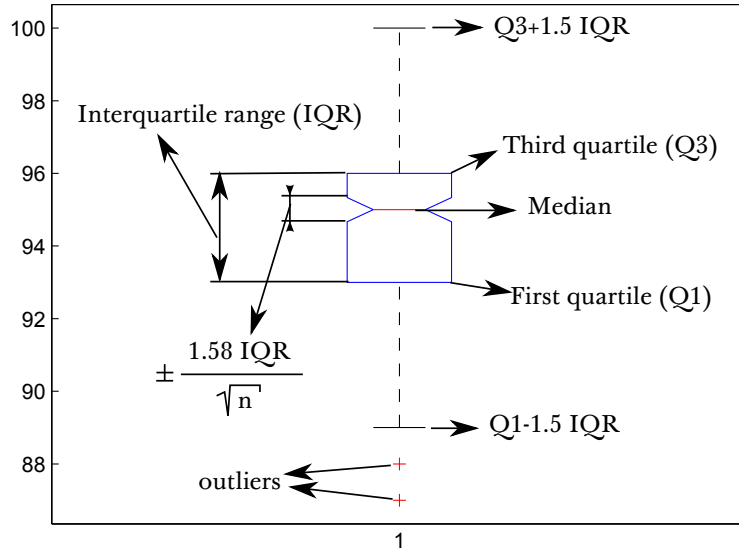


fig. 41 — Boxplot description

## A.7 Multiple data series plot

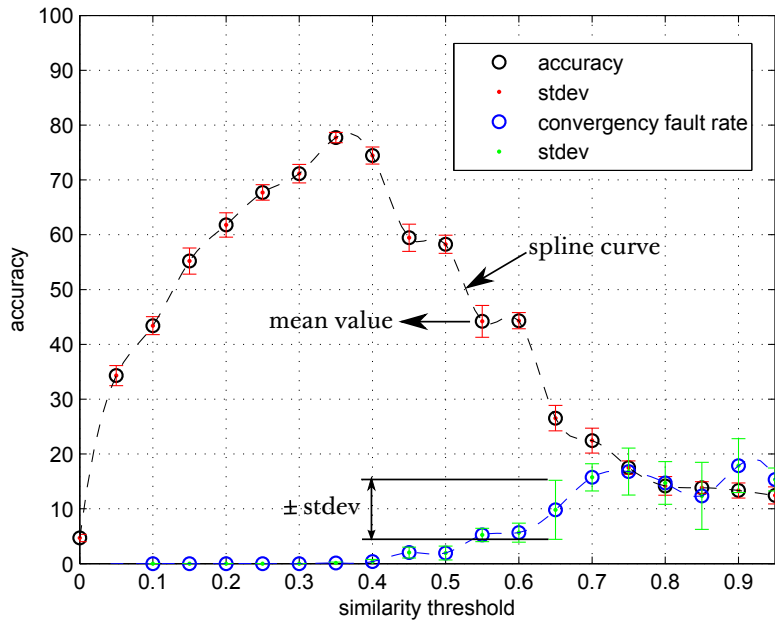


fig. 42 — Multiple series plot description

## Appendix B: Pseudocode syntax

The pseudocode is a meta-language used to describe algorithms in such well-arranged and abstract way, real languages would never achieve.

*tab 6. — pseudocode*

Syntax	Meaning
<code>a:=b</code>	b is assigned to a by value
<code>a&lt;=b</code>	b is assigned to a by reference
<code>a=b</code>	logic operation: a equal to b
<code>a&lt;b, a&lt;=b</code>	logic operation: a less than, less or equal to b
<code>a&gt;b, a&gt;=b</code>	logic operation: a greater than, greater or equal to b
<code>a sameas b</code>	object comparison
<code>if(condition), fi</code>	if clause
<code>for(i, c, a), end</code>	c like for clause
<code>do while, while, end</code>	other clauses
<code>for(o in list)</code>	for each clause
<code>function(params)</code>	function call
<code>membervar(o)</code>	getter for membevar in object o
<code>add(list,o)</code>	adds item to list
<code>remove(list,o)</code>	removes item from list
<code>Procedure: Name</code>	function declaration
<code>Input: a, b, c</code>	input parameters for function
<code>Output: d</code>	single output parameter of function

Commands can be separated either with new line or comma. Because the pseudocode is not to be executed, commands and constructions not specified in this list are allowed to be used as long as their meaning is trivial to understand.

## Appendix C: Machine specification

All tests were run on the machine with following parameters.

- Operating system: Microsoft Windows 7 SP1, 32 bit instructions
- CPU: Intel Pentium Dual-Core T4200, 2 GHz clock rate
- Operating memory: 3GB

The following development or other notable software was used.

- Java SE platform
- Mathworks Matlab (faculty licence)
- Wolfram Mathematica (faculty licence)
- X<sub>Y</sub>TEX with plain format

- Storm Type Foundry typefaces (personal licence)
- Enterprise Architect
- Microsoft SQL Server 2008

## Appendix D: DVD directory and file structure

```
|-..
|-.
|-BT_SOURCE_CODE
  |-BT
    |-src
      |-AIRS
        | AIRS.java
        | AIRSFillEngine.java
        | CSF.java
        | Pattern.java
        | Point.java
      |-CLASSIFIER
        | ClassificationAlgorithm.java
        | Test Java
      |-GAPParamSelection
        | Instance.java
        | ParamOperator.java
        | ParamSelector.java
      |-KNN
        | Core.java
        | KNN.java
        | KNNFillEngine.java
        | Pattern.java
        | Point.java
      |-NSA
        | DT.java
        | NSA.java
        | NSAFillEngine.java
        | Point.java
        | SpecialisedAntibodyGoup.java
      |-UTILITY
        | CSV.java
        | DATASET.java
        | Distance.java
        | Distance.java
        | FillEngine.java
        | MarkovChainSource.java
        | MutableDouble.java
```



```
    | MutableInteger.java
    | MutableNumber.java
    | SortableBinaryString.java
    | Statistics.java
|-bt
  | Main.java
|-database
  | BTException.java
  | BTDatabase.java
  | Database.java
|-preprocess
  | ClassIdentifier.java
  | Identification.java
  | SortableStringDoublePair.java
  | TranslationException.java
|-profile
  | ParsingException.java
  | Profile.java
  | ProfileException.java
  | ProfileReader.java
  | ProfileWriter.java
  | SectionHeader.java
|-DATA
  |-csv_datasets
  | bcw.data
  | gauss.data
  | iris.data
  | lesscmplx_processed.data
  | morecmplx_processed.data
  | wine.data
  |-graphs
  | graphs.rar
  |-profiles
  | lesscmplx.profile
  | morecmplx.profile
|-TEX
  | source.rar
  | BT.pdf
```

## References

- [1] S. Forrest and A. S. Perelson and L. Allen and R. Cherukuri, “Self-Nonself Discrimination in a Computer”, in Proceedings of the 1992 IEEE Symposium on Security and Privacy, 1994.
- [2] P. D’haeseleer, “An immunological approach to change detection: theoretical results”, in Proceedings of the 9th IEEE Computer Security Foundations Workshop, 1996.
- [3] Andrew Watkins, Jon Timmis, and Lois Boggess. (2004). “Artificial Immune Recognition System (AIRS): An Immune-Inspired Supervised Learning Algorithm”. *Genetic Programming and Evolvable Machines*, 5 (3): 291-317, September 2004
- [4] Andrew B. Watkins. (2001). “AIRS: A resource limited artificial immune classifier.”, Master’s thesis, Mississippi State University, MS. USA., December 2001.
- [5] Andrew Watkins and Jon Timmis. (2004). “Exploiting Parallelism Inherent in AIRS, an Artificial Immune Classifier”. In Proceedings of the 3rd International Conference on Artificial Immune Systems (ICARIS2004) held in Catania, Italy, 13-16 September, 2004. *Lecture Notes in Computer Science (LNCS)*, Number 3239. Edited by G. Nicosia, V. Cutello, P. Bentley, and J. Timmis, pages 427-438. Springer. 2004.
- [6] Kephart, J. O. (1994). “A biologically inspired immune system for computers”. *Proceedings of Artificial Life IV: The Fourth International Workshop on the Synthesis and Simulation of Living Systems*. MIT Press. pp. 130–139.
- [7] Dasgupta, Dipankar; Nino, Fernando (2008). CRC Press. pp. 296. ISBN 978-1-4200-6545-9.
- [8] Jason Brownlee (2011). *Clever Algorithms: Nature-Inspired Programming Recipes*. ISBN 978-1-4467-8506-5.
- [9] Levenshtein distance. In Wikipedia. Retrieved October 10, 2011, from [http://en.wikipedia.org/wiki/Levenshtein\\_distance](http://en.wikipedia.org/wiki/Levenshtein_distance)
- [10] Andrew Watkins, Jon Timmis. (2004). “Exploiting Parallelism Inherent in AIRS, an Artificial Immune Classifier”.
- [11] Farmer, J. D., N. H. Packard and A. Perelson. “The Immune System, Adaptation, and Machine Learning.” *Physica D* 22(1-3) (1986): 187-204.
- [12] A Brief History of Artificial Immune Systems. AISWeb. 29 Aug. 2011 <<http://www.artificial-immune-systems.org/people-new.shtml>>
- [13] Navara M.: “Pravděpodobnost a matematická statistika”, ČVUT, Praha, 2007

- [14] Kroupa T.: Úvod do teorie informace: “Matematické základy komprese a digitální komunikace”, ČVUT, Praha, Prosinec 2011
- [15] V. Jarník: “O jistém problému minimálním (About a certain minimal problem)”, *Práce Moravské Přírodovědecké Společnosti*, 6, 1930, pp. 57–63. (in Czech)
- [16] Joseph. B. Kruskal: “On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem.” In: *Proceedings of the American Mathematical Society*, Vol 7, No. 1 (Feb, 1956), pp. 48–50
- [17] Borůvka, Otakar: “O jistém problému minimálním (About a certain minimal problem)” (in Czech, German summary). *Práce mor. přírodověd. spol. v Brně III* 3, 1928: 37–58.
- [18] Levenshtein V. I. (1966). “Binary codes capable of correcting deletions, insertions, and reversals”. *Soviet Physics Doklady* 10: 707–10.
- [19] de Castro, L. N.; Von Zuben, F. J. (2002). “Learning and Optimization Using the Clonal Selection Principle”. *IEEE Transactions on Evolutionary Computation*, Special Issue on Artificial Immune Systems (IEEE) 6 (3): 239–251.
- [20] J. Greensmith and U. Aickelin and S. Cayzer, “Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection”, in *Proceedings of the Fourth International Conference on Artificial Immune Systems (ICARIS 2005)*, 2005.
- [21] T. Knight and J. Timmis, “AINE: An Immunological Approach to Data Mining”, in *First IEEE International Conference on Data Mining (ICDM’01)*, 2001.
- [22] L. N. de Castro and J. Timmis, “An artificial immune network for multimodal function optimization”, in *Proceedings of the 2002 Congress on Evolutionary Computation (CEC’02)*, 2002.
- [23] Petr Olšák. “Úvod do algebry, zejména lineární”, FEL ČVUT, Praha 2007, ISBN 978-80-01-03775-1
- [24] David E Goldberg. “Genetic Algorithms in Search, Optimization and Machine Learning”, Kluwer Academic Publishers, Boston, MA 2007.