Insert here your thesis' task.

Czech Technical University in Prague
Faculty of Information Technology
Department of Knowledge Engineering

Master's thesis

Automatic Text Summarization

Bc. Šimon Hlaváč

Supervisor: doc. RNDr. Ing. Marcel Jiřina, Ph.D.

3rd May 2015

Acknowledgements

I am using this opportunity to express my gratitude to everyone who supported me throughout the course of this master thesis project. I am thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

I would also like to thank my project supervisor Mr. Marcel Jiřina for help he had provided me.

Declaration

In Prague on 3rd May 2015 . . . .

Citation of this thesis

Hlaváč, Šimon. *Automatic Text Summarization.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2015.

Abstrakt

V této práci jsou představeny základní metody využívané v automatické sumarizaci textu a genetických algoritmech. Dále je zde navržen systém automatické sumarizace založený na grafových strukturách a Markovských řetězcích, který byl rovněž implementován a řádně otestován. Práce se dále zabývá učením správného nastavení vah důležitosti jednotlivých metod používaných v sumarizaci pomocí naivního přístupu a genetických algoritmů, které byly rovněž naimplementovány, včetně možnosti paralelního zpracovávání a využití cache pro zrychlení systému, a řádně otestovány.

Klíčová slova automatická sumarizace textu, genetické algoritmy, markovské řetězce, graf, řízené učení

Abstract

This work presents the basic methods used in automatic text summarization and genetic algorithms. Furthermore, system of automatic summarization based on graph structures and Markov chains was designed, implemented and properly tested. This study also discusses learning of proper setting of importance weights of individual methods used in summarization by naive approach and genetic algorithms, which were also implemented and properly tested. System also includes possibility of parallel processing and use of caching to speed up its process.

Keywords automatic text summarization, genetic algorithms, markov chains, graph, supervised learning

Contents

List of Figures

List of Tables

Introduction

Magnitude of information in written format on internet, in magazines, news, scientific articles and so on is rapidly overgrowing our human capability to keep up with reading them. Therefore, a set of new scientific methods was developed and still is. Many of those methods belongs to field of study called automatic text summarization. Main purpose of this field is to obtain potentially useful and important information from text by creating short summaries of original documents.

Summaries are composed of ether composition of most important sentences from original text or rewriting and merging sentences of whole document into shorter version with preservation of most important information.

Main purpose of those summaries is to create much shorter document, which presents key understanding of original document without the need of reading it, which shortens required time in searching and obtaining desirable information.

This thesis begins with extensive analysis of current state-of-the-art of automatic text summarization, its methods, principles and ways of measuring its performance. Then analysis of genetic algorithms is presented, which are used later in this thesis for learning proper settings of system, based on current corpus of documents. These chapters lay important foundation of proposed and implemented summary system and used methods.

Next chapter is dedicated to overall system design. System is composed of single-purpose components that are linked together into a data workflow. These components are described in detail, including their workflow. This chapter also contains descriptions of main system structures used for representation of documents, their corpus, results and additional required structures.

The proposed system is based on Markov chains and graph theories, where each node in graph is represented by one sentence of a document. Edges in this graph are weighted by set of proposed measures whose outputs are combined by linear combination into resulting weight. Coefficients of this combination are determined by ether nave approach or by mentioned genetic algorithms. In resulting graph, stationary distribution is found, which represents magnitude of importance of each sentence. Based on this importance, sentences are selected into summary.

Following chapter describes some interesting or important parts of system implementation, including selection of programming language, data cleaning, creating required databases, external evaluation, caching etc.

Last chapter contains testing and evaluation of system from various points of view. System is profiled in order to identify components that take too long to process, then solution is proposed, implemented and tested if system is indeed faster and that its performance didn't drop. Following tests are designed to validate parallel implementation, caching and summarization. System is also compared to other summarization models.

Chapter 1

Analysis of text summarization

In the beginning of this work, it was required to analyse available information regarding automatic text summarization. Most of information was obtained by free access to scientific articles provided by CTU in Prague. Nowadays exists a lot of studies regarding this area of study and thus only most used, most common and most promising was selected into my research of state-of-the-art.

1.1 Automatic text summarization in general

It is possible to categorise and implement automatic text summarization by several criteria. This categorisation brings better understanding of a summarization system, [3].

1.1.1 Main approaches

There are two basic approaches of how to create summaries from document: *abstraction* and *extraction*.

1.1.1.1 Abstraction

Abstraction, [4][5], is a way of creating summary by rewriting original sentences into shorter ones with preservation of the most important information. However, this approach is strictly dependent on specific language, its grammar and syntax, which makes realization of this approach very difficult.

The main advantage is that summarization using this approach produces more readable, coherent and cohesive text.

The main obstacle is complicated implementation of natural understanding of given language, which is in this time almost impossible. This obstacle is sometimes partially overcome by extensive dictionaries or by specialization of a summarization system to some strictly formatted text or defined area, domain or purpose.

1.1.1.2 Extraction

Much easier and mostly used way of a summary generation is *extraction*, [5][6]. This way is characterized by reusing the original parts of a document (usually phrases or sentences) in a summary. General approach is to calculate importance of every part of the document by certain approaches and then construct a summary by composition of top rated parts of the document.

The greatest advantage against *abstration* is much easier implementation and vaster possibilities in usage of approaches, which doesn't have to be based on understanding given text.

In contrast, main disadvantage of *extraction* is production of summaries lacking cohesion and integrity, therefore readability. Among other disadvantages belongs for example fact, that great number of larger documents contains pronouns, which might lose connection to original entity they represent and so the summary could be misinterpreted by users. Furthermore, if basic working part of document is sentence, then there is possibility, that some long and complex sentence will be selected to summary, but only main cause of this sentence will be informative and useful and the rest of that sentence will only prolong the summary. Also, information might be spread throughout whole document evenly and a summary will have to be ether informative but long or short and lacking some important information.

1.1.2 Detail of summary

From this point of view, summarization is divided into two categories. First one is *indicative* summarization, whose purpose is to hold only key ideas of original text. Given summary is usually a very short extract of the original document and it is mostly used to give a user a hint whether to read the original text or not (similarly as an abstract).

Second category is *informative* summarization, which is considered to be a standalone text, independent of its original by the fact, that it already has all important information with a difference that it is shorter than the original text (for example without redundant information).

1.1.3 Content of summary

Summarizations are divided by the way of how it is evaluated, what is important to have in summary.

Common way is a *general* content. There are nether additional information about content of the text nor about any topic for summarization to focus. Summarization is therefore an *extraction* or an *abstraction* of a whole document with all topics weighted as equally important.

More specific way of summarization is called *query-based*, [7][5][6], also called *personalized* summarization. These summaries are generated as a form of response to ether user-defined question or by a set of acquired or inserted and weighted keywords or phrases. Weight of those keywords is afterwards taken into account when summarization system evaluates importance of information in certain sentences to create proper personalized summary.

1.1.4 Limitations

Rather than limitations of particular summaries, they are limitations of input text given by a set of used approaches to evaluate importance of sentences.

First category limits input text to *domain dependant*. Possible input texts are therefore limited to certain knowledge domain (for example medicine, brain science, military etc.). Used algorithms are mostly based on some database or a corpus from specific domain to properly learn meaning and significance of particular words and phrases in context to given domain.

Second category restricts input text to a specific structure. It's called *genre specific*. It is important for all input documents to have the same structure of text (for example news article, scientific article, assembly manual etc.). Used algorithms work with particular parts of the text and benefit from deeper knowledge of its structure (introduction, methodology, conclusion etc.) to generate summary.

Last mentioned category is used for those summarization tools, that don't restrict input text in any domain or structural way and therefore those tools are genre and structural *independent*.

1.1.5 Number of input documents

Summarizations are divided into two categories. First is *single-document* summarization, where summarization tool accepts only one document per summary.

Second category belongs to summarization tools allowing *multi-document* input, which they summarize into single compact summary. In most cases it is crucial, that all input documents share the same topic. This summarization category in widely used by news aggregators to form summaries from articles about same topic from various news servers.

1.1.6 Language

Depending on input text, summarization tools can be categorized into *mono lingual* or *multi lingual*. In *mono lingual* tools, input text and generated summary are created in the same language and those tools are mostly even dependant on a certain language (for example because of usage of synonym thesaurus).

In contrast, *multi lingual* tools can summarize multiple input documents at a time, where every one of them can be in different language and generated summary is then created in one of the selected language.

1.1.7 Summary generation

In *abstraction* tools, process of summary generation is strongly dependent on used methods and algorithms.

Extraction tools usually use one of the following approaches to create summary. The most simple way is to select certain number of sentences satisfying given rule (number of sentences, number of words, importance measure etc.).

Another approach is called *Global Summary Selection*, [6]. It searches ideal combination of sentences to create summary by certain rule or rules (for example maximization of information, redundancy minimization, length etc.). It selects the overall best summary. Finding that summary is *NP-hard* problem, but approximate solutions can be found using dynamic programming algorithms.

Last mentioned approach is *Maximal Marginal Relevance*, [6], which could be considered as a heuristic of *Global Summary Selection*. It is iterative algorithm and each iteration adds most important sentence into summary. After each iteration, importance weights are recalculated in a way, to discredit remaining sentences that are similar to those already selected for summary. Iterative process cycles until some stop criteria is met (see first method).

1.2 Approaches used in summarization

Summarization approaches divide into three basic groups according to rate of understanding text by *NLP*. Shown from the least dependent to the most dependent on *NLP*.

1.2.1 Statistical approaches

Methods from this category are based on statistics, which aren't based on any knowledge of word meaning. These methods are easily implemented and the least language dependent.

1.2.1.1 Automatic extraction

Automatic extraction, [7][8], is based on evaluation of sentence importance in a way, that every sentence is divided into tokens (usually words or phrases) and by using some set of approaches a set of weights is assigned to every token. Resulting weight of token is then obtained by linear combination of those weights and a sentence weight is then obtained by a sum of weights of all tokens, usually divided by the number of tokens in sentence. Sentences are then ordered by its weight and defined number of sentences with the highest weight are then selected. These sentences in the order of original text then form a summary.

Used methods are following:

- *Cue method*, [5] Weight of token is defined by a cue dictionary and if given token is absent in dictionary its weight is zero.
- *Frequency method* The more frequent tokens are in document the more weight is assigned to them.
- *Location method* It's based on assumption that sentences located in the beginning of document and each paragraph are more important than others and therefore more weight is assigned to them.
- *Title method* Tokens in sentence that are also in document title are more important than others and therefore more weight is assigned to them. It's based on assumption that those sentences are referring to main topic of document.
- *Sentence Length Cut-O method*, [5] Sentences with less than predefined number of tokens are excluded from summary.
- *Uppercase method*, [5] Tokens beginning with uppercase letter have more weight. And similarly *Numerical data m.*, [4], *Quotation m.*, [4], *Date m.*, [4], and *Font-based m.*, [5].
- *Proper Noun method*, [5] Proper nouns are names of persons, places and other unique entities. Sentences containing those nouns have higher chance to be in summary.
- *Biased Word method*, [5] If a token is in biased words list, then its weight is higher. This list is previously defined and may contain domain specific words.
- *Occurrence of non-essential information method*, [5] This method is used to decrease weight to sentences containing certain words or conjunctions (for example because, furthermore, additionally etc.). Those words might indicate that it is sentence providing additional minor information and therefore its chance to be included in summary is decreased.

1.2.1.2 Machine Learning approach

This category contains any method that is able to categorize sentences into two classes (include to or exclude from summary) by a set of features, [5][9]. For feature extraction, any approach can be used. Furthermore it's important to have train and test sets of a document, where every sentence is labeled into two mentioned classes by a user. Learning step then proceeds in usual way according to selected algorithm.

Among most used algorithms belongs Bayes classifier, artificial neural networks and support vector machines.

The main disadvantage of this approach is to obtain labeled dataset. Its disadvantage lays in time-consuming work of labeling sentences in a document and subjective and inconsistent opinion of users about what sentence is suitable for a summary.

1.2.1.3 Swarm-based approaches

These approaches of artificial intelligence characterized by collective behaviour of decentralized agents are in automatic text summarization mainly used for searching for coefficients of linear combination of token weights in sentences (see automatic extraction in 1.2.1.1), [10]. Their purpose is to find proper settings to optimize generation of summaries.

It is supervised method, where by some evaluation function (for example ROUGE-1, see section 1.3) the best setting of coefficients is searched. Individual agents of swarm iteratively search solution space in particular directions while remembering so far the best find solution of coefficients. After a number of iteration, the best found solution is used to generate summaries.

1.2.1.4 Term Frequency - Inverse Document Frequency

tf-idf is used for determining weight of tokens by their frequency in document and in whole corpus of documents, [5][6]. Weight of a particular token rises with the number of occurrences of this token in given document and decreases with the number of documents also containing this token. Weight of *tf-idf* represents importance of token in a contrast to a particular topic in given document and discredits tokens that are generally known or common in given corpus (e.g. domain knowledge).

One of many definitions of *tf-idf* is following:

- $tf(word)$ - the number of occurrences of *word* in given document
- N_d - the number of all documents in corpus
- $df(word)$ - the number of documents in corpus containing this *word*
- $idf(word) = \log \frac{N_d}{df(word)}$ - inverse frequency of *word* in corpus
- $tf\text{-}idf(word) = tf(word) * idf(word)$

1.2.1.5 Latent Semantic Analysis

Latent Semantic Analysis is unsupervised robust method for finding relationships between words, without any knowledge about their mutual meaning, [7][5][6][11][12].

This method is used to cluster similar documents, to extract keywords and to find out most important sentences in document.

Its simplified definition is following:

Matrix $\mathcal{A}$ - every row of this matrix represents word from a document and every column represents sentence in a document. Value of each cell is equal to zero if that word is not present in that sentence, otherwise the value is equal to *tf-idf* of that word.

By applying SVD on matrix $\mathcal{A}$, new set of matrices is obtained - $\mathcal{U}$, $\mathcal{S}$ a $\mathcal{V}'$. Columns of $\mathcal{U}$ form mapping between space of words and space of topics, matrix $\mathcal{S}$ represents weights of those topics and matrix $\mathcal{V}'$ is new representation of sentences in those topics.

Then, dimensionality reduction is applied in order to decrease number of rows and columns. Resulting matrix $\mathcal{S} * \mathcal{V}'$ represents weights of sentences for each topic and from every of those topics, some best-weighted sentences are selected to form a summary.

By multiplying corresponding matrices with its own transposition it's possible to obtain similarities between each word (or sentence).

1.2.1.6 Scientific Articles summarization

Summarization of scientific article proceeds in a way, that every other article citing given article is found, [6]. Then, for every found article the algorithm tries to identify section, which is being cited. Those sections are then consider important and should be included in summary.

1.2.2 Linguistic approaches

Linguistic approaches are usually based on searching in ether dictionaries or graph structures representing relations in certain language. These methods are language-dependent on level of existence of those dictionaries in another language.

One of the most popular thesaurus for English is *WordNet*, [13]. It is mostly used to look up synonyms and connections between words to determine their mutual relationship.

1.2.2.1 Lexical Chains

This method runs through entire document and forms lexical chains by grouping words and phrases that have similar or same meaning (synonyms, hyponym - hyperonym relations) by some dictionary, for example *WordNet*. Cardinality and homogeneity in those chains are then analysed, in order to find strongest chain. For every chain having its strength greater than some threshold, few best representative members are selected to form summary, [7][6].

1.2.2.2 Clustering

Clustering is in automatic text summarization used mostly for large-scale documents containing more than one topic, [5][6][4]. With clustering those topics are identified and separated. Every found cluster is then summarized with any other single-topic summarization method which creates summary for every topic. Summaries are then composed into one final summary.

1.2.2.3 Graph-based approaches

Input text is represented by a graph structure, where nodes represent words, phrases, causes, sentences or whole paragraphs, [7][6][4][14][15][16][17]. Nodes are connected to each other by weighted edges according to some similarity measure.

From those graphs, the most important nodes are determined by some algorithm and content they hold will be included into summary.

Common algorithms used for selecting the most important nodes are:

- All weights from all used similarity measures are summed for each node and this total number is used for determining the most important nodes. This sum is sometimes normalized by number of non-zero-weighed edges.

- Instead of weighting edges, it is possible to connect only relevant nodes together (having higher mutual similarity than some defined threshold). Nodes with the highest degree are then selected as the most important. This algorithm is also often used to identify different topics using dense sub-graph analysis.

- If first option is altered so each weight (value of some similarity) in direction from particular node is normalized to make sum equal to one, graph could be then considered to be Markov chain, in which it is possible to calculate *stationary distribution* of every node by iterative computation. This distribution of probability could be then used as indicator of the most important nodes in graph.

1.2.3 Rhetorical approaches

Rhetorical structural analysis is strongly dependent on specific language, meaning of words and phrases, [7][4][18]. This method is based on finding so called rhetorical relationships, which are relationships between two non-overlapping sets:

- *Nucleus* represents important information in a text. It is independent part of a text.
- *Satellite* represents additional information required in order to explain and understand important information in *Nucleus*. Therefore, *Satellite* in dependent on *Nucleus*.

Relationships are represented in form of binary tree capturing mutual connections in a text. Algorithm runs through input text and splits it accordingly by clause ending, conjunctions, certain phrases or words indicating additional information, comas etc. Those parts are then considered to be nodes in binary thee, where edges are created by a certain set of rules. Then, *Nucleus* and *Satellite* nodes are identified and tree is pruned. The result of pruning is then converted into a summary.

1.3 Evaluation of summary

Evaluation of automatically generated summaries is divided to *internal* and *external* evaluation. *Internal* evaluation is conducted by human. Its main advantage is possibility of evaluation in readability, understandability and consistency. On the other hand, this type of evaluation is time consuming and every person has different subjective opinion about how the correct summary should look.

External summarization is done automatically by computer. In this type of evaluation, however, human work is also required ether to label sentences in text, that they think are important for summarization (text can be used by any dichotomous classification algorithm) or to write their own summaries to given text. In most cases, those user-written summaries are then by some metric (for example *ROUGE*, [19]) compared with system-generated summaries.

In most articles that were analysed, evaluation is done by *ROUGE* or *Recall-Oriented Understudy for Gisting Evaluation*. *ROUGE* is a set of *NLP* metrics designed to compare human-written summaries with system-generated summaries. Metrics are following, [20]:

- **ROUGE-N** - N-gram based co-occurrence statistics.
- **ROUGE-L** - Longest Common Subsequence (LCS) based statistics. Longest common subsequence problem takes into account sentence level

structure similarity naturally and identifies longest co-occurring in sequence n-grams automatically.

- **ROUGE-W** - Weighted LCS-based statistics that favors consecutive LCSes.
- **ROUGE-S** - Skip-bigram based co-occurrence statistics. Skip-bigram is any pair of words in their sentence order.
- **ROUGE-SU** - Skip-bigram plus unigram-based co-occurrence statistics.

Output of each of those metrics is *Recall*, *Precision* and *F-score* expressing mutual similarity of two summaries.

Chapter 2

Analysis of genetic algorithms

Genetic algorithms are very popular optimization method given its simplicity, minimal requirements, flexibility and parallel capabilities. As a result of this it is widely used in many different situations and is able to solve difficult problems, where conventional optimization fails.

This algorithm is space search heuristic using methods inspired by natural evolution by mimicking inheritance, crossover, mutation and selection of the fittest. It is especially used for problems that are little know of because of its general approach, [21] [1].

The algorithm iterates through generations where each generation is represented as population of individuals representing possible solutions of given problem from its search space. Depending on nature of given problem, properties of each individual are coded into some alphabet, usually binary or decimal. This coding of each property is represented by gene and composition of those genes makes chromosome.

On those chromosomes composed of ordered genes are then used natural-inspired techniques like mutation, crossover and selection in order to create better chromosomes and therefore better solutions to given problem.

2.0.1 Life cycle of genetic algorithms

As mentioned before genetic algorithms work in iterations and their basic life cycle is following:

1. Create a population zero
2. Loop until stop criteria is met
    a) Evaluation - Calculate fitness of individuals in a population
    b) Selection - Select parents for crossover
    c) Crossover - Combine parents to create children for a new population

d) Mutation - Mutate parents and children
e) Elitism - Select best E_{GA} individuals
f) Compose next population by merging the best individuals, parents and children together

Those steps are described in a detail in following sections.

2.0.2 Creation of population zero

Initial population is generated in this step. Common approach is to generate P_{GA} random individuals from whole search space, however if we know approximately where the solution might be located in search space we can adjust random generation to that specific region.

This region-specifying should find best solution more quickly, however it might have negative effects of converging to a local optimum instead of global, [22].

Also size of population depends on nature of problem and on computational complexity of evaluation (see next section 2.0.3). Usually its size is between hundred and tens of thousands.

2.0.3 Evaluation

As in natural evolution is quoted ”survival of the fittest” by Herbert Spencer, genetic algorithms also need some indicator of how good specific individual is.

Calculation of fitness is basically the only thing directly connected to specific problem and all other parts of genetic algorithms are general to all problems with the same chromosome coding.

Individual chromosomes and their genes are converted back to solution representations which are evaluated by some problem-specific system determining how good the solution is in decimal scale.

2.0.4 Selection

After evaluation of individuals, some part of population is selected for breeding and as part of next generation. Those selected individuals are called parents. Selection of parents is based on their fitness and the higher the fitness is the bigger probability for selection is.

Selection of individuals should be random-based to fitness because selecting only the best candidates leads to quick convergence to local optimum. On the other hand, without consideration of fitness - pure random selection leads to very slow or none convergence to better solution at all.

Balancing both these conditions makes next population diverse and therefore not prone to converging to a local optimum but also makes it to get closer to proper solution.

There are several used algorithms to make selection, [23]:

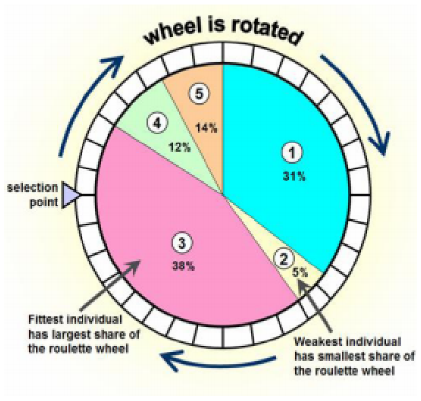
2.0.4.1 Roulette selection

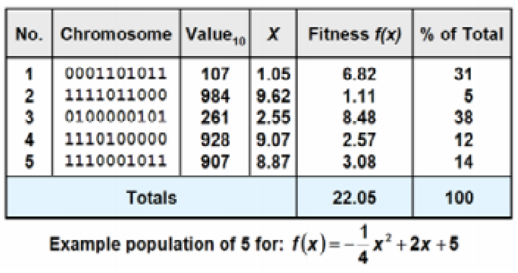
Roulette selection is based on random selection of individuals to population, where probability of selecting each individual is directly proportional to its fitness.

Basic idea is to create spinning wheel, where every individual in population is, and its size in this wheel is proportional to its fitness. The wheel is then spun and the winner is selected to next generation as a parent. This process is repeated S_{GA} times and is displayed in figure 2.1.

This approach prefers better individuals but worse could be selected as well. If distances between fitness of top few individuals and rest of population are too great, genetic algorithms might converge very fast to local optimum (as mentioned in general section 2.0.4).

Figure 2.1: Roulette wheel selection mechanism, [1]


wheel is rotated
5
14%
4
12%
1
31%
selection point
3
38%
2
5%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel



No.	Chromosome	Value_{10}	X	Fitness $f(x)$	% of Total
1	0001101011	107	1.05	6.82	31
2	1111011000	984	9.62	1.11	5
3	0100000101	261	2.55	8.48	38
4	1110100000	928	9.07	2.57	12
5	1110001011	907	8.87	3.08	14
Totals				22.05	100

Example population of 5 for: $f(x) = -\frac{1}{4}x^2 + 2x + 5$

2.0.4.2 Stochastic selection

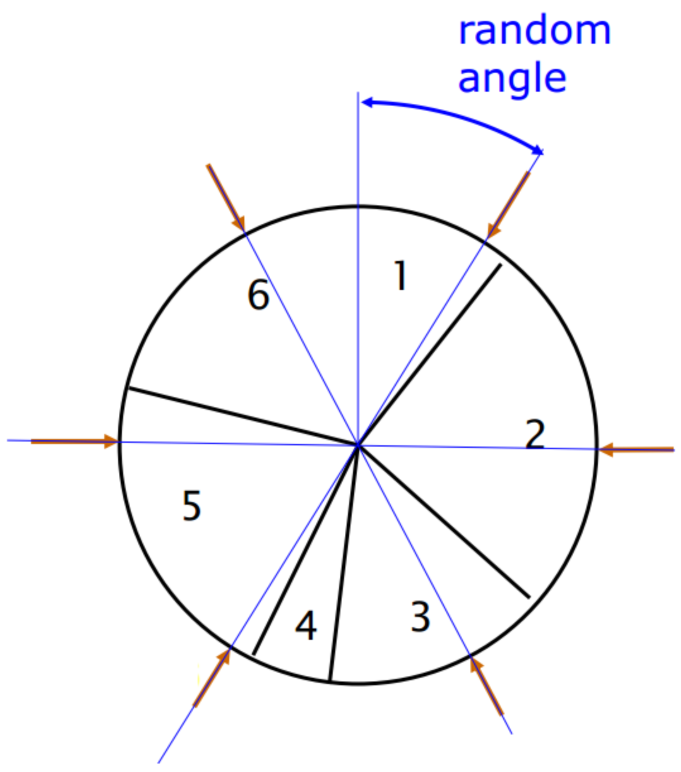
Stochastic universal sampling works with individuals mapped into segments of wheel, such as that every individual has its segment proportional to its fitness as in roulette selection (see section 2.0.4.1).

Selection works in a way, that if S_{GA} individuals are required to be selected and wheel's circumference is 1, then first selection point is randomly selected between 0 and $1/S_{GA}$ and rest $S_{GA} - 1$ selection points are then equally spaced with distance $1/S_{GA}$ between them from that first point. This process is showed on figure 2.2.

2.0.4.3 Rank selection

In rank selection, individuals in population are sorted according to their fitness values and their fitness is then reassigned in a way that it depends only on

Figure 2.2: Stochastic selection mechanism, [2]



their rank (opposite to order) and not its actual value.

This fitness reassignment overcomes the scaling problem of the roulette wheel selection (see section 2.0.4.1). The reproductive range is therefore limited, so that no individual generates an excessive number of offspring.

Ranking introduces a uniform scaling across the population and provides a simple and effective way of controlling selective pressure. It behaves in more robust manner than roulette wheel selection and, thus, is frequent method of choice. (Best individuals are not that preferred and weak individuals have bigger chance to be selected).

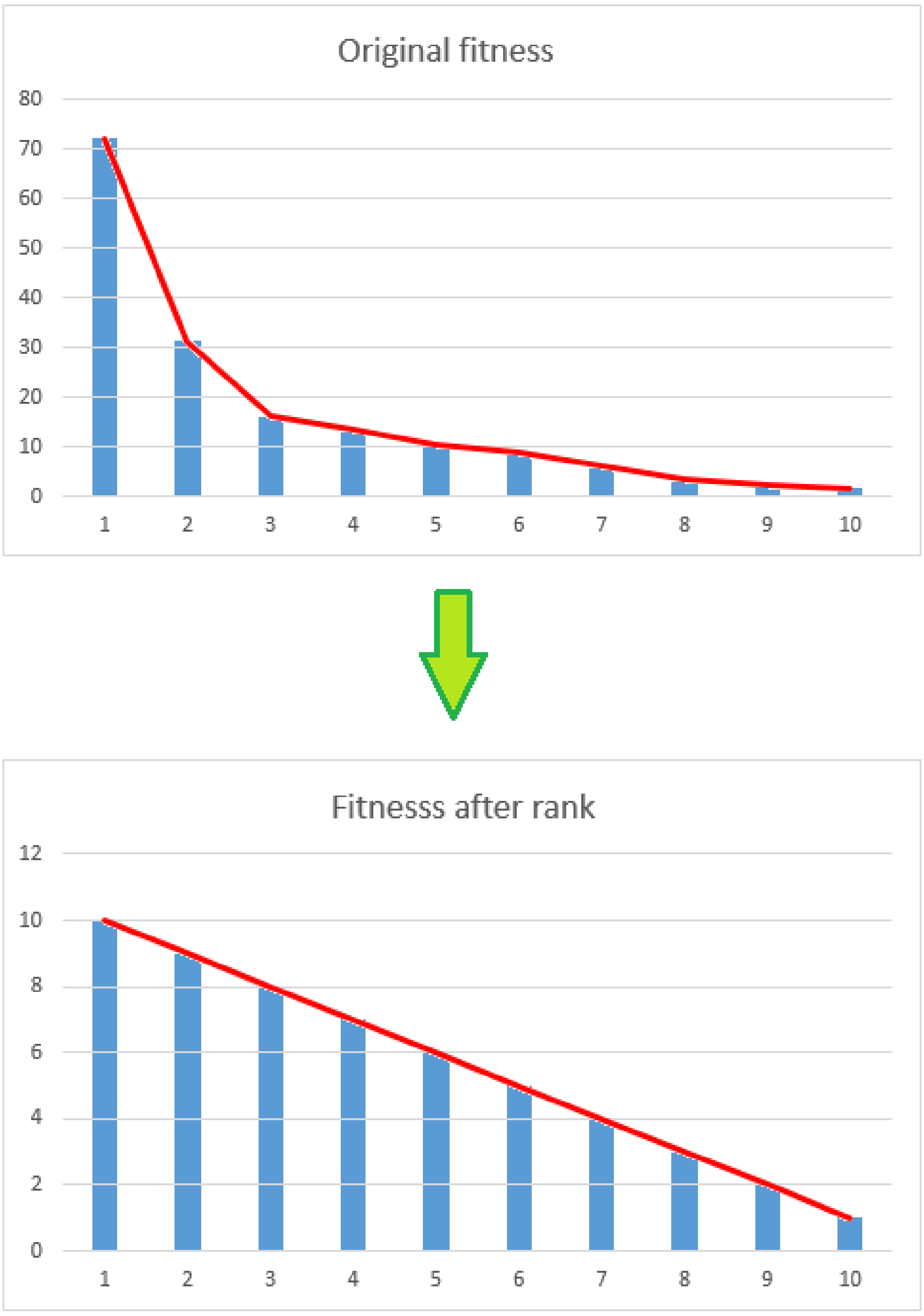
The process is demonstrated in figure 2.3.

2.0.4.4 Tourney selection

Tourney selection randomly picks T_{GA} individuals from population to tourney. Individual with the highest fitness from tourney is then selected for breeding and for next generation. This tourney is then repeated S_{GA} times.

Parameter T_{GA} widely affects behaviour of genetic algorithms. Small T_{GA} is making selection more random and higher T_{GA} behaves in selecting only the best individuals. Effects of this was described in general section 2.0.4.

Figure 2.3: Rank selection mechanism



2.0.5 Crossover

The idea is by combining two parents properly it is possible to create better individual. The better the parents are the better individual can be created.

This combination called crossover creates new individuals from previously selected parents by randomly combining them. Combination can be achieved in several ways from simple random selection of two parents and copying one part of chromosome from first parent and another part from second parent to complex function combining more than two parents in order to create child.

In binary chromosomes, one of the mostly used approaches how to combine two parents into child is N-point crossover. The algorithm randomly selects N points in chromosome at which the continuous copying of chromosome switches from one parent to another. Another approach is unified crossover where for every part of chromosome is randomly decided whether it will be copied from first or second parent. Those approaches are display in figure 2.4.

For real-valued chromosomes are used several other methods. Simple Arithmetic Crossover copies first R (random number) genes from first parent and rest is weighted average of both parents. Whole Arithmetic Crossover creates child as arithmetic average of both parents. Discrete Crossover works exactly same as uniform crossover for binary chromosomes but copies whole genes, [24].

Crossover and mutation (see next section 2.0.6) are means of exploring search space.

2.0.6 Mutation

Main purpose of mutation is to keep population diverse and therefore prevent all individuals to fall into same local optima with minimising possibility of damaging some important part of genetic information, [25].

Mutation operation highly depends on used encoding to convert solution to chromosome. General approach is that there is small probability M_{GA} that given part of chromosome of given individual will be changed.

Mutation in binary chromosome is usually implemented as negation of given bit. Real-valued mutation is based on same random increment or decrement of given gene, for example from normal distribution, [24].

High probability M_{GA} damages useful informations in chromosomes and low M_{GA} does not keep population diverse which make it prone to stuck in local optima.

2.0.7 Elitism

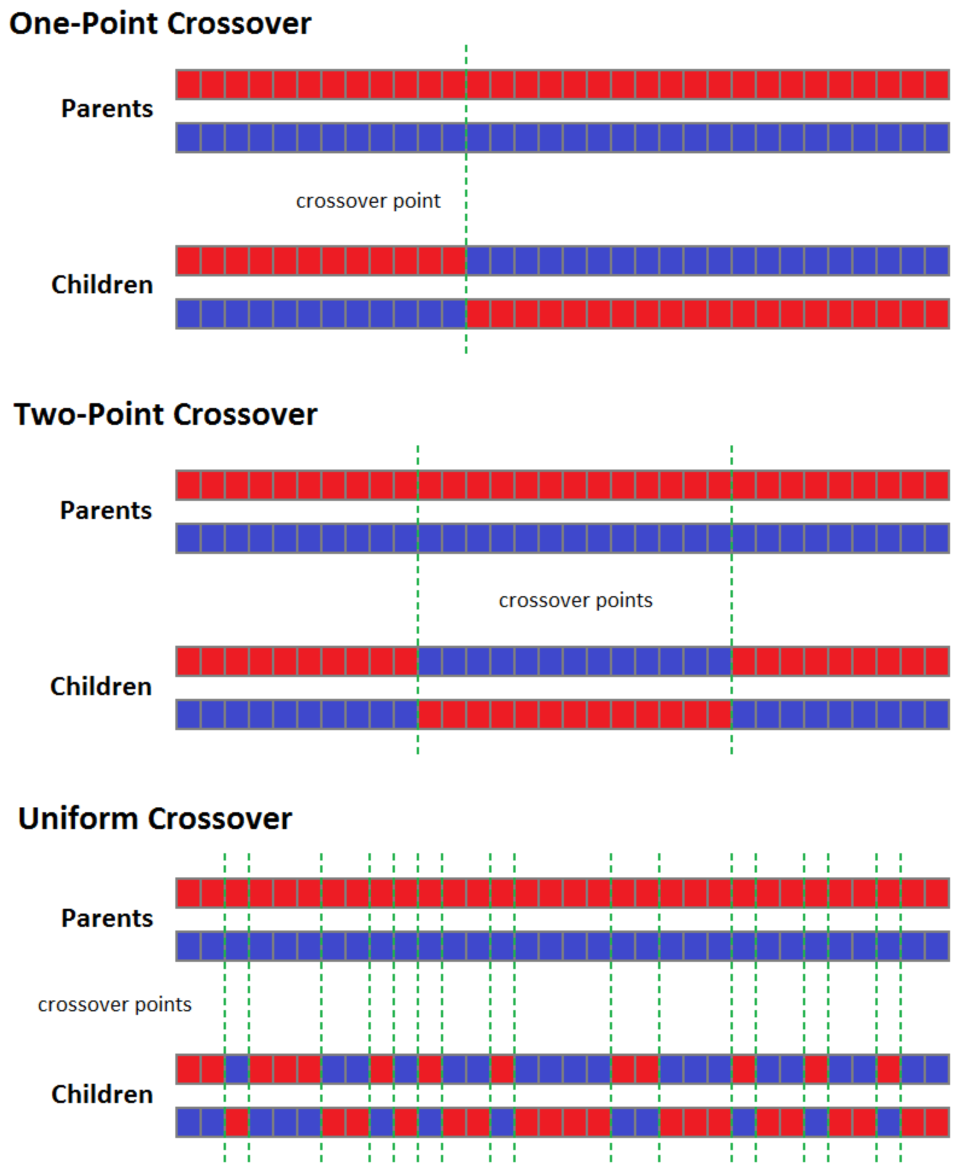
By cause of selection and mutation best individuals can be lost thus elitism approach will copy E_{GA} best individuals into next population without mutation.

2.0.8 Stop criteria

General stop criteria are time of execution, maximal number of generations and quality of found solution.

There might be additional stop criteria based on current problem definition.

Figure 2.4: Crossover mechanism



Chapter 3

Design

This chapter is focused on describing building blocks of the summary system and its structure, including workflow of data through it.

Proposed system for automatic text summarization is a combination of several algorithms stated in state-of-the-art and several designed algorithms. It is based on graph theory, where every node represents single sentence, and on principle of finding probability of visiting each node by stationary distribution of Markov chains.

This system is an *extraction*-based summary generation tool based on evaluation of *sentence importance* and *sentence similarity* (see section 3.4.4. From the point of *detail*, summaries can be scaled to arbitrary number of words not exceeding original document length and therefore it is possible to obtain ether *indicative* or *informative* summary.

System does not create *personalized* summaries and therefore generates summaries with *general* content. Due to a fact, that several methods analysing sentences importance are based on corpus frequency analysis, it is required to have a lot of documents to create corpus and dictionaries, from which particular frequencies are obtained. The system could be therefore considered as *domain dependant* if and only if all documents in corpus were specialized on certain topic or category of topics. Otherwise there is no further limitations on input text, which makes the system genre and structural *independent.* The system creates summaries from *single document* and works only with *English* written texts.

Selection of most important sentences into a summary works by gradually selecting most important sentences into summary until required number of words is reached. If the last sentence to be added exceeds required number of words it could be ether excluded (summary is shorter than required) or included (summary is longer than required).

After extensive testing, some parts of system were rewritten.

3.1 Workflow of system

There are two main system workflows: corpus preparation and document processing. They are composed by sequence of components, that are described in following sections 3.4 and 3.5.

3.1.1 Workflow of components for corpus preparation

Corpus is processed in the system in following order:

1. Selection of input documents to form corpus
2. Conversion of input documents into standardised structure
3. Preprocessing of documents for feature extraction
4. Frequency analysis of all documents in corpus
5. Massive feature extraction from documents in corpus
6. Training of weight settings

Training part can be skipped in order to significantly reduce time of corpus preparation at the expense of possible loss in quality of resulting summaries. This workflow is depicted in figure 3.1 and 3.2.

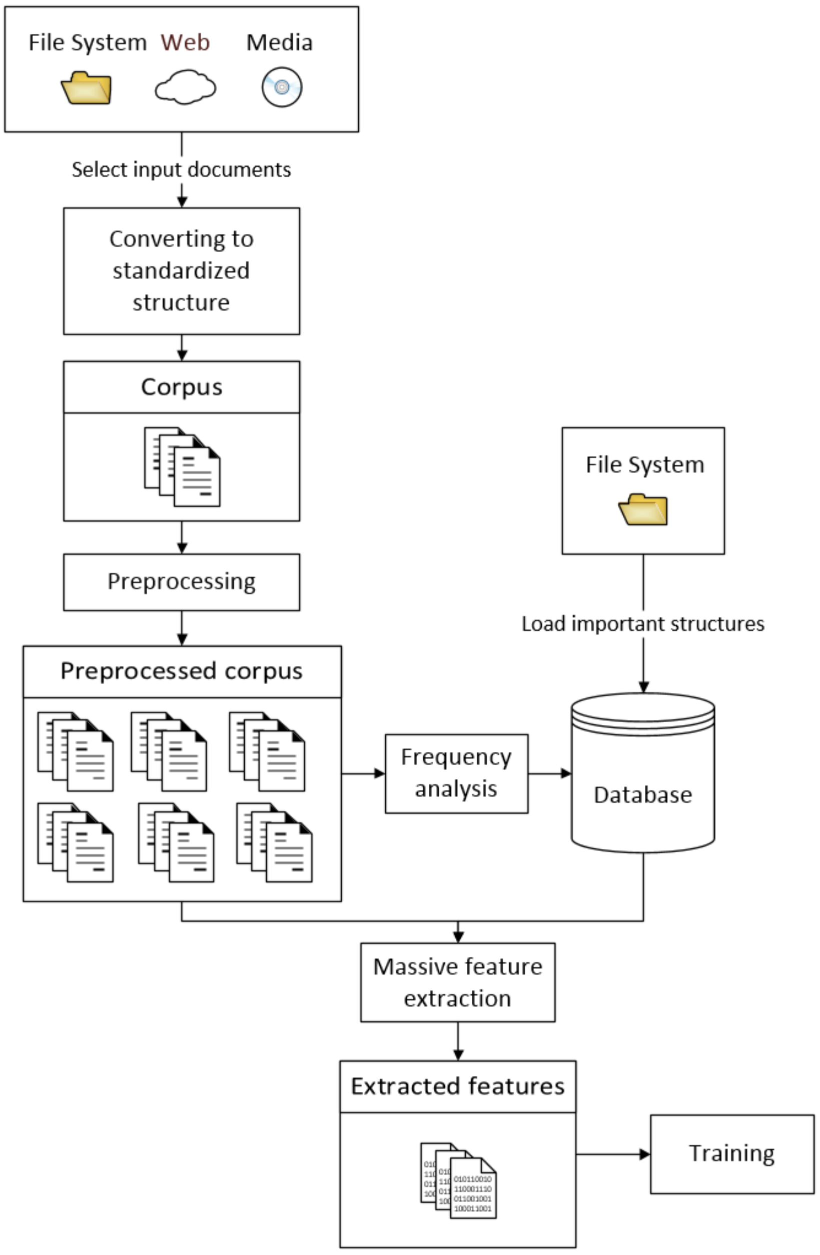
3.1.2 Workflow of components for processing one document

After corpus is prepared, single documents can be processed in order to generate summaries. The workflow is following:

1. Selection of input document
2. Conversion of input document into standardised structure
3. Preprocessing of document for feature extraction
4. Feature extraction from document
5. Thresholding of sentences
6. Generation of summary
7. External evaluation

Steps 2 to 4 are not necessary to perform, when selected document is from corpus. Step 7 is available only if there are some user-written summaries for given document, otherwise manual evaluation is required.

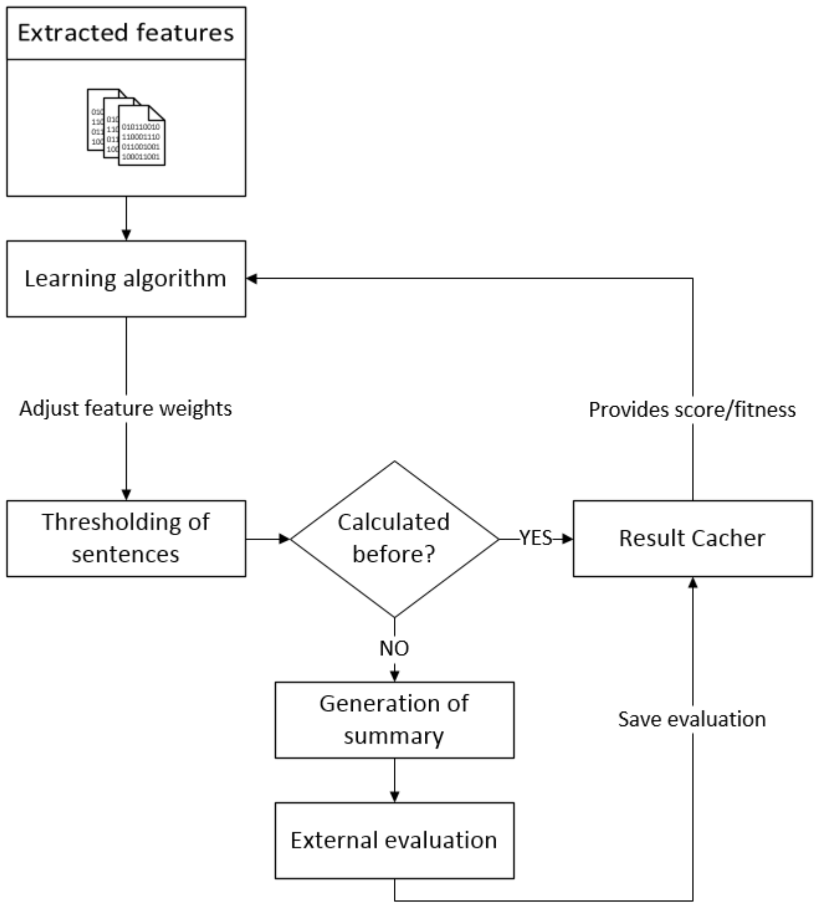
Figure 3.1: Corpus preparation workflow



3.2 Data acquisition

Since the proposed system requires user-written summaries for each document to be able to evaluate it (see section 1.3), it was imperative to obtain quite specific dataset for performance testing. From several sources, the Text Analysis Conference (former Document Understanding Conference) had the

Figure 3.2: Training process workflow



largest datasets, written and checked by large community of people. There are available datasets for various tasks in text mining including summarization, document clustering, text clustering, query-based summarization, cross-lingual multi-document summarization, question answering, textual entanglement recognition, knowledge base population and many more. Although almost all 14 annual conferences have datasets for text summarization, only years 2001 and 2002 had datasets for automatic text summarization systems matching the one that is proposed here. Together, those datasets contained approximately 1000 written documents and on average every document had 2 user-written summaries in length of 50, 100, 200, 400 words, which are important for comparison with system generated summaries. From this dataset, only documents starting with prefix AP where selected because of their rather consistent structure and only 100 words long user-written summaries are selected to test the system.

The dataset composes of 446 documents and 1062 user-written summaries.

3.3 System structures

The system is composed of many structures, where the most crucial ones determine representation of text document as object, its graph-based representation used for further calculations, structure used for keeping and processing evaluation result, dataset structure used for evaluation and structures holding important dictionaries, thesauruses and databases.

These structures are described in detail below.

3.3.1 Document structure

This structure is used to handle with a text documents. *Document* composes of *user-written summaries*, *title of document* and its *body*, which is composed by arbitrary number of *paragraphs*. *Paragraphs* further divides into *sentences*. Each of those structures of document contains *attributes*, where all outputs of preprocessing function (see section 3.4.2) are stored for further use by feature extraction functions (see section 3.4.4).

3.3.2 Graph structure

Graph is parent structure of *document*. The *document* is represented as *complete oriented weighted graph with loops*[1], where nodes are created by a *document* sentences and edges. Links[2] are calculated by *sentence similarity* measures (see section 3.4.4.2) and loops[3] by *sentence importance* measures (see section 3.4.4.1).

Due to the higher number of measures, the initial graph is represented by a set of graphs, each representing the results of one of these measures - this representation could also be interpreted as *multigraph*, where each two nodes have as many edges as there are measures, each representing the result of particular measure.

3.3.3 Dataset structure

Dataset is structure allowing selection of arbitrary number of random documents from whole corpus, which are then randomly divided into *training set* and *testing set* in given ratio. Those sets are mutually exclusive and exhaustive.

[1] *Complete oriented weighted graph with loops* is a graph, where each node is connected to all nodes (including itself) by exactly one edge in both directions. Those edges are weighted, which means that every edge has a value assigned to it, representing some magnitude of connection between nodes and since the graph is oriented, each direction (from node A to node B and from node B to node A) might have assigned different weight.

[2] Edge connecting two different nodes.

[3] Edge that connects node to itself.

Training set is used for learning the proper settings of feature weights by genetic algorithms of trainer algorithm - see section 3.4.7 and *testing set* is used to evaluate how well learnt some particular weights are.

Is also allows saving and loading some particular dataset, which is useful for tuning some functionality in the system.

3.3.4 Result structure

This structure stores everything that is dependent on summary generation and learning of feature weights - see section 3.4.7 for more detail.

Therefore, there are stored results of all calculations over document's graph structures, generated summaries and resulting comparisons between user-written and generated summaries and its evaluation.

3.3.5 Databases, dictionaries and thesauruses

The system requires several databases, dictionaries and thesaurus in order to be fully functional. These structures were refined to fit requirements of certain functions and to optimize performance in handling with them.

Those structures are following:

- *Stop words* list, created by merging of several existing sources, [26] - see section 3.4.2.
- *Nouns dictionary* containing words in singular and plural form for identification of nouns in a document and their conversion from plural form into singular form - see section 3.4.2. Dictionary was created using preprocessing of WordNet's list of regular nouns in singular form and list of irregular words in both singular and plural form, [13]. Plural forms of regular nouns were generated from singular using English grammar - see section 4.3.2.
- *English morphological dictionary* used by Snowball stemmer, [27] - see section 3.4.2, containing over 300 000 transitions between regular word and its stem. This stemmer also creates stems using some certain set of rules.
- *Corpus frequency databases* used for storing calculated frequencies from corpus and used for *tf-idf*, sentence similarity and other measures - see section 3.4.3.
- *WordNet* thesaurus capturing relations between words in hyponym to hypernym direction and vice versa (for example apple, pear and orange are hyponyms of fruit and fruit is hypernym of apple, pear and orange etc.), used by several word distance functions - see section 3.4.4.2.5. Thesaurus captures these relations between so called synsets, which are

groups of words with the same or similar meaning (synonyms). Those groups are then linked between each other by hyponym and hypernym relations.

3.4 Main system components

The system composes of functional blocks called components, which hold certain coherent purpose. Main components are directly connected to automatic text summarization process and works in main life cycle of this system from cleaning and transforming raw documents to summary generation.

Their detailed description follows:

3.4.1 Input document converter

Input document converter is used to clean documents and user-written summaries and transform them into strictly defined structures for this system to unify working with documents for components that follow.

3.4.2 Preprocessing of documents

All preprocessing functions in document get text input and output is also text, which allows system to arbitrary combine or chain those methods together. The preprocessed words are separated by normalized white spaces so if necessary, token splitting is not an issue.

Used preprocessing functions are following:

- *identity* - This function returns input text without any modifications. Its purpose is to ease constraint rules of the system on required document preprocessing and arbitrary chaining of functions.
- *stop words elimination* - This function removes all words that are present in stop words list. Purpose of this elimination is to purge document of words, prepositions, articles etc. that are too common in given language (for example: a, the, all, in, where, you, too, yes, no) and therefore don't have any informational value.
- *nouns and numbers extraction* - This function removes all words, that are not in nouns singular or plural dictionary, are not proper nouns (having capital letter) and don't contain any number. Its purpose was to keep only key information in text.
- *stemmer* - This function returns stem[4] of inserted word. Its purpose is to identify related words as same word (for example: fishing, fisher, fish, fished have all stem fish). Snowball stemmer, [27] was used.

[4]Part of word, that is usually its root.

3.4.3 Frequency analysis of corpus

This component calculates frequencies of words of all documents in corpus. There are 3 frequency analyses done.

Unigram analysis counts occurrences of every word in all documents in corpus, which is used for *tf-idf* method (see section 1.2.1.4).

Binary word document analysis counts for each word the number of documents, where this word occurs. This method is also used in *tf-idf* method.

Bigram frequency counts for every possible combination of two words in corpus, how often they occur in same sentence. This frequency is used for determining similarity of two sentences (see section 3.4.4.2).

Those analyses are performed on preprocessed text by all possible and logical combinations of preprocessing functions (see section 3.4.2).

3.4.4 Feature extraction of document

This component performs feature extraction from *document* sentences in order to calculate similarities between graph nodes. The initial set of graphs is created (see section 3.3.2).

Feature extraction is divided into two categories: *sentence importance* and *sentence similarity.*

3.4.4.1 Sentence importance

Sentence importance measures are used for calculating magnitude of information or significance. Those measures are used to calculate weights of loops[5] in document graph thus these measures creates graphs with only loops. Given the fact that those measures were introduced in state-of-the-art, only additional information are presented here.

Input of these measures is sentence of document and output is real number representing sentence magnitude.

Used measures are following:

3.4.4.1.1 Location measure Calculation of the location measure is:

$$Location_{s_i} = \frac{1}{4 + DOC_{s_i}} + \frac{1}{10 + PAR_{s_i}},$$

where DOC_{s_i} and PAR_{s_i} are indexes of orders of given sentence s_i in the document and the paragraph, where i is index of sentence in the document.

Measure is set in a way that sentence location in the document is more important than in the paragraph. Sentences in both beginnings of document and paragraphs are valued the most.

[5]Edge starting and ending in same node.

3.4.4.1.2 TF-IDF measure This measure is calculated in following way:

$$TF\text{-}IDF_{s_i} = \frac{\sum_{i=1}^{|\mathcal{S}_i|} tf\text{-}idf(word_j)}{|\mathcal{S}_i|},$$

where $\mathcal{S}_i$ is set of words in given sentence s_i and $word_j$ is particular word from that set and *tf-idf* is explained in section 1.2.1.4.

This measure presents overall specificity of words in given sentence with regard to whole corpus.

3.4.4.1.3 Title, Uppercase and Numeric measures Those measures are calculated almost the same. The difference is in to what they compare, whether to document title or simple dictionary of rules.

$$Title_{s_i} = \frac{|word : word \in \mathcal{S}_i \wedge word \in \mathcal{D}|}{|\mathcal{S}_i|},$$

where $\mathcal{S}_i$ is set of words in given sentence s_i and $\mathcal{D}$ is set of words in document title, or virtual set of all words with capital letter or with number.

Measures *TF-IDF*, *Title*, *Uppercase* and *Numeric* are parametrized by all previously mentioned methods in *preprocessing* (see section 3.4.2).

3.4.4.2 Sentence similarity

Sentence similarity measures are used for determining magnitude of dependence or similarity between two different sentences. Those measures are used to calculate weight of edges in document graph, that are not loops and therefore these measures create graphs with edges without loops.

Input of these measures are two different sentences from graph and output is real number representing magnitude of mutual sentence dependence or similarity.

Designed measures are following:

3.4.4.2.1 Same Tokens Ratio measure This measure compares words that are same in both sentences. Calculation is following:

$$SameTokensRatio_{s_i,s_j} = \frac{|\mathcal{S}_i \cap \mathcal{S}_j|}{|\mathcal{S}_i \cup \mathcal{S}_j|},$$

where $\mathcal{S}_i$ and $\mathcal{S}_j$ are sets of words in given sentences s_i and s_j.

The more similar these sentences are, the higher the output will be.

3.4.4.2.2 Consistency measure This measure is used to link consecutive sentences to increase chance, that they will appear together in summary and thus increase its readability and consistency. The closer the sentences are in document and the closer the paragraphs containing those sentences are, the higher the output will be.

Calculation is following:

$$Consistancy_{s_i,s_j} = \frac{1}{doc_{diff_{s_i,s_j}} * (par_{diff_{s_i,s_j}} + 1)},$$

where $doc_{diff_{s_i,s_j}}$ and $doc_{diff_{s_i,s_j}}$ is differences of positions between sentences s_i and s_j, and differences of paragraph positions containing those sentences, respectively.

3.4.4.2.3 Bigram frequency measure This measure uses precalculated corpus frequencies explained in section 3.4.3. Basically from input sentences, complete bipartite graph[6] is constructed from words of each sentence, where every edge weight is equal to calculated bigram frequency of those words it connects. Those weights are then summed and normalized.

Computation is following:

$$BigramFreq_{s_i,s_j} = \frac{\sum_{x \in \mathcal{S}_i} \sum_{y \in \mathcal{S}_j} frequency_{bigrams}(x, y)}{|\mathcal{S}_i \cap \mathcal{S}_j|},$$

where S_i and S_j are sets of words from both sentences s_i and s_j and $frequency_{bigrams}$ is function returning bigram frequency of two given words.

This measure is similar to *Same Tokens Ratio measure*, but it favours sentences whose words frequently appear together in corpus.

3.4.4.2.4 Cosine distance measure This measure is based on classic cosine distance function between two one-dimensional vectors. Vector is represented as frequencies of words in given sentence. Its length equals cardinality of set of words from both sentences. Each position in vector represents *tf-idf* value of specific word in that set and zero on positions representing words that are not present in current sentence.

Calculation is following:

$$\cos\theta = \frac{\vec{f_i} \times \vec{f_j}}{||\vec{f_i}|| * ||\vec{f_j}||},$$

where $\vec{f_i}$ and $\vec{f_j}$ are *tf-idf* vectors of sentences s_i and s_j.

[6]Complete bipartite graph is graph, whose nodes are divided into to disjoint set. Each node from one set is connected to all nodes from the other set and there is no connection between nodes in the same set.

3.4.4.2.5 Word distance measures Those measures are used to measure semantic similarity and relationship between two words. All of these used measures are based on information retrieved from lexical database *WordNet*. For these measures, two *Java* packages were used: JWS, [28], and WS4J, [29]. Both packages provide similar measures. Only fast measures and those with output range between 0 and 1 were selected.

Selected measures are following:

- *Path*: This measure expresses similarity as reciprocal function to distance between given two words in hierarchical thesaurus of synonyms.
- *Lin*: This measure expresses similarity rate between information gain of the most specific word, whose meaning given two words share and difference of information between those two given words.
- *WuPalmer*: This measure expresses similarity as depth of shortest path to most specific word, whose meaning given two words share. Normalized by depth of both of those words.

Measures *Same Tokens Ratio*, *Bigram frequency* and *Cosine distance* are parametrized by all possible combinations from preprocessing. *Word distance* measures are parametrized only by combinations excluding stemming due to its destructive nature to words.

3.4.5 Massive feature extraction of corpus

Feature extraction of one document took long time and given the frequent testing of system, it was convenient that all documents would be processed only once and extracted features would be stored into file. Hence time to summarize one document would significantly drop.

3.4.6 Thresholding of sentences

After document feature extraction, which is represented as set of graphs, it is necessary to merge all those graphs into one. It is performed in three steps. In first step, all graphs are independently normalized, that sum of all weights of edges coming out of a node is equal to one (normalization to same scale).

Second step merges graphs with only loops, and graphs with edges without loops into two graphs and also normalizes sums of all weights of edges coming out of a node to be equal to one. This normalization, however, can be performed in two ways: *without training*, where each outgoing weight is divided by sum of corresponding weights from all graphs or *with training* (see section 3.4.7), where all graphs (their weights) are firstly multiplied by values (feature weights) acquired from training and then divided. Feature weights indicate which measures should be prioritized in order to maximize results, otherwise those measure are considered to be equal.

Last step merges remaining two graph together with same technique, thus creating graph that satisfies conditions of Markov chain.

Because graphs are represented by adjacency matrices, this Markov chain is represented by its *stochastic matrix* $\mathcal{P}$ created by the normalization.

Then, selecting sentences for summary works in such a way, that this *stochastic matrix* is repeatedly squared until values in it converge to some steady numbers. By that approximation of stochastic matrix after infinite number of executed transitions in chain between arbitrary nodes is acquired. After that, it is easy to calculate *stationary distribution* π (overall probability of visiting all nodes) from axioms of probability and rules in Markov chain, because after squaring of that matrix, each column of matrix $\mathcal{P}^{\infty}$ consist of the same values (${\mathcal{P}^{\infty}}_{1,j} = {\mathcal{P}^{\infty}}_{2,j} = \ldots = {\mathcal{P}^{\infty}}_{n,j}$).

From statement about *stationary distribution* arises, that $\pi\mathcal{P} = \pi$. After multiplying and modifying of this equation, following relationship for each π_i emerges:

$$\pi_i = \mathcal{P}_{*,i} * (\pi_1 + \pi_2 + \ldots + \pi_n),$$

where $\mathcal{P}_{*,i}$ is arbitrary number from column i, because they are all the same. And because of sum of all probabilities is from definition equal to one ($\sum \pi_i = 1$), it implies that $\pi_i = \mathcal{P}_{*,i} * 1$, thus $\pi_i = \mathcal{P}_{*,i}$.

This way, the vector π is obtained, which illustrates degree of importance of each sentence.

Simplified process is depicted in figure 3.3.

3.4.7 Training of feature weights

This component is used to fine-tune proper setting of feature weights, used in previous section 3.4.6 for prioritizing certain measures in order to maximize performance of extraction of significant sentences from document in certain domain (corpus).

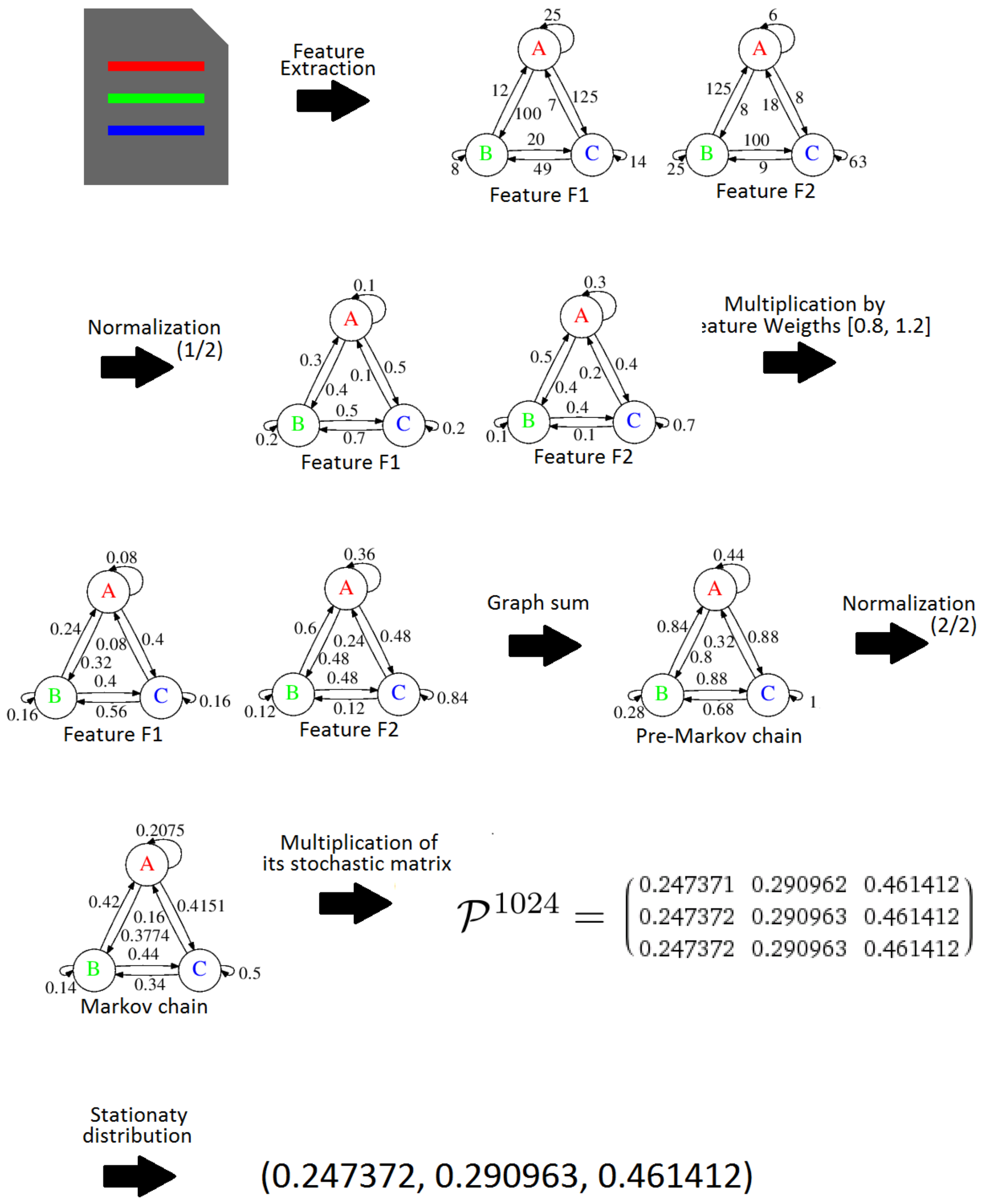
Two basic training mechanisms were developed:

3.4.7.1 Trainer

This algorithm was designed to be easily implemented. When considering all logical and possible features, 56-dimensional space would have to be searched. All features are visible in table 5.3.

Dimensionality reduction is achieved by gradually selecting each setting of one feature extraction measure (feature weight of that measure with particular setting is one, other weights are set to zero) and running summarization process on whole train set. Resulting score (see section 3.4.9) is saved for all settings and only the best performing setting is selected for that measure. This procedure is repeated for each measure. By this process, search space is reduced from 56 to 10 dimensions.

Figure 3.3: Simplified process of sentence thresholding



$$\mathcal{P}^{1024} = \begin{pmatrix} 0.247371 & 0.290962 & 0.461412 \\ 0.247372 & 0.290963 & 0.461412 \\ 0.247372 & 0.290963 & 0.461412 \end{pmatrix}$$

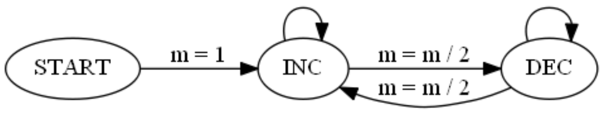
(0.247372, 0.290963, 0.461412)

Given the long processing of train set, instead of searching large portion of vector space, proposed method is used to find the closest local optimum from initial start point. It is achieved in following way:

1. Since range of values in settings vector doest affect output and only proportions between each value does, as initial start point of search, vector composed of high-scores achieved in dimensionality reduction was selected.

2. This vector is then adjusted by min-max normalization so lowest value would start at one.

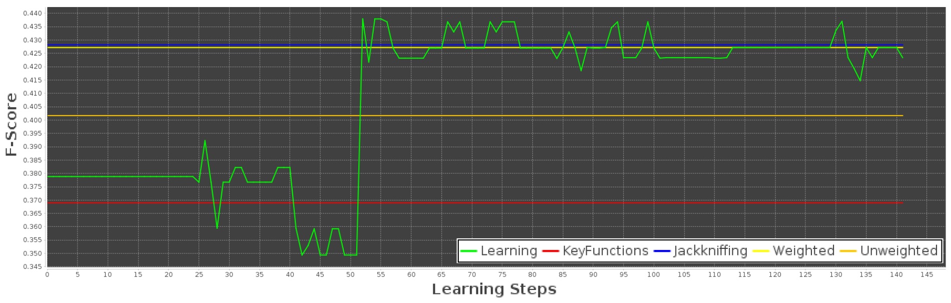
3. Algorithm then iteratively searches for all weights, from highest initialized weight to lowest, to find better solution in a way, that other weights are fixated and following automaton is executed until $m \geq \frac{1}{8}$.

   In each state of the automaton, given value of weight is adjusted according to parameter m (INC: $weight = weight * (1 + m)$ and DEC: $weight = weight/(1 + m)$). Then new score is calculated (see sections 3.4.6 and 3.4.9) from newly adjusted weight vector on train set. If new score is better than previous, the automaton continues in same state, otherwise it transitions to another state. Also if score doesn't change in 5 last iterations automaton execution ends.

4. This automaton is executed for all 10 weights one after another. Example of learning process is display in figure 3.4.

Figure 3.4: Sample of trainer algorithm process



3.4.7.2 Genetic algorithms

Given the high dimensionality and vast search space it is not possible to use complete space search algorithm, thus genetic algorithms are used. They are well suited for this problem because of their flexibility and independence to search space cardinality.

From initial testing there were some features that seemed more important than others but in order to keep diversity high, prevent bias and converging of generation to local minima, generation zero is initialized randomly in whole search space.

Chromosomes are represented as vectors of 56 real-valued weights[7] from 0 to W_{GA}. They represent feature weights and no further encoding is needed. Those weights are visible in table 5.3 and their index in chromosome is identical to ID, that is visible in that table.

Fitness of each individual is represented as *F-score* of ROUGE-1 metric calculated as similarity between user-written summaries and system-generated summaries in training set of documents. This score is in range from 0 to 1 and is typically around 0.4. In order to increase distance between the strongest and the weakest individual this score was raised to 10. This is important for roulette selection to increase probability of selecting better individuals (see section 2.0.4.1).

Both roulette and tourney selection are implemented and tested.

Creation of children from parents via crossover was inspired by uniform crossover and Whole Arithmetic Crossover (see section 2.0.5). Combining function for each gene is randomly selected from subtraction, minimum, average, maximum and addition. This way whole child is created and if any gene value is out of range it is set to its closest boundary.

Same process as in crossover is also used in mutation. Random number from gene range is picked and is combined with that particular gene by randomly selected function.

Elitism is also used to keep best individuals in population. Stop criterion is only number of generation G_{GA}.

Whole process of genetic algorithms is depicted in figure 3.5 and sample of genetic algorithms process is shown in figure 3.6. This picture shows performance in each generation, showing best and worst individuals from population evaluated on train set (blue and yellow) and test set (green and red). There are also plotted two lines, lighter red visualizing performance without any learning (all features are considered equally important) and orange visualizing quality of user-written summaries.

[7]Note that as proposed in [24], not only evolution strategies but also genetic algorithms can have real-valued chromosomes.

Figure 3.5: Workflow of genetic algorithms in automatic text summarization

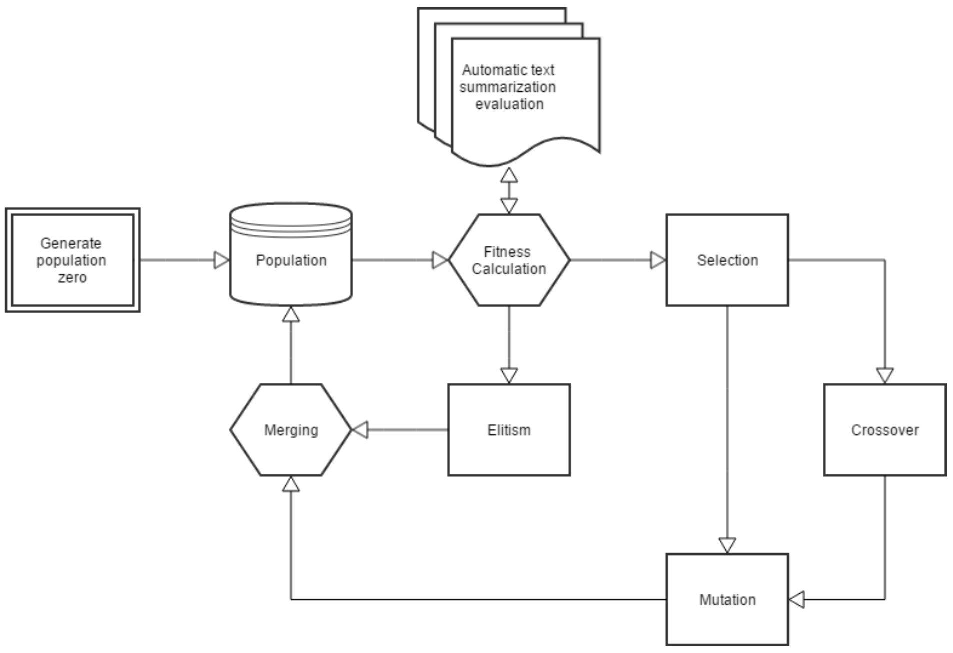
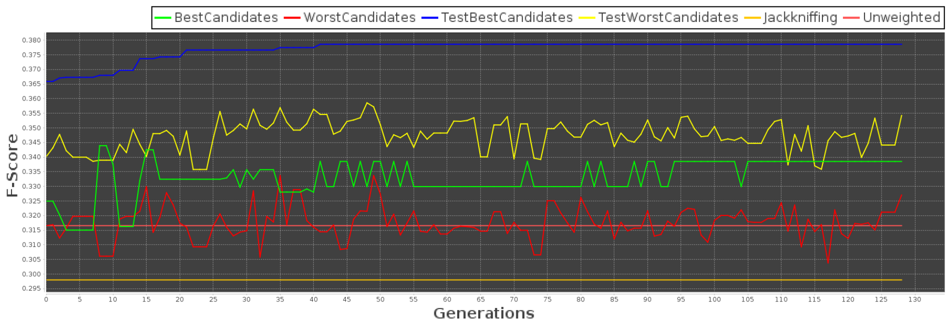


Figure 3.6: Sample of genetic algorithms process



3.4.8 Generation of summary

Input of this component is previously obtained vector π (see section 3.4.6) and required number of words in summary including flag if this number could be exceeded by last added sentence.

Sentences are then selected for a summary, order from most important, until exact number of required words is reached or until last added sentence exceeds this number, then this sentences is ether added or discarded, depending of input flag.

Sentences form summary are then sorted to match their original order and this summary is then saved to file system.

3.4.9 External evaluation

External evaluation is performed by *ROUGE* metrics. First step before running this algorithm is to convert user-written and system-generated summaries into defined format, [30], for *ROUGE* and create settings file, containing metadata specifying way of evaluation.

Hence, similarity measuring is done by one metric ROUGE-1, formerly across all metrics (see section 5.1). Resulting *F-score*, *Precision* and *Recall* expressing similarity between user-written and system-generated summaries are loaded back into the system and used as an indicator of the quality of the generated summarization.

Overall *score*, used to evaluate quality of system-generated summaries is represented as difference between arithmetic average of *F-score* of all used metrics and *F-score* of so called *jackknifing*.

Jackknifing is method used to estimate quality of user-written summaries by evaluating their mutual similarity. If there is K user-written summaries for one document, for every of this summary, ROUGE metrics are calculated between it and remaining $K - 1$ summaries. Final result is average of all those results. This value indicates mutual information and information consistency in all user summaries and that it is unlikely, that similarity between those summaries and system-generated will be higher.

This component also cooperates with *result cacher* (see section 3.5.5) to prevent the system from computing the same calculations again by storing them into cacher and recalling them if necessary.

3.5 Additional system components

Some of additional system components are used as support components to main system components: user settings in *settings holder*, caching results in *result cacher*, manipulation with text in file system by *text handler*, logging by *logger* and loading of required structures by *loader*.

Another components work to ease testing of evaluation and understanding of the system by printing charts by *chart plotter* and capturing computational time by *time capturer*.

Last component *BBC summarizer* is used for demonstrating summarization process on BBC articles.

Detailed description follows.

3.5.1 Settings holder

Settings holder keeps system settings in one place and allows to set up the system in following:

- verbosity
- path of
  - stop words dictionary
  - raw documents
  - cleaned documents
  - precalculated documents - see section 3.4.5
  - cached results - see section 3.5.5
  - rouge evaluator
  - temporary files
    - of user-written summaries
    - of system-generated summaries
    - of settings for rouge metric
    - of rouge calculations
  - nouns dictionary - see section 3.3.5
  - corpus frequency database - see section 3.3.5
  - results including charts
- rouge metrics to be used in comparison - see section 1.3
- whether to evaluate also on test set during training (for chart output, not for learning)
- charts resolution
- whether to delete temporary files after computations
- thread count

3.5.2 BBC Summarizer

This component serves for summarizing BBC articles. It was developed in order to perform and validate the summarization system on real data instead of just prepared datasets.

When link on BBC article is provided, all other BBC articles that are being linked to from given article, are also downloaded. This set of documents then serves as corpus on which steps from workflow in section 3.1.1 are performed. Then given article is processed and summarized by steps from workflow in section 3.1.2. Excluding components for training feature weights and external evaluation because of lack of user-written summaries.

Output of this component is html document displaying summarized sentences in bold and other sentences in normal font.

3.5.3 Time capturer

Time capturer builds hierarchically organized structure of every part of the system (for example training iteration, dimensionality reduction, testing etc.) during computation and captures time spent in these particular parts. After the computation, time capturer computes overall time spend in each part of system[8] that can be seen in snippet 3.1 including real-time in last row.

Snippet 3.1: Time capturer sample output

```
[TRA][D25][R0] - 15 minutes 56 seconds 751 ms
  Test - 0 minutes 6 seconds 136 ms
    Jackknifing - 0 minutes 0 seconds 2 ms
    Testing - 0 minutes 6 seconds 134 ms
      KeyFunctions - 0 minutes 0 seconds 978 ms
      Weighted - 0 minutes 0 seconds 517 ms
      Unweighted - 0 minutes 4 seconds 639 ms
  Train - 15 minutes 50 seconds 615 ms
    Learning - 3 minutes 39 seconds 773 ms
      TrainStep-152 - 0 minutes 1 seconds 673 ms
      ...
      TrainStep-1 - 0 minutes 1 seconds 686 ms
    DimensionalityReduction - 12 minutes 10 seconds 842 ms
      Numeric-Stemmer-StopWords - 0 minutes 16 seconds 786 ms
      TF-IDF-Stemmer - 0 minutes 8 seconds 81 ms
      Consistency - 0 minutes 22 seconds 985 ms
      Bigrams-StopWords - 0 minutes 3 seconds 531 ms
      Uppercase-Nouns - 0 minutes 40 seconds 46 ms
      Cosine - 0 minutes 13 seconds 229 ms
      Distance-WS4J-WuP-Nouns - 0 minutes 10 seconds 55 ms
      Location - 0 minutes 7 seconds 344 ms
      ...

Done in 5 minutes 56 seconds.
```

[8]This time is not real-time but time spent on every core of CPU together.

3.5.4 Chart plotter

Chart plotter is used in order to visualize results from the system. It draws charts of thread computational time divided into several categories including training and testing and charts of learning cures, whether from trainer algorithm or genetic algorithms - see section 3.4.7.

Charts of learning curves are displayed in figure 3.4 and 3.6.

3.5.5 Result cacher

Result cacher is a database, which for every document in corpus records all results of ROUGE comparison between user-written and system-generated summaries. If system tries to evaluate any generated summary again, cacher recollects previous results without the need of costly computation by ROUGE.

3.5.6 Text handler

Text handler contains all necessary functions for loading raw text documents and user-written summaries and for saving textual data like computational time, results of comparison of summaries by ROUGE and logs.

3.5.7 Logger

Logger stores ongoing progress in current run of system, if the settings allow it. Logs can be printed ether to standard output or log file.

3.5.8 Loader

Loader is used to load preprocessed documents, dictionaries, databases and thesaurus in text format into the system's object format.

Chapter 4

Realisation

This chapter is focused on implementation of whole automatic text summarization system. Due to the fact, that analysis and design was described in detail (see chapter 1 and 3), only important or interesting parts of implementation are mentioned here.

Sections are ordered chronologically as they were implemented and as they are executed in system.

4.1 Programming Language

Programming language was selected by following criteria:

- personal experience,
- portability of language,
- ease of debugging,
- ease of creation of user interface,
- simplicity of scalability,
- efficient creation of graphic elements,
- availability of external libraries.

After a closer selection of programming languages, I had to decide between C#, C++ and Java. The main disadvantage of the C# language is its development and support mainly by Windows platform. Another disadvantage is that support for other than business libraries almost does not exist.

Unlike C#, C++ language is supported on both major platforms - Windows and Linux, but its debugging is not so perfect and effective. Furthermore, making the user interface and other simple graphic element in the C++ language requires an external library and advanced knowledge of this language.

The best choice for the implementation of my work therefore remains Java, with which I have the most experience. Java is also portable across platforms and is easily debugged. In addition, it has great support in text editors, profilers and external libraries. Also library for creating simple user interface and other design elements is already integrated.

4.2 Documents conversion

Document conversion is necessary in order to standardise and clean documents to unify their further processing.

AP prefixed documents obtained from DUC have following structure:

- Documents
  - Documents are saved as text files. Whole document is enclosed in tag `DOC`, which contains many tags including `HEAD` and `TEXT`. `HEAD` tag contains document title and could appear multiple times and `TEXT` contains article with paragraphs separated by at least one newline.
- Summaries
  - Summaries are saved as text files with content enclosed in tag `SUM`. This tag contains many attributes. Three of them are important: `DOCREF`, `TYPE` and `SIZE`. `DOCREF` value is name of summarized document, `TYPE` determines whether it is written summarization from multiple documents (value `MULTI`) or single document (value `PERDOC`) and `SIZE` determined summary length.

Standardized structure is following:

- Documents
  - Text file name is composed with document name and postfix `.docu`, for example `ap900802-0118.docu`. Document starts with arbitrary tag `TITLE` containing document heading followed by several tags `PARAGRAPH` containing sentences separated by tag `SENTENCE`.
- Summaries
  - Text file name composes of source document name and postfix `.user.sum.` followed by number of summary, for example `ap900802-0118.user.sum.3`. Whole summary is enclosed in tag `USER-SUMMARY` and its contents are separated sentences by tags `SENTENCE`.

To convert files from their original format to new format, several steps were performed.

1. Identify proper files
   - Summaries of `TYPE PERDOC` and `SIZE` 100 with existing source documents were selected.
   - Documents having at least one summary were selected.
2. Identify structures and remove all unnecessary tags
   - Summaries doesn't contain any paragraphs, thus only sentences were separated.
   - Content of first `HEAD` was selected as document title (other `HEAD` tags usually contained only additional info, url etc.). Then paragraphs were separated and finally sentences in each paragraph.
3. Enclosed identified structures with defined tags
4. Save documents and summaries

Whole conversion is performed solely by regular expressions. Separating sentences was firstly performed by GATE's ANNIE Sentence Splitter, but its performance was low in both speed and quality. One of its mistakes was to separate `"This is direct speech."` as two sentences `"This is direct speech.` and `"`, another mistake was to separate sentences by dots, that followed abbreviations. Sentence splitting is thus performed by regular expression that was tuned to given dataset of documents and summaries.

4.3 Building databases

4.3.1 Stop words list

Stop words list was created by merging approximately 20 found stop words lists from various sites. All of those lists had to be converted into the same format before merging - lowercased words, one on each line.

`HashSet` collection was used for merging, which doesn't allow duplicates to be inserted. All words from all list were then loaded into this set and saved afterwards into one final stop words list.

4.3.2 Nouns lists

Nouns list was used to identify nouns in text. To do so, nouns in both singular and plural form were required.

Most of those words were extracted from *WordNet* thesaurus. It contains two list `index.noun` and `noun.exc`. First file contains all regular nouns in singular form and second file contains irregular nouns in both singular and plural form.

In order to obtain plural forms of regular nouns, English grammar rules for transforming singular nouns to plural were used. Those rules were following:

- IF singular word ends on *consonant* followed by letter *y*
  THEN plural form is created by replacing *y* with *i* and adding *es*
- ELSE IF singular word ends on *s*, *x*, *z*, *ch*, or *sh*
  THEN plural form is created by adding *es*
- ELSE plural form is created by adding *s*.

4.4 Preprocessing

4.4.1 Chaining of preprocessed functions

It is possible to chain preprocessing functions to get accordingly preprocessed data, for example stemmed nouns or frequencies of all words in document, excluding stop words.

This chaining is implemented by required parameter in `constructor` in preprocessing function, requiring another preprocessing function, which will be performed before this function. Identity function had to be implemented, in order to end chain of required preprocessing functions.

4.4.2 Preprocessing on-demand

Given the number of possible combinations in chaining preprocessing functions, all of these functions are run on-demand, meaning that they are executed only if some measure performing feature extraction tries to access preprocessed data that were not yet preprocessed.

Preprocessing functions are evoked by the `AttributeResolver` and results are stored into the `Attribures` structure (see next section 4.4.3), therefore each method of preprocessing is run at most once.

In order to properly use the `AttributeResolver`, each feature extraction function has list of required attributes and flags indicating what part of document has to be preprocessed (sentences, paragraphs, title or/and whole document) and by what function. For example function calculating *tf-idf* needs only whole document to contain frequencies of all words, thus frequencies of all words in each individual sentence are not required and therefore not calculated.

The `AttributeResolver` then recursively passes through chained preprocessing function until its end is reached and then starts preprocessing required parts of document in reverse order.

4.4.3 Storing preprocessed data

After documents are preprocessed, it is necessary to store resulting texts into some structure with relations to original sentences, in order to identify connection with measured similarities of preprocessed sentences and original ones that will be afterwards selected for summary.

As a solution, abstract class `Attributes` was implemented, containing `HashMap` what would store any data to associated `string` key. Every preprocessing function has its own `string` name, which is chained with name of any preceding preprocessing function performed on input text and this resulting name was used as `string` key to store and recall preprocessed text.

`Attributes` were then implemented by all structures of document (sentences, paragraphs, title and whole document) to hold appropriate preprocessed data.

4.5 Feature extraction

4.5.1 Symmetry

In order to speed up feature extraction, every similarity measure has flag whether it is symmetrical, meaning that similarity between sentences a and b is the same as similarity between sentences b and a.

If functions are symmetric, only part above diagonal of adjacency matrix is calculated and rest is copied.

4.6 Sentence thresholding

4.6.1 Stationary distribution calculation

As described in section 3.4.6, *stationary distribution* is calculated by powering stochastic matrix until its values stabilize.

Since matrix is dense, there is no need for any special representation of it. This matrix is implemented as 2D array of type `double` and calculation is performed by using another 2D array, which serves as cache for results. To significantly speed up process of powering, calculation is performed by multiplying result of previous multiplication with itself instead of original matrix. Results from iterations are then $\mathcal{P}^1$, $\mathcal{P}^2$, $\mathcal{P}^4$, $\mathcal{P}^8$, $\mathcal{P}^{16}$ etc. instead of $\mathcal{P}^1$, $\mathcal{P}^2$, $\mathcal{P}^3$, $\mathcal{P}^4$, $\mathcal{P}^5$ etc.

First try was to calculate as many iteration as possible to create most precise *stationary distribution* vector. After approximately 20 iterations ($\mathcal{P}^{1048576}$), numerical errors started to appear and resulting matrix was after several more iterations filled with infinities or zeros.

To compensate this numerical error, type `double` was replaced with type `BigDecimal` which is capable of calculation with arbitrary finite precision.

Calculation was then able to iterate more than thousand iterations, but it was considerably slower. Since this operation is one of the most crucial and most frequent, `BigDecimal` was abandoned.

Final type is again `double`, but in addition, two stopping criteria were added to power iterations. First is to stop calculation after 100 iterations and second is to stop calculation when sum of any row of resulting matrix vary by more than 10^{-4} from one.

Values in columns of resulting matrix differ usually by less than 10^{-3} and resulting stationary distribution is computed as column-wise average.

4.6.2 Parallelization

Since sentence thresholding is performed for every document in dataset per every fitness evaluation of individuals in genetic algorithms or by every step in dimensionality reduction and following learning, it was required to maximize parallel processing capabilities of this component and reduce overhead and synchronization traffic.

That was achieved by separating static and dynamic data of document in a way that static data were accessible by all threads and dynamic data were created inside threads, without any accessibility from outside of that particular thread. This way, no synchronization is necessary, because static data are used only for reading and dynamic data are handled locally in threads (accept barrier collecting final results).

Static data are text document, user-written summaries, extracted features and first step of feature normalization.

Dynamic data are everything related to computation: final normalization, Markov chain, multiplied matrix, stationary distribution vector, generated summaries and final evaluations.

4.7 External evaluation

4.7.1 Rouge

External evaluator *ROUGE* is written in perl. This perl script needs files of user-written summaries, system-generated summaries and settings to be created and saved to file system in order to process them (passing documents via parameters in not possible). This format was used from tutorial in [20].

Summaries are formatted in HTML structure, each sentence wrapped in `ahref` tag with unique id.

Settings is `xml` file containing *eval* points (ie. independent measuring sessions) specifying *peer* files (ie. system-generated summaries) and *model* files (ie. user-written summaries). *Eval* points are used to separately calculate similarity between user-written and system-generated summaries and *jackkniffing.*

After file generation, script is externally run from Java, capturing its output:

Snippet 4.1: Example of ROUGE output

```
ap880217-0175.136+ ROUGE-1 Average_R: 0.65714
 (95%-conf.int. 0.65714 - 0.65714)
ap880217-0175.136+ ROUGE-1 Average_P: 0.52273
 (95%-conf.int. 0.52273 - 0.52273)
ap880217-0175.136+ ROUGE-1 Average_F: 0.58228
 (95%-conf.int. 0.58228 - 0.58228)
...
```

After all files from dataset are evaluated, those outputs are loaded back into the system and parsed into more readable format, keeping only *F-score*:

Snippet 4.2: Example of parsed ROUGE outputs

```
+-------------------------+--------+
|F-score                  |ROUGE-1 |
+-------------------------+--------+
|ap880217-0175.136+       | 0.58228|
|ap880217-0175.96-        | 0.55277|
|ap880318-0051.110+       | 0.33970|
...
|ap901130-0060.99-        | 0.36875|
|ap901203-0166.111+       | 0.17416|
|ap901203-0166.94-        | 0.19195|
+-------------------------+--------+
|ap880217-0175.jack       | 0.52775|
|ap880318-0051.jack       | 0.44993|
|ap880324-0193.jack       | 0.32758|
...
|ap901127-0119.jack       | 0.28800|
|ap901130-0060.jack       | 0.43060|
|ap901203-0166.jack       | 0.37796|
+-------------------------+--------+
```

4.7.2 Dataset evaluation

When genetic algorithms or trainer algorithm require evaluation of given feature weights, dataset is evaluated on train set. Test set is used to evaluate final learnt feature weights.

Each time dataset is evaluated, new summaries needs to be generated according to supplied feature weight vector. Those summaries with additional required data (see previous section 4.7) are then evaluated and results are parsed into the system. Resulting *F-score* is then calculated as average of F-scores of individual documents in given set.

4.8 Result cacher

Since genetic algorithms and trainer algorithm takes long time to compute, datasets used for testing were small to decrease time required for testing. In order to test on bigger datasets, it was required to somehow optimize those algorithms. From previous testing it was discovered, that longest time of execution is by computing ROUGE metrics between summaries.

First idea was to precalculate all possible summaries with given number of words from every document with simple branch and bound algorithm allowing only summaries in given range of words. It was discovered that number of possible combinations of sentences from documents is too vast. Most documents were computationally unprocessable by this way.

Thus, direct preprocessing was not possible and more indirect approach was taken. Before summary is generated from given vector of weights, binary vector representing sentences from document in summary is looked up in hash table, if it was not calculated before and if so, no further calculation is required and results are taken from that cache. If that combination of sentences was not calculated before, system has to generate summary from given binary vector and document, store that summary and user summaries to file system, generate settings file and execute ROUGE metric, parse results from file and then store this result into that cache.

Loading of documents (when building dataset) also loads caches for each document from file system and after computation, all caches are updated and saved back into file system.

Results are saved in concurrent `HashMap`, which copies itself before writing operation and this copy is then used for writing and original `HashMap` for reading. All operations are therefore concurrent and non-blocking, which makes usage of this map perfectly suited for this task. After writing is done, original map is replaced by its up-to-date copy.

4.9 Training of feature weights

4.9.1 Trainer algorithm

4.9.1.1 Dimensionality reduction

Dimensionality reduction is used to select best setting for each feature extraction function and therefore reduce dimensionality from 56 (functions with all settings) to 10 (best settings only). All features are visible in table 5.3.

This is done by implemented class `FeatureSet` which serves as array for functions with settings. For each function, this array is created containing all settings of this function. Then all `FeatureSets` are evaluated and by method `getMax()`, best performing function setting is selected as final representative of that function.

Those representatives form new final `FeatureSet`, which can be converted to feature weights vector. This vector is then passed to trainer algorithm, which tries to fine-tune its weights to achieve better performance.

4.10 Time capturer

Time capturer is used for measuring both *thread* time and *real* time as described in section 3.5.3.

In order to capture time, class `StopWatch` was implemented. It has two methods `start()` and `stop()`, which returns elapsed time between calls of `start()` and `stop()`.

Real time is captured by `StopWatch` class. Hierarchical thread time is implemented by `TimeObject` class, which holds tree-like structure, similar to file system. Each component of system has its `StopWatch` clock and after finishing its job, it submits its time with its component categorization path (for example `/Training/Genetic/Generation15Sample88/`). `TimeObject` creates this path in tree-like structure and assigns elapsed time to leaf node (`Generation15Sample88`). After all computations are done, `printTime()` method is called, which evokes recursive summarization of all leaf nodes to their parents and so on. Resulting fully computed tree is then saved into file `time.out`. Its part is visible in snippet 3.1.

4.11 Chart plotter

Chart plotter was used for plotting 2 types of charts: learning curves and times as described in section 3.5.4.

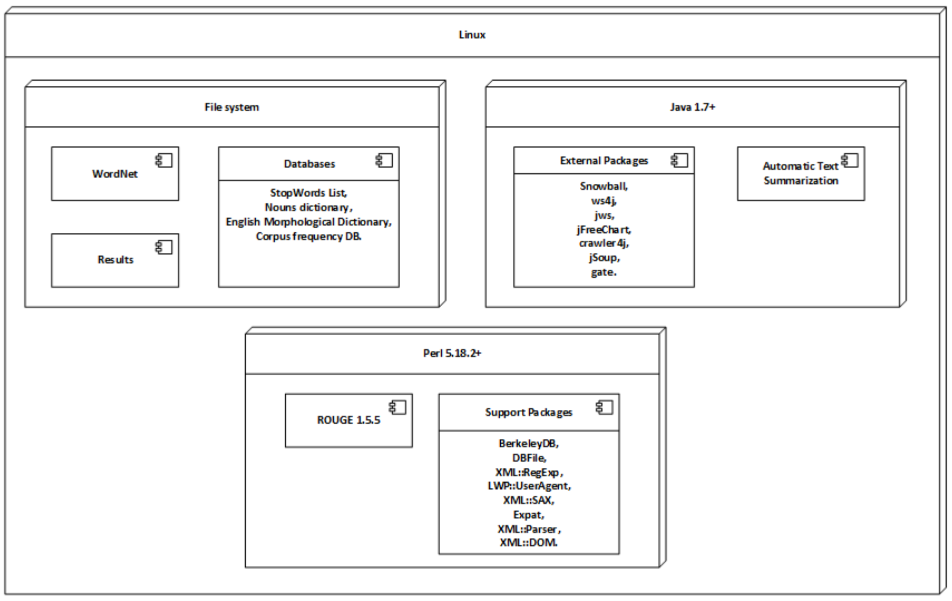
Chart logic is used from Java library `jFreeChart`. In order to plot data, they had to be preprocessed in way to match `jFreeChart` data structures. All data are sent to function in `HashMap`, where keys was also labels to be printed and values was used as amounts in the chart.

Resulting chart is then saved as `png` image.

4.12 Deployment

In deployment diagram in figure 4.1 is visible, that proposed system runs on Linux platform and requires file system for loading required databases and storing result, Java machine to run entire system and some external packages and perl for running external evaluation tool *ROUGE* and packages it requires.

Figure 4.1: Deployment diagram of proposed system



Chapter 5

Testing

This chapter is focused on testing important parts of summarization system in order to make it more efficient, quicker to compute and accurate.

First four tests are designed to speed up summarization system by removing unnecessary steps with further evaluation, if system performance will not be affected. Next test focuses on capabilities of parallel processing and its difference to linear processing. Then, *result cacher* is tested whether it speeds up summarization. Following tests are focused on genetic algorithms performance and overall system performance with comparison to other summary system. Last section show 6 examples of BBC article summarization.

5.1 Testing of external evaluation performance

Since training of feature weights was very slow, system was profiled in detail. On prepared corpus with already extracted features it was discovered, that the longest portion of time was spent in external evaluation component. Following table 5.1 shows simplified results of profiling.

Table 5.1: System profiling of training related component[9]

Component	Elapsed Time
External evaluation	88.6 %
Thresholding of sentences	11.3 %
Generation of required files for ROUGE	0.07 %
Generation of summary	0.03 %

Since external evaluation took the longest time, it was further tested whether it is required to perform all available *ROUGE* metrics, thus what is their mutual dependence or correlation.

[9]Times of initialization and non-related components were neglected.

5.1.1 Test methodology

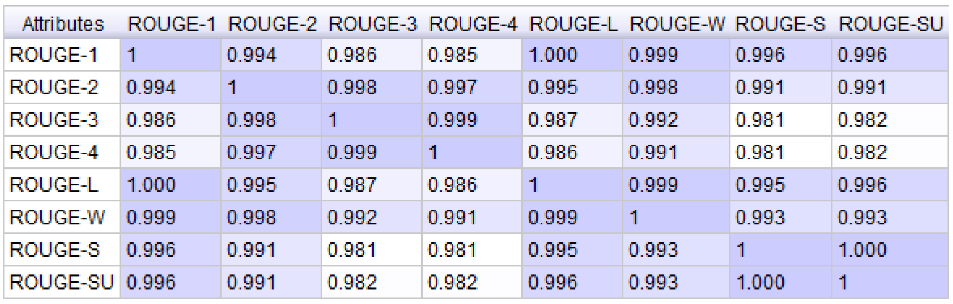
Test was performed by collecting all results of external evaluation from all previous test and those results were thereafter analysed in *RapidMinder*, [31], by calculation of correlation matrix.

5.1.2 Results

From matrix in figure 5.1 is apparent that all metrics are strongly dependent on each other - lowest correlation coefficient is 98.1%. Therefore it seems unnecessary to measure all but some arbitrary subset. If only one measure was selected, computational time of training related components could drop by 77.5%. Which means that training on set of 300 documents with approximately 150 iterations would take 17 minutes instead of 75 minutes.

Testing whether this reduction didn't affect performance and really reduced computation time is described in section 5.4.

Figure 5.1: Correlation matrix of mutual dependence in *ROUGE* metrics



Attributes	ROUGE-1	ROUGE-2	ROUGE-3	ROUGE-4	ROUGE-L	ROUGE-W	ROUGE-S	ROUGE-SU
ROUGE-1	1	0.994	0.986	0.985	1.000	0.999	0.996	0.996
ROUGE-2	0.994	1	0.998	0.997	0.995	0.998	0.991	0.991
ROUGE-3	0.986	0.998	1	0.999	0.987	0.992	0.981	0.982
ROUGE-4	0.985	0.997	0.999	1	0.986	0.991	0.981	0.982
ROUGE-L	1.000	0.995	0.987	0.986	1	0.999	0.995	0.996
ROUGE-W	0.999	0.998	0.992	0.991	0.999	1	0.993	0.993
ROUGE-S	0.996	0.991	0.981	0.981	0.995	0.993	1	1.000
ROUGE-SU	0.996	0.991	0.982	0.982	0.996	0.993	1.000	1

5.2 Testing of feature extraction performance

Massive feature extraction took very long time to complete, thus related components were profiled. It was discovered that the most portion of time took *word distance* measures. Those measures take long time to compute because they constantly search through tree-like structure of word relations containing 120,000 nodes and 210,000 edges. Simplified results of profiling are in table 5.2.

Profiling in table 5.2 shows, that *word distance* measures take significant amount of time in comparison to other measures. It is currently computed in 12 settings (3 preprocessing options $\times$ 3 distance measures $\times$ 2 libraries). Most ideal solutions seems to be reduction of computed settings. Selecting only one library would reduce feature extraction by 49.5% and massive feature extraction of 450 documents would then took 4.5 hours instead of 9.

Table 5.2: System profiling of feature extraction related components[10]

Measure	Elapsed Time
Word distance	99.09 %
Bigram frequency	0.545 %
Cosine distance	0.334 %
Same tokens ratio	0.021 %
TF-IDF	0.004 %
Uppercase	0.003 %
Consistency	0.002 %
Numeric	0.00197 %
Title	0.001 %
Location	0.00001 %

Testing whether this reduction didn't affect performance and really reduced computation time is described in section 5.4.

5.3 Testing of feature extraction importance

From previous testing is section 5.2 was discovered, that it would be beneficial to remove one library of *word distance* measures. In order to remove least performing one, testing of feature importance was designed.

5.3.1 Test methodology

16 independent test runs were made on corpus subset composed of 50% of all available documents (different subset each test run). In every run, each setting of each feature extracting measure was independently tested (its weight was set to one and weight of all other measures was set to zero) and the result of evaluation was recorded. Those results were then averaged.

5.3.2 Results

Table 5.3 shows importance of each measure and its setting. The higher the measure is in table the better performance it has. This result shows for analysed corpus from DUC, that best performing measures are *TF-IDF* and *Uppercase* and among the worst performing measures are *Consistency* and *word distance*, where measures from library *WS4J* are on average better than from *JWS*.

[10]Times of initialization and non-related components were neglected.

Table 5.3 also contains item *All-With-Same-Weights*, which represents all measures together in unweighted combination (all measures are taken as equally performing ie. all have feature weight set to one).

Table 5.3: Performance of each measure and its setting[11]

ID	Measure-Setting	ROUGE-1	ROUGE-2	ROUGE-L
0	Uppercase-Identity	0.44145	0.22621	0.41175
22	TF-IDF-Stemmer-StopWords	0.43955	0.22512	0.41002
19	TF-IDF-StopWords	0.43955	0.22512	0.41002
20	TF-IDF-Nouns	0.43943	0.22497	0.40988
23	TF-IDF-Stemmer-Nouns	0.43943	0.22497	0.40988
21	TF-IDF-Stemmer-Identity	0.43929	0.22497	0.40975
18	TF-IDF-Identity	0.43929	0.22497	0.40975
24	Location	0.43929	0.22497	0.40975
3	Uppercase-Stemmer-Identity	0.43902	0.22477	0.40952
5	Uppercase-Stemmer-Nouns	0.43786	0.22354	0.40832
2	Uppercase-Nouns	0.43786	0.22354	0.40832
1	Uppercase-StopWords	0.43786	0.22354	0.40832
4	Uppercase-Stemmer-StopWords	0.43786	0.22354	0.40832
32	Bigrams-StopWords	0.42667	0.20892	0.39752
35	Bigrams-Stemmer-StopWords	0.42525	0.20759	0.39624
33	Bigrams-Nouns	0.42331	0.20638	0.39518
36	Bigrams-Stemmer-Nouns	0.42247	0.20521	0.39371
15	Title-Stemmer-Identity	0.41984	0.21007	0.38987
–	All-With-Same-Weights	0.41857	0.19844	0.38803
16	Title-Stemmer-StopWords	0.41711	0.20700	0.38663
12	Title-Identity	0.41688	0.20767	0.38676
14	Title-Nouns	0.41445	0.20444	0.38409
17	Title-Stemmer-Nouns	0.41382	0.20411	0.38353
13	Title-StopWords	0.41204	0.20406	0.38204
29	Same-Tokens-Stemmer-StopWords	0.39932	0.18231	0.37238
10	Numeric-Stemmer-StopWords	0.39841	0.19387	0.37137
9	Numeric-Stemmer-Identity	0.39841	0.19387	0.37137
6	Numeric-Identity	0.39841	0.19387	0.37137
7	Numeric-StopWords	0.39841	0.19387	0.37137
26	Same-Tokens-StopWords	0.39808	0.18137	0.36984
8	Numeric-Nouns	0.39800	0.19354	0.37100
11	Numeric-Stemmer-Nouns	0.39800	0.19354	0.37100
27	Same-Tokens-Nouns	0.39702	0.18151	0.36952
42	Cosine-Stemmer-StopWords	0.39678	0.18243	0.36899

[11]Due to high correlation of ROUGE metrics only 3 are displayed to maintain clarity. Results show *F-score*.

43	Cosine-Stemmer-Nouns	0.39593	0.18118	0.36804
40	Cosine-Nouns	0.39539	0.18146	0.36748
30	Same-Tokens-Stemmer-Nouns	0.39456	0.17995	0.36731
48	Word-Dst-WS4J-Path-Nouns	0.39244	0.18345	0.36382
39	Cosine-StopWords	0.39234	0.17987	0.36431
41	Cosine-Stemmer-Identity	0.38992	0.17850	0.36220
47	Word-Dst-WS4J-Lin-Nouns	0.38620	0.17801	0.35957
54	Word-Dst-JWS-Path-Nouns	0.38460	0.17954	0.35688
49	Word-Dst-WS4J-WuP-Nouns	0.38229	0.18007	0.35431
38	Cosine-Identity	0.38094	0.17164	0.35328
45	Word-Dst-WS4J-Path-StopWords	0.37841	0.17252	0.34953
44	Word-Dst-WS4J-Lin-StopWords	0.37636	0.16873	0.34856
55	Word-Dst-JWS-WuP-Nouns	0.37418	0.17462	0.34719
51	Word-Dst-JWS-Path-StopWords	0.37091	0.16861	0.34381
53	Word-Dst-JWS-Lin-Nouns	0.36437	0.16550	0.33938
46	Word-Dst-WS4J-Wup-StopWords	0.36372	0.16391	0.33582
52	Word-Dst-JWS-WuP-StopWords	0.36033	0.16229	0.33318
50	Word-Dst-JWS-Lin-StopWords	0.35436	0.15587	0.32855
28	Same-Tokens-Stemmer-Identity	0.34930	0.14653	0.32225
25	Same-Tokens-Identity	0.34526	0.14406	0.31852
31	Bigrams-Identity	0.33078	0.13859	0.30525
34	Bigrams-Stemmer-Identity	0.33070	0.13817	0.30494
37	Consistency	0.30096	0.11577	0.28283

5.4 Testing impact of proposed changes

This section is focused on testing proposed speed ups in section 5.1 and 5.2 and on their impact on performance to evaluations and summary generations.

In order to maximize clarity, following abbreviations were used:

- **R+** All *ROUGE* metrics calculated (8 in total)
- **R-** One *ROUGE* metric calculated (ROUGE-1)
- **WD+** All *world distance* functions used (12 in total)
- **WD-** Half of *world distance* functions used (6 in total) - package WS4J

- **Unweighted** All features have the same weights during evaluation
- **Weighted** Weights of features are learnt by trainer algorithm
- **Key Functions** Feature weights are set only to features with settings selected in dimensionality reduction as best candidates.

- **Performance** *F-score* of ROUGE-1

5.4.1 Test methodology

First test was designed in 10 runs over random 10% documents from corpus. In each run 70% of documents were used in training and 30% in testing (evaluating) and each run was composed of 4 runs of automatic text summarization with following settings:

- R+, WD+
- R+, WD-
- R-, WD+
- R-, WD-

This test was used for determining time and performance of learning.

Second test was designed in two runs each doing massive feature extraction of all 446 documents. Runs differed in extraction of WD+ and WD- features and both runs extracted all other features. This test was used for determining time of massive extraction.

5.4.2 Results

5.4.2.1 Time

5.4.2.1.1 Massive feature extraction From time perspective, massive feature extraction is not dependent on ROUGE metrics which are used only for evaluation.

By reducing *word distance* measures from 12 to 6, time required for extraction was by 31% lower, but expected reduction of required time was about 50%. Feature extraction of one document dropped from 75 seconds to 50 seconds.

Removed package from processing was *JWS* due to its worse performance than *WS4J*.

Measured results are visible in table 5.4.

Table 5.4: Duration of massive feature extraction of 446 documents

	Elapsed Time	Faster or slower by
WD+	33148.797 s	-45.03 %
WD-	22856.908 s	31.05 %

5.4.2.1.2 Learning In this case, time of computation is dependent on both ROUGE metrics and *word distance* measures. Given the fact, that features are already extracted, *word distance* measures slowdown learning process only by small portion during their matrix normalization. This operation is the same for all similarity measures, so reduced time is proportional to number of similarity measures plus some overhead. Duration of calculation was approximately by 2.5% faster for reduced feature set (`WD-`).

Process of learning is strictly dependent on external evaluation. In section 5.1 was measured, that external evaluation takes about 88% of every training step. It was proposed that by reducing all evaluation metrics from 8 to 1, time required to take one learning step will be by 77% shorter. However this time was only by 54% shorter due to some overhead in loading dictionaries and initialization of ROUGE algorithm in every step.

In total, each learning step was by 56.5% faster and detailed results are shown in tables 5.5 and 5.6.

Table 5.5: Duration of trainer learning on 10% of corpus

Dataset ID	[R+][WD+]	[R+][WD-]	[R-][WD+]	[R-][WD-]
1	1765.587 s	1714.168 s	847.299 s	817.823 s
2	1782.125 s	1718.867 s	960.622 s	932.97 s
3	2194.535 s	2129.261 s	841.702 s	808.783 s
4	2139.015 s	2072.535 s	922.488 s	890.538 s
5	2018.541 s	1966.006 s	874.565 s	834.371 s
6	2227.371 s	2130.818 s	863.241 s	806.767 s
7	2232.855 s	2193.807 s	1053.901 s	1023.074 s
8	2259.036 s	2195.994 s	869.4 s	840.009 s
9	2107.932 s	2060.649 s	1002.113 s	979.328 s
10	1736.364 s	1678.874 s	936.813 s	898.439 s

Table 5.6: Relative speed up of learning in different configurations

Dataset ID	[R+][WD+]	[R+][WD-]	[R-][WD+]	[R-][WD-]
1	100 %	102.91 %	152.01 %	153.68 %
2	100 %	103.55 %	146.10 %	147.65 %
3	100 %	102.97 %	161.65 %	163.15 %
4	100 %	103.11 %	156.87 %	158.37 %
5	100 %	102.60 %	156.67 %	158.66 %
6	100 %	104.33 %	161.24 %	163.78 %
7	100 %	101.75 %	152.80 %	154.18 %
8	100 %	102.79 %	161.51 %	162.82 %
9	100 %	102.24 %	152.46 %	153.54 %
10	100 %	103.31 %	146.05 %	148.26 %
Average	100 %	102.96 %	154.74 %	156.41 %

5.4.2.2 Performance

5.4.2.2.1 Word distance measures From results is obvious, that removed *word distance* measures always performed so poorly that they weren't selected as *Key Functions* or didn't obtained high weight from learning. This resulted in fact, that their removal is independent to performance.

5.4.2.2.2 Rouge Results of performance confirmed that all ROUGE measures are greatly correlated so whether the score/fitness of weights is set as average of all ROUGE measures or only ROUGE-1, the search of feature weight space should be almost the same.

Standard deviation of performance of learning with evaluation solely by ROUGE-1 is less than 2% and detailed results are shown in tables 5.7 and 5.8.

5.4.3 Discussion

Both approaches yield significant reduction of process time required to learning and processing data. Given the fact that reduction of *word distance* measures don't effect performance, it is highly recommended to remove them permanently from system.

In comparison of reduce time by 54% and standard deviation of performance by less than 2% both ways, the reduction of process time outweighs this performance loss. It is highly recommended to reduce ROUGE measures to solely ROUGE-1.

Both of those propositions were set as default settings in summarization system.

5.5 Testing of parallel processing

This section is focused on testing speed up of parallel implementation of genetic algorithms and trainer algorithm.

5.5.1 Test methodology

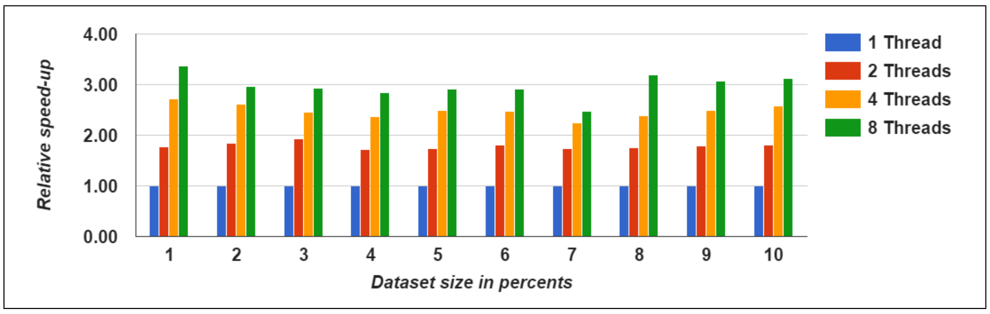
There were two tests, one testing genetic algorithms and second trainer algorithm. Both tests were run 10 times with increasing dataset size from 1% size of all available documents to 10% increasing by one percent. In each of those runs genetic algorithms and trainer algorithm were tested with one, two, four and eight threads on same dataset. In total, there were 80 test results.

5.5.2 Results

5.5.2.1 Genetic algorithms

From measured results (see table 5.9) was calculated relative speed up to time of calculation by one thread (see table 5.10). In chart displaying those data (see figure 5.2) is noticeable, that using threads is beneficial and doubling number of threads increases speed of execution almost linearly with slowly decreasing speed. Due to a fact that everything that takes time in this algorithm is parallelized, this algorithm should be twice as fast with doubling thread count. Therefore, there is big overhead in switching threads. To decrease synchronization, all data linked to evaluation are created locally in each thread.

Figure 5.2: Chart of relative speed up of genetic algorithms



5.5.2.2 Trainer algorithm

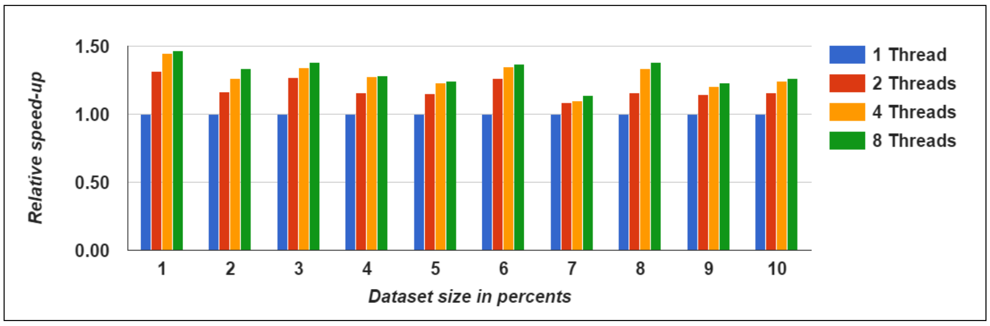
Similarly to genetic algorithms, calculated speed ups of trainer algorithm (see table 5.11) from measured data (see table 5.12) implies that using threads is beneficial, but only in small amount (see figure 5.3). Main part of training algorithm is sequential and therefore cannot be computed in parallel. Only part that can be computed in parallel is dimensionality reduction that has fixed number of computation to make (56 exactly) and takes about 1/5 of computational time in average.

5.5.3 Discussion

Genetic algorithms are designed to run in parallel and are practically unusable on one thread, therefore it is practical to use this algorithm on multicore system.

Trainer algorithm has only small portion of code that can run in parallel and therefore it is not necessary to use this algorithm on multicore system.

Figure 5.3: Chart of relative speed up of trainer algorithm



5.6 Testing of result cacher component

This section is focused on testing speed up of system by caching output of ROUGE measure for given summary so they don't have to be computed more than once.

5.6.1 Test methodology

5.6.1.1 Trainer algorithm

There were two tests, first testing linear (1 thread) and second parallel execution (6 threads) of trainer algorithm. Each test consisted of 5 runs of trainer algorithm without caching and 5 runs of trainer algorithm with caching. All runs was made on same dataset composed of 10% of all available documents. In total 20 runs were made.

Before every test, cache was cleared so first run in both tests had to start with computing everything anew. Following runs already had everything precalculated.

5.6.1.2 Genetic algorithms

There was one test consisting of 5 runs of genetic algorithms without caching and 5 runs of genetic algorithms with caching, tested on same dataset as trainer algorithm and with same policy.

5.6.2 Results

5.6.2.1 Trainer algorithm

As seen from both figures 5.4 and 5.5, first run with caching is always slower because cache does not contain any results.

From figure 5.4 and 5.5 and table 5.13 and 5.14 is visible, that computation is almost 4 times faster with cache than without cache even if no results were previously stored into cache and if cache contains all results, then computation is 6 times faster.

Figure 5.4: Chart of computational time of trainer algorithm using one thread

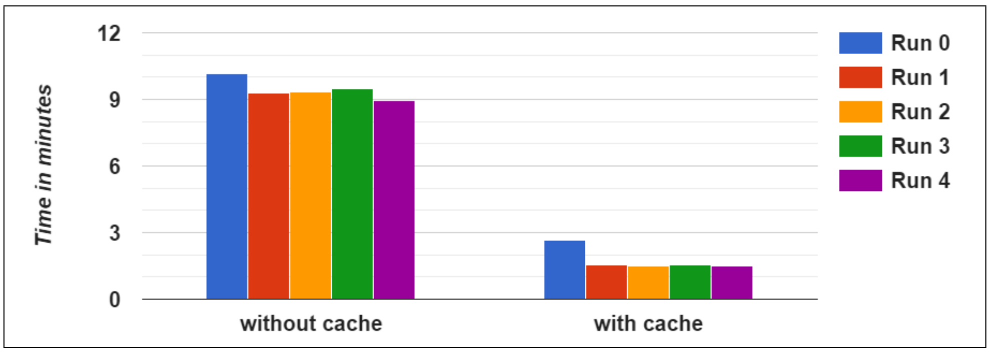
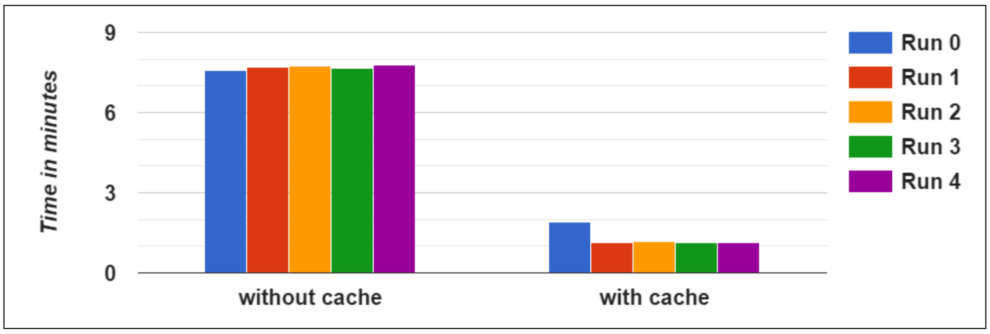


Figure 5.5: Chart of computational time of trainer algorithm using six threads



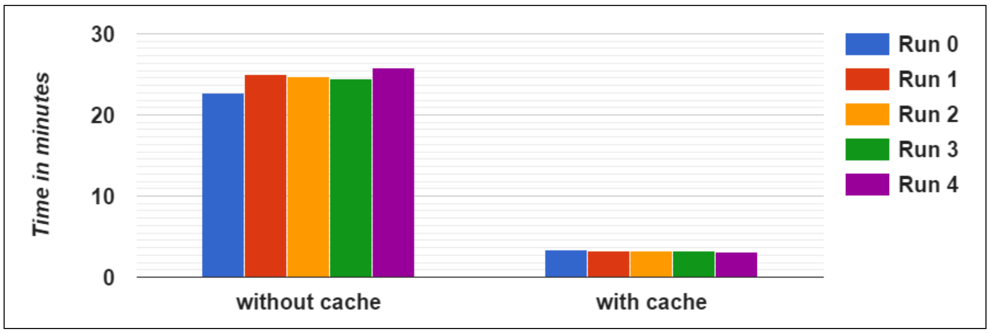
5.6.2.2 Genetic algorithms

From figure 5.6 and table 5.15 is visible that given its randomness, genetic algorithms don't use much of precalculated results from previous runs. Even though, caching makes genetic algorithms run on average 7.5 times faster than without it.

5.6.3 Discussion

From all measured results is visible that using cache significantly decreases time of execution. The most positive observation is, that significant reduction of computational time occurs even if no previous results are stored in cache which speeds up whole system and not just cases that are being recomputed again.

Figure 5.6: Chart of computational time of parallel run of genetic algorithms



5.7 Testing of genetic algorithms parameters

This section is focused on testing time consumption and performance of genetic algorithms based on different parameter settings.

5.7.1 Test methodology

This test was composed of 5 runs over randomly picked datasets. Each run has dataset consisting of 25% randomly selected documents from all available documents.

Tested parameters were: number of generations G_{GA} (see section 3.4.7.2), population size P_{GA} (see section 2.0.2), weight upper boundary W_{GA} (see section 3.4.7.2), elitism size E_{GA} (see section 2.0.7), mutation probability M_{GA} (see section 2.0.6), selection size S_{GA} (see section 2.0.4) and tourney size T_{GA} (see section 2.0.4.4).

Those parameters were tested one by one in each test run to following values:

- G_{GA} = 16, 32, 64, 128
- P_{GA} = 16, 32, 64, 128
- W_{GA} = 16, 128, 512
- E_{GA} = 2, 4, 8, 16
- M_{GA} = 0.1%, 2%, 10%, 50%
- S_{GA} = 10%, 30%, 50%, 70%, 90% of P_{GA}
- T_{GA} = 5%, 10%, 20%, 30% of P_{GA}

Every of those parameter tests were run twice, once with roulette selection and once with tourney selection.

Default parameters in test were 32 generations, 64 individuals in population, weight upper boundary 128, 4 elite individuals, mutation probability 2%, selection size 70% and tourney size 20%.

In total, there were 260 test results measuring time and performance.

5.7.2 Results

5.7.2.1 Time

Time of computation is mainly dependent on complexity of fitness calculation, other parts of the system don't take much time to complete. Fitness is calculated for each feature weight vector only once, therefore if there are several same individuals or if some individuals are selected for next generation, their fitness don't have to be calculated again.

As shown in figure 5.7 and 5.8, time of execution increases linearly with number of individuals in system (either by more generations or bigger populations).

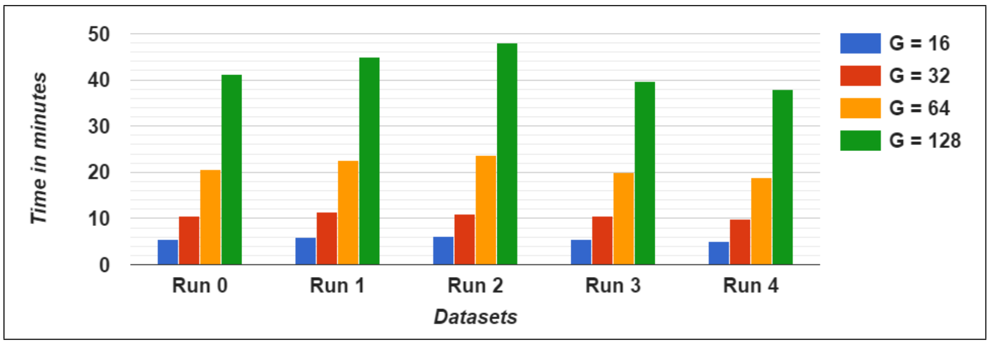
Figure 5.9 shows that with increasing number of individuals carried to next generation without any change, computation time decreases. This is caused because those individuals cannot be modified by mutation so their fitness is not calculated again.

Next figure 5.10 shows similar behaviour as previous one. It is because of the bigger the selection size is, the smaller amount of individuals are created for which fitness needs to be calculated.

Last figure 5.11 displays that with increasing probability of mutation, time of computation rises. Since for every new weight vector fitness needs to be calculated. It is logical, that increasing probability of mutation will produce more previously unseen individuals.

Type of selection (roulette or tourney), size of tourney and weight upper boundary don't have any effect on computation time. All results are in table 5.16.

Figure 5.7: Chart of time consumption based on number of generations



5.7.2.2 Performance

Results in table 5.17 and 5.18 are calculated proportionally to *F-score* of ROUGE metric achieved with no learning, considering all feature equally significant. Even though the differences between performances seem insignificant, some dependencies are visible.

Figure 5.8: Chart of time consumption based on size of population

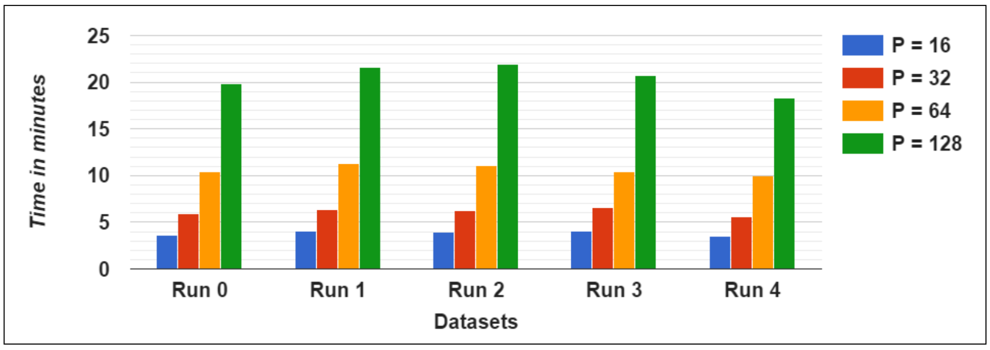


Figure 5.9: Chart of time consumption based on size of elitism

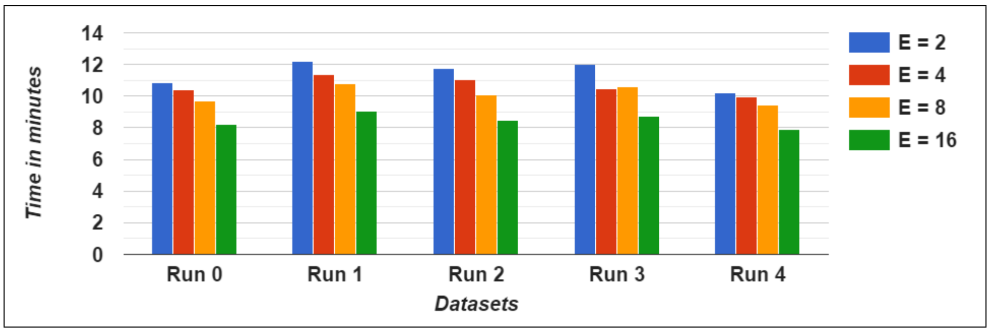
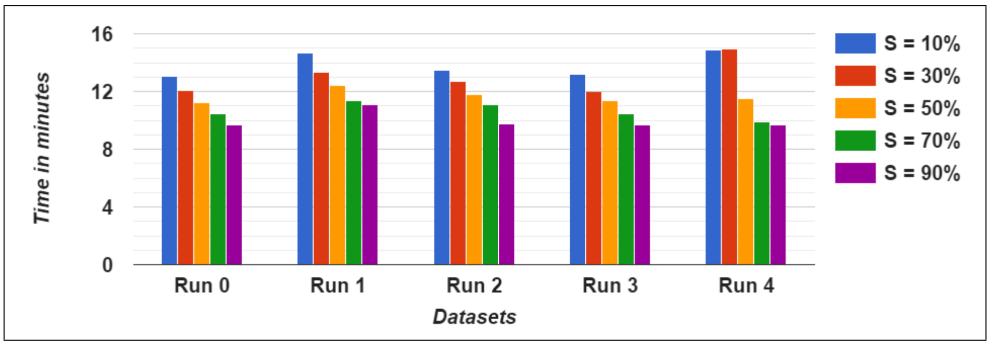


Figure 5.10: Chart of time consumption based on selection size



From figure 5.12 is visible that increasing size of population leads to better performance because vaster space can be searched so there is higher probability of finding proper weight vector.

Another figure 5.14 shows that with increasing upper weight boundary performance declines. This might be caused because of with increasing domain size, search space expands and possibility of finding proper solution decreases.

On next figure 5.15 is seen that best performance is achieved with 10% chance of mutation. Algorithm is then balancing random search and convergence to some optimum. Higher mutation destroys current population and lower mutations are prone to converge to local minimum.

Last interesting figure 5.13 displays that elite size 8 is best possible parameter settings. Size of 16 makes one quarter of population thus there is much smaller possibilities for space exploration. Smaller sizes don't carry that much information to next population to make significant difference.

Overall performance was about 2% better than without learning.

5.7.3 Discussion

As seen from all performed performance tests, results seem independent to changes in parameters. This might by caused by several reasons. Firstly, because of testing was done on laptop, it wasn't possible to test genetic algorithms on bigger dataset. Datasets were so small that they might have completely different behaviour in each test run. Also genetic algorithms are random based, to make a proper testing, every test would need to be repeated atleast 50 times which was not possible due to its time consumption.

Secondly, fitness function is discontinuous function even though its domain is continuous. Change in fitness occurs only when weights are so different, that some sentence in generated summary is exchanged with another, but because every used feature has very similar performance, changes in sentences are very small even with big changes in weights.

Weights are basically translated to binary vector representing sentences that are chosen to generated summary. This vector is completely different for each document so it might not be even possible to generalize this choosing for multiple documents. Genetic algorithms are mainly used to optimize one instance of specific problem. It might not be very usable in learning how to generalize weights for whole set of documents based on given corpus.

Recommendations for future development would be to try different crossover and mutation functions, for example convert real-valued chromosomes to binary and implement basic N-point crossover and mutation by negation.

Figure 5.11: Chart of time consumption based on mutation probability

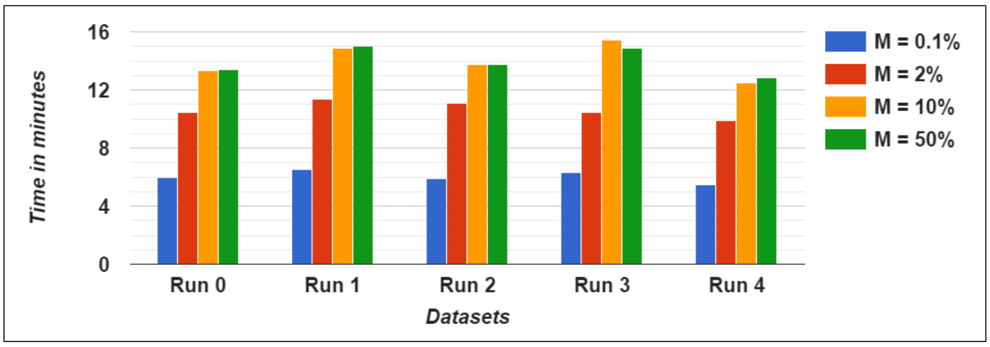


Figure 5.12: Chart of relative performance based on population size

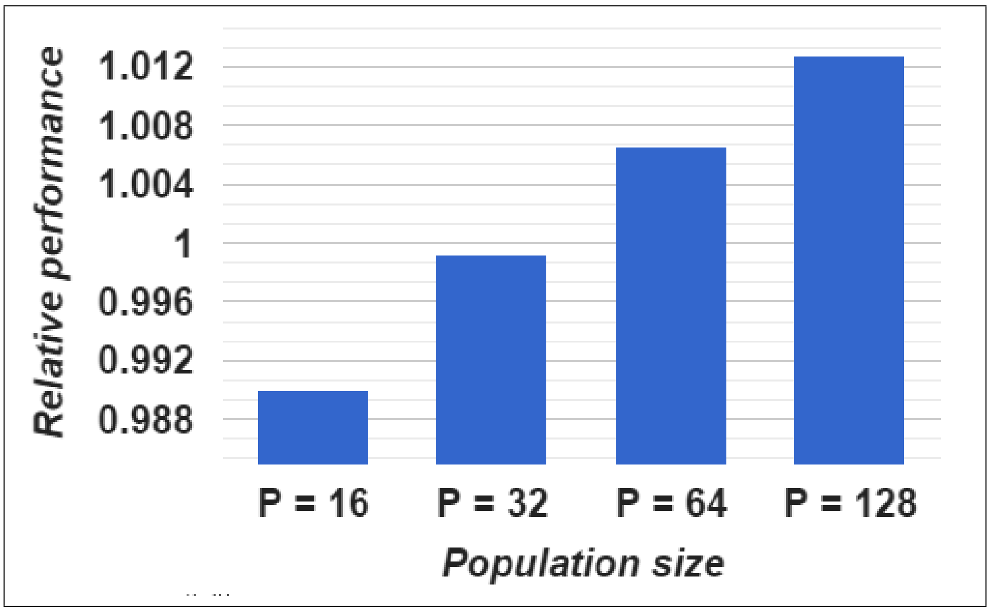


Figure 5.13: Chart of relative performance based on elitism size

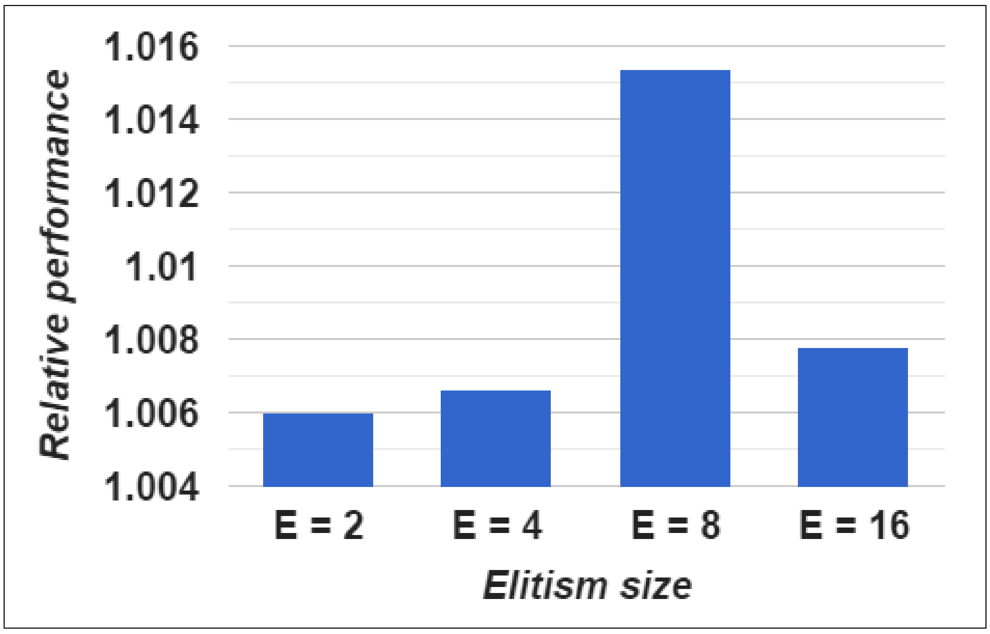


Table 5.7: Performance of *Weighted*, *Unweighted* and *Key Functions*

Weighted	[R+][WD+]	[R+][WD-]	[R-][WD+]	[R-][WD-]	Jackknifing
Dataset 1	0.39689	0.39689	0.39575	0.39575	0.42695
Dataset 2	0.37085	0.37085	0.36737	0.36737	0.36314
Dataset 3	0.39414	0.39414	0.39736	0.39736	0.39658
Dataset 4	0.36283	0.36283	0.3724	0.3724	0.39811
Dataset 5	0.41699	0.41699	0.40962	0.40962	0.43968
Dataset 6	0.38909	0.38909	0.3808	0.3808	0.39819
Dataset 7	0.42543	0.42543	0.42543	0.42543	0.45805
Dataset 8	0.38177	0.38177	0.3836	0.3836	0.42319
Dataset 9	0.38488	0.38488	0.36394	0.36394	0.42013
Dataset 10	0.39537	0.39537	0.40282	0.40282	0.45595
Key Functions	[R+][WD+]	[R+][WD-]	[R-][WD+]	[R-][WD-]	Jackknifing
Dataset 1	0.4085	0.4085	0.40681	0.40681	0.42695
Dataset 2	0.36764	0.36764	0.36553	0.36553	0.36314
Dataset 3	0.38947	0.38947	0.39139	0.39139	0.39658
Dataset 4	0.35514	0.35514	0.37085	0.37085	0.39811
Dataset 5	0.4171	0.4171	0.42106	0.42106	0.43968
Dataset 6	0.38688	0.38688	0.39454	0.39454	0.39819
Dataset 7	0.42791	0.42791	0.42286	0.42286	0.45805
Dataset 8	0.38053	0.38053	0.38256	0.38256	0.42319
Dataset 9	0.36426	0.36426	0.36388	0.36388	0.42013
Dataset 10	0.40115	0.40115	0.40115	0.40115	0.45595
Unweighted	[R+][WD+]	[R+][WD-]	[R-][WD+]	[R-][WD-]	Jackknifing
Dataset 1	0.41886	0.41886	0.41886	0.41886	0.42695
Dataset 2	0.36738	0.36738	0.36738	0.36738	0.36314
Dataset 3	0.38134	0.38134	0.38134	0.38134	0.39658
Dataset 4	0.35015	0.35015	0.35015	0.35015	0.39811
Dataset 5	0.4092	0.4092	0.4092	0.4092	0.43968
Dataset 6	0.38921	0.38921	0.38921	0.38921	0.39819
Dataset 7	0.41646	0.41646	0.41646	0.41646	0.45805
Dataset 8	0.38517	0.38517	0.38517	0.38517	0.42319
Dataset 9	0.34925	0.34925	0.34925	0.34925	0.42013
Dataset 10	0.41386	0.41386	0.41386	0.41386	0.45595

Table 5.8: Relative performance improvement to *jackkniffing*

Weighted	[R+][WD+]	[R+][WD-]	[R-][WD+]	[R-][WD-]	Diff. of R+ to R-
Dataset 1	−7.04 %	−7.04 %	−7.31 %	−7.31 %	0.27 %
Dataset 2	2.12 %	2.12 %	1.16 %	1.16 %	0.96 %
Dataset 3	−0.62 %	−0.62 %	0.20 %	0.20 %	−0.81 %
Dataset 4	−8.86 %	−8.86 %	−6.46 %	−6.46 %	−2.40 %
Dataset 5	−5.16 %	−5.16 %	−6.84 %	−6.84 %	1.68 %
Dataset 6	−2.29 %	−2.29 %	−4.37 %	−4.37 %	2.08 %
Dataset 7	−7.12 %	−7.12 %	−7.12 %	−7.12 %	0.00 %
Dataset 8	−9.79 %	−9.79 %	−9.36 %	−9.36 %	−0.43 %
Dataset 9	−8.39 %	−8.39 %	−13.37 %	−13.37 %	4.98 %
Dataset 10	−13.29 %	−13.29 %	−11.65 %	−11.65 %	−1.63 %
Key Functions	[R+][WD+]	[R+][WD-]	[R-][WD+]	[R-][WD-]	Diff. of R+ to R-
Dataset 1	−4.32 %	−4.32 %	−4.72 %	−4.72 %	0.40 %
Dataset 2	1.24 %	1.24 %	0.66 %	0.66 %	0.58 %
Dataset 3	−1.79 %	−1.79 %	−1.31 %	−1.31 %	−0.48 %
Dataset 4	−10.79 %	−10.79 %	−6.85 %	−6.85 %	−3.95 %
Dataset 5	−5.14 %	−5.14 %	−4.23 %	−4.23 %	−0.90 %
Dataset 6	−2.84 %	−2.84 %	−0.92 %	−0.92 %	−1.92 %
Dataset 7	−6.58 %	−6.58 %	−7.68 %	−7.68 %	1.10 %
Dataset 8	−10.08 %	−10.08 %	−9.60 %	−9.60 %	−0.48 %
Dataset 9	−13.30 %	−13.30 %	−13.39 %	−13.39 %	0.09 %
Dataset 10	−12.02 %	−12.02 %	−12.02 %	−12.02 %	0.00 %
Unweighted	[R+][WD+]	[R+][WD-]	[R-][WD+]	[R-][WD-]	Diff. of R+ to R-
Dataset 1	−1.89 %	−1.89 %	−1.89 %	−1.89 %	0.00 %
Dataset 2	1.17 %	1.17 %	1.17 %	1.17 %	0.00 %
Dataset 3	−3.84 %	−3.84 %	−3.84 %	−3.84 %	0.00 %
Dataset 4	−12.05 %	−12.05 %	−12.05 %	−12.05 %	0.00 %
Dataset 5	−6.93 %	−6.93 %	−6.93 %	−6.93 %	0.00 %
Dataset 6	−2.26 %	−2.26 %	−2.26 %	−2.26 %	0.00 %
Dataset 7	−9.08 %	−9.08 %	−9.08 %	−9.08 %	0.00 %
Dataset 8	−8.98 %	−8.98 %	−8.98 %	−8.98 %	0.00 %
Dataset 9	−16.87 %	−16.87 %	−16.87 %	−16.87 %	0.00 %
Dataset 10	−9.23 %	−9.23 %	−9.23 %	−9.23 %	0.00 %
Average	−6.53 %	−6.53 %	−6.50 %	−6.50 %	−0.03 %
Average StDev*	–	–	–	–	1.84 %

Difference of WD+ to WD- is always zero.

* Calculated without Unweighted set.

Table 5.9: Time of execution of genetic algorithms

	1% of DS	2% of DS	3% of DS	4% of DS	5% of DS
1 Thread	102986 *ms*	176311 *ms*	371856 *ms*	428149 *ms*	489082 *ms*
2 Threads	58391 *ms*	95587 *ms*	192135 *ms*	250241 *ms*	282415 *ms*
4 Threads	37764 *ms*	67608 *ms*	151518 *ms*	181392 *ms*	196054 *ms*
8 Threads	30596 *ms*	59471 *ms*	126935 *ms*	151047 *ms*	167998 *ms*
	6% of DS	7% of DS	8% of DS	9% of DS	10% of DS
1 Thread	669833 *ms*	873893 *ms*	779662 *ms*	781909 *ms*	895019 *ms*
2 Threads	370348 *ms*	502879 *ms*	444733 *ms*	435066 *ms*	493134 *ms*
4 Threads	270042 *ms*	389292 *ms*	327080 *ms*	313830 *ms*	347237 *ms*
8 Threads	230608 *ms*	353550 *ms*	244568 *ms*	254129 *ms*	286118 *ms*

Table 5.10: Relative speed up of execution of genetic algorithms

	1% of DS	2% of DS	3% of DS	4% of DS	5% of DS
1 Thread	100 %	100 %	100 %	100 %	100 %
2 Threads	176 %	184 %	194 %	171 %	173 %
4 Threads	273 %	261 %	245 %	236 %	249 %
8 Threads	337 %	296 %	293 %	283 %	291 %
	6% of DS	7% of DS	8% of DS	9% of DS	10% of DS
1 Thread	100 %	100 %	100 %	100 %	100 %
2 Threads	181 %	174 %	175 %	180 %	181 %
4 Threads	248 %	224 %	238 %	249 %	258 %
8 Threads	290 %	247 %	319 %	308 %	313 %

Table 5.11: Time of execution of trainer algorithm

	1% of DS	2% of DS	3% of DS	4% of DS	5% of DS
1 Thread	64625 *ms*	98430 *ms*	237756 *ms*	282145 *ms*	294508 *ms*
2 Threads	48998 *ms*	84754 *ms*	187590 *ms*	243314 *ms*	256220 *ms*
4 Threads	44583 *ms*	77742 *ms*	177195 *ms*	221038 *ms*	239465 *ms*
8 Threads	43957 *ms*	73775 *ms*	171996 *ms*	220423 *ms*	237102 *ms*
	6% of DS	7% of DS	8% of DS	9% of DS	10% of DS
1 Thread	455677 *ms*	808552 *ms*	466081 *ms*	508528 *ms*	531627 *ms*
2 Threads	359832 *ms*	746085 *ms*	402154 *ms*	443069 *ms*	458930 *ms*
4 Threads	338330 *ms*	736325 *ms*	349146 *ms*	421417 *ms*	427333 *ms*
8 Threads	332267 *ms*	711244 *ms*	336630 *ms*	413659 *ms*	420682 *ms*

Table 5.12: Relative time of execution of trainer algorithm

	1% of DS	2% of DS	3% of DS	4% of DS	5% of DS
1 Thread	100 %	100 %	100 %	100 %	100 %
2 Threads	132 %	116 %	127 %	116 %	115 %
4 Threads	145 %	127 %	134 %	128 %	123 %
8 Threads	147 %	133 %	138 %	128 %	124 %
	6% of DS	7% of DS	8% of DS	9% of DS	10% of DS
1 Thread	100 %	100 %	100 %	100 %	100 %
2 Threads	127 %	108 %	116 %	115 %	116 %
4 Threads	135 %	110 %	133 %	121 %	124 %
8 Threads	137 %	114 %	138 %	123 %	126 %

Table 5.13: Time and speed up of linear run of trainer algorithm

	without cache	with cache	speed up
Run 0	10.15 *min*	2.65 *min*	383.47 %
Run 1	9.27 *min*	1.56 *min*	595.58 %
Run 2	9.33 *min*	1.52 *min*	612.20 %
Run 3	9.50 *min*	1.56 *min*	609.47 %
Run 4	8.95 *min*	1.52 *min*	588.93 %

Table 5.14: Time and speed up of parallel run of trainer algorithm

	without cache	with cache	speed up
Run 0	7.58 *min*	1.90 *min*	398.90 %
Run 1	7.71 *min*	1.14 *min*	677.84 %
Run 2	7.75 *min*	1.18 *min*	654.91 %
Run 3	7.68 *min*	1.14 *min*	672.00 %
Run 4	7.80 *min*	1.15 *min*	676.77 %

Table 5.15: Time and speed up of parallel run of genetic algorithms

	without cache	with cache	speed up
Run 0	22.74 *min*	3.39 *min*	670.88 %
Run 1	25.04 *min*	3.29 *min*	761.97 %
Run 2	24.66 *min*	3.25 *min*	758.88 %
Run 3	24.50 *min*	3.19 *min*	768.51 %
Run 4	25.76 *min*	3.17 *min*	813.63 %

Table 5.16: Computation time of genetic algorithms with different parameters

Generations	Run 0	Run 1	Run 2	Run 3	Run 4
G = 16	5.4274 *min*	5.9792 *min*	6.0467 *min*	5.4093 *min*	5.0071 *min*
G = 32	10.4346 *min*	11.3471 *min*	11.0562 *min*	10.4570 *min*	9.9256 *min*
G = 64	20.6738 *min*	22.6809 *min*	23.7064 *min*	20.0521 *min*	18.9225 *min*
G = 128	41.2559 *min*	44.9392 *min*	48.0655 *min*	39.786 *min*	37.8291 *min*
Population	Run 0	Run 1	Run 2	Run 3	Run 4
P = 16	3.6136 *min*	4.0823 *min*	3.9712 *min*	4.0165 *min*	3.5165 *min*
P = 32	5.9388 *min*	6.3816 *min*	6.2784 *min*	6.5987 *min*	5.6335 *min*
P = 64	10.4346 *min*	11.3472 *min*	11.0562 *min*	10.457 *min*	9.9256 *min*
P = 128	19.803 *min*	21.5831 *min*	21.8811 *min*	20.7682 *min*	18.347 *min*
Weights	Run 0	Run 1	Run 2	Run 3	Run 4
W = 16	10.5915 *min*	11.5747 *min*	12.0515 *min*	11.3752 *min*	9.9227 *min*
W = 128	10.4346 *min*	11.3472 *min*	11.0562 *min*	10.457 *min*	9.9256 *min*
W = 512	10.6791 *min*	11.6565 *min*	11.7716 *min*	10.9953 *min*	9.8026 *min*
Elitism	Run 0	Run 1	Run 2	Run 3	Run 4
E = 2	10.8719 *min*	12.1955 *min*	11.728 *min*	12.0356 *min*	10.209 *min*
E = 4	10.4346 *min*	11.3472 *min*	11.0562 *min*	10.457 *min*	9.9256 *min*
E = 8	9.7102 *min*	10.8094 *min*	10.0915 *min*	10.6112 *min*	9.4668 *min*
E = 16	8.2165 *min*	9.04875 *min*	8.4966 *min*	8.7028 *min*	7.8809 *min*
Mutation	Run 0	Run 1	Run 2	Run 3	Run 4
M = 0.1%	5.9501 *min*	6.503 *min*	5.9122 *min*	6.2846 *min*	5.5003 *min*
M = 2%	10.4346 *min*	11.3472 *min*	11.0562 *min*	10.457 *min*	9.9256 *min*
M = 10%	13.367 *min*	14.8427 *min*	13.7545 *min*	15.4227 *min*	12.4909 *min*
M = 50%	13.39 *min*	15.0404 *min*	13.7681 *min*	14.9082 *min*	12.8558 *min*
Selection	Run 0	Run 1	Run 2	Run 3	Run 4
S = 10%	13.082 *min*	14.6495 *min*	13.4683 *min*	13.1734 *min*	14.8973 *min*
S = 30%	12.1 *min*	13.3498 *min*	12.6756 *min*	12.0277 *min*	14.9219 *min*
S = 50%	11.258 *min*	12.4105 *min*	11.7581 *min*	11.3648 *min*	11.5383 *min*
S = 70%	10.4346 *min*	11.3472 *min*	11.0562 *min*	10.4568 *min*	9.9256 *min*
S = 90%	9.6873 *min*	11.1156 *min*	9.7784 *min*	9.6723 *min*	9.7098 *min*
Tourney	Run 0	Run 1	Run 2	Run 3	Run 4
T = 5%	10.6758 *min*	12.5577 *min*	10.972 *min*	10.6383 *min*	10.8097 *min*
T = 10%	10.4337 *min*	12.6613 *min*	10.908 *min*	10.4918 *min*	10.4749 *min*
T = 20%	10.4808 *min*	11.5413 *min*	10.7421 *min*	10.2963 *min*	9.6532 *min*
T = 30%	10.5763 *min*	12.3514 *min*	10.7894 *min*	10.547 *min*	10.1363 *min*

Figure 5.14: Chart of relative performance based on upper weight boundary

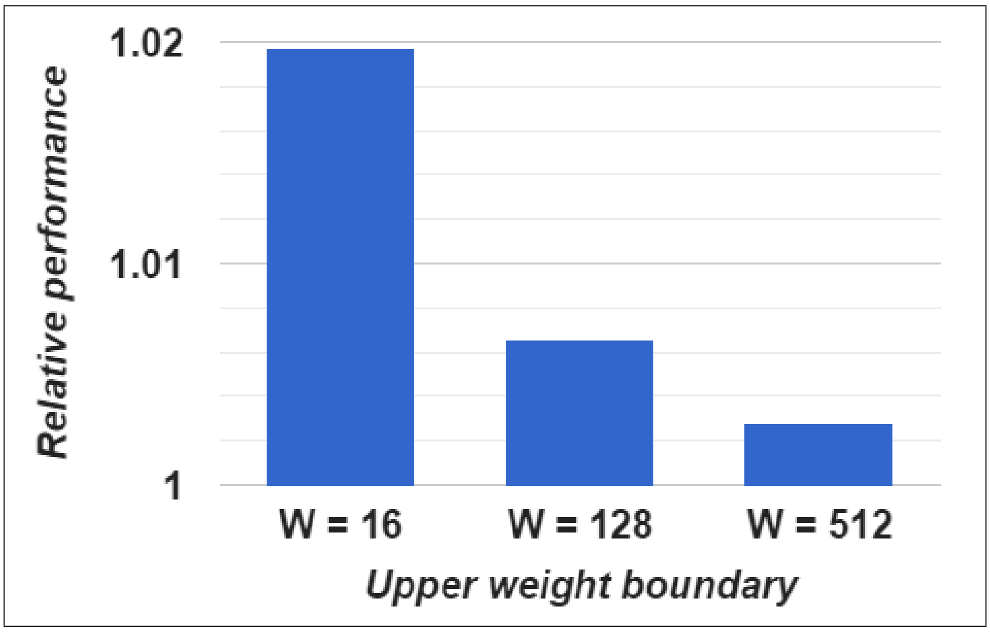


Figure 5.15: Chart of relative performance based on mutation probability

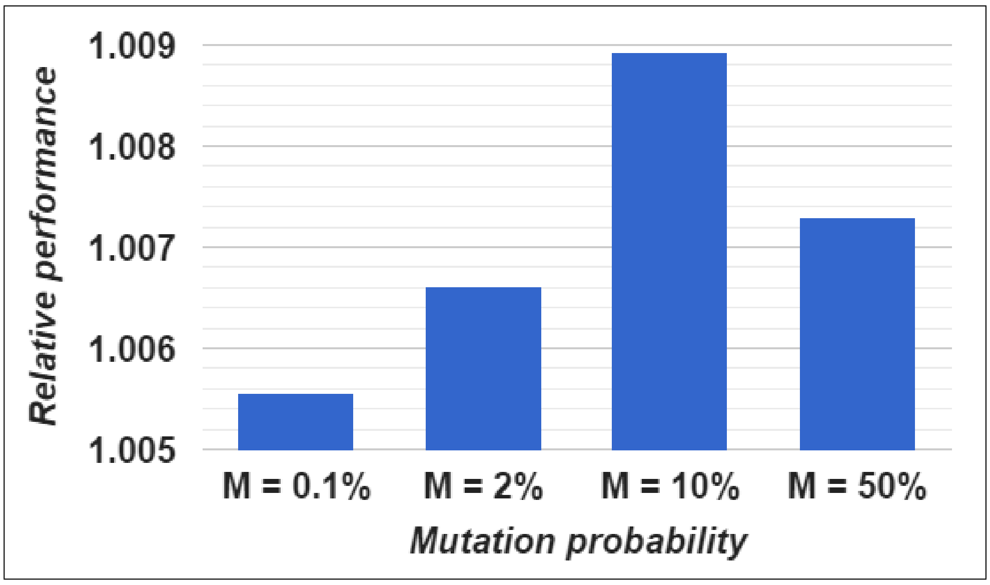


Table 5.17: Relative performance of genetic algorithms based on different parameters to *Unweighted* performance - roulette selection

Generations	Run 0	Run 1	Run 2	Run 3	Run 4	Average
G = 16	101.05 %	101.69 %	98.11 %	101.92 %	101.87 %	100.93 %
G = 32	100.06 %	101.08 %	100.21 %	100.12 %	99.76 %	100.25 %
G = 64	100.14 %	102.73 %	100.41 %	103.14 %	104.36 %	102.15 %
G = 128	102.52 %	106.93 %	101.86 %	100.03 %	102.48 %	102.76 %
Population	Run 0	Run 1	Run 2	Run 3	Run 4	Average
P = 16	99.85 %	99.99 %	97.22 %	97.20 %	99.89 %	98.83 %
P = 32	98.43 %	103.31 %	100.6 %	98.05 %	100.46 %	100.17 %
P = 64	100.06 %	101.08 %	100.21 %	100.12 %	99.76 %	100.25 %
P = 128	97.35 %	106.3 %	100.14 %	99.21 %	103.46 %	101.29 %
Weights	Run 0	Run 1	Run 2	Run 3	Run 4	Average
W = 16	102.94 %	103.51 %	101.53 %	97.99 %	99.21 %	101.04 %
W = 128	100.06 %	101.08 %	100.21 %	100.12 %	99.76 %	100.25 %
W = 512	98.92 %	97.56 %	101.06 %	100.03 %	99.82 %	99.48 %
Elitism	Run 0	Run 1	Run 2	Run 3	Run 4	Average
E = 2	99.85 %	104.38 %	99.01 %	98.9 %	104.36 %	101.3 %
E = 4	100.06 %	101.08 %	100.21 %	100.12 %	99.76 %	100.25 %
E = 8	99.59 %	102.91 %	99.71 %	100.55 %	101.13 %	100.78 %
E = 16	98.36 %	102.56 %	96.63 %	99.78 %	106.42 %	100.75 %
Mutation	Run 0	Run 1	Run 2	Run 3	Run 4	Average
M = 0.1%	101.05 %	99.75 %	100.11 %	100.72 %	96.7 %	99.67 %
M = 2%	100.06 %	101.08 %	100.21 %	100.12 %	99.76 %	100.25 %
M = 10%	101.72 %	100.79 %	99.83 %	100.03 %	105.91 %	101.66 %
M = 50%	99.44 %	99.74 %	96.97 %	100.03 %	106.42 %	100.52 %
Selection	Run 0	Run 1	Run 2	Run 3	Run 4	Average
S = 10%	99.96 %	107.96 %	97.76 %	103.83 %	100.78 %	102.06 %
S = 30%	101.05 %	101.49 %	95.01 %	101.44 %	100.06 %	99.81 %
S = 50%	99.44 %	103.91 %	96.75 %	100.25 %	99.55 %	99.98 %
S = 70%	100.06 %	101.08 %	100.21 %	100.12 %	99.76 %	100.25 %
S = 90%	98.5 %	99.78 %	101.8 %	102.8 %	99.19 %	100.41 %
					Best	102.76 %

Table 5.18: Relative performance of genetic algorithms based on different parameters to *Unweighted* performance - tourney selection

Generations	Run 0	Run 1	Run 2	Run 3	Run 4	Average
G = 16	99.02 %	106.77 %	99.26 %	100.59 %	99.14 %	100.95 %
G = 32	101.05 %	106.06 %	102.75 %	99.9 %	95.61 %	101.08 %
G = 64	102.59 %	103.43 %	98.73 %	98.9 %	102.58 %	101.25 %
G = 128	97.35 %	100.86 %	101.53 %	98.66 %	100.69 %	99.82 %
Population	Run 0	Run 1	Run 2	Run 3	Run 4	Average
P = 16	97.75 %	100.5 %	100.69 %	98.07 %	98.77 %	99.16 %
P = 32	98.93 %	102.05 %	100.41 %	98.96 %	98.05 %	99.68 %
P = 64	101.05 %	106.06 %	102.75 %	99.9 %	95.61 %	101.08 %
P = 128	99.59 %	105.06 %	100.05 %	102.92 %	98.65 %	101.25 %
Weights	Run 0	Run 1	Run 2	Run 3	Run 4	Average
W = 16	98.43 %	105.31 %	100.92 %	104.01 %	105.91 %	102.92 %
W = 128	101.05 %	106.06 %	102.75 %	99.9 %	95.61 %	101.08 %
W = 512	99.59 %	104.26 %	100.71 %	101.6 %	99.21 %	101.07 %
Elitism	Run 0	Run 1	Run 2	Run 3	Run 4	Average
E = 2	100.84 %	102.75 %	92.63 %	100.07 %	103.19 %	99.9 %
E = 4	101.05 %	106.06 %	102.75 %	99.9 %	95.61 %	101.08 %
E = 8	100.84 %	105.88 %	101.69 %	103.24 %	99.82 %	102.3 %
E = 16	101.05 %	101.16 %	100.6 %	100.25 %	100.95 %	100.8 %
Mutation	Run 0	Run 1	Run 2	Run 3	Run 4	Average
M = 0.1%	99.41 %	105.37 %	98.74 %	100.03 %	103.67 %	101.44 %
M = 2%	101.05 %	106.06 %	102.75 %	99.9 %	95.61 %	101.08 %
M = 10%	100.64 %	98.6 %	103.09 %	100.25 %	98.05 %	100.13 %
M = 50%	101.05 %	102.63 %	97.92 %	100.57 %	102.53 %	100.94 %
Selection	Run 0	Run 1	Run 2	Run 3	Run 4	Average
S = 10%	101.02 %	101.23 %	97.72 %	100.07 %	99.99 %	100.01 %
S = 30%	104.1 %	104.73 %	102.88 %	105.13 %	97.87 %	102.94 %
S = 50%	100.91 %	101.46 %	97.99 %	100.39 %	100.46 %	100.25 %
S = 70%	101.05 %	106.06 %	102.75 %	99.9 %	95.61 %	101.08 %
S = 90%	100.84 %	104.68 %	98.53 %	98.9 %	99.4 %	100.47 %
Tourney	Run 0	Run 1	Run 2	Run 3	Run 4	Average
T = 5%	99.44 %	102.5 %	92.52 %	100.03 %	100.31 %	98.96 %
T = 10%	99.59 %	104.8 %	96.95 %	101.89 %	102.8 %	101.19 %
T = 20%	101.05 %	106.06 %	102.75 %	99.9 %	95.61 %	101.08 %
T = 30%	100.84 %	105.73 %	100.41 %	100.25 %	102.11 %	101.87 %
					Best	102.94 %

5.8 Testing of overall system performance

This section is focused on performance testing of proposed system. Several system evaluation techniques will be compared by their performance including random model and external summarization system.

5.8.1 Test methodology

This test was composed of 5 runs over 5 subsets of 25% randomly selected documents from corpus.

Each run composed of several runs of evaluation of learning algorithms, including some external tools for comparison. Algorithms which were run multiple times had their resulting scores averaged into one final performance score. Single run was composed of these algorithms:

- unweighted evaluation - 1 run
- key features selection and its evaluation - 1 run
- trainer algorithm and its evaluation - 1 run
- genetic algorithms and their evaluation - 5 runs
- random sentence selection and its evaluation - 100 runs
- external summary tool *TextRank*, [32], and its evaluation - 1 run

5.8.2 Results

From result in figure 5.16 is visible, that trainer, genetic, *Key Functions* and *Unweighted* algorithms used for training and summary generation perform almost the same, however, trainer algorithm and genetic algorithms require a lot of time for learning and thus their usability rapidly decreases with such low performing results in comparison to *Key Functions* and *Unweighted* algorithms which are computed much quickly.

As upper bound was used jackkniffing (see section 3.4.9). From its definition and calculation, it is highly unlikely that any summarization tool will achieve higher *F-score* when comparing to the same user-written summaries and from this perspective, performance of all implemented algorithms is very satisfactory.

System was also tested against other summarization tool, *TextRank*, which is based on same principal as *PageRank* and from its key ideas is very similar to patterns used in this system. Results indicate that this algorithm performs approximately by 21% worse than here implemented system and its performance is slightly better than random sentence selection, which was also implemented.

More detailed results are visible in table 5.19.

Figure 5.16: Chart of average overall performance

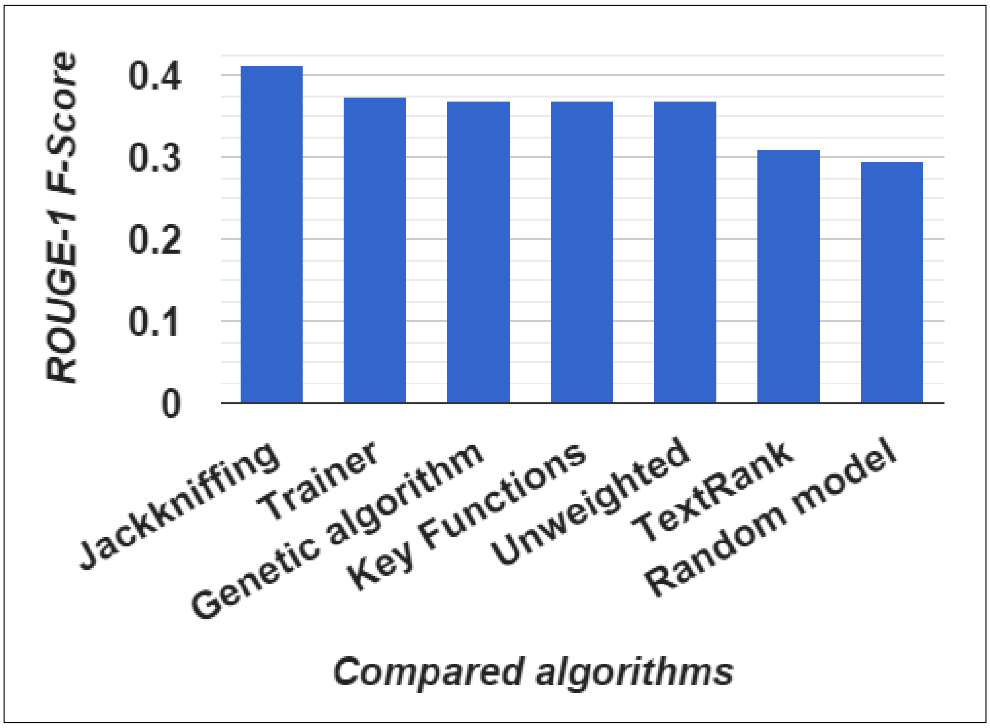


Table 5.19: ROUGE-1 *F-score* performance of several algorithms

Algorithm	Run 0	Run 1	Run 2	Run 3	Run 4	Average
Jackkniffing	0.3951	0.4087	0.4163	0.4368	0.4007	0.4115
Trainer	0.3788	0.3839	0.3423	0.3746	0.3845	0.3728
Genetic algorithms	0.3682	0.3764	0.3461	0.3707	0.3840	0.3691
Key Functions	0.3624	0.3781	0.3421	0.3786	0.3822	0.3687
Unweighted	0.3749	0.3655	0.3481	0.3746	0.3774	0.3681
TextRank	0.3076	0.3386	0.2825	0.3054	0.3083	0.3085
Random model	0.2857	0.2922	0.3035	0.2981	0.3000	0.2959

5.9 BBC article summarization

BBC Summarizer component was tested by several BBC articles and six of those results are shown in this paper in figure 5.9.1, 5.9.2, 5.9.3, 5.9.4, 5.9.5 and 5.9.6. Highlighted sentences were selected in summary, others not.

5.9.1 Islamic State: 'Kidnapped' Dutch children taken to Syria

A Chechen woman living in the Netherlands has taken her two young children against their father's will to join Islamic State (IS) militants in Syria, Dutch prosecutors say.

The mother, a boy aged eight and a seven-year-old girl are believed to have travelled using false passports.

The divorced father, a Dutchman, had warned the authorities of their imminent departure.

The authorities say it is the first case of its kind.

The 32-year-old woman, who was not identified, had been living in the southern Dutch city of Maastricht.

She and her two children had not been seen since the end of October.

They are understood to have taken a flight from Belgium to Athens and the woman is reported to have contacted her mother in January, saying they were in the IS stronghold of Raqqa in northern Syria.

Prosecutors are treating it as a case of kidnapping and have issued an international arrest warrant.

But they acknowledge that if the woman and her children have crossed the border into Syria, there is very little they can do to bring the family home.

An estimated 200 Dutch nationals, including some minors, are known to have joined IS in Iraq and Syria.

IS has become one of the most dangerous jihadist groups, accused of mass killings and persecution of ethnic and religious minorities in areas it controls.

Available at `http://www.bbc.com/news/world-europe-31915142`

5.9.2 French government orders website block

The French authorities have used new powers to block five websites, which they claim condone terrorism, without a court order.

Internet service providers have 24 hours to comply.

The chairman of European Internet Service Provider OVH tweeted that his firm had not been given any warning.

The new powers apply to sites suspected of commissioning or advocating terrorism or distributing indecent images of children.

The rules were approved along with other counter-terrorism measures by the French parliament last year.

It is the first time they have been put to use to block websites without going through a court process.

Visitors to the sites affected are now directed to a page from the French Interior Ministry, containing a graphic of a big red hand.

Following the attack on French magazine Charlie Hebdo in January, interior ministers from 11 European countries including France expressed concern in a joint statement at "the increasingly frequent use of the internet to fuel hatred and violence".

"They added that they were determined to ensure that "the internet is not abused to this end, while safeguarding that it remains...a forum for free expression.

Available at `http://www.bbc.com/news/technology-31904542`

5.9.3 Afghanistan suicide bomb in Jalalabad leaves many dead

At least 33 people have been killed and 100 injured in a suicide bomb attack in the eastern Afghan city of Jalalabad.

The blast happened outside a bank where government staff and military personnel were collecting their salaries.

A spokesman for a group claiming to represent Islamic State in Afghanistan said it carried out the attack, though the BBC cannot verify the claim.

The BBC's Shahzeb Jillani in Kabul says it is the largest attack in Jalalabad in many months.

Children are said to be among the victims.

President Ashraf Ghani called it a "cowardly and heinous terrorist act".

The Taliban did not claim responsibility for the attack. Daesh [IS] claimed responsibility."On a visit to north-eastern Badakhshan province, he said: "Who claimed responsibility for the horrific attack in Nangarhar [province] today?

One eyewitness, Jaweed Khan, said: "I saw many people, dead bodies and injured people on the ground.

"Ambulances arrived very late, and many people died of their wounds."

Police said another bomb was discovered nearby, and was destroyed in a controlled explosion.

Another blast was also reported outside a shrine in Jalalabad on Saturday morning. There were no reported casualties.

Shahidullah Shahid, who claims to be a spokesman for Islamic State (IS) in Afghanistan, said the group was behind the attack on the bank.

He also named a man who he said was the attacker.The BBC cannot confirm either claim.If confirmed, it would be Islamic State's first major attack in Afghanistan.

Mr Shahid was a spokesman for the Pakistani Taliban until he was fired for pledging allegiance to IS last year.

A spokesman for the Taliban, Zabihullah Mujahid, told news agencies the group was not behind the bank blast.

"It was an evil act.We strongly condemn it," he told Reuters.

Jalalabad has been a repeated target for the Taliban in the past year.

After years of Nato intervention, the challenge of fighting extremist groups is now solely in the hands of Afghan forces, but a new report indicates the number of troops and police is falling and creating new security risks.

Only a few thousand Nato troops remain in the country, largely in training roles, after their combat role ended in December.

Available at `http://www.bbc.com/news/world-asia-32363749`

5.9.4 Australian teenagers held over alleged Melbourne terror plot

Police in Australia say they have foiled an Islamic State-inspired plot to carry out an attack at a World War One centenary event.

Police arrested five teenage suspects, charging one 18-year-old with conspiring to commit a terrorist act.

The men were planning to target police at an Anzac memorial event in Melbourne next week, police said.

About 200 police officers took part in the counter-terrorism operation in the city early on Saturday.

Acting Deputy Police Commissioner Neil Gaughan told reporters that evidence suggested the suspects had been influenced by Islamic State.

One of the men, Sevdet Besim, appeared briefly in Melbourne Magistrates Court on Saturday.

Police say a second man held on terrorism-related offences is also likely to be charged.

A third man, also 18, was arrested on weapons charges and two other teenagers, aged 18 and 19, were in custody and assisting with inquiries.

Officials referred to possible attacks using "edged weapons", but Mr Gaughan said there was no evidence to suggest there was "a planned beheading".

The men were "associates" of Abdul Numan Haider, a teenager shot dead in September after he stabbed two officers, police said.

Anzac Day is an annual day of remembrance for servicemen and women from Australia and New Zealand.A series of events are planned for next week to coincide with the 100th anniversary of the landings at Gallipoli, Turkey.

Australian Prime Minister Tony Abbott urged people to turn up to memorial events as planned.

"The best thing we can do to counter terrorism...as individuals is to lead normal lives," he said, adding that the authorities were doing everything possible to keep people safe.

Police said that although officers were the primary target of the alleged plot there was also a threat to the public.

Search operations were continuing at several addresses in the south-east of the city on Saturday.

The premier of Victoria, Daniel Andrews, said the police presence at Anzac Day events would be "significantly increased".

"These individuals arrested today are not people of faith, they don't represent any culture," he added.

"This is not an issue of how you pray or where you were born...this is simply evil, plain and simple."

Australia raised its threat level to high last September and has since carried out a series of counter-terrorism raids.

Available at `http://www.bbc.com/news/world-australia-32361721`

5.9.5 Ukraine crisis: Rebel leader warns truce 'could fail'

A senior separatist leader in eastern Ukraine has claimed an agreed ceasefire deal will fail unless Kiev recognises the independence of rebel-held areas.

Aleksandr Zakharchenko told the BBC he wanted to expand the self-proclaimed Donetsk People's Republic (DNR).

He said the truce agreement brokered by the West in Minsk in February was not being properly implemented by Kiev.

"Ukraine doesn't want to resolve all the issues," he said.Kiev has repeatedly denied the claim.

In other developments:

"If you agree to resolve something, then you need to act and move forward, and resolve everything that's included," said Mr Zakharchenko, the DNR head.

""If that doesn't happen, then the Minsk agreement is unfulfilled, and it renders all the meetings in Minsk pointless.

He also accused Kiev of preparing for war - a charge Ukraine denies.

Under the Minsk agreement, backed by France, Germany and Russia, Ukraine's government claims that the rebel-held east will remain part of the country.

But Mr Zakharchenko insisted it must be legally recognised as an independent territory.

He said: "Ukraine has stopped paying welfare, pensions and other payments that are obligatory for a state to pay its citizens.

""They don't do it, so they've de facto recognised us.

The BBC's Tom Burridge reports that throughout this week shelling could be heard in central Donetsk, a sign that the Minsk deal had not brought real peace to the region.

On Saturday, Ukraine said the rebels had attacked government positions in the eastern Luhansk region - but the assault was repelled.

Ukraine and the rebels both claim to have withdrawn heavy weapons from the line of contact.

The UN says at least 6,116 people have been killed since the fighting began in the Donetsk and Luhansk regions last April - a month after Russia annexed the Crimean peninsula.

Ukraine accuses Russia of arming the rebels and sending Russian troops over the border - a claim Moscow denies.

Available at `http://www.bbc.com/news/world-europe-32363766`

5.9.6 US paratroopers to train Ukraine army as Russia complains

About 300 US paratroopers have come to western Ukraine to train with Ukrainian national guard units, the US Army says.

Kremlin spokesman Dmitry Peskov warned that the move "could seriously destabilise the situation" in Ukraine.

He said the US military presence "is a long way from helping towards a settlement of the conflict", Russia's RIA Novosti news agency reported.

The US and its Nato allies accuse Russia of sending soldiers and weapons to the separatists in eastern Ukraine.

Russian President Vladimir Putin again dismissed the Western charges on Thursday, telling millions of Russians that "there are no Russian troops in Ukraine".

The rebels in the east signed a ceasefire agreement with the Ukrainian government in February, but recently the number of violations has escalated.

There has been further shelling on the outskirts of rebel-held Donetsk and in the village of Shyrokyne, near the southern port city of Mariupol.

Ukraine's national guard has been involved in the fighting.It includes various volunteer units who are now being integrated with the Ukrainian regular army.

The US Army said the US paratroopers were part of the 173rd Airborne Brigade.

The training will take place at Yavoriv, near Lviv in western Ukraine.The US forces will begin training three battalions of Ukrainian troops over the next six months, the statement said.

The brigade trained with Ukrainian forces in international exercises in Ukraine last September.

Available at `http://www.bbc.com/news/world-europe-32349308`

Conclusion

Purpose of this work was to design, implement and evaluate automatic text summarization system for English texts based on research about current methods used in this field.

At the beginning of this work, thorough analysis about current state-of-the-art was done, information about classifications and approaches of text summarization and its systems were collected. Found information was used as basis for proposed and implemented summary system.

In design, summary system was specified and categorized as single document summarization system using corpus based database performing over English written texts by extracting most valuable sentences with arbitrary length of summary. All required and supporting components of system were also specified.

Main summarization core is based on graph theories and Markov chains, which are built from document sentences and their mutual similarities which were measured by 10 designed and implemented measures. In those chains, stationary distributions, expressing probabilities of node (sentence) visitation in infinite number of steps, is found and used as indicator of sentence value. System further performs optional fine-tuning of these values by adjusting preliminary weights of used similarity measures by learning.

After that, system was implemented and interesting parts of this implementation was captured in this thesis. Most important components of system were then subjected to testing of performance, time consumption, parallel capabilities and validation and found bugs and errors were fixed and results were recorded. It was discovered that it is possible to significantly speed up feature extraction, learning and overall document processing. Implementation was accordingly adjusted to those findings to perform faster.

System was design to be easily upgraded if needed and as a result, this thesis managed to fulfil all requirements for this work.

Bibliography

[1] Bajpai, P.; Kumar, M. Genetic algorithm–an approach to solve global optimization problems. *Indian Journal of computer science and engineering*, volume 1, no. 3, 2010: pp. 199–206.

[2] Petr Fier, J. S. Simulated Evolution I. - Genetic Algorithms. *Problems and Algorithms*, 2014.

[3] Gholamrezazadeh, S.; Salehi, M. A.; Gholamzadeh, B. A Comprehensive Survey on Text Summarization Systems. *2009 2nd International Conference on Computer Science and its Applications*, 2009: pp. 1–6. Available from: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5404226`

[4] Das, D.; Martins, A. F. A survey on automatic text summarization. *Literature Survey for the Language and Statistics II course at CMU*, volume 4, 2007: pp. 192–195.

[5] Gupta, V.; Lehal, G. S. A Survey of Text Summarization Extractive Techniques. *Journal of Emerging Technologies in Web Intelligence*, volume vol. 2, no. issue 3, 2010-08-20: pp. –. Available from: `http://www.academypublisher.com/ojs/index.php/jetwi/article/view/3180`

[6] Nenkova, A.; McKeown, K. A survey of text summarization techniques. In *Mining Text Data*, Springer, 2012, pp. 43–76.

[7] Wang, S.; Li, W.; Wang, F.; et al. A Survey on Automatic Summarization. *2010 International Forum on Information Technology and Applications*, 2010: pp. 193–196. Available from: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5635127`

[8] Uddin, M. N.; Khan, S. A. A study on text summarization techniques and implement few of them for Bangla language. *2007 10th In-*

ternational Conference on Computer and Information Technology, 2007: pp. 1–4. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4579374

[9] Devasena, C.; Hemalatha, M. Automatic text categorization and summarization using rule reduction. In *Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on*, IEEE, 2012, pp. 594–598.

[10] Binwahlan, M. S.; Salim, N.; Suanmali, L. Swarm Based Text Summarization. *2009 International Association of Computer Science and Information Technology - Spring Conference*, 2009: pp. 145–150. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5169327

[11] Ai, D.; Zheng, Y.; Zhang, D. Automatic text summarization based on latent semantic indexing. *Artificial Life and Robotics*, volume vol. 15, no. issue 1, 2010: pp. 25–29. Available from: http://link.springer.com/10.1007/s10015-010-0759-x

[12] Mashechkin, I. V.; Petrovskiy, M. I.; Popov, D. S.; et al. Automatic text summarization using latent semantic analysis. *Programming and Computer Software*, volume vol. 37, no. issue 6, 2011: pp. 299–305. Available from: http://link.springer.com/10.1134/S0361768811060041

[13] Fellbaum, C. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[14] Antiqueira, L.; Oliveirajr, O.; Costa, L.; et al. A complex network approach to text summarization. *Information Sciences*, volume vol. 179, no. issue 5, 2009-02-15: pp. 584–599. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0020025508004520

[15] Wei, Y. Document summarization method based on heterogeneous graph. *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*, 2012: pp. 1285–1289. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6234047

[16] Ferreira, R.; Freitas, F.; de Souza Cabral, L.; et al. A Four Dimension Graph Model for Automatic Text Summarization. *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, 2013: pp. 389–396. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6690041

[17] Cheng, K.; Li, Y.; Wang, X. Single Document Summarization Based on Triangle Analysis of Dependency Graphs. *2013 16th International Conference on Network-Based Information Systems*, 2013: pp. 38–43. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6685374

[18] Chengcheng, L. Automatic Text Summarization based on Rhetorical Structure Theory. *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, 2010: pp. V13–595–V13–598. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5622918

[19] Lin, C.-Y. *ROUGE: Recall-oriented understudy for gisting evaluation.* software, Information Sciences Institute of University of Southern California, California, 2003.

[20] ROUGE: (metric). In *Wikipedia*, 2014. Available from: http://en.wikipedia.org/wiki/ROUGE_(metric)#Metrics

[21] Sivanandam, S.; Deepa, S. *Introduction to genetic algorithms.* Springer Science & Business Media, 2007.

[22] Rehorek, T. Genetic programming. *Basics of Artificial Intelligence*, 2013.

[23] Pohlheim, H. Evolutionary Algorithms. *The Genetic and Evolutionary Algorithm Toolbox for Matlab*, 2006.

[24] Larose, D. T. *Genetic Algorithms.* John Wiley & Sons, Inc., 2006, ISBN 9780471756484, pp. 240–264, doi:10.1002/0471756482.ch6. Available from: http://dx.doi.org/10.1002/0471756482.ch6

[25] Kordik, P.; Slapak, M. Introduction to Evolutionary Algorithms. *Computational Intelligence Methods*, 2015.

[26] Collection of stop words lists. Available from: http://www.webpageanalyse.com/dev/stopwords/en,http://dev.mysql.com/doc/refman/5.1/en/fulltext-stopwords.html,http://www.ranks.nl/stopwords,http://www.lextek.com/manuals/onix/stopwords1.html,https://code.google.com/p/stop-words/,http://norm.al/2009/04/14/list-of-english-stop-words/

[27] Porter, M. F. Snowball: A language for stemming algorithms. Published online, October 2001, accessed 11.03.2008, 15.00h. Available from: http://snowball.tartarus.org/texts/introduction.html

[28] XML Schema Similarity Mapping: JWS. 2008. Available from: https://code.google.com/p/xssm/

[29] WordNet Similarity for Java. 2013. Available from: `https://code.google.com/p/ws4j/`

[30] Ganesan, K. Basics of Setting up ROUGE Toolkit for Evaluation of Summarization Tasks. In *Http://kavita-ganesan.com/*, 2010. Available from: `http://kavita-ganesan.com/rouge-howto`

[31] RapidMiner. 2014. Available from: `http://sourceforge.net/projects/rapidminer/`

[32] TextRank. 2013. Available from: `https://github.com/davidadamojr/TextRank`

Appendix A

Acronyms

NLP	Natural Language Processing
CPU	Central Processing Unit
DUC	Document Understanding Conference
tf-idf	Term Frequency - Inverse Document Frequency
SVD	Singular Value Decomposition

Appendix B

User Guide

B.1 Installation

To run summary system, it is required to install some perl libraries and Java runtime environment. The summary system is compatible with linux-based platforms, especially `Ubuntu 14.04.2`. In order to execute installation, perl needs to be installed on desired platform.

To run installation script, execute following commands:

```
cd <path to media>/ATS
./SetupFiles/install.sh
```

This process takes about 10 minutes and requires occasional user interaction in accepting default settings, licences and entering password for `sudo` command.

B.2 Running demo

To run summary system demo, composed of 10 documents with user-written summaries processed by all steps in workflows in figures 3.1.1 and 3.1.2, execute following commands:

```
cd <path to media>/ATS
java -jar dist/AutomaticTextSumarization.jar
```

This process takes about 8 minutes and requires no further interaction. The system prints ongoing work into `terminal`. Most important part is last printed section containing performance and locations of system-generated summaries.

Appendix C

Contents of enclosed DVD

ATS	source codes, summary system and libraries
└─ Crawler	crawler libraries
└─ Data	documents, summaries and preprocessed files
└─ dist	runnable system distribution
└─ javadoc	system documentation
└─ lib	system required libraries
└─ AutomaticTextSumarization.jar	the runnable demo
└─ JFreeChart	chart plotter libraries
└─ Nouns	processed and unprocessed nouns lists
└─ ROUGE	ROUGE script
└─ SetupFiles	required files to run the demo
└─ install.sh	installation script
└─ src	source codes
└─ Stemming	Snowball stemmer
└─ StopWords	processed and unprocessed stop words lists
└─ WordDistance	word distance libraries
└─ WordNet	WordNet thesaurus
thesis	thesis
└─ source	LaTeX source codes of the thesis
└─ text	thesis text
└─ MT_Hlavac_Simon_2015.pdf	the thesis in PDF format