Doctoral Thesis

Development of Optimization Method with the Use of Genetic Algorithms for Natural Language and Related Models

September2018

Pawel C. Lempa

KITAMI INTITUTE OF TECHNOLOGY

Graduate School of Engineering

September2018

Table of Contents

List of Tables

List of Figures

Abstract

Language models are an indispensable element of Natural Language Processing (NLP) research. They are used in machine translation, speech recognition, part-of-speech tagging, handwriting recognition, syntactic parsing, information retrieval and others. In short, language models are probability distributions over sequences of words. There are countless numbers of NLP solutions, algorithms and programs applying language models in specific tasks. Unfortunately, often these are not optimized, but rely on default, most commonly used sets of parameters. For example, many of them use numerous objective functions with different variables but without proper weights applied to them. Users usually set these variables themselves, which causes the results not to exceed a certain mediocre level. In case of small number of variables, users can adjust them manually, but optimization of objective functions with massive number of variables, especially multi-objective functions is difficult and time consuming. This was the motivation to propose an application of a Genetic Algorithms (GAs) to optimize the weighting process.

GAs are subset of Evolutionary Algorithms (EAs), inspired by the process of natural selection known from nature. They use bio-inspired operators such as selection, crossover and mutation to generate solutions for optimization and search problems. This way GAs represent randomized heuristic search strategies simulating natural selection process, where the population is composed of candidate solutions. They are focused on evolving a population from which strong and diverse candidates can emerge via mutation and crossover (mating). There exist different types of GAs, moreover the same type of GA can bring different quality of solutions, depending on multiple variables, which include starting population, number of generations or fitness function. Finding the best starting parameters and type of GA the most appropriate for a given optimization problem is a next challenge. For that reason, I created a library that automatically applies multiple types of GAs in optimization purposes.

The library was created in C++ language, with the use of .NET environment. Its main goal is to be used with different secondary programs and applications, without significant interfering in the original structure of the solution. Basic function of library allows the use of several different kinds of GAs like: Simple GA, Uniform Crossover GA, n-point Crossover GA, GA with sexual selection, GA with chromosome aging and so forth. User can freely define starting parameters for GA including: population size, starting population, number of generations, type of mutation and crossover. Advanced functions of the library allow the use of multithreaded processing for running several GAs in the same time. Basic option of multithreading runs the same type of GA with different starting parameters, advanced version allows to exchange information between different threads every set number of generations. In case of large number of variables to compute, it is also possible to separate a mutation and crossover for several threads running at the same time.

The most important functionality of the library is its easy adjustability in optimization of different kinds of applications. The library is used to run the original program in every generation of GA with new weights for variables generated from natural selection. Time of program running is closely related with original program processing time. It depends on the type of original solution and the time of processing one generation is similar to one run of the optimized program.

During creating and testing the library, numerous experiments have been carried out.

In preliminary experiments the library was used for optimization of construction of mechanical elements. Later the application was tested on natural language processing and related solutions. One part of the research was optimizing Quantitative Learner's Motivation Model. The goal of this experiment was to optimize the formula for prediction of learning motivation by means of different weights for three values: interest, usefulness in the future and satisfaction. For this optimization, an application in C# using GA library was created. Data sets for the experiments were acquired from questionnaires enquiring about the above three elements in actual university classes. The results of the experiment showed improvement in the estimation of student's learning motivation up to over 17 percentage points of Fscore.

The final experiment aimed to optimize the implementation of Support Vector Machines (SVMs) for the problem of pattern recognition in natural language data. SVMs are a machine learning algorithm based on statistical learning theory. They are applied to large number of real-world applications, such as text categorization, hand-written character recognition, etc. Original program was created in C++. For this application numerous different types of GAs were tested with different number of generations, weight range and starting parameters. Optimization was successful, with different scale of improvement based on previously mentioned conditions, with the highest achieved improvement of over 6 percentage points of recall comparing to baseline and reaching 78%. All experiments data are included in this work.

Chapter 1

Introduction

1.1 Background and goal of this work

Language models are inseparable element of Natural Language Processing (NLP) research. They are used in machine translation, speech recognition, part-of-speech tagging, handwriting recognition, syntactic parsing, information retrieval and others. In short, language models are a probability distribution over sequences of words. There are countless numbers of NLP solutions, algorithms and programs applying language models in specific tasks. Unfortunately, often these are not optimized, but rely on default, most commonly used sets of parameters. For example, many of them use numerous objective functions with different variables but without proper weights applied to them. Users usually set these variables themselves, which causes the results not to exceed a certain mediocre level. In case of small number of variables, users can adjust them manually, but optimization of objective functions with many variables, especially multi-objective functions is difficult and time consuming. This motivated me to propose an application of a Genetic Algorithm to optimize the weighting process.

1.2 Previous research

Natural Language Processing (NLP) is an interdisciplinary field connecting artificial intelligence and linguistics. NLP focuses on a number of topics like language generation, machine translation, dialogue systems etc. Researches around the world have proposed different approaches to solve problems within previously mentioned topics. One of them is to use genetic algorithms. A genetic algorithm is an algorithm looking for the best solution using heuristic searching based on natural selection known in genetics. This approach has proven its utility in many cases, sometimes replacing other tools used in artificial intelligence, but also often used as a part of them to resolve a given research problem. Therefore genetic algorithms are often used with e.g neural networks. In the following subsections I will describe the application of genetic algorithms in selected papers related with this topics.

1.2.1 Dialogue Systems

Dialogue systems are computer systems which purpose is to converse with the human in most natural way. Often such systems are used for communication based on text, speech or symbols. Dialogue systems tend to be a part of more general solutions. For example, full functional robot for whom dialogue system is used for communication with external environment. This kind of robots are described in papers by Nakadai et al. (2008) [4] and (2010) [5]. In both works genetic algorithms are used for parameters selection in audition subsystems.

Genetic algorithms, are also frequently used for their most practical function, namely, for optimization. In the context of dialogue systems genetic

algorithms can help optimize the settings of time and accuracy of the used methods [3] or cost function [6].

More advanced versions of genetic algorithms are also used. Thanks to large diversity and ease in adoption of GA, they were used in more specified situations. Genetic Algorithm with Sexual selection (GAwSS) was used by Araki and Kuroda (2006) [1]. Thanks to that, a system based on the method using GAwSS, in its initial state can be trained to sufficient level without any prior language information (vocabulary or grammar).

Another example of system which does not need a predefined training data is described Kimura et al. (2001) [2]. In this case conversation rules for the dialog agent are created on base of statements and the system responses.

The last case of GA used in dialogue system is described in [7]. The system proposed in the paper uses a dynamic weight connection of Artificial Intelligence learning algorithm with neural network and genetic algorithm. At first they are created as separate modules. In this case there were used two versions of genetic algorithms, basic and mutated. The goal was to find the best option for this solution.

1.2.2 Language Generation

Language generation is a process which aims to create natural language from a data representation such as a knowledge base. Language generators are often used to create textual version of natural language.

The paper by Montero and Araki (2007) [8] describes a method which is based on a small number of phrases chosen randomly from a databese of phrases. From these phrases genetic algorithm generates and evaluates trivial dialogue phrases.

Language generation can also be used to create very sophisticated texts like poetry. Examples of this kind of systems are described in [9] and in [11], where a genetic algorithm is used to find a solution that satisfies the constraints of grammaticality, meaningfulness, and poeticalness.

Good example of the use genetic algorithm is presented in [10]. This paper describes a system of opportunistic text generation. The system reacts react on user choices and adapts to them. In this case a genetic algorithm is used for finding highly-valued as possible tree of facts.

The last example of a paper describing genetic algorithms applied to NLP, [12] describes the use of a genetic algorithm in story generator. In this case GA is utilized for story planer to find space of possible stories. Main reason for using it here is to greatly reduce the risk of getting stuck in local optima.

1.2.3 Machine Learning

Machine learning is intended to create algorithms that can learn on the base of initial data. Often this initial data needs to be very large. In this situation a genetic algorithm is a useful solution which can significantly reduce of required information. A good example is presented by Echizenya et al. in [13], [14] and [15]. Thanks to that the number of translation examples for machine translation was greatly reduced. Also important thing is that, the different solutions can use different versions genetic algorithms adjusted to their needs.

The last example could be [17]. In this paper the authors presented a system YALE. It is free open-source environment for KDD and machine learning. It contains a large number of useful features, but for this survey the most important part is Text mining and tracking drifting concepts. Again

as presented in section on Dialogue Systems, genetic algorithms are used for feature selection.

1.3 Terminology

This section describes a terminology used in this work. Topics from Natural Language Processing were described in section 1.2.

1.3.1 Language Model (LM)

Language model is a probability distribution over sequences of words. For a sequence of the length m, it assigns a probability $P(w_1, .., w_m)$ (observing sentence $w_1, .., w_m$) to the whole sequence. This can be useful in many natural language processing applications, especially when an output is text.

Language modeling among others is used in:

- speech recognition
- part-of-speech tagging
- machine translation
- information retrieval
- handwriting recognition
- parsing

Example of the use of Language Model can be shown in speech recognition. The system (computer, robot etc.) tries to understand human words

and context to distinguish between words and phrases that sound similar is provided by LM.

The apple and pair salad

The apple and pear salad

Also LM can be used for a prediction of missing words.

Please turn off your cell _____

Or for connotation recognition.

You are stupid (Negative connotation)

I love you (Positive connotation)

In building language models a major problem is data sparsity. During training, most possible word sequences will not be observed. To resolve this one option is to assume that the probability of a word only depends on the previous n words. This is known as an n-gram model or unigram model when n = 1.

Unigram model

A unigram model is used in information retrieval. It can be treated as the combination of several one-state finite state machine.

It splits the probabilities (P) of different terms(t) in a context, e.g. from 1.1

$$P(t_1t_2t_3) = P(t_1)P(t_2|t_1)P(t_3|t_1t_2)P(t_1t_2t_3) = P(t_1)P(t_2|t_1)P(t_3|t_1t_2) \quad (1.1)$$

to unigram probability (P_{uni}) in 1.2

$$P_{uni}(t_1t_2t_3) = P(t_1)P(t_2)P(t_3)P_{uni}(t_1t_2t_3) = P(t_1)P(t_2)P(t_3) \quad (1.2)$$

In this model, the probability of each word only depends on that word's own probability in the document, so there is only one-state finite state machine as units. The machine itself has a probability distribution over the entire vocabulary (P(term)) of the model (1.1), summing to 1 (equ. 1.3) .

Table 1.1: Unigram model

Term	Probability
a	0.1
word	0.25
to	0.08
check	0.04
for	0.2
...	...

$$\sum_{\text{term in doc}} P(term) = 1 \tag{1.3}$$

The probability generated for a specific query (P(query)) is calculated as in 1.4.

$$P(\text{query}) = \prod_{\text{term in query}} P(\text{term}) \tag{1.4}$$

1.3.2 Quantitative Lerner's Motivation Model (QLMM)

QLMM is a composition of three elements, which represent the attitude of students towards the attended courses ([18]). Quantification of those elements represents the general level of learning motivation. Te three elements called interest, usefulness in the future, and satisfaction in original model

have the same level of importance. However, experience shows that they are not equally important. For the purpose of estimation of learning motivation in students original questionnaire was created. The designed questionnaire consists of ten questions, including both choice-, and free answer-questions.

Prediction procedure

$$Q = \{q_1, q_2, ..., q_i, ..., q_m\} \tag{1.5}$$

$$X = \{x_1, x_2, ..., x_j, ..., x_n\} \tag{1.6}$$

where:

- questionnaire Q (1.5) consists of m questions(q)
- learner group X (1.6) contains n number of people, where x_n is group of n people

In the step 1, he values of evaluation $r_{i,j}$ from the answers of learners x_j for the questionnaire q_i (i is questionnaire number, j is number of learner) are collected and for each question mean average μ_i and standard deviation σ_i of the respondents are calculated. In this case, μ_i is calculated according to equation (1.7), and σ_i is calculated according to equation (1.8).

$$\mu_i = \frac{1}{n} \sum_{j=1}^{n} r_{i,j} \tag{1.7}$$

$$\sigma_i^2 = \frac{1}{n-1} \sum_{j=1}^{n} (r_{i,j} - \mu_i)^2 \tag{1.8}$$

In the step 2, classify $r_{i,j}$ for each of the three evaluated items using as threshold σ_i, as in equation (1.9) and produce scores for all elements $s_{i,j}$. Next, calculate the scores $s_{i,j}$ of each learner x_i for each question i times. Finally, sum the $s_{i,j}$ for each learner and predict the learner's motivation M_j, according to equation (1.10).

$$s_{i,j} = \begin{cases} 1 & \text{if } r_{i,j} > \mu_i + \sigma_i \\ -1 & \text{if } r_{i,j} < \mu_i - \sigma_i \\ 0 & \text{otherwise} \end{cases} \tag{1.9}$$

$$M_j = \sum_{i=1}^{m} s_{i,j} \tag{1.10}$$

In the step 3, M_j obtained in step 2 has the following properties: $M_j|j \leq |m|$. Therefore, it classify the learning motivation M_j into three classes according to equation (1.11).

$$\text{Learner's motivation is } \begin{cases} \text{high} & \text{if } M_j > 1 \\ \text{low} & \text{if } M_j < 1 \\ \text{neither high nor low} & \text{otherwise} \end{cases} \tag{1.11}$$

1.3.3 Accuracy, precision and recall

For calculating accuracy, precision and recall are used number of correct and incorrect answers. In tab 1.2 are shown actual positive (p) and negative answers (n) and predicted positive (p') and negative (n') answers.

Table 1.2: Values for Accuracy, Precision and Recall

	p' (Predicted)	n' (Predicted)
p (Actual)	True Positive (TP)	False Negative (FN)
n (Actual)	False Positive (FP)	True Negative (TN)

Accuracy

Accuracy describes what % of the classifications are correct (1.12).

$$A = \frac{TP + TN}{TP + FP + FN + TN} \tag{1.12}$$

Precision

Precision describes of the number of words selected, what percentage of them did the system classify correctly (1.13).

$$P = \frac{TP}{TP + FP} \tag{1.13}$$

Recall

Recall describes the number of words classified correctly, what percentage of them have been selected (1.14).

$$R = \frac{TP}{TP + FN} \tag{1.14}$$

F-score

F-score is the weighted harmonic mean of Precision and Recall (1.15).

$$F = \frac{1}{(\alpha)(\frac{1}{P}) + (1 - \alpha)(\frac{1}{R})} \tag{1.15}$$

1.3.4 Support Vector Machines (SVMs)

SVMs are a machine learning algorithm based on statistical learning theory. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

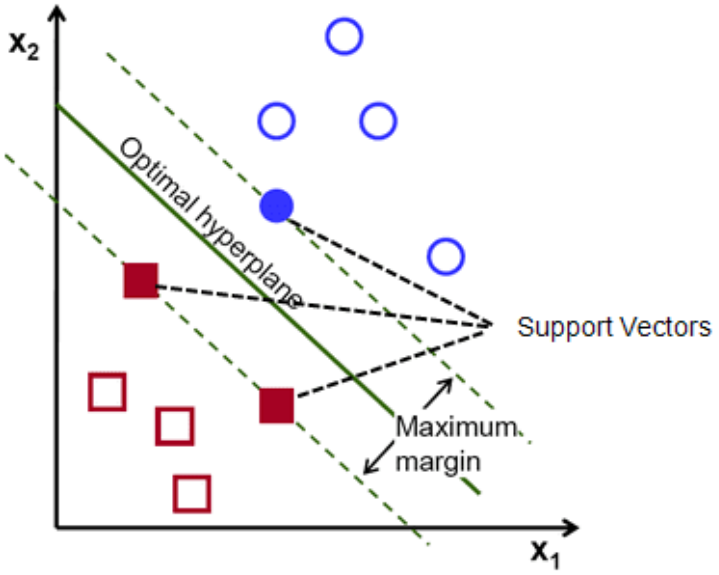
Types of SVMs:

- Linear
- Polynomial
- Radial based function
- Sigmoid

Linear SVM

For linear svm the original function which finds the maximum-margin hyperplane dividing the samples (fig. 1.1). It works best with language data.

Figure 1.1: SMV Linear



1.4 Structure of dissertation

This dissertation is composed of 5 chapters.

Chapter 1 provides background and an outline of this dissertation and survey on a use of GAs in NLP.

Chapter 2 describes Genetic Algorithms (GAs), definitions connected to it and different types of GAs used in this work.

Chapter 3 describes the proposed solution for optimizing Natural Language and Related Models and describes a GAs library created for that purpose.

Chapter 4 discusses the results of an experiments. The preliminary experiment to optimize Quantitative Learner's Motivation Model and main experiment to optimize Support Vector Machines for the problem of pattern recognition in natural language data.

Chapter 5 presents the general conclusions of this research. I carry out a comprehensive consideration of the solution proposed in chapter 3 and discuss of outcome of experiments described in chapter 4. I also reveal a challenges derived from the results of the evaluations, propose some methods for resolving.

Chapter 2

Genetic Algorithms (GAs)

The GA is a probabilistic search algorithm that iteratively transforms a set (called a population) of mathematical objects (typically fixed-length binary character strings), each with an associated fitness value, into a new population of offspring objects using the Darwinian principle of natural selection and using operations that are patterned after naturally occurring genetic operations, such as crossover (sexual recombination) and mutation. It was developed in 1970's by J. Holland, K. DeJong, D. Goldberg in USA.

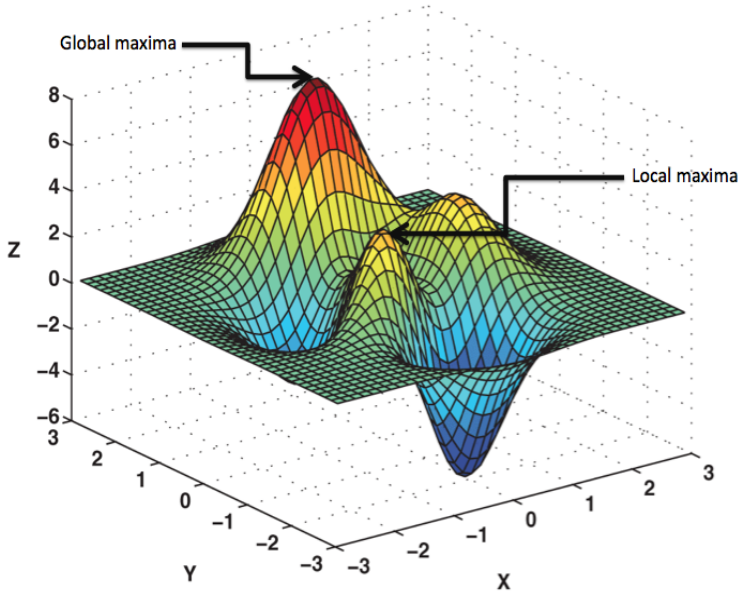
GAs are often used for optimization (scheduling, design, etc.) problems, though can be used for many other things as well. Good problem for GA are for example scheduling air traffic, but bad for problems like finding large primes (if the fitness landscape is not continuous it is hard to differentiate between something 1 bit away from a prime, and something that is in the complement of a prime) or 2D pathfinding (2D pathfinding is bad because the search space for it is small).

Genetic algorithms work best when the "fitness landscape" is continuous (in some dimensions as shown in fig. 2.1). This is also true of standard search, e.g. A*. Intuitively, this just means that it is possible to find a heuristic that gives a rough idea of how close a candidate is to being a solution.

Advantages of this solution are faster speed of computing (and lower memory requirements) than searching a very large search space and easiness,

in that if your candidate representation and fitness function are correct, a solution can be found without any explicit analytical work. But disadvantages are that algorithm can get stuck on local maximas, though crossover can help mitigate this and it can be hard to work out how best to represent a candidate as a bit string.

Figure 2.1: Fitness landscape



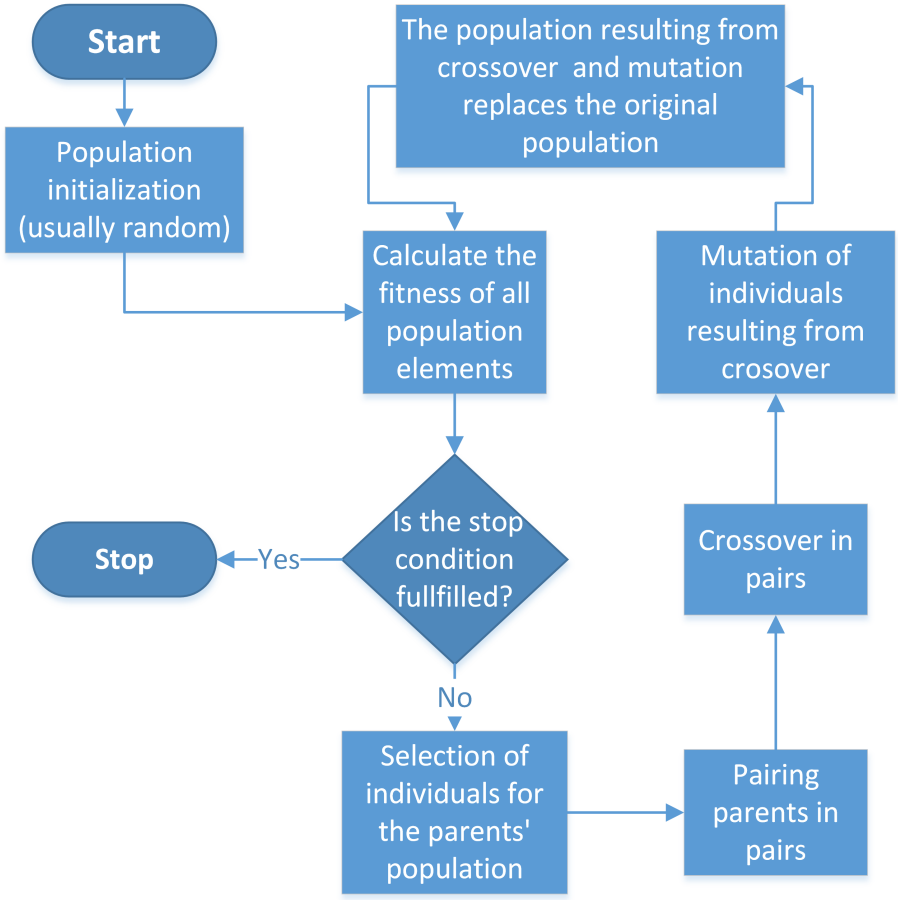
2.1 Successor functions

- Mutation – Given a candidate, return a slightly different candidate.
- Crossover – Given two candidates, produce one that has elements of each.

There is not always generate successor for each candidate. Rather, a successor is generated from population based on the candidates in the cur-

rent population, weighted by fitness. Given a population P, generate P' by performing crossover —P— times, each time selecting candidates with probability proportional to their fitness. Then get P" by mutating each candidate in P' and return P".While the best candidate so far is not a solution, create new population using successor functions and evaluate the fitness of each candidate in the population. In fig. 2.2 is shown a flowchart of typical GA.

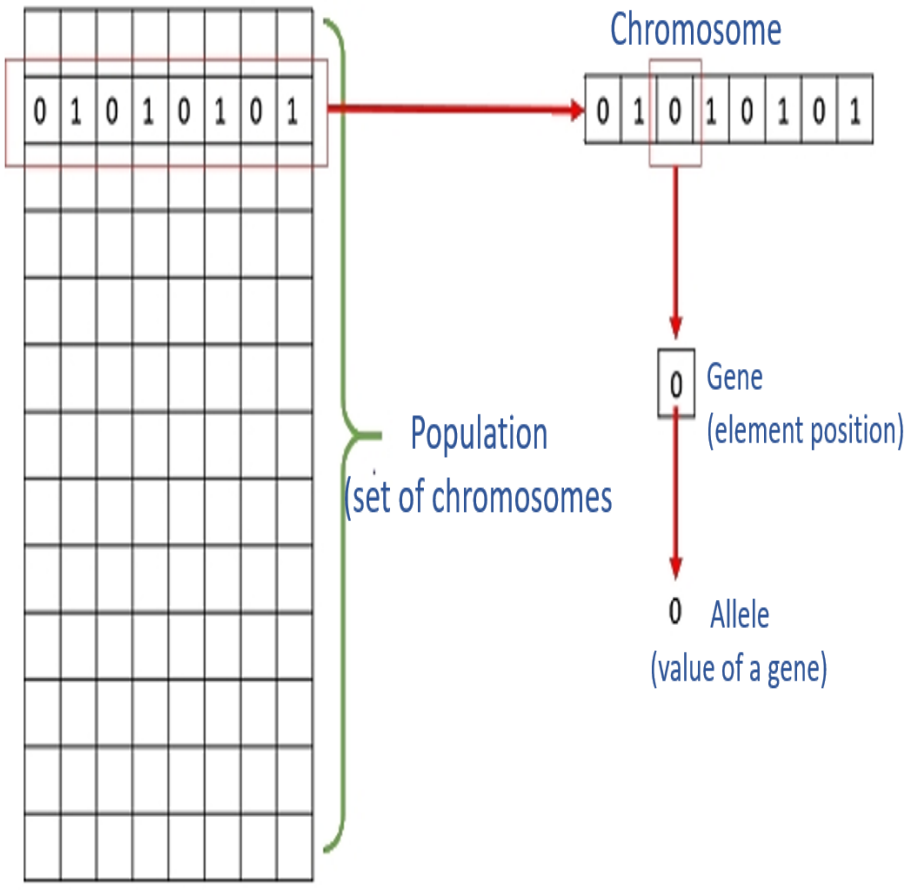
Figure 2.2: Genetic Algorithm flowchart



2.2 Population

Population is a subset of solutions in the current generation (iteration). It can be called a set of chromosomes (2.3).

Figure 2.3: Graphical representation of population

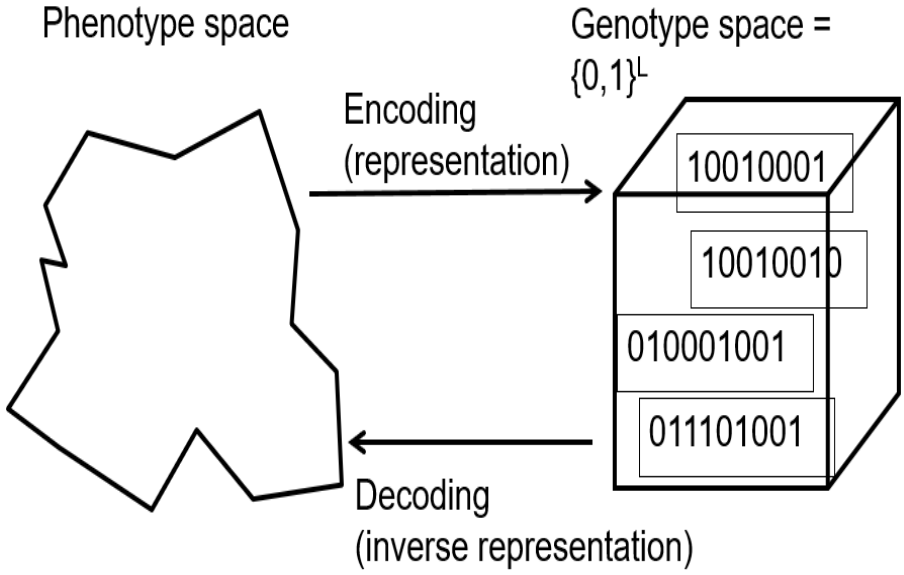


- Chromosomes is one solution to the given problem.
- Gene is one element position of a chromosome.
- Allele is the value a gene takes for a particular chromosome.

- Genotype is the population in the computation space. In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a computing system.
- Phenotype is the population in the actual real world solution space in which solutions are represented in a way they are represented in real world situations.

For simple problems, the phenotype and genotype spaces are the same. However, in most of the cases, the phenotype and genotype spaces are different. Decoding is a process of transforming a solution from the genotype to the phenotype space, while encoding is a process of transforming from the phenotype to genotype space. Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation (fig. 2.4).

Figure 2.4: Coding and decoding data in GA



For example, consider the 0/1 Knapsack Problem. The Phenotype space

consists of solutions which just contain the item numbers of the items to be picked. However, in the genotype space it can be represented as a binary string of length n (where n is the number of items). A 0 at position x represents that x-th item is picked while a 1 represents the reverse. This is a case where genotype and phenotype spaces are different.

The diversity of the population should be maintained otherwise it might lead to premature convergence. The population size should not be kept very large as it can cause a GA to slow down, while a smaller population might not be enough for a good mating pool. Therefore, an optimal population size needs to be decided by trial and error.

- Random Initialization - populate the initial population with completely random solutions.
- Heuristic initialization - populate the initial population using a known heuristic for the problem.

It has been observed that the entire population should not be initialized using a heuristic, as it can result in the population having similar solutions and very little diversity. It has been experimentally observed that the random solutions are the ones to drive the population to optimality. Therefore, with heuristic initialization, we just seed the population with a couple of good solutions, filling up the rest with random solutions rather than filling the entire population with heuristic based solutions. It has also been observed that heuristic initialization in some cases, only effects the initial fitness of the population, but in the end, it is the diversity of the solutions which lead to optimality.

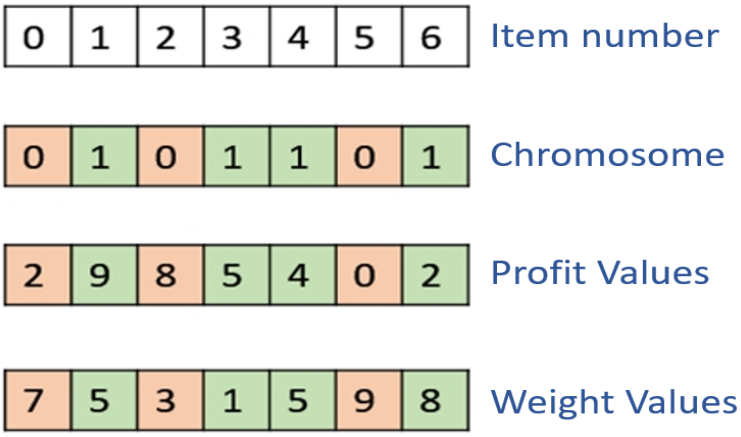
2.3 Fitness function

The fitness function is analogous to a heuristic that estimates how close a candidate is to being a solution. In general, the fitness function should be consistent for better performance. However, even if it is, there are no guarantees. This is a probabilistic algorithm. In our classification rule example, one possible fitness function would be information gain over training data.

In the optimization theory is used the concept of the objective function. If in task the objective function is positive and the goal is to maximize it, the fitness function can be used to assume the objective function (of course including binary coding by chromosomes).

The figure 2.5 shows the example of the fitness calculation for a solution of the 0/1 Backpack. It is a simple fitness function which just sums the profit values of the items being picked (which have a 1), scanning the elements from left to right till the backpack is full (capacity 15).

Figure 2.5: Backpack problem



2.4 GA stop condition

There exist three types of stop conditions for GA. First, maximum number of iterations (generations) that when the generation reaches to this predefined value, it stops and provides the best solution in the last generation.

Second, the algorithm proceeds until the best solution during the evolution process does not change to a better value for a predefined value of generations. This predefined value can be 20% or 30% of the generation number which the best solution has found so far. I.e. the algorithm reaches to a value of 200 at generation 50, then this value doesn't change for 15 generations (30% of 50), so the algorithm stops.

The second method requires good knowledge of the problem under some tests by first stop criteria and only recommended for problems with expensive computationally fitness functions. In the most studies, the fist stop criteria has been used.

Third method is when objective function has reached a certain pre-defined value.

2.5 Parents selection

Better-matched individuals have a better chance of being selected. There are used several types of a selection.

2.5.1 Fitness Proportional selection

Algorithm draws randomly s times the individual (with return) from the original P^t population to the population of parents T^t, with the probability

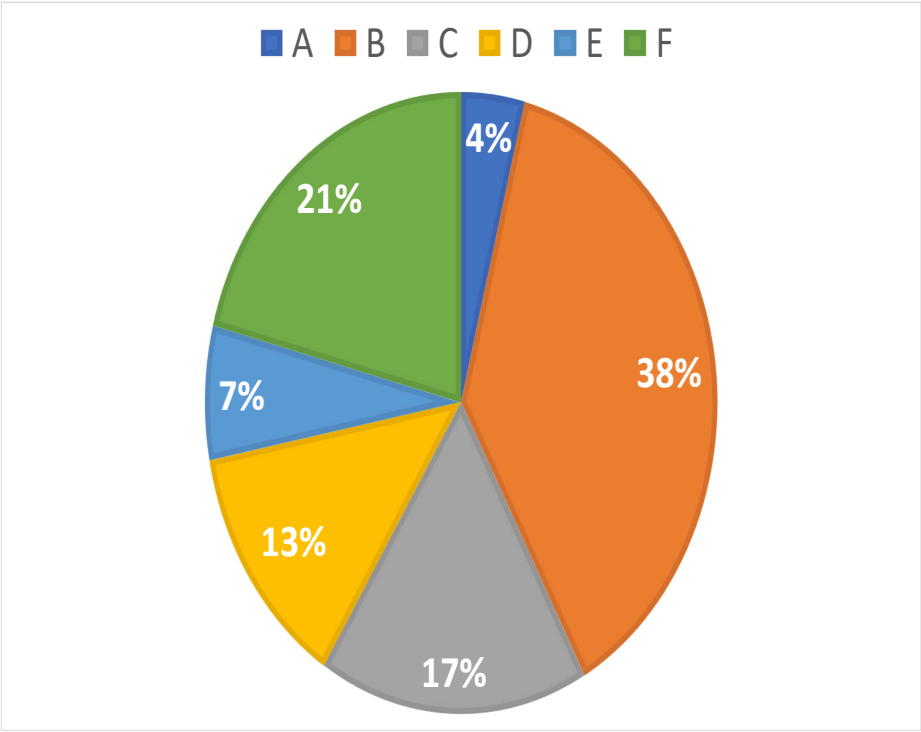
of drawing $p(y_i)$ (2.1),where f is fitness function.

$$p(y_i) = \frac{f(y_i)}{\sum_{i=1}^{s} f(y_j)} \tag{2.1}$$

2.5.2 Roulette selection

For each y_i on the interval [0,1], algorithm leaves the section with the length $p(y_i)$. Times it randomly draws a random number from the uniform distribution on the interval [0,1]. It selects the element for the parents population, on which section "encounters" draw random number r. Example of that roulette is shown in figure 2.6. Different chromosomes (A, B, C, E, E, F) with their representation in roulette depended on their fitness are a pool for selection.

Figure 2.6: Roulette selection

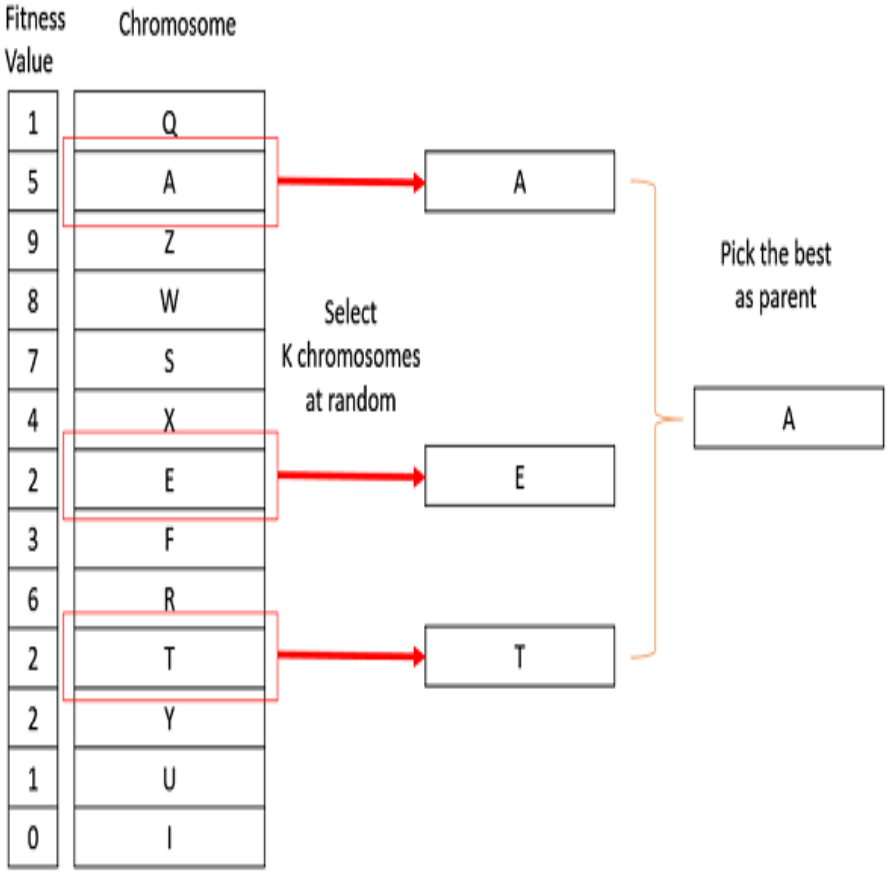


2.5.3 Tournament selection

In tournament selection algorithm select randomly q individuals (without return) from the P^t population, where tournament size (q) is an algorithm parameter than copy the best of selected individuals to the population of parents T^t. At the end it returns number (q) individuals back to P^t. It organize the tournament S times by receiving a complete population of parents T^t about the cardinality S shown in figure 2.7. The higher the value of parameter q, the greater the selective pressure.

The advantages for this selection are: no restrictions on the optimized function known from the selection proportional, it does not require knowledge of the optimized function (only need it know when one individual is better than another - it is useful in applications such as game theory) and does not require global population statistics, and is therefore easier to use implementations on parallel machines.

Figure 2.7: Tournament selection



2.5.4 Ranking selection

An attempt to circumvent the defects of proportional selection by basing the probability of selection not on the absolute value of the matching function, but on its comparison with the matching of other individuals. Algorithm sorts individuals in the population based on the value of the matching function. The individual with the highest fit receives the rank of S, the individual with the match of the second-largest rank S-1, ..., the worst-ranked individual receives 1. Algorithm can easily deal with problems such as the objective function achieves values less than 0 or requires minimization but requires sorting, with the computational cost O (S * logS), but it is usually small compared to the cost of calculating the function matching. $r(y_i)$ is ranking of individual y. The probability of selection p(y_i) is calculated solely on the basis of $r(y_i)$, and not on the basis of $f(y_i)$. If it is already know $p(y_i)$, then algorithm continue to use the roulette wheel method, or better the SRS or SUS algorithm.

2.5.5 Sexual selection

In a classical GA, chromosomes reproduce asexually: any two chromosomes may be parents in crossover. But in Sexual selection Crossover takes place only between chromosomes of opposite sex. For each generation the population is divided into males and females chromosomes by tournament selection. Thanks to that an advance instinctive speciation starts working - escape from local optima with directional forms of sexual-selective drift. It can speed up the optimization by increasing accuracy of the mapping from phenotype to fitness and grow reproductive variance in populations. Each female is con-

sidered as niche in population.

2.6 Paring parents

For every new child algorithm decides if it will result from crossover by random probability. If yes, then GA selects two parents, eg. through roulette wheel selection or tournament selection. The two parents make a child, then GA mutates it with mutation probability and add it to the next generation. If no, then GA selects only one "parent" to clone it, mutate it with probability and add it to the next population.

2.7 Crossover

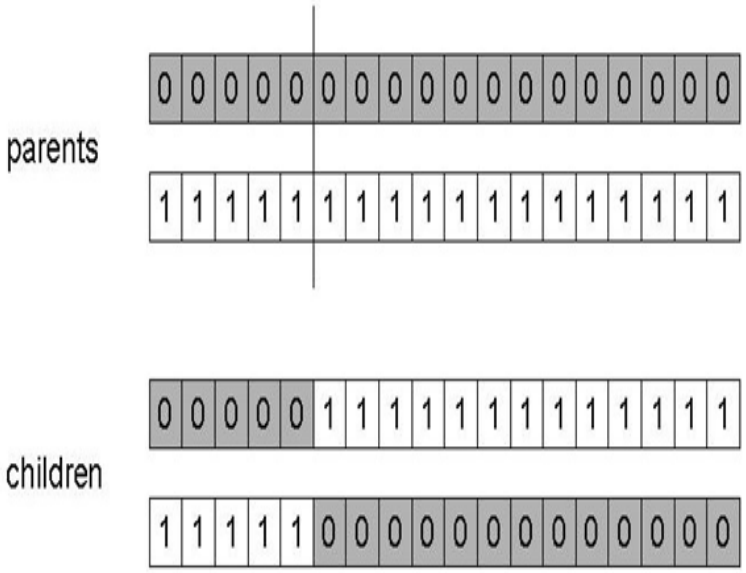
Crossover is analogical to reproduction and biological crossover. Offspring (children) are produced using the genetic material of the parents, usually applied with high probability p_c.

2.7.1 One-point Crossover

The crossover operator is a binary operator. Prior to its use, individuals in the T^t parents population formed by selection are randomly combined into pairs. For each pair with a probability p_c (p_c is an algorithm parameter, typically between 0.6 and 1) crossover is performed (shown in figure 2.8):

- Select a crossing point randomly in the range from 1 to L.
- Separate parents. Descendants are received as a result of exchanging fragments between parents.

Figure 2.8: One-point Crossover



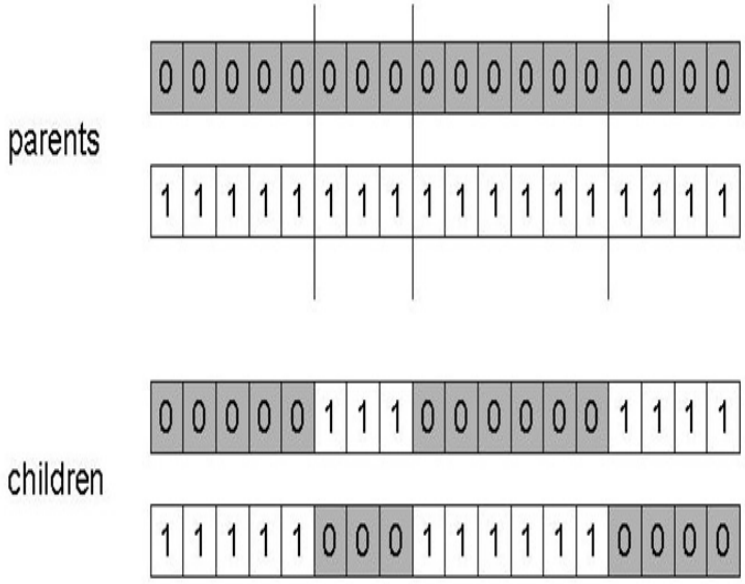
2.7.2 N-point Crossover GA

Algorithm chooses n random crossover points, splits chromosome along those points, than glue parts, alternating between parents (shown in figure 2.9).

2.7.3 Uniform crossover

Algorithm assigns 'heads' to one parent, 'tails' to the other, than flips a coin for each gene of the first child. It makes an inverse copy of the gene for the second child. Inheritance is independent of position (shown in figure 2.10).

Figure 2.9: N-point Crossover



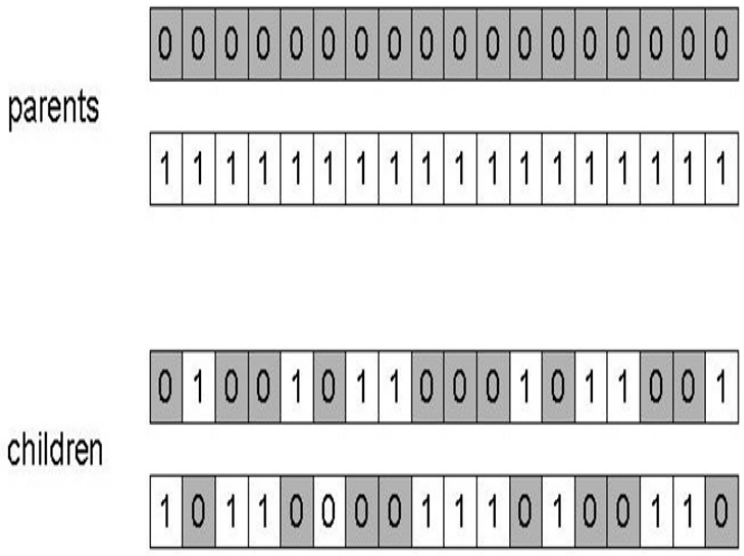
2.8 Mutation

Mutation is a small random tweak in the chromosome to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability – pm. If the probability is very high, the GA gets reduced to a random search. Mutation is the part of the GA which is related to the "exploration" of the search space. It has been observed that mutation is essential to the convergence of the GA while crossover is not.

2.8.1 Bit Flip Mutation

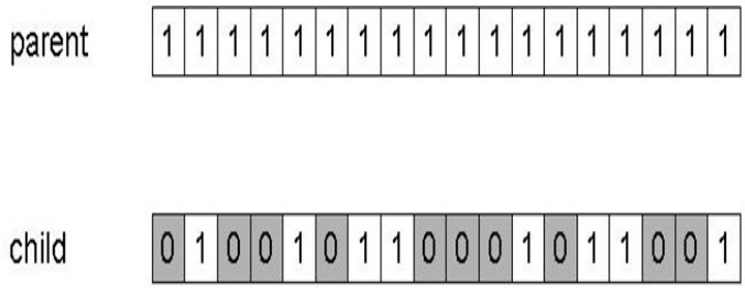
For each element of the O^t population created as a result of intersection, algorithm scans all bits. For each bit with the probability p_m (p_m is the parameter of the algorithm) it changes the value to the opposite. p_m is a

Figure 2.10: Uniform Crossover



small number (eg 0.001 or a value between $\frac{1}{S}$ and $\frac{1}{L}$). It is small because the mutation should lead to small changes in chromosomes. For implementation, for each bit in each chromosome, it draws the number r from compartment [0,1) and then negates bit, when $r < p_m$ (fig. 2.11).

Figure 2.11: Mutation



2.8.2 Random Resetting

Random Resetting is an extension of the bit flip for the integer representation. In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

2.8.3 Swap Mutation

In swap mutation, algorithm selects two positions on the chromosome at random, and interchange the values. This is common in permutation based encodings.

2.8.4 Scramble Mutation

Scramble mutation is also popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.

2.8.5 Inversion Mutation

In inversion mutation, algorithm selects a subset of genes like in scramble mutation, but instead of shuffling the subset, it merely inverts the entire string in the subset.

2.9 New Population generation

New Population generation is a decision to whom kick out and whom to keep in the next generation. The Survivor Selection Policy determines which individuals are to be kicked out and which are to be kept in the next generation.

It is crucial as it should ensure that the fitter individuals are not kicked out of the population, while at the same time diversity should be maintained in the population.

Some GAs employ Elitism. In simple terms, it means the current fittest member of the population is always propagated to the next generation. Therefore, under no circumstance can the fittest member of the current population be replaced.

The easiest policy is to kick random members out of the population, but such an approach frequently has convergence issues, therefore the following strategies are widely used.

2.9.1 Age Based Selection

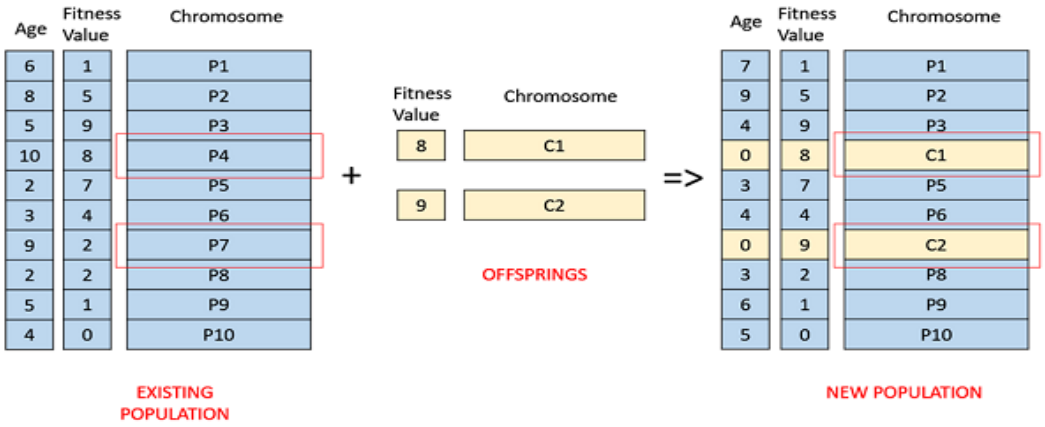
In Age-Based Selection, algorithm does not have a notion of a fitness. It is based on the premise that each individual is allowed in the population for a finite generation where it is allowed to reproduce, after that, it is kicked out of the population no matter how good its fitness is.

For instance, in the example fig. 2.12, the age is the number of generations for which the individual has been in the population. The oldest members of the population i.e. P4 and P7 are kicked out of the population and the ages of the rest of the members are incremented by one.

2.9.2 Fitness Based Selection

In this fitness based selection, the children tend to replace the least fit individuals in the population. The selection of the least fit individuals may be done using a variation of any of the selection policies described before –

Figure 2.12: Age Based Selection



tournament selection, fitness proportionate selection, etc.

For example, in the fig. , the children replace the least fit individuals P1 and P10 of the population. It is to be noted that since P1 and P9 have the same fitness value, the decision to remove which individual from the population is arbitrary.

Figure 2.13: Fitness Based Selection

2.10 Simple Genetic Algorithm (SGA)

GA developed by J. Holland, was a first type of genetic algorithms. It main application was in discrete optimization. Characterized by slow speed but good heuristic for combinatorial problems. Traditionally emphasizes combining information from good parents (crossover), it had many variants, e.g., reproduction models, operators. The summary of SGA is represented in table 2.1.

Table 2.1: SGA technical summary tableau

Representation	Binary strings
Recombination	N-point or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Specialty	Emphasis on crossover

Chapter 3

Solution implementation

3.1 Solution

The main goal of optimization method with the Use of Genetic Algorithms for Natural Language and Related Models was to created a easy to use method for already existing models. It should be as least invasive as possible which reduce a requirement knowledge of model and work to change it to minimum.

3.1.1 Method

The idea of this solution is simple - from existing model find it elements that can influence a results and try to optimize it, usually by implementing appropriate weights. It does not matter if it is single or multi-objective function, genetic algorithm can work with any of it. Of course a user need to have at least a basic knowledge of a model, especially a code or main idea of model has to be used in changed form in optimization (for example, to define a fitness function).

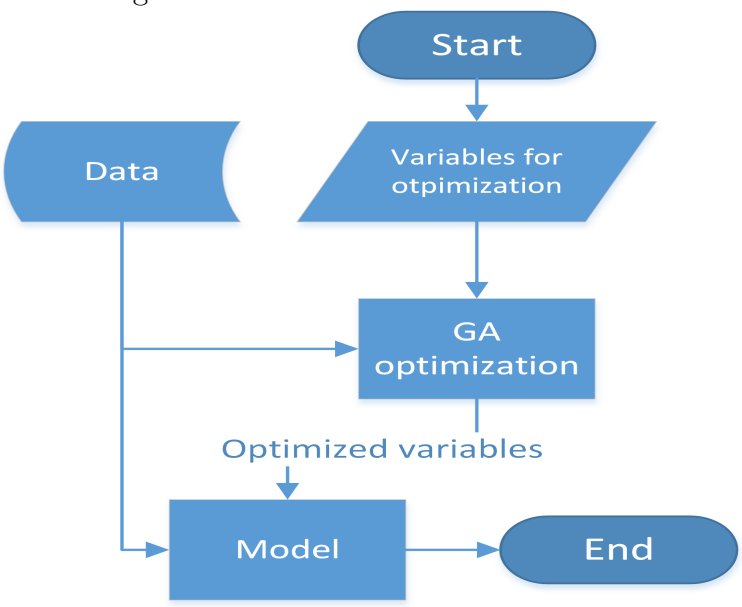
The user after deciding what part of a model he want to optimize, have to choose a GA type and starting parameters and run it with model. The problem of deciding of parameters values can be reduced by using predefined optimization sets, but user it still need to work with it.

3.1.2 Two types of implementation

Depending on model to optimize, there is two ways to use this solution.

The first case is in when we need to run a model for optimization only once or twice. It means that it is possible for us do define a fitness function and work on possible optimization without using a model every generation. In that case one final run at the end of optimization, after implementing obtained weights is enough. Sometime model have to be started also at the beginning of optimization to receive data to work. This case dramatically reduce the computing time, but instead user have to spend a much more time in setting process, because the fitness function and very often other parameters have to be prepared by user based on optimized model (fig. 3.1).

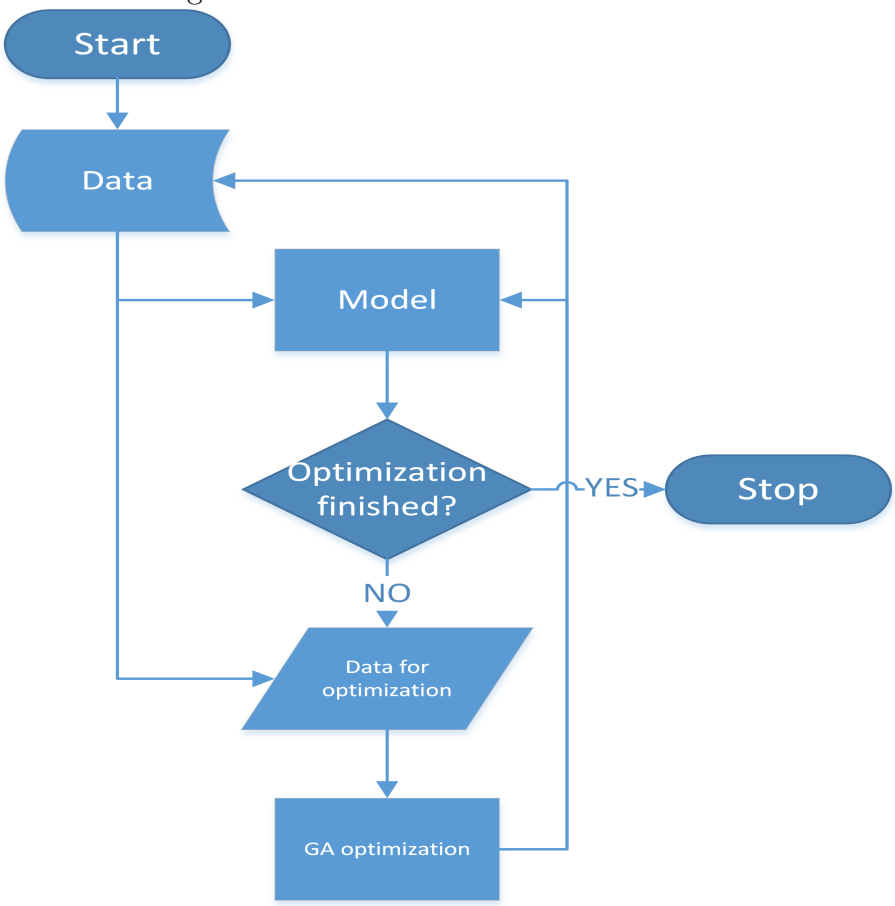
Figure 3.1: Predefined Fitness function



The second case is situation when a model have to be launched in every generation. Usually it is caused by using a model results as a fitness function

for GA. In that case a computing time is much longer (computing time of each generation contains a time for one run of a model plus GA operations), but a Setting time is reduced and a user does not have to have extensive knowledge how model works. User have to only focus on which variable needs to be optimize. This process can be speed up if the data that should be optimized is separate to the program of the model (fig. 3.2).

Figure 3.2: Model used for fitness function



3.1.3 Multi-threading

Multi-threading was implemented for two main reasons:

1. reducing computing time

2. automation of starting parameters selection

Reducing computing time is possible by separating to independent threads all processes doing the same type of operations on data. For example selection of parents, crossover and mutation. After choosing chromosomes for this operations, they can be conducted simultaneously because they do not have to wait for other results. Of course each GA operations have to finish computing all threads before it can move to next operation.

Quite often it is difficult to define the optimal starting parameters for GA. Instead just using an experience and trial and error method it is easier to automate it by running the same GA optimization, with the same data but different parameters in separate threads. It not only allows to reduce time of running the same optimization again and again, but also threads can exchange information between each other, which allows quicker reduction of inadequate parameters.

3.2 GA library

The library was created in C++ language with Open License Software guidelines in mind. It can be easily imported to any program that allows to use external C++ libraries. It was tested in C# and C++ programs running in Windows and Linux systems. It can not run separately - it requires to define a starting parameters, data and fitness functions. Also multi-threading

depends on system and processor (it use a pthread library for it). Library is user friendly - all setting options are packed in easy to use functions with big range of predefined variables, so a user just can use the options he need. The basic knowledge about Genetic Algorithms is required when using this library to understand setting process.

3.2.1 Genetic Algorithms parameters

In it possible to define every parameter of GA in the program. For number values it is possible do define them by user or to chose from some of prepared sets. For option like selection type, crossover and mutation type etc. it is possible to choose one of them from wide range of options. There are methods for user to quickly encode and decode chromosomes.

1. Genetic type
    - Simple Genetic Algorithm
    - Uniform Crossover Genetic Algorithm
    - 2-point Crossover Genetic Algorithm
    - 2-point Crossover Genetic Algorithm with Limited lifespan
    - 3-point Crossover Genetic Algorithm
    - 3-point Crossover Genetic Algorithm with Limited lifespan
    - 5-point Crossover Genetic Algorithm
    - 5-point Crossover Genetic Algorithm with Limited lifespan
    - n-point Crossover Genetic Algorithm
    - n-point Crossover Genetic Algorithm with Limited lifespan

- Genetic Algorithm with Limited lifespan
- Genetic algorithm with Sexual reproduction
- Genetic algorithm with Sexual reproduction and limited lifespan

2. Chromosome representation
    - binary
    - string
    - floating point
3. Population - for it can be used predefined sets or adjusted manually
    - Random initiation (with defined range)
    - Heuristic initiation(not recommended)
    - Mixed initiation(with defined percent of heuristics initiation)
4. Fitness function - defined by user, possible to use predefined most common formulas (like max, min etc.)
5. Stop condition
    - Max number of generation set by user or chosen from predefined sets (like 100, 300, 500, 1000, 2000, etc.)
    - Max number of generation without improvement
    - Minimal objective function value achieved
6. Parents selection (with predefined parameters or with option to set the by user)

- Fitness Proportional selection
- Roulette selection
- Tournament selection
- Ranking selection
- Sexual selection

7. Crossover (with presets probability and option to define it by user)
    - One-point Crossover
    - N-point Crossover (predefined sets: 2-point, 3-point, 5 point)
    - Uniform crossover
8. Mutation (with presets probability and option to define it by user)
    - Bit Flip Mutation
    - Random Resetting
    - Swap Mutation
    - Scramble Mutation
    - Inversion Mutation
9. New Population generation (with presets percentage values and option to define it by user)
    - Replacing the whole old generation
    - Fitness Based Selection
    - Age Based Selection

Chapter 4

Experiments

4.1 Preliminary experiment - Quantitative Learner's Motivation Model (QLMM)

QLMM is a composition of three elements, which represent the attitude of students towards the attended courses. Quantification of those elements represents the general level of learning motivation. Te three elements called interest, usefulness in the future, and satisfaction in original model have the same level of importance. However, experience shows that they are not equally important. Therefore, I performed an optimization experiment to find the best weights for these three elements.

For the experiment, I used two pieces of software. Te first was the one used in Nobuta et al. [18] , made for calculation of QLMM, which I modified by adding settable values of the weights for interest, usefulness, and satisfaction. Te second software used our own library of Genetic Algorithm to find the best weights for the three values applied in QLMM. For the optimization purpose, I used four types of genetic algorithms. Te simple genetic algorithm, GA with limited lifetime of chromosomes, GA with sexual selection in reproduction and algorithm with both limited lifetime and sexual selection.

4.1.1 Genetic Algorithm type

To perform the optimization, a Simple Genetic Algorithm (SGA) was used at first [24]. In this algorithm, the chromosomes v(t) were used in the floating-point form, with crossover (4.1 and 4.2).

$$v_1(t) = v_1(t-1)a_1 + (1-a_1)v_2(t-1) \tag{4.1}$$

$$v_2(t) = v_2(t-1)a_2 + (1-a_2)v_1(t-1) \tag{4.2}$$

where a_1, a_2, a - random numbers in the range [0..1], and the mutation is in the form (4.3):

$$v(t) = av(t-1) \tag{4.3}$$

For the selection of the size of the population, a solution has been proposed [25] in which the number of chromosomes of the population is a variable depending on the value of the evaluation function of entire population. This helps to avoid the situation in which too small population of solutions could lead to a convergence to the local optimum, and too high population to a significant increase in the calculation time. It is also assumed that, at different stages of evolution, the optimal number of population may be different. Because this parameter of chromosome age has been inserted, which is an integer in the range [0..max age], where max age is the maximum age of a chromosome and after exceeding it, chromosome will not be taken into account in further calculations. After each iteration of the algorithm, the age is increased by 1, while during the selection - chromosomes with higher age

than the maximum are rejected. In this situation, there is no need to define the selection algorithm because the age of a chromosome is determined by the objective function. Te age of a chromosome determines which chromosomes will take part in the creation of new solutions. In the calculation of the lifetime of a chromosome, a linear assignment was used in the form 4.4.

$$MinLT + (MaxLT - MinLT)\frac{eval(v_i(t)) - AbsFitMin}{AbsFitMax - AbsFitMin} \tag{4.4}$$

where:

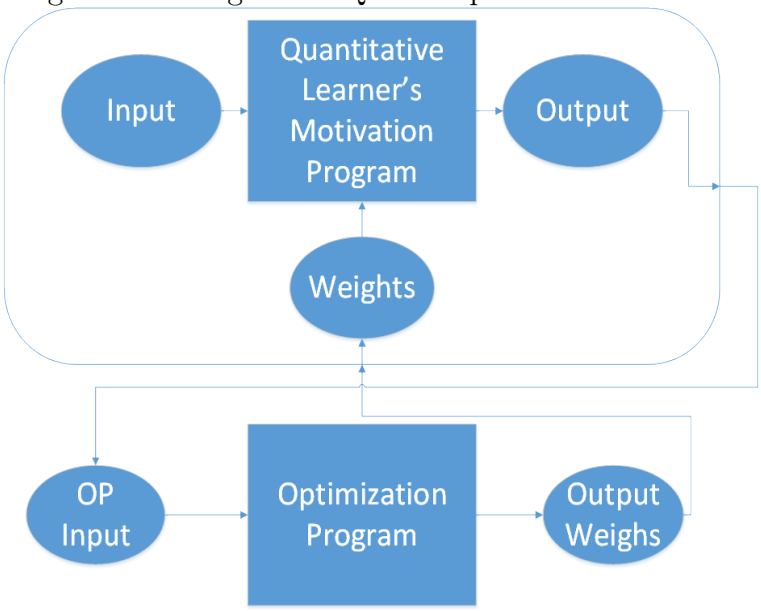
- MinLT – minimum possible age of a chromosome,
- MaxLT – maximum possible age of a chromosome,
- eval(vi) – value of the adaptation function of a chromosome,
- AbsFitMin – the lowest value of the adaptation function found during all previous iterations,
- AbsFitMax – the largest value of the adaptation function found during all previous iterations.

4.1.2 Experiment

The data used for the experiments was collected from questionnaires for nine different courses for undergraduate students of 1st to 3rd year in the Kitami Institute of Technology. There was 5,040 answers collected at the beginning and the end of the school year. Range values for the three elements of QLMM were from 1 to 5.

For the experiment, I used two pieces of software. Te first one, written in C#, was developed previously for the quantification of learner's motivation [18]. Te only change was made for adding settable values of weights for the three values of QLMM. Te second software, written in C#, used our own library of Genetic Algorithm made in C++ [27]. The function of the second software was finding the best weights for QLMM. Te diagram of the solution is showed in Fig. 4.1.

Figure 4.1: Diagram of QLMM optimization solution



Solution steps:

1. Uploading data from excel files containing questionnaire answers to a program.

2. Calculating QLMM in the first program without any weights for interest, usefulness in the future and satisfaction.

3. Uploading results from the first program to the second.

4. Calculating weights for the three elements of QLMM with the use of genetic algorithm.

5. Calculating QLMM again in the first program with weights received from the second program.

6. Uploading results from the first to the second program again.

7. Repeating steps 4 to 6 until the end of all generations of used GA

I used four different types of genetic algorithms for optimization. Te basic one was a simple genetic algorithm (SGA). Te second algorithm used the limited lifetime of chromosomes. Te third one was SGA with sexual selection in reproduction [28]. Te last algorithm used both solutions used in the second and third type of GA. All of them had the same starting parameters.

- The type of representation floating-point
- Crossover 2-point, with a probability 0 9
- The probability of mutation 0 1,
- Te size of population 100,
- Te number of algorithm iterations 2000
- Fitness function (4.5)

$$f = max(A, P) \tag{4.5}$$

Table 4.1: Improvement in results after optimization with GA

Genetic algorithm type	Improvement in Precision	Improvement in Recall
Simple Genetic Algorithm (SGA)	6%	7%
GA with limited lifetime	13%	15%
GA with sexual selection	11%	11%
GA with limited lifetime and sexual selection	17%	21%

where: f- fitness function, A - accuracy, p - precision.

All the results were compared to the output from basic program without optimization of the weights. All four solutions showed improvement in relation to original results. In Table 4.1, I showed the percentage values of improvement of precision and recall.

4.1.3 Discussion

From results of experiment it is visible that optimization with GA worked with different results depending on used type of algorithm. To make results of different types of GA relevant to each other, all parameters not typical for chosen types was equated to the same values (e.g number or generations, population size etc.). In all cases optimization was successful, but more sophisticated type of GA gave better results. The best optimization result was achieved with a use of Genetic Algorithm with limited lifetime (lifespan) and sexual selection. Thank to testing different types of GA, it was possible to find the best optimization solution for a QLMM. Thanks to the prepared software (GA library) after preliminary adjustment of the model for optimization purpose, it was possible to easy test different GA types, just by changing some parameters in a program. But the best values for starting

parameters were chosen by user, so there was possibility that their were not optimal. In next experiments, the selection of optimal starting parameters of GA will be proceed by multithreading function in GA library.

4.1.4 Conclusions

In the experiments, the program based on the research of QLMM used weights calculated in a separate program which used different types of a genetic algorithm to find optimal weights for interest, usefulness and satisfaction from QLMM. All the four solutions resulted in improvements of precision and recall used for the calculation of optimization of motivation model. Te best solution was reached by means of the genetic algorithm with sexual selection in reproduction of chromosomes and their limited lifetime. Te way of using a program allows for reusing it after a proper adjustment in other research for optimization of, e.g., binary classifiers [29]. It means that the proposed solutions successful, efficient and possible to use in other cases.

4.2 Main experiments - Optimization of Support Vector Machines for the problem of pattern recognition in Natural Language Data

This experiment aimed to optimize implementation of Support Vector Machines for the problem of pattern recognition in natural language data. Original SVMs program was created in C++. For this application there were tested numerous different types of Genetic Algorithms with different number of generations, weight range and starting parameters.

4.2.1 Dataset

The dataset was focused on detection of the Cyberbulling (using technology to ridicule or humiliate others). The data was collected by Internet patrol (annotated by experts) from unofficial school forums (BBS). It was provided by a human rights center in japan (Mie prefecture), according to the definition by Japanese ministry of Education (MEXT). There was 1,490 harmful and 1,508 non-harmful entries.

Example:

Jap: Shinde kureeee, daibu kiraware-mono de yuumei,
subete ga itaitashii
Eng: Please, dieeee, you're so famous for
being disliked by everyone, everything in you is so pathetic

4.2.2 Optimization

The SVMs model use learning and test data stored in separate files (fig. 4.2, second column was added to figure, to show more data).

Figure 4.2: Part of a file with data for a model

```
+1 201:1 3148:1 3983:1 4882:1
-1 874:1 3652:1 3963:1 6179:1
-1 1331:1 3084:1 3957:1 4514:1
-1 643:1 1870:1 3957:1 4367:1
-1 96:1 3332:1 3956:1 6018:1
+1 1168:1 3318:1 3938:1 4481:1
+1 350:1 3082:1 3965:1 6122:1
-1 42:1 2145:1 3963:1 5876:1
-1 99:1 3057:1 3957:1 5838:1
-1 1005:1 3488:1 3957:1 4486:1
+1 690:1 2401:1 3956:1 6297:1
-1 1117:1 3481:1 3963:1 5784:1
-1 96:1 2977:1 3963:1 5090:1
+1 561:1 1676:1 3983:1 5335:1
-1 537:1 2677:1 3957:1 4047:1
+1 657:1 2248:1 3957:1 5378:1
-1 561:1 1550:1 3963:1 6171:1
+1 569:1 1863:1 3957:1 4080:1
-1 962:1 3033:1 3963:1 4048:1
-1 1252:1 2602:1 3957:1 5446:1
-1 1091:1 2845:1 3965:1 4569:1
+1 561:1 2728:1 3957:1 4483:1
-1 871:1 1880:1 3969:1 4048:1
-1 257:1 3012:1 3963:1 6467:1
-1 1011:1 3330:1 3965:1 5267:1
+1 96:1 3226:1 3957:1 4432:1
-1 1105:1 2528:1 3963:1 4208:1
-1 96:1 2977:1 3963:1 5756:1
-1 392:1 3495:1 3963:1 5579:1
-1 153:1 2688:1 3963:1 4368:1
-1 55:1 2911:1 3963:1 5416:1
+1 96:1 3753:1 3955:1 6238:1
-1 96:1 3530:1 3963:1 6460:1
```

In this file each entry is in separate line and starts +1 or -1 that represents a positive (+1) and negative (-1) connotation. It is followed by set of numbers

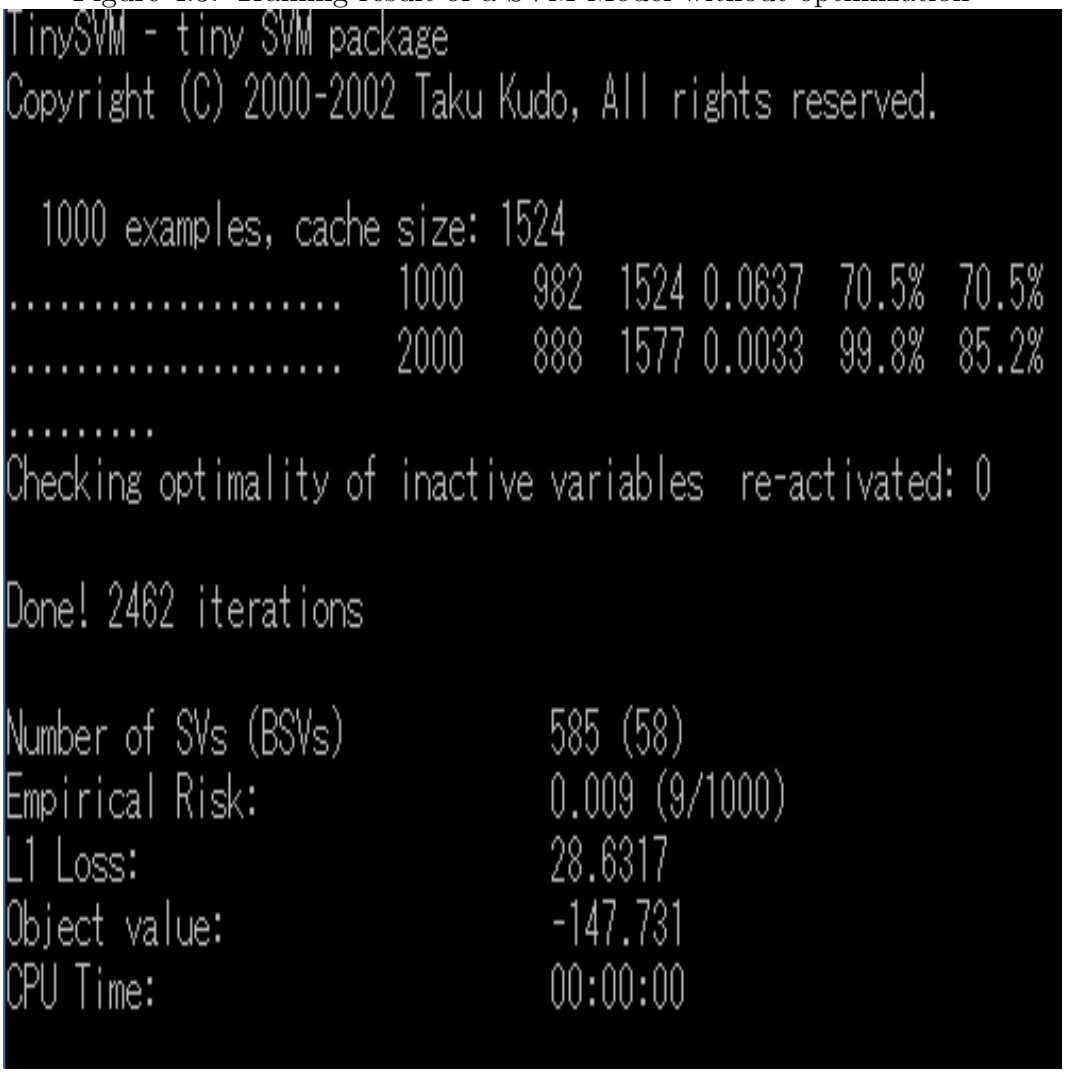
divided by colons. Each set represents a word used in the entry. First number represents index of the word, the number after colon is a weight assigned always as 1 (model prepared and described in).

The idea of optimization in this case was to change weights range from only one to bigger diversity. After some preliminary experiments and research in literature, best results were achieved with three ranges:

- from 1 to 3 (1,2,3),
- from 1 to 5 (1,2,3,4,5)
- from 1 to 10 (1,2,3,4,5,6,7,8,9,10)

Original program for a model was created by Taku Kudo in C++ and is called *TinySVM - tiny SVM package* (source [46]). Program is launched in console, and all results are also showed in console. Model use two data files. First for training a model, second for testing a results. Training and testing are invoked by two separate commands. In fig. is shown a result of a training without optimization (fig. 4.3).

Figure 4.3: Training result of a SVM Model without optimization



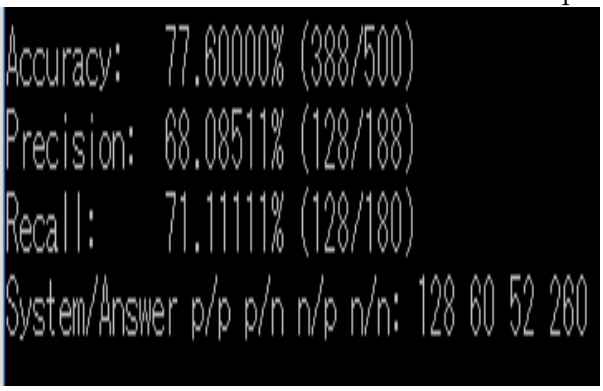
Description of the training result (each line described):

- Name of the program
- Copyrights of the program
- Number of examples and the size of the cache assigned to a program

- Messages printed during the training iterations
  - 1st column: One "." means 50 iterations
  - 2nd column: Represents the total iterations processed
  - 3rd column: Represents the size of current working set. It will become smaller during the shrinking process
  - 4th column: Represents the current cache size
  - 5th column: Max KKT violation value - if this value reaches termination-criterion (default 0.001), it means that the 1st stage of the learning process has completed
  - 7th column: Cache hit rate during last 1000 iterations
  - 8th column: Cache hit rate during all iterations
- Number of Support Vectors (Bunded Support Vectors)
- In general, the risk R(h) cannot be computed because the distribution P(x,y) is unknown to the learning algorithm (this situation is referred to as agnostic learning). However, it can be computed an approximation, called empirical risk, by averaging the loss function on the training set
- L1-norm loss function is also known as least absolute deviations (LAD), least absolute errors (LAE). It is basically minimizing the sum of the absolute differences (S) between the target value (Y_i) and the estimated values $f(x_i)$
- A vector of predicted object values

After training the model, it have to use a test data to receive information about efficiency of a model. The result is shown in fig. 4.4.

Figure 4.4: Test result of a SVM Model without optimization



Description of test result (each line described):

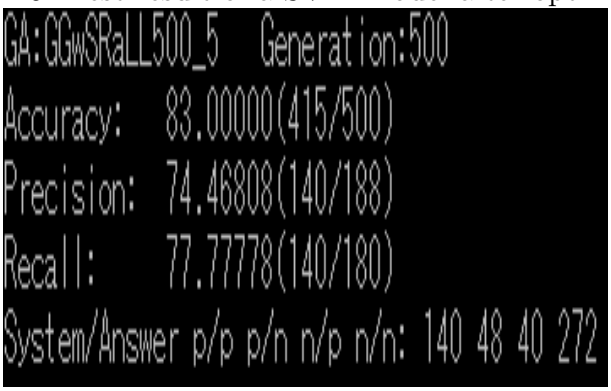
- Accuracy (equation 1.12) - percent value, number of p/p(TP) + n/n(TN), number of all answers
- Precision (equation 1.13) - percent value, number of p/p (TP), p/p(TP) + p/n(FP)
- Recall (equation 1.14) - percent value, number of p/p (TP), p/p(TP) + n/p(FN)
- Exact number of p/p (TP), p/n (FP), n/p (FN), n/n (TN)

The three values (Accuracy(A), Precision(P), Recall(R)) are used for optimization by GA. Thanks to possibility to change the weights in separate data files, model does not need to be adjusted for optimization. The results of model computation was used as fitness function for GA (4.6).

$$f = max(A, P, R) \tag{4.6}$$

It made computation time longer (Second case of the solution), but allowed to avoid changing of original code of the model. That's why additional program was created that implements a GA library and invokes SVMs script. Results of experiment are described in next section, but in figure 4.5 is shown the result of optimization in the console (there is always shown only result of the final generation).

Figure 4.5: Test result of a SVM Model after optimization



Description of training result (each line described):

- Shortcut describing used algorithm (explanation in tables 4.2 and 4.3), number of generations used by GA
- Accuracy after optimization (equation 1.12) - percent value, number of p/p(TP) + n/n(TN), number of all answers
- Precision after optimization(equation 1.13) - percent value, number of p/p (TP), p/p(TP) + p/n(FP)
- Recall after optimization (equation 1.14) - percent value, number of p/p (TP), p/p(TP) + n/p(FN)

- Exact number of p/p (TP), p/n (FP), n/p (FN), n/n (TN) after optimization

For the purpose of generating optimal starting parameters another that defined in experiment (population size, type of crossover and mutation probability, lifespan length and some of selection types) multi threading automatic selection was used. The biggest amount of different parameters was tested for three first genetic algorithms described in table 4.2 (total 160 separate threads per algorithm, run by 8 in the same time). From that information the four best sets were chosen. There were used as base sets for other algorithms. Each of the base set was running with five different sets with range close to base sets and two compliantly random. For final optimization was chosen the eight best different sets and run in the same time on separate threads one more time. The result are described in table 4.4.

Figure 4.6: Optimization process

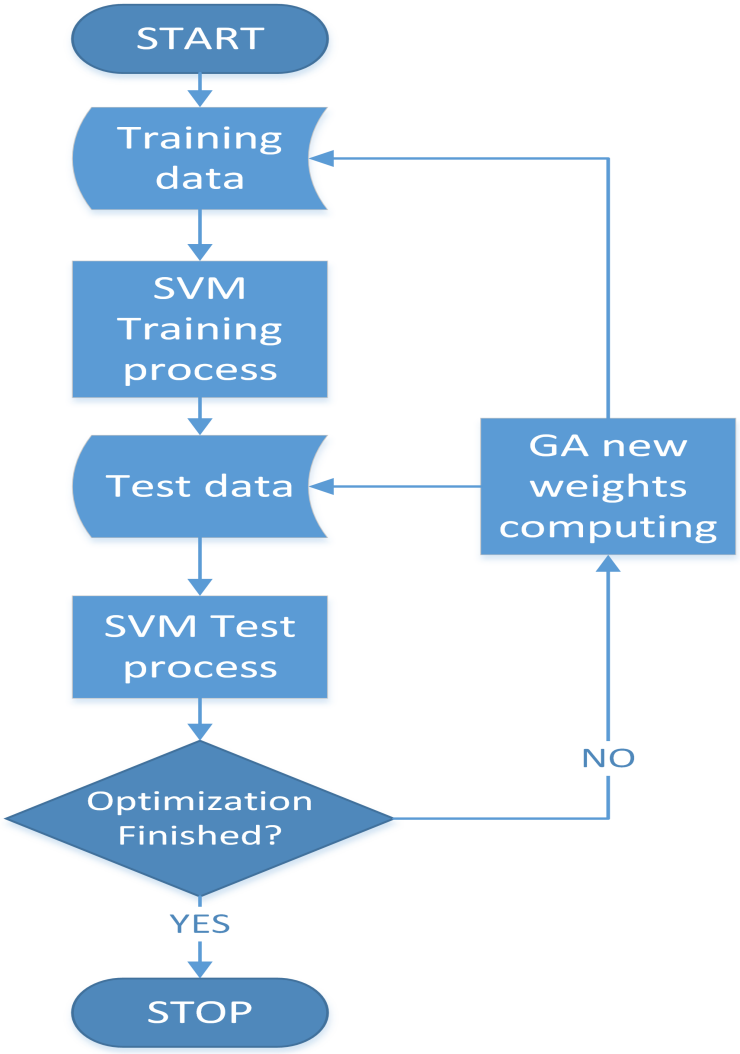


Table 4.2: Genetic Algorithm shortcuts explanation

Name	**Shortcut**
Simple Genetic Algorithm	SGA
Uniform Crossover Genetic Algorithm	UCGA
2-point Crossover Genetic Algorithm	2pCGA
3-point Crossover Genetic Algorithm	3pCGA
5-point Crossover Genetic Algorithm	5pCGA
Genetic Algorithm with with limited life	GAwLL
Genetic algorithm with sexual reproduction	GGwSR
GA with sexual reproduction and limited life	GGwSRaLL

4.2.3 Experiment

In the experiment several different types of Genetic Algorithms were used. In table 4.2 are names of all used types with distinguish shortcuts for each of them. This shortcuts will be used in rest of description of experiment.

To distinguish examples using different starting parameters for each GA, additional informations was added to shortcuts. The structure is described in formula 4.7.

$$NameofGA(Shortcut)+NumberofGenerations(NoG)+weightRadius(WR) \tag{4.7}$$

The examples are described in table 4.3. Each of GAs used parameters as described in optimization of experiment. The experiment was carried out on the same type of machine and the same data for all types of GAs.

Table 4.3: GA shortcuts extensions examples

Shortcut	**NoG**	**WR**
SGA100_3	100	1-3
SGA100_5	100	1-5
SGA100_10	100	1-10
SGA300_3	300	1-3
SGA300_5	300	1-5
SGA300_10	300	1-10
SGA500_3	500	1-3
SGA500_5	500	1-5
SGA500_10	500	1-10
SGA1000_3	1000	1-3
SGA1000_5	1000	1-5
SGA1000_10	1000	1-10
UCGA100_3	100	1-3
2pCGA300_5	300	1-5
3pCGA500_10	500	1-10
GAwLL500_3	500	1-3
GGwSR_100_5	100	1-5
GGwSRaLL_100_10	100	1-10

In graph 4.7 is shown best results of optimization. It can be seen that in the most cases minimal amount of generations to shown an improvement was 300, but best results were in 500 generations. There was not any noticeable changes in case of bigger number of generations (in most cases the optimal optimization result was achieved between 420 to 460,470 generations), but still experiment was proceeded on 1000 generations to avoid possible local maxima. The most suitable weight range was from 1 to 5, followed by range from 1-10 and 1-3.

In graph 4.8 is shown progress of improving a fitness value through 500 generations for Genetic Algorithm with Sexual Selection and Limited Lifespan.

Table 4.4: Results of optimization - chosen results

Algorithm	Acc.	Prec.	Rec.	Change Acc.	Change Prec.	Change Rec.
Baseline	77,60%	68,09%	71,11%	-	-	-
SGA300_5	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
SGA500_5	78,80%	69,68%	72,78%	1,20%	1,60%	1,67%
SGA500_10	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
SGA1000_5	78,80%	69,68%	72,78%	1,20%	1,60%	1,67%
UCGA500_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
UCGA500_5	80,40%	71,80%	75,00%	2,80%	3,72%	3,89%
UCGA500_10	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
2pCGA500_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
2pCGA500_5	78,80%	69,68%	72,78%	1,20%	1,60%	1,67%
2pCGA500_10	78,40%	69,15%	72,22%	0,80%	1,06%	1,11%
3pCGA500_3	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
3pCGA500_5	81,20%	72,87%	76,11%	3,60%	4,79%	5,00%
3pCGA500_10	80,00%	70,74%	73,89%	2,40%	2,66%	2,78%
GGwSR500_3	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
GGwSR500_5	81,60%	73,40%	76,67%	4,00%	5,32%	5,56%
GGwSR500_10	80,20%	71,28%	74,44%	2,60%	3,19%	3,33%
GGwSRaLL500_3	80,20%	71,28%	74,44%	2,60%	3,19%	3,33%
GGwSRaLL500_5	83,00%	74,47%	77,78%	5,40%	6,38%	6,67%
GGwSRaLL500_10	81,20%	72,87%	76,11%	3,60%	4,79%	5,00%

4.2.4 Discussion

Similar like in preliminary experiment, results showed that different types of GA have a different effectiveness in optimization of the problem. In main experiment was tested bigger number of GA types and additionally with different sets of parameters for GA (Generations and range for weights). It allowed to check how that different values influence final results. It was also important to make that values constant (especially number of generations) for easier comparison of results between different types of GA.

Important part was finding optimal other starting parameters (for ex-

ample population size, types of selection, mutation probability etc.) with a use of GA library. It showed that multithreading function is useful for optimization and simplifies that process (especially for not advanced users).

Genetic Algorithm was successful with overcoming a local maxima, what can be seen in graph 4.8 showing improvement of fitness function values during generations.

Last thing observed during experiment was different scale of optimization and values of starting parameters compared to preliminary experiment. it proved that each case of optimization require different approach and with the use of GA library the work for this part can by greatly reduced by different already created methods and optimization functions in that library.

4.2.5 Conclusions

Optimization was successful, with different scale of improvement based on used genetic algorithm and starting parameters, with the highest achieved improvement of over 6 percentage points of recall comparing to baseline and reaching 78% with a use of Genetic Algorithm with Sexual selection and limited lifespan (GGwSRaLL). GA library after preliminary adjustments for optimization purpose of SVMfor the pattern recognition in NLP was easy to use with different types of GAs. Finding optimal starting parameters with the use of multithreading was also successful and efficient. All experiments data are included in this work.

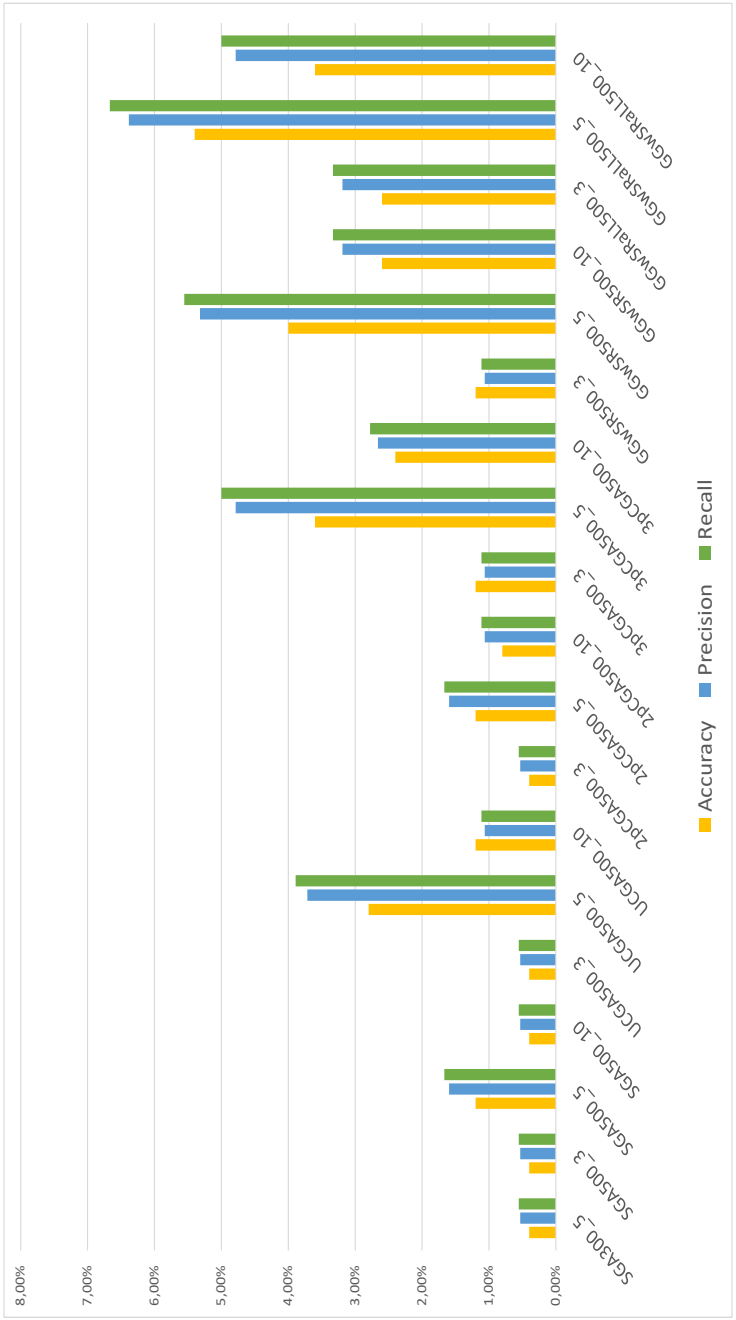
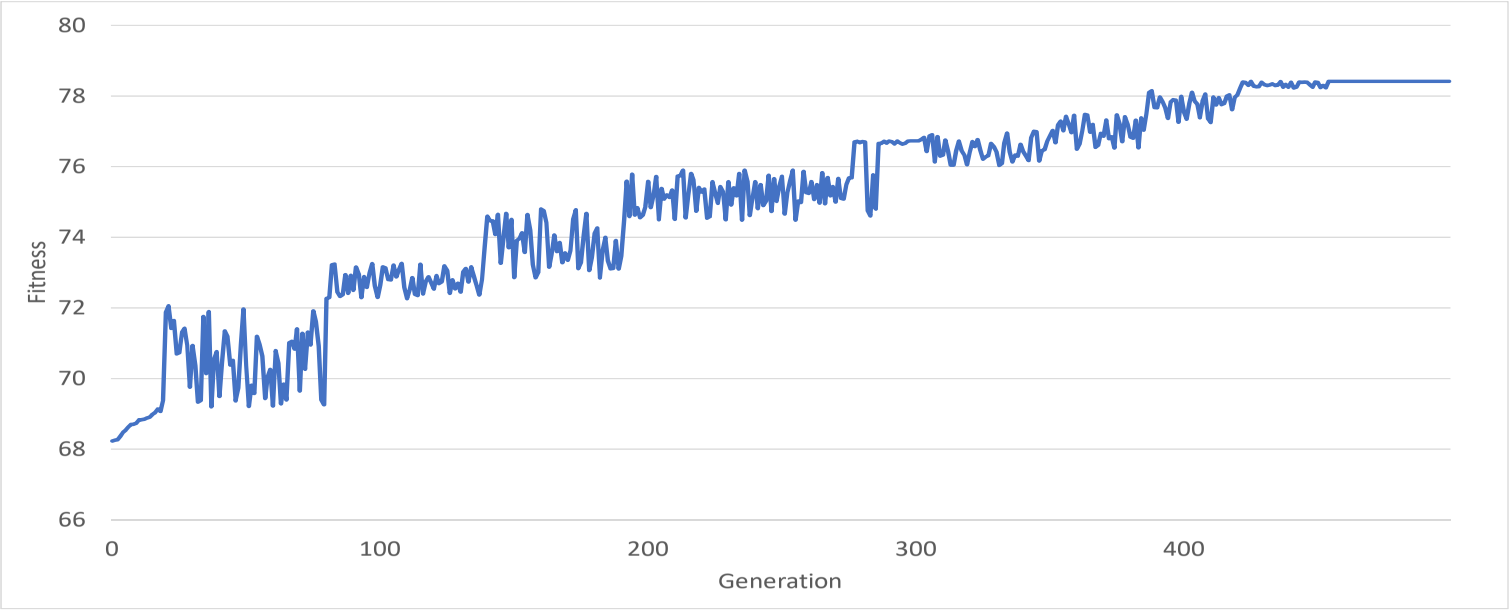


Figure 4.7: Experiment results - chosen GAs

Figure 4.8: Fitness improvement during all generations



Chapter 5

Conclusions and Future Work

Conclusions of the dissertation and thoughts about possible future improvement of the optimization method and GA library.

5.1 Conclusions

I pursued this research with hypothesis that with correctly use of Genetic Algorithms it is possible to optimize existing Natural Language and related models without big interference in their existing codes.

In part of the chapter 1 I described my survey of a existing use of GAs in NLP. I found out that not numerous but they exist examples of it. In most cases a Genetic Algorithm was used for optimization of the one part of the solution. This confirmed me in the belief that it is worth testing my method in NLP models.

In chapter 3 I described a my proposed method of optimization with GAs and library wrote in C++ to use with other solutions. The most important part of my method was to make it easy to use with a small amount or even none of modifications for existing models. My solution allows to use a big range of different types of GAs with easily to set starting parameters and flexible successor functions. The use of multi-threading allows to speed up computing time. It also allows to find the optimal parameters for used

GA, thanks to running simultaneously the same algorithm with different parameters and comparing it results in real time.

The library for optimization was written in C++ with keeping guidelines for open source software, which allows to use it in Windows and Linux systems. User can in simple way define GAs types and their parameters, turn on or off multi-threading (in current version up to 8 separate processes) and connect it to his program without big changes in existing code.

In chapter 4 are included all conducted experiments. The preliminary experiment was on optimization of Quantitative Learner's Motivation Model. The goal of this experiment was to optimize the formula for prediction of learning motivation by means of different weights for three values: interest, usefulness in the future and satisfaction. For this optimization, an application in C# using GA library was created. Data sets for the experiments were acquired from questionnaires enquiring about the above three elements in actual university classes. The results of the experiment showed improvement in the estimation of student's learning motivation up to over 17 percentage points of Fscore. That experiment was conducted for prove the usefulness of proposed optimization method. It did not used multi-threading, and amount of GAs types was limited, but thanks to receiving successful results confirmed my belief that it is worth using my solution.

The main experiment showing all features of my method. For optimization was chosen Support Vector Machines (SVM) for the problem of pattern recognition in natural language data. Existing solution that use machine learning with a big amount of data that can be easily modified was very helpful on reducing work on requirement implementation of GA library. Experiment was focused on finding best GAs type and parameters for SVM.

There were tested different types of GA, number of generations, weight range and starting parameters but always on the same data. The multi-threading was used for speeding up operations like: selection, crossover and mutation. It was also implemented in choice of starting parameters like population size. Optimization was successful, with different scale of improvement based on previously mentioned conditions, with the highest achieved improvement of over 6 percentage points of recall comparing to baseline and reaching 78%.

Experiment proved usefulness of optimization by GA. This method can be used for different Natural Language and related models with minimal work for adjusting it to GA library. Each model required separate approach, but after preliminary adjustment it can work with any types of data that model used. In some cases it is enough to optimize a model one time (of course using the most optimal GA and parameters), but in some cases optimization have to be conducted every time when the data is changed. Because of the structure of this solution, the computing time is strictly connected to computing time of the model. The possible speeding up depends on fitness function of GA - if it is possible to define it outside of used model, the computing time can be drastically reduced. But in situation when fitness function is calculated from result of model running, it is need to run it after every generation of GA.

5.2 Future work

I am planing future development of GAs library: to add more pre-set types of GAs, different selections, crossovers and improve multi-threading (to reduce time of computation and add more flexibility to data exchange between threads). Also includes a tool for monitoring an optimization during com-

puting with it numerical and graphical representation.

I also plan to implement my solution in other Natural Language Models.

References

[1] Kenji Araki, Michitomo Kuroda. 2006. Generality of spoken dialogue system using SeGA-IL for different languages. *Proceedings of the IASTED*, San Franisco, 2006.

[2] Yasutomo Kimura, Kenji Araki, Yoshio Momouchi, and Koji Tochinai. 2001. Spoken dialogue processing method using inductive learning with genetic algorithm. *Systems and Computers in Japan*, Vol. 35, No. 12, 2004.

[3] Donn Morrison, Ruili Wang, Liyanage C. De Silva. 2007. Ensemble methods for spoken emotion recognition in call-centres. *Speech Communication*, Elseviere, vol. 49, pp. 98-112, 2007.

[4] Kazuhiro Nakadai, Shunichi Yamamoto, Hiroshi G. Okuno,Hirofumi Nakajima, Yuji Hasegawa, Hiroshi Tsujino. 2008. A Robot Referee for Rock-Paper-Scissors Sound Games. *EEE International Conference on Robotics and Automation*, Pasadena, pp. 3469-3474, 2008.

[5] Kazuhiro Nakadai, Toru Takahashi, Hiroshi G. Okuno, Hirofumi Nakajima, Yuji Hasegawa, Hiroshi Tsujino. 2010. Design and Implementation of Robot Audition System 'HARK' — Open Source Software for Listening to Three Simultaneous Speakers. *Advanced Robotics*, vol. 24, Issue 5-6, pp. 739–761, 2010.

[6] V. Fischer, J. Fischer, H. Niemann. 1996. An Algorithm for Any-Time

Speech Understanding. *Proceedings of the 3rd German-Slovenian Workshop on Speech and Image Analysis*, Ljubljana, 1996.

[7] Noraini Seman, Zainab Abu Bakar, and Nursuriati Jamil. 2013. Improving Speech Recognizer Using Neuro-genetic Weights Connection Strategy for Spoken Query Information Retrieval. *Information Retrieval Technology*, Lecture Notes in Computer Science, vol. 8281, pp. 528-539, 2013.

[8] Calkin S. Montereo, Kenji Araki. 2007. Unsupervised language independent genetic algorithm approach to trivial dialogue phrase generation and evaluation. *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, Vol.4592, pp.388-394, 2007.

[9] Ruli Manurung, Graeme Ritchie, Henry Thompson. 2012. Using Genetic Algorithms to Create Meaningful Poetic Text. *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 24, issue 1, pp. 43-64, 2012.

[10] Chris Mellish, Mick O'Donnell, Jon Oberlander, Alistair Knott. 1998. An Architecture for Opportunistic Text Generation. *Proceeding of the 9th International Workshop on Natural Language Generation*,1998.

[11] Hisar Maruli Manurung. 2003. An evolutionary algorithm approach to poetry generation. *Doctoral Thesis*, Institute for Communicating and Collaborative Systems,School of Informatics University of Edinburgh, 2003.

[12] Neil McIntyre, Mirella Lapata. 2010. Plot Induction and Evolutionary Search for Story Generation. *Proceedings of the 48th Annual Meeting of*

the Association for Computational Linguistics, Stroudsburg, pp. 1562-1572, 2010.

[13] Hiroshi Echizen-ya, Kenji Araki, Yoshio Momouchi, Koji Tochinai. 1996. Machine Translation Method Using Inductive Learning with Genetic Algorithms. *Proceeding of the 16th COLING*, Vol. 2, pp. 1020-1023, 1996.

[14] Hiroshi Echizen-ya, Kenji Araki, Yoshio Momouchi, Koji Tochinai. 2004. Machine Translation Using Recursive Chain-Link-Type. *Systems and Computers in Japan*, Vol. 35, No. 2, pp. 1-15, 2004.

[15] Hiroshi Echizen-ya, Kenji Araki, Yoshio Momouchi, Koji Tochinai. 2002. Study of practical effectiveness for machine translation using recursive chain-link-type learning. *Proceeding of the 19th COLING*, Vo. 1, pp. 1-7, 2002.

[16] Colin R. Reeves. 1995. A Genetic Algorithm for Flowshop Sequencing. *Computers & Operations Research*, vol. 22, issue 1, pp. 5-13, 1995.

[17] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, Timm Euler. 2006. YALE: Rapid Prototyping for Complex Data Mining Tasks. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, pp. 935-940, 2006.

[18] Nobuta Y., Masui F., Ptaszynski M, 2015.Modeling Learning Motivation of Students Based on Analysis of Class Evaluation Questionnaire. *Technical Transactions*, vol 2-M, 193–201.

[19] Ekbal, A., Saha, S., 2016. Simultaneous feature and parameter selection using multiobjective optimization: application to named entity recogni-

tion. *International Journal of Machine Learning and Cybernetics*, Volume 7, Issue 4, 597–611.

[20] Calkin S. Montereo, Araki K., 2007. Unsupervised language independent genetic algorithm approach to trivial dialogue phrase generation and evaluation. *Lecture Notes in Computer Science, Springer*, Berlin, Heidelberg, Vol. 4592, 2007, 388–394.

[21] Manurung R., Ritchie G., Tompson H., 2012. Using Genetic Algorithms to Create Meaningful Poetic Text. *Journalof Experimental & Teoretical Artifcial Intelligence*, Vol. 24, Issue 1, 43–64.

[22] Manurung H.M., 2003. An evolutionary algorithm approach to poetry generation. *Doctoral Tesis*, Institute for Communicating and Collaborative Systems, School of Informatics, University of Edinburgh.

[23] McIntyre N., Lapata M., 2010. Plot Induction and Evolutionary Search for Story Generation. *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Stroudsburg, 1562–1572.

[24] Goldberg D.E., Holland J.H., 1988. Genetic algorithms and machine learning. *Machine learning*, vol. 3(2), 95–99.

[25] Langdon WB, Poli R., 2002. Foundations of genetic programming. *Springer*.

[26] Melanie M., 1999. An introduction to genetic algorithms. *Cambridge*, Massachusets London, England, Fifth printing.

[27] Ladd SR., 1995. Genetic algorithms in C++. *Hungry Minds*, Incorporated.

[28] Deslauriers W.A. Asexual Versus Sexual Reproduction in Genetic Algorithms. *Carleton University.*

[29] Wu Ch. H., Tzeng G.H., Goo Y.J., Fang W.C., 2007. A real-valued genetic algorithm to optimize the parameters of support vector machine for predicting bankruptcy. *Expert Systems with Applications*, Vol. 32, Issue 2, 397–408.

[30] Keller J.M., Kopp T., 1987. Application of the ARCS model of motivational design. M. Reigluth (Ed.), *Instructional theories in action: Lessons illustrating selected theories and models*, Lawrence Erlbaum Associates, USA.

[31] Keller J.M., Suzuki K., 1988. Use of the ARCS motivation model in courseware design (Chapter 16), [in:] D.H. Jonnasen (Ed.). *Instructional designs for microcomputer course-ware*, Lawrence Erlbaum Associates, USA.

[32] Asif Ekbal, Sriparna Saha, 2014. Simultaneous feature and parameter selection using multiobjective optimization: application to named entity recognition. *Int. J. Mach. Learn. & Cyber*, Springer-Verlag Berlin Heidelberg 2014.

[33] Krenich S., 2017. Multi-thread evolutionary computation for design optimization. *Technical Transactions*, vol. 9, 197-206.

[34] Grefenstette J., 1986. Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 16, No. 1, 122–128.

[35] Osmera P., Lacko B., Peter M., 2003. Parallel Evolutionary Algorithms, *Proceedings IEEE International Symposium on Computational Intelligence in Robotics and Automation.*

[36] Xin Zhang, Qinglian Zhang, Xiu Zhang, 2017.Nonuniform antenna array design by parallelizing three-parent crossover genetic algorithm. *EURASIP Journal on Wirelss Communications and Networking*,Springer, 106.

[37] Azam Marjani, Saeed Shirazian2, Mehdi Asadollahzadeh, 2016. Topology optimization of neural networks based on a coupled genetic algorithm and particle swarm optimization techniques (c-GA–PSO-NN). *The Natural Computing Applications Forum*, Springer.

[38] Adem Alpaslan Altun, 2013. A combination of Genetic Algorithm, Particle Swarm Optimization and Neural Network for palmprint recognition. *Neural Comput & Applic*, Springer, vol. 22, S27–S33.

[39] Lempa P., Ptaszyński M., Masui F., 2018. The use of genetic algorithm to optimize quantitative learner's motivation model. *Technical Transactions*, Vol. 4, 189–194.

[40] Lempa P., Ptaszyński M., Masui F., 2016. A Survey on the Use of Genetic Algorithms in Natural Language Processing. *Proceedings of The 22nd Annual Meeting of The Association for Natural Language Processing (NLP-2016)*, pp. 39-40, Sendai, Japan.

[41] Lempa P., Lisowski E., 2013. Genetic algorithm in optimization of cycloid pump. *Technical Transactions*, vol. 1-M, 230-236.

[42] Wang, N.F. and Tai, K., 2010. Target matching problems and an adaptive constraint strategy for multiobjective design optimization using genetic algorithms. *Computers and Structures*, Scopus, vol. 88, num. 19-20, 1064-1073.

[43] Kita, E. and Tanie, H., 1996. Shape optimization of continuum structures by genetic algorithm and boundary element method. *Engineering Analysis with Boundary Elements*, Scopus, vol. 19, num. 2, 129-136.

[44] Melanie Mitchell, 1999. An introduction to genetic algorithms. *Cambridge*, Massachusetts London, England, Fifth printing, vol. 3.

[45] Stroustrup B., 2000. The C++ Programming Language. The C++ Programming Language (Special Edition). *Addison-Wesley*.

[46] Taku Kudo, http://chasen.org/ taku/software/TinySVM/

Publication List

1. Lempa P., Ptaszyński M., Masui F., *Cyberbullying Blocker Application for Android* In Proceedings of 7th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics (LTC'15), The First Workshop on Processing Emotions, Decisions and Opinions (EDO 2015), pp. 408-412, November 27-29, 2015, Poznan, Poland.

2. Lempa P., Ptaszyński M., Masui F., *A Survey on the Use of Genetic Algorithms in Natural Language Processing* In Proceedings of The 22nd Annual Meeting of The Association for Natural Language Processing (NLP-2016), pp. 39-40, Sendai, Japan, March 07-11, 2016.

3. Lempa P., Ptaszyński M., Masui F., *The use of genetic algorithm to optimize quantitative learner's motivation model*, Technical Transactions, Vol. 4, 2018, pp. 189–194, DOI: 10.4467/2353737XCT.18.066.8378

4. Lempa P., *Optimization of QLM Models with the use of Genetic Algorithm*, International Conference on Computer Aided Mechanical Engineering (CAME 2017), Krakow, May 25-26, 2017

Acknowledgments

I would like to express my deepest gratitude to Professor Fumito Masui for his invaluable help and guidance during the course of my Doctor of Philosophy Degree. I would like to thank him for his advice and kind support throughout my research, and his understanding and openness to different cultures. Without him it would be impossible to finish my Ph.D.

I would also like to thank Professor Michał Ptaszyński for his invaluable insight into the world of Natural Language Processing, his precious inputs to my research and most of all for his help into easing me into the Japanese way of life, work and study. I am really thankful that I always could count on him.

I wish to thank the members of my dissertation committee: Koichi Hirayama, Sharif Ullah and Toshio Eisaka for generously offering their time, support, guidance and good will throughout the preparation and review of this document.

I would also like to thank Professor Edward Lisowski for his help in Cracow University of Technology. Thanks to his inspiration and support I could start my scientific career. I am really thankful for his patience and guidance.

My studies in Kitami Institute of Technology would not have been possible without the support of the Japanese Ministry of Education which awarded me this scholarship. Therefore, I would like to express my gratitude to the

Ministry of Education and the Embassy of Japan in Poland for giving me this chance.

Finally, I would like to thank my family for having supported me through all these years, and for making me who I am today.

Lastly, I want to thank all my friends, close and distant, for all the help they have provided.

Appendix A

Tables contains the data results from experiment described in chapter 4. It contains shortcut of used algorithm (Algorithm), obtained accuracy (Acc.), precision(Prec.), recall(Rec.) and comparison to results without optimization.

Table 5.1: All results from experiment in chapter (Part 1)

Algorithm	**Acc.**	**Prec.**	**Rec.**	**Change**		
				Acc.	**Prec.**	**Rec.**
Base	77,60%	68,09%	71,11%	-	-	-
SGA100_3	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
SGA100_5	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
SGA100_10	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
SGA300_3	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
SGA300_5	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
SGA300_10	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
SGA500_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
SGA500_5	78,80%	69,68%	72,78%	1,20%	1,60%	1,67%
SGA500_10	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
SGA1000_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
SGA1000_5	78,80%	69,68%	72,78%	1,20%	1,60%	1,67%
SGA1000_10	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
UCGA100_3	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
UCGA100_5	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
UCGA100_10	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
UCGA300_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
UCGA300_5	78,80%	69,68%	72,78%	1,20%	1,60%	1,67%
UCGA300_10	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
UCGA500_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
UCGA500_5	80,40%	71,80%	75,00%	2,80%	3,72%	3,89%
UCGA500_10	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
UCGA1000_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
UCGA1000_5	80,40%	71,80%	75,00%	2,80%	3,72%	3,89%
UCGA1000_10	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
2pCGA100_3	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
2pCGA100_5	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
2pCGA100_10	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
2pCGA300_3	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
2pCGA300_5	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
2pCGA300_10	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
2pCGA500_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
2pCGA500_5	78,80%	69,68%	72,78%	1,20%	1,60%	1,67%
2pCGA500_10	78,40%	69,15%	72,22%	0,80%	1,06%	1,11%
2pCGA1000_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
2pCGA1000_5	78,80%	69,68%	72,78%	1,20%	1,60%	1,67%
2pCGA1000_10	78,40%	69,15%	72,22%	0,80%	1,06%	1,11%

Table 5.2: All results from experiment in chapter (Part 2)

Algorithm	**Acc.**	**Prec.**	**Rec.**	**Change**		
				Acc.	**Prec.**	**Rec.**
3pCGA100_3	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
3pCGA100_5	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
3pCGA100_10	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
3pCGA300_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
3pCGA300_5	80,40%	71,80%	75,00%	2,80%	3,72%	3,89%
3pCGA300_10	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
3pCGA500_3	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
3pCGA500_5	81,20%	72,87%	76,11%	3,60%	4,79%	5,00%
3pCGA500_10	80,00%	70,74%	73,89%	2,40%	2,66%	2,78%
3pCGA1000_3	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
3pCGA1000_5	81,20%	72,87%	76,11%	3,60%	4,79%	5,00%
3pCGA1000_10	80,00%	70,74%	73,89%	2,40%	2,66%	2,78%
5pCGA100_3	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
5pCGA100_5	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
5pCGA100_10	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
5pCGA300_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
5pCGA300_5	80,40%	71,80%	75,00%	2,80%	3,72%	3,89%
5pCGA300_10	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
5pCGA500_3	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
5pCGA500_5	81,20%	72,87%	76,11%	3,60%	4,79%	5,00%
5pCGA500_10	80,00%	70,74%	73,89%	2,40%	2,66%	2,78%
5pCGA1000_3	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
5pCGA1000_5	81,20%	72,87%	76,11%	3,60%	4,79%	5,00%
5pCGA1000_10	80,00%	70,74%	73,89%	2,40%	2,66%	2,78%
GAwLL100_3	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
GAwLL100_5	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
GAwLL100_10	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
GAwLL300_3	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
GAwLL300_5	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
GAwLL300_10	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
GAwLL500_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
GAwLL500_5	78,80%	69,68%	72,78%	1,20%	1,60%	1,67%
GAwLL500_10	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
GAwLL1000_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
GAwLL1000_5	78,80%	69,68%	72,78%	1,20%	1,60%	1,67%
GAwLL1000_10	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%

Table 5.3: All results from experiment in chapter (Part 3)

Algorithm	**Acc.**	**Prec.**	**Rec.**	**Change**		
				Acc.	**Prec.**	**Rec.**
GGwSR100_3	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
GGwSR100_5	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
GGwSR100_10	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
GGwSR300_3	78,00%	68,62%	71,67%	0,40%	0,53%	0,56%
GGwSR300_5	80,40%	71,80%	75,00%	2,80%	3,72%	3,89%
GGwSR300_10	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
GGwSR500_3	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
GGwSR500_5	81,60%	73,40%	76,67%	4,00%	5,32%	5,56%
GGwSR500_10	80,20%	71,28%	74,44%	2,60%	3,19%	3,33%
GGwSR1000_3	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
GGwSR1000_5	81,60%	73,40%	76,67%	4,00%	5,32%	5,56%
GGwSR1000_10	80,20%	71,28%	74,44%	2,60%	3,19%	3,33%
GGwSRaLL100_3	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
GGwSRaLL100_5	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
GGwSRaLL100_10	77,60%	68,09%	71,11%	0,00%	0,00%	0,00%
GGwSRaLL300_3	78,80%	69,15%	72,22%	1,20%	1,06%	1,11%
GGwSRaLL300_5	81,20%	72,87%	76,11%	3,60%	4,79%	5,00%
GGwSRaLL300_10	80,00%	70,74%	73,89%	2,40%	2,66%	2,78%
GGwSRaLL500_3	80,20%	71,28%	74,44%	2,60%	3,19%	3,33%
GGwSRaLL500_5	83,00%	74,47%	77,78%	5,40%	6,38%	6,67%
GGwSRaLL500_10	81,20%	72,87%	76,11%	3,60%	4,79%	5,00%
GGwSRaLL1000_3	80,20%	71,28%	74,44%	2,60%	3,19%	3,33%
GGwSRaLL1000_5	83,00%	74,47%	77,78%	5,40%	6,38%	6,67%
GGwSRaLL1000_10	81,20%	72,87%	76,11%	3,60%	4,79%	5,00%