An Automatic Text Summarizer

Alexander Bogdanovski

Supervisor: Dr. Rob Gaizauskas

COM3010

This report is submitted in partial fulfilment of the requirement for the degree of Bachelor of Science with Honours in Computer Science by Alexander Bogdanovski

May 3, 2006

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross–referencing to author, work and page(s). Any illustrations which are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Alexander Bogdanovski

Signature: .....

Date: May 3, 2006

Abstract

Automatic text summarization is a process in which a computer takes a text document as input and outputs a summary of that document. There are various approaches to it, some of which have been around for more than 40 years. In general we talk about single document summarization and multi–document summarization.

This project focuses on multi–document summarization. Its aim is to create a theoretical foundation for the implementation of a multi–document summarization system. I will review some of the more recent findings in the area and then I will further discuss the approaches that are particularly suitable for this project. Finally the design, implementation and evaluation of the new system will be documented.

Acknowledgements

Many thanks to Dr. Rob Gaizauskas for his words of advice.

I would also like to thank David Jarzebowski for the work he has done on the project.

Contents

1 Introduction

In this chapter I will give a basic definition of automatic summarization as well as explain some basic notions in the field of automatic summarization. Finally I will review some of the more popular approaches to this problem.

1.1 What is Automatic Summarization?

As defined by Mani (2001) automatic summarization is an automated process in which a computer takes a piece of information, also called the *source* (i.e. an electronic document), selects the most important content in it, and presents that content to the user in a condensed form. This usually involves a *computer* and some *specialized software* to carry out the actual summarization. Such piece of software is called a summarization system, or summarizer. A summarizer is a system which produces a condensed version of a document's content in a form readable by humans. A good summarizer should select only the important information in the text and extract it. This notion of importance is somewhat vague and can be challenging for both humans and computers. Humans often disagree on what part of the information is important and computers have no idea what is worth extracting and what is not. Summarization systems are simply computer programs which tell the computer what information to extract from the source document(s).

Automatic summarization has been around for more than 40 years now. Research in this field has been also very active in recent years. The idea of having a computer produce an automatic summary of a document for you has always looked appealing for computer scientists. The need for text summarization is even greater today because with the ever growing Internet comes the problem of information overload. People have access to vast amounts of information and yet not enough time to digest all of it. Therefore summaries can be extremely useful and time–saving. There are already many places where you can see summarized text (not computer generated) — newspaper headlines, abstracts of research papers, even tables of football game scores.

1.2 Concepts and Definitions

Automatic summarization may be applied to both text documents and multimedia. Examples of document summarization can be news summaries or scientific abstracts. Summaries of TV programmes, movie trailers or a short version of a sound clip are examples of multimedia summarization. However multimedia summarization is still in an early stage of research.

There are two distinct forms of automatic summarization, namely *extracts* and *abstracts*. The main difference between the two is the output of the summarizer. Extracts contain parts of the same content as it appears in the source document. This is because the

summarizer selects only the salient sentences in the document and extracts them as they are — nothing is changed. An abstract is a "summary at least some of whose material is not present in the input" (Mani, 2001). It is usually produced by paraphrasing the source.

Summarizers fall into two categories: *single document* and *multiple document*. Here the difference between the two is mainly in their input. A single document summarizer takes one document at a time as input. And the inputs of a multi–document summarizer are two or more documents containing similar information. The focus of this project will be on multi–document summarizers.

A summarizer can usually produce either *generic* or *user–focused* summaries. A user–focused summary is the one in which specific information is selected in order to satisfy the requirements of a particular user group. As opposed to that, a generic summary is more suitable for the average reader.

There are also two types of summaries depending on their function. There can be *informative* and *indicative* summaries. The purpose of the first type is to deliver as much information as possible to the user and can also serve as a substitute for the source. Indicative summaries on the other hand are only meant to help the user decide whether or not to read the source document.

Finally I would like to discuss a few important concepts related to automatic summarization. These are *coherence and redundancy*. Coherence is important because it is a feature of a good summary. A summary is said to be incoherent if it contains random sentences which have no link between them or dangling anaphora. Redundancy is also a big issue especially in multi–document summarization. A summary contains redundancy if it has more than one sentence with the same meaning in it.

1.3 Approaches

There are a few different approaches to the problem of automatic summarization. We can characterize them into two main groups — single document and multi–document approaches. As defined by Mani and Maybury (1999) there can be *surface-level, corpus based, discourse structure based* and *knowledge based* approaches for single document summarization. All these approaches could successfully applied to multi–document summarization. Recent research papers have used domain specific knowledge to build multi–document summarizers (McKeown and Radev, 1995).

The surface–level approach requires shallow understanding of the text and this usually involves analysis of the syntactic structure of sentences. It is used to extract salient information by taking into account some key features of a sentence. Such features are the location of a sentence in the paragraph, presence of frequent salient terms and presence of keywords found both in the sentence and in the title or headings. Surface–level approaches

are covered in Section 2.1.1.

Corpus based approaches involve statistical analysis of large bodies of text (corpora) to find specific features about the documents in them. These approaches are covered in Section 2.1.2.

Human abstractors create a mental discourse model of a document while reading it. Discourse-level approaches try to create similar model of the discourse structure of a document which can later be used for the generation of a summary (Mani, 2001). See Section 2.1.3.

Knowledge based approaches are used for the creation of summarization systems which act in a specific domain (i.e. domain dependent). Such systems usually produce high quality summaries but do not have the ability to adapt to different types of documents (domains). See Section 2.1.4 for more details.

1.4 Historical Background

Text summarization has been around since late the 1950's, when some of the first systems were developed. In the 60's some systems began to emerge which used simple syntactic analysis of text. One of the early pioneers in the field was Luhn (1958) who used statistical analysis which was based on the term frequencies. In the 70's some of the first commercial applications ware available to the public. Those were based on surface–level approaches with the use of cue phrases. The 80's spawned the first discourse–level and hybrid approaches. Finally the late 90's were a very fruitful period for text summarization. Recently, a lot of research has been going on in new areas such as multi–document, multilingual and multimedia summarization (Mani and Maybury, 1999).

1.5 Aims and objectives

The aims of the project are to acquire a certain amount of knowledge in the area of text summarization and to utilize this knowledge in order to produce a software system for automatic text summarization. The focus will be on multi–document summarization techniques but classical approaches will be discussed as well. I will take into account the work that David Jarzebowski has done before me, and try to contribute as much as I can to the further development of the project.

The objective for this stage of the project is to analyze the problem and come up with a possible solution. The objectives for the rest of the project fall into two groups — research objectives and development objectives. Research objectives are:

- Familiarize with the topic by reading related literature.

- Produce an initial research document (this document).

  Objectives related to the development stage are listed below:

- Build a multi–document summarization system which implements extraction techniques to produce summaries from multiple documents.
- Produce two additional baseline systems.
- Evaluate the system by comparing it to two baseline systems.
- Analyze the results and findings in the final document.

2 Literature Survey

In this chapter I will review some of the literature about automatic document summarization. There is a vast amount of publications in the field and the following sections represent a brief survey of the literature available. I will start off with some of the classical single document approaches and explore the different techniques that have been used there. Then I will go on to multi–document summarization where I will talk about the most recent achievements in this area. Finally I will discuss some evaluation methods and frameworks.

2.1 Single Document Summarization

2.1.1 Classical Approaches

The papers mentioned here are considered classics because they have created the foundations for modern applications and have also been a source of motivation for researchers. These work in those papers has been done using shallow linguistic analysis, primarily on the surface level of text.

Luhn's paper on automatic abstracting provides a simple method for creating abstracts from specialized literature (Luhn, 1958). Here the author motivates the reader by expressing the advantages of automatic abstracting — inexpensive and requires much less intellectual effort.

Luhn (1958) used an algorithm which scans the source document for the most salient information. In this algorithm a weight is assigned to each sentence according to the term frequencies in the text. As the document is being scanned, pronouns, prepositions and other common words are filtered by using a stop–list and then the remaining terms are sorted alphabetically. Next statistical analysis is applied to the list of sorted terms where pairs of succeeding words from the input document are compared letter by letter. This allows for similar words to be found (e.g. *differ, difference, different*). All similar words are grouped together and are called *significant* words. The weight of a sentence, or its *significance factor* is determined by the formula:

$$sf = \frac{(number\ of\ significant\ words)^2}{the\ total\ number\ of\ words} \tag{1}$$

The sentences which have the highest weight value are extracted to produce the "auto–abstract". The system produced by Luhn was of reasonable quality given that, at that time, there were not many documents in electronic form. Luhn calls the final output an abstract, but it is essentially a summary produced by extraction. This should be made clear since we usually refer to abstracts as summaries "at least some of whose material is not present in the input" (Mani, 2001).

The work of Edmundson (1969) is one of the most influential in the area of automatic summarization. He created a framework for developing extraction based summarizers and also provided an evaluation for his system, the results of which were indeed very useful.

The important innovation in Edmunson's work was the introduction of three new parameters for calculating the weights of sentences. Those were the *sentence position* in text, *cue words* and *title and heading words*.

The sentence position parameter indicates the place in the paragraph where a given sentence is located (e.g. the beginning or the end). This parameter was also used for assigning weights to sentences depending on their position in text. For example a positive score would be given to a sentence which was found in the first or last paragraphs of the document. This was a sensible ranking system as most informative sentences tend to be located either at the beginning ("Introduction") or at the end of the text ("Conclusion").

Examples of cue words are "significant," "impossible," and "hardly." Cue words were stored in a cue dictionary which consisted of three smaller dictionaries: bonus words (positive relevance), stigma words (negative relevance) and null words (irrelevant) (Edmundson, 1969). Thus each word was ranked according to its relevance. The final weight for a sentence equals the sum of the weights of the words in it. Keywords, on the other hand, are words that do not appear in the cue dictionary but are specific to a document.

The title is very important part of a document because it can reveal the subject matter of that document. The assumption made by Edmunson was that authors usually use informative titles. While this is true for most cases, there can some exceptions.

Edmunson used a large corpus of 200 documents of scientific papers which were chemistry related. Apart from that he used another corpus of 200 documents which were related to other fields like physical science, life science and information science. This was used for purposes of statistical analysis (e.g. common words, sentence position etc.) and also determined the initial weights and parameters (Edmundson, 1969). The final weight, $W(s)$, for a sentence s was calculated by a linear function containing the sum of the three parameters (C cue words, K keywords, L location, T title) above and the keyword parameter:

$$W(s) = \alpha C(s) + \beta K(s) + \gamma L(s) + \delta T(s) \tag{2}$$

Edmunson also experimented with the parameters in the above equation and, through evaluation, he found that keywords were not as good feature as the other three, and also that the combination of cue words, title and location gave the best performance. He also found that location was the best and keywords were the worst individual features (Mani, 2001, p. 49).

Pollock and Zamora (1975) focused on automatic extraction from specialized documents. They tried to demonstrate that using a genre–specific algorithm for extraction

yields better results that a more general approach. The aim of the paper was to develop a system which outputs a summary which conforms to the standards of the Chemical Abstracts Service (CAS).

Pollock and Zamora (1975) used an interesting algorithm which was used for sentence *rejection* rather than *selection*. The final output is an indicative summary, about 10–20% the size of the source. The idea behind the algorithm is that each word can be ranked using "semantic codes" which determine how suitable for extraction the word is. For that purpose Pollock and Zamora prepared a long list of words with more than 700 terms in it. Each term in this list was given a semantic code. Semantic codes were a sort of marking system and decided whether a word or a phrase is an indicator for informativeness. For example the phrase "our results" is given the code "I" which means "very positive" which is also the highest mark. If a word is not suitable it is given a lower code like "B" (negative), or even "M" (super–negative, delete sentence). Once each word is marked accordingly, sentences are rejected of selected respectively, depending on their overall score.

The first two papers discussed above proposed a similar solution to the problem by assigning weights to sentences. Luhn's system was very simple since it used only the term frequencies as a feature for extraction. Edmunson's system showed much better results in comparison to Luhn's. This was because Edmunson realized that features like sentence location and title were important features for extraction. His paper also shows that using keywords as the only feature for extraction will give poor results (Edmundson, 1969). Then finally the third paper had a different approach to the problem of producing genre specific summaries (Pollock and Zamora, 1975). The algorithm here rejected sentences that were considered less informative. This technique proved to be quite effective for chemistry related documents.

2.1.2 Statistical (corpus–based) Approaches

In the previous section I talked about some of the early work that was done in the field. The approaches used there were simple and yet effective in most cases, but the analysis phase was done only on the single source document. In this section I review some papers on corpus based techniques for automatic summarization.

A corpus is a collection of documents. Usually the documents in the collection are of different varieties. Corpus based approaches are different than other approaches in their analysis phase. This means that they analyse an entire corpus of documents instead of a single document (i.e. the source). Machine learning techniques are often used in order to "learn" important information about the documents in the corpus. For example features like sentence location may have different values for different types of documents like newspapers and scientific papers (Mani, 2001). So if a learning algorithm is applied to a corpus of newspapers articles, for example, it will learn that the $L(s)$ term in equation (2) will have a higher value if sentence s is in the beginning of the article. Sometimes

the term frequency alone is not a satisfactory measure of the importance of terms. That is why the $tf.idf$ measure is used. It is also widely used for the purposes of information retrieval. The tf part stands for "term frequency" — the number of times a term T occurs in a document. The idf part stands for "inverse document frequency" and is calculated by the formula:

$$idf = \log_2 \frac{N}{n} \tag{3}$$

,

where n is the number of document the term T appears at least once, and N is the number of documents in the collection. This is particularly useful because if a word appears a lot in one document but rarely in other documents then it is a relevant keyword for that document. The $tf.idf$ value is usually only calculated for words that are not in the stop list.

Kupiec et al. (1995) presented a more advanced extractive summarization system. It used two corpora — a test corpus and a training corpus. The training corpus contained document/summary pairs. These summaries were *abstracts* created by professional abstractors. The algorithm was a Bayesian classifier which calculated the probability of a sentence being relevant. The formal definition for the classifier is derived from Bayes' rule. Here S is the summary to be produced and F_1 to F_k are the features:

$$P(s \in S|F_1, F_2, \ldots F_k) = \frac{P(F_1, F_2, \ldots F_k|s \in S)P(s \in S)}{P(F_1, F_2, \ldots F_k)} \tag{4}$$

This means that probability that a sentence s will be selected for extraction depends on the features F_1 to F_k. Kupiec et al. (1995) used five main features — sentence length, cue phrases, position of a sentence in paragraph (paragraph–initial, paragraph–final etc.), thematic words (most frequent words) and an uppercase word feature (proper names).

In order to train their summarizer, Kupiec et al. (1995) used a sentence matching technique which would find correspondence between manual summary sentences and sentences in the original document (Kupiec et al., 1995). Thus a sentence from the manual summary could be a *direct match* with another sentence in the source, a *direct join*, meaning two sentences were used from the source to produce one in the manual summary, or it could be *unmatchable*.

The evaluation results in Kupiec et al. (1995) showed that the system produced good summaries with a high percentage of relevant sentences in them. The conclusion made at the end was that the best combination of features was paragraph-cue phrase-sentence length, and the use of the keywords feature only decreased the overall performance. These observations were in agreement with Edmundson (1969)'s (see Section 2.1.1).

Aone et al. (1999) used similar techniques to the ones presented in Kupiec et al. (1995) above. Their work goes beyond the typical frequency–based summarization systems and they used multi–word phrases as the basic text unit, instead of words.

A huge corpus (800 MB) of newspaper articles was pre-processed and tagged by Aone et al. (1999). A database was created from this corpus containing multi–word phrases and names. And since words were extracted along with their context (i.e. surrounding words) the database had different records for the same word or name. For example the company "Ford" was different from President "Ford". The system also incorporated some knowledge of the corpus and was able to statistically derive *collocation phrases* (e.g. "computer chips", "potato chips"), find *signature words* by calculating *idf* values and recognize *associated phrases* (e.g. "Bayer" and "aspirin"). By gathering knowledge about the corpus, the system was able to adapt to different domains automatically. Another feature of the system was that it could be trained to better recognize signature words by using Bayesian statistics (see (Kupiec et al., 1995)).

The system explained by Aone et al. (1999) was implemented as a client–server application. The evaluation was carried out in two phases — first without training and then with the trained system. The results showed that the system was able to do better extraction when person names were removed from the text that was processed (but appeared in the summary). This was due to the fact that names have high *idf* values but do not indicate any relevant topics in the document. The conclusion was that names of people did not make good keywords and were rather misleading. The trained system gave better overall results in the tests and in particular it had a better precision and recall scores.

Finally I will briefly discuss the work of Hovy and Lin (1999). They presented a summarization system called SUMMARIST which used *topic identification*, *interpretation* and *generation* operations to produce summaries. The system combined statistical techniques, knowledge about the corpus and was designed to create both abstracts and extracts.

Interpretation was the second step in the summarization process in Hovy and Lin (1999). Here two or more topics were "fused" into one concept. This process was considered to be the most difficult part of the summarization process because it requires knowledge about the world which is rarely included in the text explicitly.

Generation was the final step in the process of summarization. The SUMMARIST system was able to produce extracts without generation by simply reproducing the sentences selected in the topic identification stage. Also it could also output topic lists with all keywords and fused concepts. And finally a sentence generator together with a sentence scorer were be used to produce abstracts.

In this section we saw how complex summarization methods and techniques have emerged based on corpus statistics. With the increasing amount of electronic publications and documents available, corpus based approaches become more popular and statistical analysis becomes a natural approach to the problem of automatic summarization. Nevertheless problems like coherence still exist when producing extracts and more advanced

approaches are needed for natural language generation.

2.1.3 Discourse Structure Based Approaches

Discourse structure approaches try to model the strategies that professional human abstractors use for producing abstracts. By studying the way humans create abstracts we can gain a better insight into the process of creating summaries. Then we can use this knowledge to create a better summarization system. Abstracts are usually very condensed summaries which try to follow the internal structure of the source document.

Before I start discussing the advances in this area I will explain the difference between *text coherence* and *text cohesion*. Text coherence represents the relationships between sentences and clauses in the text. We can also think of text coherence as being related to the notion of a *theme*. And text cohesion "involves relations between words, word senses, or referring expressions, which determine how tightly connected the text is" (Mani, 2001, p. 92). It also provides a way of finding the meaning of the text by examining linguistic relations such as anaphora, synonymy, hypernymy ("kind of") and meronymy ("part of"). Text cohesion represents the relationships between words in the text, as opposed to relationships between sentences (text cohesion).

Boguraev and Kennedy (1997) explored phrasal analysis and the anaphoric relations in text. The paper presented a system for summarization based on discourse structure. The system produced so called "capsule overviews" — a set of key phrases and sentences from the original document. The system architecture contained several components: *preprocessing, linguistic analysis, discourse segmentation, phrasal analysis, anaphora resolution, calculation of discourse salience* and *topic identification*. These components worked together such that the output of one was the input of the next and so on. The capsule overview was in the form of a list of sentences, or parts of sentences, grouped together by topic.

Barzilay and Elhadad (1999) proposed a way of exploiting *lexical chains*. Barzilay and Elhadad created an algorithm which used several knowledge sources: the WordNet thesaurus, a part–of–speech tagger, a shallow parser and a segmentation component. In the process of summarization, text was first segmented (tokenized) and then lexical chains were produced. The procedure for chain construction was selecting a set of words and then for each one finding a related chain. A "relatedness" criterion was used for that purpose and if a related chain was found then the word was inserted into the chain.

The paper suggested that a lexical chain of low frequency words could carry the same salient information as the use of high frequency words. The evaluation showed that this method outperformed many commercial applications and could be used for building good quality summaries. The metric used for evaluation was the *percent agreement*. It measured the agreement among human judges (see Section 2.3). Results were better for a 10% summary with 96% average agreement, and for the 20% summary the agreement was 90%.

Marcu (1995) used rhetorical structure theory to build trees. His *rhetorical parsing algorithm* used cue phrases to derive a rhetorical structure in the form of a tree. The nodes were labeled with names of rhetorical relations (e.g. elaboration, concession) and the leaves of the tree contained elementary textual units. Each node in that tree was either a *nucleus* or a *satellite*. Nuclei nodes were assumed to hold more salient information than satellite nodes.

Marcu (1995) presented a discourse–based summarizer which took rhetorical structure trees and used them to construct the final summary of a document. This was possible because the formalized structure of these trees allowed for the salience of clauses to be computed (Mani and Maybury, 1999).

As part of the evaluation, Marcu (1995) compared his system with Microsoft's AutoSummarize which was part of the Office97 package. The results expressed the percent agreement between human judges and the system with respect to the most important parts of the text. The tests showed that the discourse–based summarizer in Marcu (1995) created summaries with 60% precision and recall. Microsoft's commercial summarizer performed in the range of 40% precision and recall. And finally it was shown that the best performance was achieved by Marcu's system when manually constructed rhetorical trees were used — 78% precision and 67% recall.

2.1.4 Knowledge Based Approaches

So far I have reviewed mostly domain–independent approaches to automatic summarization. The early works of Edmundson (1969) and Luhn (1958) presented more generic systems for text extraction. These systems had little knowledge about the type of document they were processing. Corpus based approaches use statistical analysis on text documents of various types in order to extracted common features from them.

The difference between all of the summarization systems discussed so far and knowledge rich systems is that, the latter are domain specific and incorporate a great deal of knowledge about a certain domain. This makes them very effective for creating summaries of documents in that domain. The main disadvantage of knowledge rich systems is that they do not adapt easily to different types of documents, which is also a limitation.

Some of the knowledge based systems discussed below output structured data and not ready–to–use summaries (Lehnert, 1981). In relation to that, others like McKeown et al. (1995) take structured data as input and generate natural language summaries from that data. This data represents the most salient information in a document and serves as a basis for creating the final summary. This decreases the complexity of the summarizer since sophisticated linguistic techniques like building rhetorical structure trees are not used (Mani and Maybury, 1999).

Lehnert (1981) talks about plot units as a way of representing the structure of a narrative stories. The motivation behind this idea is that when humans read a narrative they create a mental representation of the story. And a lot of the information we learn about the story is actually inferred and is not explicitly present in the narrative. This means that by using classical summarization techniques we can only extract the information which is explicitly present in a document. In other words no inferred propositions will be recognized.

Lehnert realized that the events in a story may have either a *positive*, *negative* or *neutral* effect on the reader. This is why she proposed *affect states* as building blocks of a plot unit. An affect state could be "+" (positive), "–" (negative) or "M" (neutral mental state). State diagrams could be produced by simply connecting positive, neutral and negative states. A transition, or *casual link*, from a negative affect state to a neutral affect state was described as *motivation*, and a transition from a neutral state to a positive state was *actualization* (Lehnert, 1981). There were also two more casual links — *termination* and *equivalence*.

Primitive plot units could be constructed from the above casual links and affect states. There were a number of different plot units like *success*, *loss*, *resolution*, *problem* etc. Thus sentences from a narrative could be labeled with a corresponding plot unit. For example the sentence "I fixed a flat tire today" is labeled as "success".

More complex plot units were constructed from the primitive ones. For example the complex plot unit "giving up" consists of three primitive plot units — "failure" followed by "problem" followed by "change of mind".

Plot units provided a means of "chunking" the information and then a summary could be produced by combining those "chunks" (Lehnert, 1981). Although Lehnert did not provide a full implementation of the system proposed, she created a framework for high–level analysis and summarization. The author suggested that plot units are good for generalization tasks and that they could also serve as a basis for natural language generation.

McKeown et al. (1995) also used the knowledge based approach. They described two summarization systems — STREAK and PLANDOC. The first application was a summary generator which used structured data from basketball games, and the second generated summaries of telephone network plans. It took data files produced by the Bellcore PLAN software tool as input. The goal that both systems were trying to achieve was to produce condensed summaries that contained as much data as possible.

The STREAK system comprised three main components: *sentence scorer*, *lexicalizer* and *sentence reviser*. The PLANDOC system had a different architecture. It used discourse planning and look–ahead operations such as conjunction and repetition deletion. There were several modules that carried out the plan processing in PLANDOC: *fact generator*, *ontologizer*, *discourse planner*, *lexicalizer* and *sentence generator*.

The sentence scorer module in STREAK took a set of facts as input. These facts were produced by a fact generator which generated facts from a database of game scores. Based on these facts the sentence scorer created a semantic tree which was passed on to the lexicalizer. The lexicalizer processed that tree and mapped it onto a lexicalized skeletal syntactic tree (McKeown et al., 1995). The combined output of the sentence scorer and the lexicalizer modules, was the *first draft*. This draft and the facts served as the input of the sentence reviser module which produced the *final draft*.

In the PLANDOC system a set of facts was passed to the ontologizer. The facts here were again generated by a fact generator. The role of the ontologizer was to enrich these facts with domain specific knowledge and then send them to the discourse planner. The discourse planner took the enriched facts and converted them to more complex facts. Finally the set of complex tasks was fed into the lexicalizer which did the same job as the one in STREAK (explained above). The final summary consisted of sentences automatically generated from syntactic trees.

In comparison to STREAK the PLANDOC system used a simpler and more traditional approach to natural language generation. Both systems implemented opportunistic methods for generating summaries. Although evaluation was not carried out in a formal manner, the two systems showed that the input of a summarization system was not limited to full text only. On the contrary, summarized or structured data could successfully be processed and turned into natural language by knowledge rich approaches (McKeown et al., 1995).

As will be seen in the next section, knowledge based approaches are successfully used in multi–document summarization (McKeown and Radev, 1995).

2.2 Multi–document Summarization

This is a relatively new but very popular research area in automatic summarization. With the growing number of documents available electronically comes the need for some organization of information. Online news sources, for example, publish news articles every day and some of them have different versions of a story. Some may contradict, others may give exactly the same information. And since it is impossible for a user to read all the news on all the web sites, a summary of the news is desirable. Such a summary could give the users an overview of many news sources and inform them about uncertain facts by explicitly showing contradictions in the sources. Multi–document summarization could be used to solve this problem.

As the name suggests, the number of documents used as source range from two to many. The difference between multi–document approaches and corpus based approaches is that in the latter the corpus is usually composed of various types of documents, whereas here there should be at least two documents in the corpus that are on the same topic.

One solution to the problem is to use a *clustering* algorithm to group similar documents together. This is done when there are many document in a corpus on different topics. Once the documents have been grouped, each *cluster* is processed and a summary is produced.

There are several challenges in this area which were not an issue in previous single document approaches. These are:

- redundancy — eliminating redundancy is very important when processing many documents on the same topic.
- grouping — in order to group documents together by topic we need similarity measures for comparing them.
- evaluation — human abstractors do not normally produce summaries from multiple documents so comparison between these and automatically generated ones can be problematic.

The following literature review focuses on the current state of research in this area. The papers herein are also discussed with respect to the challenges above.

The first paper described here is by McKeown and Radev (1995). They present a system called SUMMONS which summarizes related news articles. The SUMMONS was a genre specific system which operated in the terrorist domain. They goal of the system was to generate fluent, variable–length summaries.

SUMMONS was based on a traditional language generation architecture and had two main modules for doing content planning and linguistic operations. The content planner consisted of *paragraph planner* and *combiner*. The linguistic component was made up of a *lexical chooser*, *ontologizer* and a *sentence generator*. A similar architecture was seen in McKeown et al. (1995) (see Section 2.1.4). The input of the system came in the form of MUC (Message Understanding Conference) templates which were directly fed to the combiner component. These templates contained blank fields which had to be filled with some salient information from a *single* document. For example a template could contain the fields "victim" and "perpetrator", which would be filled in by the system in the process of summarization. The output was a paragraph of automatically generated natural language text.

The role of the content planner was to determine the information which should be included in the summary. A set of planning operators was provided and it was used by the content planner. The operators were: *change of perspective*, *contradiction*, *addition*, *refinement*, *agreement*, *superset*, *trend* and *no information*. Each one of these was essentially a manually written rule that linked two templates and as a result of that a third template was produced. For example if two news articles were contradicting each other

then a contradiction operator will be used on their corresponding templates. Thus a third template will be created which would contain the difference of the initial two.

In general the whole linguistic component was reused from the PLANDOC system (McKeown et al., 1995). It contained grammar rules and constraints which were applied to words to produce natural language. The lexical chooser managed the structure of each sentence by choosing appropriate words for each semantic role (McKeown and Radev, 1995). Finally the sentence generator produced natural language sentences by linearizing that syntactical structure.

The algorithm outlined by McKeown and Radev (1995) had several steps: *preprocessing*, *combination*, *discourse planning*, *format conversion*. First templates were sorted in chronological order. Then the templates were combined using any of the planning operators (contradiction, refinement, etc.) and the newly produced templates were sorted by priority. In the final step the sentence generator created the summary paragraph, which had variable length.

As part of a testing stage in McKeown and Radev (1995), SUMMONS was given manually produced templates but no formal evaluation was carried out as a part of this project.

In the next paper, Radev et al. (2003) proposed a new approach to multi–document summarization. It was called *centroid based summarization* and was implemented in the MEAD system. This approach used *clusters* which were created by grouping similar documents together. A document was determined to be part of an existing cluster if its vector of highest $tf.idf$ values was close to the vector of the *centroid* of that cluster. As we have seen already in Aone et al. (1999), the $tf.idf$ measure was used for single document summarization (see Section 2.1.2). Radev et al. (2003) proved that it can be used for multi–document summarization with the same success.

It should be made clear that the term "clustering" is used in the sense of grouping random document together by topic. Clusters are also used in the summarization process itself. This involves comparison between a sentence and the centroid of the cluster it is part of.

A *centroid* is defined by Radev et al. (2003) as "set of words that are statistically important to a cluster of documents". If we think of clusters as being a circular area of space the a centroid would be the centre of that area. And the border of that area is know as the *threshold*. It is important to note that a document D was only included in a cluster C if it was significantly similar to the documents in that cluster. Similarity between document D and cluster C was calculated by the cosine similarity measure.

Every cluster had a centroid which was represented by a list of $tf.idf$ values. A centroid contained only values above a certain threshold. The overall value of a centroid was equal

to the sum of these $tf.idf$ values. It was used along with two more parameters — *positional value* (P) and *first–sentence overlap* (F) — to give the final score for a sentence:

$$score(s_i) = w_c C_i + w_p P_i + w_f F_i \tag{5}$$

By assigning a score to each sentence, the process of summarization was simplified to just picking the first n sentences with the highest score. Various other scoring functions were also used: *position*, *centroid* and *overlap with first sentence*. There results showed that combining these three features in a single scoring function produced the best summaries. The formula for that function is shown below:

$$score(s_i) = C_i + 2P_i + F_i \tag{6}$$

In order to cope with redundancy, Radev et al. (2003) used an algorithm which performed redundancy checks on sentences. If a sentence was considered redundant (i.e. contained overlapping words with another sentence), it was "penalized" by subtracting a *redundancy penalty* R_s from the overall score for that sentence. This method was similar to Carbonell and Goldstein (1998)'s MMR (Maximal Marginal Relevance) but was modified to work with multiple documents.

The evaluation framework used by Radev et al. (2003) was innovative and effective. Two new techniques, *relative utility* and *information subsumption*, were introduced. Human judges were also present, and their job was to mark each sentence in a cluster with an importance score from 0 to 10 (0 being "unimportant" and 10 — "very important").

In other words the score for a sentence s_i depended on the centroid (C_i), position (P_i) and the first sentence overlap (F_i) features. In general the MEAD system performed well and the summaries produce contained informative sentences. The only drawback was that utility based evaluation required a significant amount of effort from the judges.

Saggion and Gaizauskas (2004) also had a system based on clusters. Their system used extraction techniques to build personal profiles from clusters of documents. The system was presented in the Document Understanding Conference in 2004 and took part in the competition(see Section 2.3.1.

There were two approaches covered by Saggion and Gaizauskas (2004) because the system was tested on two different tasks — 2 and 5 from DUC (2004). Although both approaches were based around the idea of clustering similar documents together, the second task required more knowledge about the domain than the first.

The features used in Saggion and Gaizauskas (2004) were *sentence cluster similarity*, *sentence lead–document similarity* and *absolute document position*. Documents were included in a cluster if the sentences in them were similar to the centroid of that cluster. A cosine similarity measure was used.

The goal of the system in Saggion and Gaizauskas (2004) was to create a profile of a person by extracting relevant information about that person from a set of documents. We have seen how McKeown et al. (1995) populated blank templates of attribute/value pairs with data from the terrorist domain. Here the goal was similar, but used a cluster of documents as the source. The personal profile was constructed from the information present in a list of *facets* (e.g. background, education, nationality etc.), which had to be populated by the system.

This final summary was created on the basis of the personal profile, once any redundancy was removed. This was achieved by calculating an *n–gram similarity* between text fragments (Saggion and Gaizauskas, 2004). The idea behind that was that if two fragments contained more than a certain number of identical text units (words), then one of those fragments was discarded as being redundant, and the other one was included in the summary.

The evaluation stage in Saggion and Gaizauskas (2004), involved both human judges and the use of automatic summary evaluation tools such as SEE and ROUGE (see Section 2.3). The results from SEE with respect to task 2 were very promising but the text quality was average. Unfortunately the results for task 5 were not as high as expected.

2.3 Evaluation Methods

All summarization systems discussed thus far produced summaries of different kinds. And for the past 40 years people have worked hard to create better summarizer. But how do we decide which system produces the best summaries? Comparing summaries is not straightforward since there are many factors (coherence, relevance etc.) that have impact on the quality of the summary. Evaluation is not only used for comparison, but mainly for understanding the advantages and disadvantages of a system and learning from them (Mani, 2001).

There is no universally established method for evaluation and research papers often use their own methods. Still we can distinguish between two types of evaluation methods: *extrinsic* and *intrinsic* (Mani, 2001, p. 223). An extrinsic evaluation is a test of the "usefulness" of a system. This could require some feedback from a third party like potential users of the system. Given the output of the system they are asked to perform a certain task (e.g. answer questions about a story) and then assessment is made on how good they perform that task. Thus the quality of the summary (the system output) can be inferred indirectly.

An intrinsic evaluation is more of a "quality" test for a system. This process of evaluation usually involves human judges who analyse the performance of a system. The evaluation itself could be a comparison between the summary and an ideal summary or

between the system and another system (Mani and Maybury, 1999). The problems with the first approach come from the fact that it is hard to define the "perfect" summary, because there can be more than one way of summarizing a particular document. This is especially the case with abstracts. Although it is easier for judges to agree on the set of most important sentences in a text, the evaluation of extracts is very much related to their compression rates. As for second approach, comparison is usually made between the target system and a *baseline system*. A baseline system is a system which is set to perform a simple summarization and is considered to have the worst performance. For example a baseline system could be one which extracts random sentences from a document. The goal is to make a system that performs better than the baseline.

We have seen examples of intrinsic evaluation in a number of papers. Saggion and Gaizauskas (2004) compared the results of their system to the results of a baseline system. Another intrinsic approach was used by Marcu (1995), where the final summary was judged by humans. Radev et al. (2003) proposed the *utility–based* evaluation method. It was a more fine–grained approach than the usual boolean judgements (Mani, 2001, p. 259). In other words instead of judging whether a sentence should be in a summary, or not, we assign to it a relative utility metric, which represents the degree of belief that a sentence should be included in the summary. Extrinsic evaluation was used by Mani and Bloedorn (2000) measured the usefulness of their system in the context of an information retrieval task.

2.3.1 DUC

The Document Understanding Conference is a major event for evaluation of both single document and multi–document summarization systems. It is sponsored by the Advanced Research and Development Activity (ARDA) and it is run by the National Institute of Standards and Technology (NIST) to further progress in summarization and enable researchers to participate in large-scale experiments (DUC, 2004). The purpose of this conference is to help researchers build better summarization system by evaluating them each year. The evaluation itself involves the use of manually produced summaries (provided by NIST) and sets of documents which are used as sources. There are also several tasks each year which have to be completed as part of the evaluation. DUC is not only a conference but also a competition. This encourages participants to improve their systems each year. The tasks given in DUC are similar each year and normally require the participants to produce automatic summaries from the source data provided. There are various restrictions on the length of the output summaries. Also there are tasks which ask the participants to produce summaries on a certain topic or event, or a query. DUC is an evaluation conference but it is also a source of valuable data: ideal human summaries, source documents and clusters of data for the purposes of multi–document summarization. This data is valuable because a lot of effort has been put into collecting it and it is very useful for testing summarization systems and improving them.

2.3.2 SEE and ROUGE

Both intrinsic and extrinsic evaluation methods involve the participation of human subjects. For intrinsic evaluations it is often the case that judges are required to compare an automatically generated summary to an ideal summary. This process requires a lot of effort and increases the expense of the evaluation. This is why several tools were developed to make the whole process easier. Such tools are SEE and ROUGE. The first one provides a standardized environment for summary evaluation. It is designed to help judges and maintain consistency of results. ROUGE is a fully automated evaluation system which takes as input pairs of auto–generated summaries and their corresponding ideal summaries. It determines the similarities between these summaries based on a number of features. This tool greatly simplifies the evaluation process and decreases its expense by eliminating the need for human subjects.

3 Requirements and Analysis

Mani (2001, p.13) describes the typical parameters for a summarization system: *compression rate* (fixed length, variable length), *audience* (user–focused, generic), *relation to source* (extract, abstract), *function* (indicative, informative), *span* (single or multi–document), *language* (monolingual, multilingual) and *genre* (genre specific, genre independent).

The goal in this project is to produce a summarization system which produces summaries by extraction and has the following parameters: multi–document, variable length, generic, indicative, monolingual, genre independent. The reason I have chosen to create a system which produces extracts is because it is easier to implement, robust and the summaries can be of decent quality. This is because no knowledge of the semantic structure of the text is required. The main disadvantage with using this technique is that is can yield an incoherent summary. In contrast, in order to produce abstracts, a summarizer has to have knowledge of the semantics of the source.

3.1 System design

The initial design of the summarization system is described below. I will explain the high–level design and then I will talk about each component and algorithm individually.

As I said the system in this project is a multi–document summarizer. This means that the input data will be in the form of clusters of documents with at least two documents in each cluster. Each of these documents is expected to be encoded either in XML (eXtensible Markup Language) or other markup language (e.g. DUC format).

Markup language is used because it is good for structuring data. It provides a description of a document in the form of metadata (data about the data). The metadata usually conveys sentence information like sentence number and position in paragraph. Also it can be used to explicitly define the beginning and the end of each sentence and paragraph using tags like `<sent></sent>` and `<par></par>`.

3.1.1 System Architecture

I propose a system which is made up of several components: *parser*, *tokenizer*, *clusterizer*, *sentence examiner*, *sentence selector* and *anaphor resolver*. The high–level architecture of the system is shown on the diagram below.

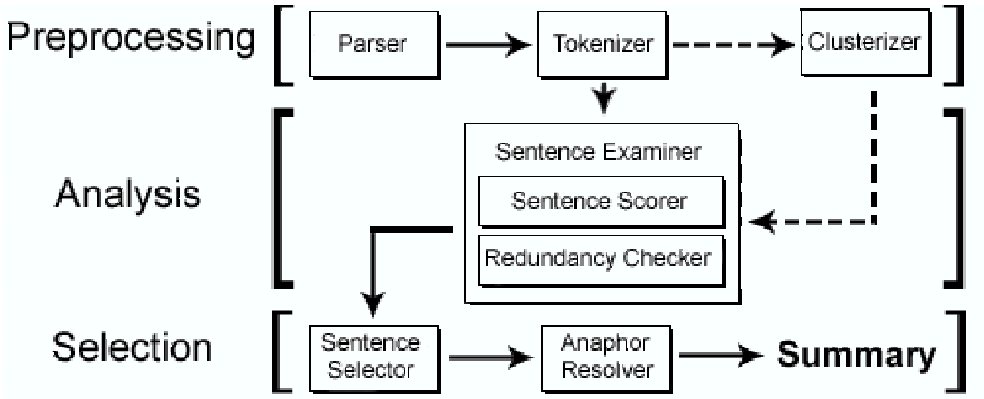

Figure 1: System flow diagram

The diagram above shows the different components of the system and the order in which they are used. There are three main stages of the summarization process: *preprocessing*, *analysis* and *selection*. In the first stage all input documents are parsed and tokenized. Stop words are also filtered using a stop list. It is assumed that the input documents are already organized into clusters beforehand. If this is not the case then the user can choose to enable the clusterizer component. The three components part of that stage are the parser, tokenizer and the optional clusterizer. In the second stage a cluster is selected and processed. Documents are analysed and the sentence scorer component assigns a score to each sentence in each document in the cluster. Then the redundancy checker compares documents and searches for redundant sentences. The third stage is selection. In this stage sentences are selected on the basis of compression rate and score. Finally the anaphor resolver component determine whether the selection of sentences is coherent (i.e. contains no unresolved anaphora) and if it is not, it tries to resolve any anaphora. Once the anaphor resolver has approved the selection, sentences are extracted and the final summary is produced.

3.1.2 Parser

The parser will be used for parsing the input documents. The output of the parser will be a parse tree which will represent the syntactic structure of the document being parsed.

3.1.3 Tokenizer

The tokenizer component will split a sentence up into words, so that each word can be processed individually. The tokenizer will also separate words from punctuation marks. For example the string "that's good." is converted into a list of five tokens:"that", " ' " "s", "good" and ".". Abbreviations like "U.N." and "U.S.S.R." are also recognized and treated as a single token. This also applies for decimal numbers like 1.234 for example. Common words like "the", "of", "and", "for", etc. will be filtered using a list of stop words.

3.1.4 Clusterizer

The clusterizer is the component which groups documents into clusters by topic. It is an implementation of the agglomerative clustering algorithm proposed by Radev et al. (1999). The first encountered document is always placed into a new cluster and all other documents can either be added to that cluster or form a new cluster. This decision depends on a similarity factor which will be explain later in more detail. At this level of processing each document is treated as a vector of words with their corresponding $tf.idf$ values. The output of the clusterizer is a set of clusters containing one or more relevant documents. The pseudo code shown below describes the actual algorithm :

Algorithm 1 DOCUMENT CLUSTERING

Require: Integers $n, m, N, M \geq 0$.

```
1: N = number of input documents
2: M = 0 (counts the number of clusters created)
3: for n = 1 to N do
4:     if M = 0 then
5:         create new cluster for document n
6:     end if
7:     for m = 1 to M do
8:         compare centroid of document n to centroid of cluster m
9:         if n is similar to m then
10:            add document n to cluster m
11:        end if
12:    end for
13:    if centroid n is not similar to any centroid m then
14:        create new cluster with document n in it
15:        m = m + 1
16:    end if
17: end for
```

Since both the document and the cluster centroid are represented as vectors of $tf.idf$ values we can compare them easily by using the cosine similarity measure. It is used for calculating the angle between two vectors. In this case the two vectors are the centroids of a document and a cluster. The size of these vectors, as well as the similarity threshold are global parameters which can be easily adjusted.

3.1.5 Sentence Examiner

The sentence examiner will consist of two sub components: *sentence scorer* and *redundancy checker* and will take one cluster of documents as input. The first will be used for assigning scores to each sentence and the second will modify these scores while checking documents

for redundancy. The score for each sentence is computed from the scores for each of the following features: *location*, *title overlap* and/or *first sentence overlap* and *centroid value*. Edmundson (1969) proved that the location feature gives good results because important sentences tend to be located at the beginning. By using the title overlap feature we assume that the author of the document has used an informative title. So if a sentence contains overlapping words with the title then it is a candidate for extraction. The first sentence overlap feature will be used either in combination with the title overlap feature or on its own (e.g. if the title is missing). The centroid value will be computed by the formula below (Radev et al., 2003):

$$C_n = \sum_w C_{w,n} \tag{7}$$

Here the centroid value C_n is represented as the sum of the $tf.idf$ values $C_{w,n}$ for the words in sentence n which are also present in the centroid for that cluster. This means that only words with high $tf.idf$ values will contribute to the centroid value for that sentence.

Another way of computing the centroid value is by comparing a sentence to the centroid of a cluster. The centroid of a cluster is shown graphically in Figure 2 below. It is a vector in multi–dimensional space. This is similar to the clustering technique explained earlier where every document is a new dimension in the vector space. Here every sentence is a different dimension in that space. The proximity of a sentence to the centroid is measured by the angle θ between the two vectors. If that angle is small then the sentence is close to the centroid and gets a higher score. In other words the closer a sentence is to the centroid the higher the centroid value will be. So by looking at Figure 2 we can see that sentences 1 and 2 will get the highest centroid values.

When the scores for all features are combined we get the total score for a sentence S_n:

$$score(S_n) = w_c C_n + w_l L_n + w_t T_n + w_f F_n \tag{8}$$

The weights w_c, w_l, w_t and w_f may be learned automatically by using machine learning techniques or they may be adjusted manually. The sentence scorer uses the formula above to assign a score to each sentence and then the redundancy checker which modifies each of these scores. The following pseudo–code shows how the sentence scorer works:

3.1.6 Redundancy checker

The redundancy checker will iterate through each document starting from the second one and check whether any of the sentences are similar to any of the sentences in the first document. If more than 50% of the words in a sentence from the first document are present in a sentence from the second document, then the latter is considered to be redundant. Redundancy in the final summary can be bad but in general it can be an indicator for salient information. For example if a documents A and B have very similar sentences then this means that these sentences are important because they are mentioned in both

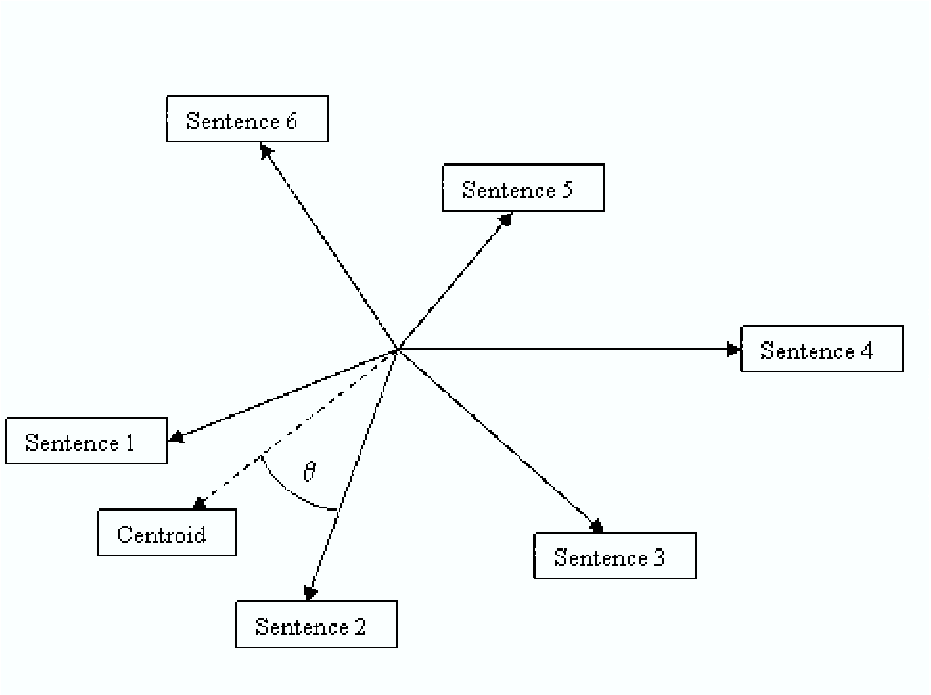

Figure 2: Conceptual centroid representation

Algorithm 2 SENTENCE SCORING

Require: Integers $n, s, N_c, S_n \geq 0$.

1: N_c = number of documents in the cluster
2: S_n = number of sentences in the n-th document of the cluster
3: **for** $n = 1$ to N_c **do**
4: **for** $s = 1$ to S_n **do**
5: $s(n) = score(s)$
6: **end for**
7: **end for**

documents. The redundancy checker will be aware of that and if it finds a redundant sentence in document B, say, then it will subtract a certain amount $P.score(B)$ from B's score and at the same time add $P.score(B)$ to the score of A. Here the variable P is the percentage of *redundancy penalty*. Thus the final version of the equation for calculating the total score of a sentence S_n becomes:

$$score(S_n) = w_c C_n + w_l L_n + w_f F_n - w_R R_s \tag{9}$$

The algorithm described below takes the first document that was added to the cluster and compares the rest to it. This method is almost identical to the one used by Radev et al. (2003) with the only difference being that the redundancy penalty is fixed instead of being dynamic.

Algorithm 3 REDUNDANCY CHECK

Require: Integers $n, s, s_1, N_c, S, S_1 \geq 0$ and $0 \leq P \leq 100$.

```
1: N_c = number of documents in a cluster
2: S_1 = number of sentences in the first document of the cluster
3: S = number of sentences in current document
4: for n = 2 to N_c do
5:     for s = 1 to S do
6:         for s_1 = 1 to S_1 do
7:             if sentence s in document n is similar to sentence s_1 in the first document
               then
8:                 score(s_1) = score(s_1) + Pscore(s)
9:                 score(s) = score(s) − Pscore(s)
10:            end if
11:        end for
12:    end for
13: end for
```

3.1.7 Sentence Selector

The sentence selector is going to select a set of relevant sentences from the source documents. This set will form the basis of the final summary. Only the top $N_c.R$ sentences are selected, where N_c is the number of sentences in the cluster and R is the *relative compression rate*. For example if there are 100 sentences in a cluster and the user wants a 10% summary we extract the first 10 sentences with the highest score.

3.1.8 Anaphor resolver

The anaphor resolver is the component which will deal with the coherence of the summary. It will use a shallow approach to the anaphora resolution problem. Its role is to find pronouns like "he", "she", "they" and find the term which they refer to. If a sentence begins with a pronoun like "he", say, we can either replace it with the term it refers to, or if that term is not found the system will choose to include the previous sentence. In order to find the antecedent term the resolver will search for the nearest noun phrase or named entity. If an anaphor cannot be resolved then the previous sentence is selected for extraction.

3.2 System Evaluation

The evaluation for the system will be intrinsic. This will be done by comparing the system to a baseline system, as well as using the evaluation package ROUGE to compare the produced summaries with ideal summaries. Additional evaluation might be carried out by using relative utility metrics as proposed by Radev et al. (2003). This approach provides a more fine–grained assessment of the system (see Section 2.3). Each sentence will be

manually assigned a utility metric (score) which will describe how relevant a sentence is. This score is usually in the range from 1 to 10 with 1 being "irrelevant" and 10 being "very relevant". Thus the produced summary will be evaluated by summing up the utility scores for each sentence in it. The same thing will be done for an ideal summary and the two results will be compared.

The system will be tested using DUC (2004) data only. This means that both clusters and model summaries will be taken from that data set. I have also decided to experiment with news wire RSS feeds as a source of input data. RSS (Really Simple Syndication) feeds are essentially XML documents which are widely used online for information distribution. The clusterizer component would very useful in this case where we have many unsorted (not yet clustered) documents coming into the system.

In general the system is meant to be genre independent but I believe that it will perform better in the news article domain. The reason for that is the sentence features used. Sentence location for example is used as a factor for increasing or decreasing the score of a sentence depending on its position in the document. If a sentence is near the beginning of the document it is regarded as more important. As we know, news articles tend to use the "pyramidal" structure for telling a story — the most salient information is presented at the beginning.

4 Implementation and Testing

By having a complete design of the system, the implementation process was just a matter of following that design. Each component of the system was developed in the order of use. Individual testing of each component developed was essential before moving on to the next one. Testing was done on DUC 2004 data and in particular the *d30002t* cluster from the TDT (Topic Detection and Tracking) collection which contains news articles about hurricane Mitch.

Some parts of the system were not implemented according to the design. This was because at the early stage of design some details about how the system should work were not yet clear. This is the case with anaphora resolution and redundancy checking. I will discuss those changes in detail later.

The system was implemented in the Perl programming language and tested on samples from the DUC (2004) data. The reason for choosing Perl was its text processing power and regular expression capabilities. It is also very good for rapid development which is particularly important in this project.

4.1 Parsing

The parser that was implemented can only parse DUC data files. This should not be considered as a restriction because it is really easy to include additional modules to the system. At the beginning of the project I planned to implement an additional RSS parser. The RSS parser was meant to be used for parsing news feed taken straight from the Web. This would have enabled the system to work "in the wild", making it a more useful application. Due to time limitations this feature was not implemented but is a reasonable extension to the system.

4.1.1 DUC File Format

DUC data files have a specific format with markup that resembles XML. An example of the structure of a DUC file is shown below:

```
<DOC>
 <DOCNO> APW19981124.0267 </DOCNO>
 <DOCTYPE> NEWS </DOCTYPE>
 <TXTTYPE> NEWSWIRE </TXTTYPE>
 <TEXT> ...  </TEXT>
</DOC>
```

Since the format of the files used in DUC was very simple and not really XML I chose not to use the modules supplied with Perl for parsing XML. Instead I have implemented a

simple parser to extract information between tags and in particular the <DOCNO> tag which contains the number of the document and the <TEXT> tag which contains the contents of the document. Other tags like <DOCTYPE> and <TXTTYPE> are ignored because they are of no importance to the system.

The text between <DOCNO> tags is used as identification of the document and also contains the date it was published. Before a document is parsed it is first validated by a syntax checker. It checks whether each opening tag has an associated closing tag. If the markup is valid the contents of the document are extracted by the parser and passed on to the tokenizer for tokenization.

4.2 Tokenization

Tokenization is the operation of splitting up a string of characters into a sequence of tokens. Tokens are the smallest units in text. A token is usually a word or punctuation mark. Each punctuation mark is treated as a separate token but such tokens are usually filtered (see next section). Apart from that we might also want to filter out frequent words known as *stop words*. Once a string of characters is tokenized we can deal with each token individually, i.e. analyze it, classify it or discard it. Without tokenization it would be impossible to process text in electronic form.

Tokens containing the full stop character "." are of particular importance because they are used in sentence splitting. These tokens are also filtered but before that they are used to separate sentences from each other. Some summarization systems use data files which mark separate sentence using special tags like <SENT></SENT> but these are not used here. The tokenizer assumes that each sentence ends with full stop and uses that as a sentence delimiter. It is the same as using white space as delimiter between words but here the full stop acts as a delimiter between sentences. Abbreviations like "Mon." and "Nov." are recognized by the tokenizer by consulting a dictionary file. This process prevents the tokenizer from prematurely splitting a sentence at the position where the abbreviation is.

Tokenization is not just a matter of treating all strings separated by white space as tokens. This would be the simplest approach but it would cause problems. For example if we treat "and," as a token the system would not recognize it as a stop word simply because it does not appear in the stop list. Instead we need to remove any punctuation which is found immediately after or within a word. Thus for example when "that's" is tokenized the output would be not one but three tokens: "that", " ' " and "s". By doing this sort of operation the system is able to analyse words correctly and more precisely.

The tokenizer in the project from previous year, done by David Jarzebowski, was implemented as a finite state machine. It had six states according to each type of input: alpha, numeric, etc. This approach is very clear and works very well for any type of text. I have decided to improve it and re–implement it so it becomes a more efficient and robust tokenizer. Now the new tokenizer recognizes abbreviations like "U.N." and "U.S.S.R.",

as well as decimal numbers (1.234) or numbers like 1,000,000 (one million). These are all treated as a single token because it would be unreasonable to decompose them further.

4.3 Stop Words

Stop words are words that are used very frequently in text but are not useful for text processing. These words belong to closed classes of words like determiners (the, a), prepositions ("at", "by", "with"), conjunctions ("and", "or"), etc. These words are filtered by the system using a *stop list*. A stop list is simply a text file containing most stop words in the English language.

4.4 Clustering

Multi–document summarization is only possible when the input documents are on the same topic. Grouping documents together by topic (or other criteria) is called clustering. The process can be done manually by humans by selecting documents (articles, reviews, etc.) on the same topic and putting them into clusters. So for example if we have five documents about the war in Iraq, and three documents about 9/11 events, clustering them would produce two clusters. This process although effective is not efficient as it requires a lot of human effort. Automatic clustering is possible but it is not as effective as manual clustering. This means that such systems often make mistakes and categorize documents incorrectly. The goal is to minimize the number of those mistakes and to have a large percentage of correctly categorized documents.

4.4.1 Algorithms

The clustering algorithm described in Section 3.1 is a type of agglomerative clustering and is described by Radev et al. (1999). The first encountered document is always placed into a new cluster and all other documents can either be added to that cluster or form a new cluster. The opposite of agglomerative clustering is divisive clustering where the process starts by placing all documents in one cluster which eventually breaks up into smaller clusters. Another popular algorithm is k-means clustering algorithm which is simple and quick which allows it to run on large data sets. Its disadvantage is that it does not yield the same result with each run, since the resulting clusters depend on the initial random assignments (Wikipedia, 2006). See Beeferman and Berger (2000) for more details about agglomerative clustering.

The clustering component of our system works by comparing centroids. Each document has its own centroid consisting of words with high $tf.idf$ values. Similarly every cluster has a centroid of words with averaged $tf.idf$ values. When the system runs for the first time it takes one document and puts it into a new cluster. At this point the centroid of the cluster is identical to the centroid of the only document in it. Later on a new document is picked and it is compared to the existing cluster. This is a matter of comparing the two centroids

which are of the same size. If the new document's centroid is similar to the cluster's then the $tf.idf$ values from the document's centroid are added to the centroid cluster and averaged. Each centroid is implemented as a hash in Perl. A hash map contains key–value pairs where keys must be unique and values are obtained from those keys. The keys of the centroid hash are words from the document and the values are their corresponding $tf.idfs$.

4.4.2 Parameters

There are two parameters that control the clustering component: *similarity threshold* and *centroid size*. The similarity threshold was set to 0.1. This means that if the centroid of an existing cluster has more than one overlapping word with the centroid of some document, then that document is included in the cluster. Setting this parameter too low will result in grouping many documents into fewer clusters, and setting it too high will put each document into its own cluster. The centroid size was set to 15. This indicates the number of words that each centroid can contain. These are only words with the highest $tf.idf$ values. A larger centroid size is better because there can be more than ten keywords for a document and this allows most of them to be included in the centroid.

4.4.3 Efficiency

The efficiency of an agglomerative clustering algorithm greatly depends on the similarity threshold chosen. If that threshold is set too high the result will be a large number of clusters with one or two documents in them and the worst case scenario will be to have a separate cluster for each document in the collection. Setting the threshold too low will result in having fewer clusters with more documents in them or, in the worst case, one cluster with all documents in it. This is why it is reasonable to try and automatically "learn" what the optimal threshold value would be. This could be achieved by using machine learning techniques. Unfortunately I was not able to incorporate machine learning into my clusterizer due to my lack of knowledge in this area and the time available for the project.

The clustering component is optional. It was developed just to make it possible for the system to work with unsorted collections of data (i.e. documents that have not yet been clustered). The summarizer works well without it, provided that the user supplies one or more clusters of documents.

4.5 Sentence Scoring

Since the summarizer works by extraction the final summary is produced by selecting a certain number of sentences from a pool of sentences. The selected sentences are ranked according to their relevance. If a sentence is very relevant it gets a high score and is selected. For each sentence stop words are filtered first before it is passed on to the scoring component. Scores are calculated depending on several factors (see Equation 8, Section

3.1.5). The total score for a sentence is the combination of these factors and determines just how relevant a sentence is.

4.5.1 Parameters

There are three parameters that control the scoring function in the system — *location weight*, *centroid value weight* and *first sentence overlap weight*. These weights are currently set to 1 which means that all features (location, centroid value, etc.) are equally important factors in the final score for a sentence. These weights can be modified manually so that optimal performance is achieved. But the best way to do this is to use machine learning algorithms to automatically adjust those weights for us. However, this feature of the system has not been implemented.

4.5.2 Centroid Value (Algorithm 1)

For the project I have implemented two different algorithms for computing the centroid value. The first one was originally created by Radev et al. (2003). When each document is processed a centroid is created for it containing the terms with highest $tf.idf$ values. This centroid is different from the centroid of the cluster because the latter contains the averaged $tf.idf$ values for the most relevant terms in the collection.

The centroid in Radev et al. (2003) value is calculated by summing up the $tf.idf$ values for all terms in the sentence (see Equation 7, Section 3.1.5). $Tf.idf$ values are pre–calculated only for words that do not appear in the stop list.

4.5.3 Centroid Value (Algorithm 2)

The second version of the algorithm for computing the centroid value indicates how far the sentence is from the centroid of the cluster. If a sentence is close to the centroid then it regarded as more relevant than a sentence that is far from the centroid. As described in Section 3.1.5 the centroid of a cluster is a vector in a multi–dimensional space where every document/sentence vector is different dimension.

In order to determine how close a sentence vector is to the centroid we need to calculate the cosine similarity between the two. In other words we have to find the angle between the two vectors.

4.5.4 Location Value

The location of a sentence in text is a good indication of its relevance which was first noticed by Edmundson (1969). News articles have pyramidal structure where the most important information is presented at the very beginning. Other documents like stories or scientific papers have different structure and may contain salient information in different portions of the document — *beginning, middle* or *end*. Since our system works primarily with news

articles it is only interested in sentences positioned at the start of the article. The first sentence of the article is given the highest score and that score decreases proportionally to the sentence number.

Location value is calculated from the formula below, where N is the total number of documents in the collection and n is the number of the sentence which is currently processed:

$$L_n = \frac{(N-n)}{N}100 \tag{10}$$

4.5.5 First Sentence Overlap Value

Normally every document has a title describing what the document is all about. This is why titles are so valuable to summarization. But when a title is absent one can use the first sentence instead. This is the case with DUC data files where titles are usually missing. The first sentence is also very relevant and as I mentioned earlier that is especially true for news articles.

The first sentence overlap value for a given sentence is calculated by comparing that sentence to the first one in the document. Stop words are again filtered first. Then both sentences are converted to vectors of terms and their term frequency (tf) values. Finally the *dot product* between the two vectors is calculated and that is the first sentence overlap value. This algorithm was is described in Radev et al. (2003).

4.5.6 Title Value

The title value T_n appears in equation 8 but is not used in the system. It can be calculated using the same algorithm described above for the first sentence overlap with no modifications required to the actual source code.

4.6 Redundancy Elimination

Creating summaries from many documents on the same topic requires some sort of redundancy elimination. When there are different sources of the same information it is very likely that the summary will contain redundant information. Redundancy checking is not an easy task and requires a lot of processing. For this project I came up with one algorithm for the task which is described in Section 3.1 above. At first it was not very clear how it was going to be implemented and there are some differences between the design and the actual implementation described below. After discussing this algorithm with my supervisor Dr. Rob Gaizauskas , I have decided to implement another algorithm for redundancy elimination. The second algorithm was taken from the Saggion and Gaizauskas (2004) paper.

4.6.1 Algorithm 1

The first algorithm is my own design and has a complexity of order at most n^2 where n is the number of documents in the collection. This is because it compares every sentence of one document to every sentence of every other document in the cluster. First the system has to decide which document to choose as the first one and then the rest will be compared to that first one. The criterion is the size of the document so that the largest document will be taken as first. By choosing the largest document there is a higher chance of it containing most of the relevant information.

The redundancy checker component is called after the sentence scorer finishes its job and there is an existing hash map of sentences and their scores. It works by taking that hash map and modifying the scores accordingly. Score modifications are controlled by two parameters — *similarity threshold* and *redundancy penalty*. When comparing sentences the system checks if they are similar using the cosine similarity measure. If two sentences are similar enough, i.e. similarity is above the threshold, then the one with the higher score in the hash map will get a percentage of the marks of the other. That percentage is the redundancy penalty and is currently set to 50%. This method ensures that the final summary contains sentences not only from the first document (the largest) but also from other documents which contain sentences similar to the ones in it. Also if an important sentence is found in some document and the largest document does not contain a similar sentences then that sentence will not get any additional marks and will have a lower chance of entering the summary. This is one of the drawbacks of this algorithm — only sentences that are redundant will get additional marks from other sentences that have been penalized. The main idea behind this method is that redundant sentences are also the ones that are most relevant. This is because in news articles the main event is described in each article with one or more sentences. For example if the news is "G. W. Bush visited Iraq" then it is very likely that other documents contain sentences with both "G. W. Bush" and "Iraq" in them. This makes them redundant but also relevant.

4.6.2 Algorithm 2

The second algorithm was developed by Saggion and Gaizauskas (2004) as part of a summarization system participating in DUC (2004). They take a different approach to redundancy elimination by pre–computing N–gram overlaps between text units. Again we have to choose two documents for comparison and compare every sentence from the first to every sentence from the second. The text fragments that we are going to compare are words but they might also be larger — sentences or paragraphs. The similarity metric used is the N–gram similarity and it is calculated using the equation below:

$$NGramSim(T_1, T_2, n) = \sum_{k=1}^{n} w_k \frac{|grams(T_1, k) \cap grams(T_2, k)|}{|grams(T_1, k) \cup grams(T_2, k)|} \tag{11}$$

Here the two fragments T_1 and T_2 can be *uni–grams*, *bi–grams*, *tri–grams* or *k–grams*,

depending on k and the maximum is n which is set to 4. The weight w_k is set to the arithmetic series $w_1 = 0.1, w_2 = 0.2, w_3 = 0.3$ and $w_4 = 0.4$. Two text fragments are considered similar if the n–gram similarity, $NGramSim(T_1, T_2, n)$ is greater or equal to α, where α is the similarity threshold with 0.1 as its initial value.

4.6.3 Comparison

Both algorithms work well but have their advantages and disadvantages. The first algorithm is my own design and has the advantage of being easy to implement. The main disadvantage is that it is not very efficient. The second algorithm is harder to implement because it calculates set intersections and unions. It is considerably faster though. The results from the ROUGE evaluation are shown in Section 5.

The first algorithm has another disadvantage. The only sentences that will get additional marks from other sentences are redundant sentences. This means that if there exist any unique relevant sentences that are not similar to any other sentences then they will have a lower chance of being selected for extraction. This is not an issue in the second algorithm. The drawback there is that n–gram sets have to be pre–computed for all sentences before they are compared.

4.7 Selection

The final summary is produced by selecting the sentences with highest rank. The process of selection is straightforward once we have calculated the total score for each sentence. A table of sentences and their scores is created by the system and is represented internally as a hash map. In our case the key is the sentence string itself and the value is the total score for the sentence. Before the actual selection takes place that table needs to be sorted. Sorting must be done by values so that we have sentences with highest scores at the top.

The actual selection is governed by a single parameter and that is the *compression rate* (summary length). If the rate is 2% and there are 100 sentences in the collection only the first two with the highest scores will be selected.

4.8 Anaphora Resolution

Anaphora resolution is a very complex task, but without applying it automatically generated summaries may be incoherent and of poor quality. The anaphor resolver implemented as part of this project uses a very simple approach to the problem.

The anaphor resolver component works in two stages: *named entity recognition* and *anaphora resolution*. In the first stage the system searches every document for named entities. A named entity is used to identify a particular person, place or event. Names can easily be recognized by the system with the help of a dictionary. Before a document is

scanned for named entities, all words that are found in the dictionary are discarded. The result of this operation is a list of words that do not appear in the dictionary and which are assumed to be named entities. Later each of these words is checked if it starts with uppercase character. This is done because in the English language all names begin with a capital letter. The main disadvantage with this approach is that if a sentence starts with a common word like "Church", referring to some person, then that will not be recognized as a name because "church" was found in the dictionary. This problem can be avoided by treating every word starting with uppercase to be a name. But then, again, this will cause a bigger problem — the first word from every sentence will be treated as a named entity, which is simply incorrect.

A dictionary file containing about 2500 terms was used in the project. The dictionary was created by combining several WordNet dictionaries containing nouns, verbs, adjectives etc. This file is loaded each time the system starts. Although the dictionary was not extensive enough the results of using such approach were positive.

Once all named entities have been recognized the system take each sentence that contains a named entity and looks at the next sentence. Every pronominal anaphor ("he", "this", "it", etc.) in the sentence to follow is replaced with the named entity that was found in the previous sentence. If there are more than one named entities in the previous sentence then the system will choose the one that is closest to the anaphor. The result of this simple and computationally inexpensive technique is a summary with higher levels of coherence.

5 Results and Discussion

The summarization system developed for this project was evaluated intrinsically and incrementally. Components were tested as they were developed and then the summaries produced by the system were evaluated with each new component put to work. The core components that are required for producing a summary are the parser, tokenizer, sentence scorer and sentence selector. Optional components are the clusterizer, redundancy checker and the anaphor resolver. The system was evaluated first with just the core components in use. Later on in the evaluation process the optional parts of the system were switched on and the summaries were re–evaluated.

Two baseline systems were developed for the purposes of evaluation. This is common practice in software evaluation because the only way to tell if one system is better than another is to compare both. Baseline systems are easy to implement and are designed to produce summaries which are generated using the simplest selection techniques — *random* and *first sentence selection*. The first system produces summaries by selecting random sentences from random documents in a cluster each time it is run. The second baseline system generates summaries by selecting the first sentence from each document in a cluster. If a cluster contains five documents and the user wants a three sentence summary the system will only extract the first sentence from the first three documents. Otherwise if the user wants six sentence summary, for example, the system will extract all first sentences from five documents and will also extract the second sentence from the first document.

5.1 ROUGE results

The results from the ROUGE evaluation system were collected for three different compression rates — 2%, 5% and 10%. Also the whole system was re–evaluated each time a new component was introduced and results were recorded. To make the evaluation easier helping scripts were created in Perl. ROUGE configuration files were also created in XML. These files contain the paths to model summaries among other system details. The model summaries used during evaluation were taken from DUC (2004) data.

The tables below show the ROUGE–L scores for the three different compression rates. ROUGE–L is the default measure computed by ROUGE. It indicates the longest common subsequence (LCS) of words that appear in two sentences. In this case one of these sentences appears in the summary generated by the system and the other is taken from a model summary which was supplied with the DUC (2004) data.

The tables contain results for both the baseline systems and the system developed for the project. The ROUGE score is given to five decimal places where the higher score is the better.

System	2% Summary	5% Summary	10% Summary
Lead Sentence	0.18273	0.31275	0.38454
Random Sentence	0.14809	0.28163	0.41767
Project	0.18223	0.33434	0.42169

Table 1: Project system running without redundancy elimination (centroid value algorithm 1)

System	2% Summary	5% Summary	10% Summary
Lead Sentence	0.18273	0.31275	0.38454
Random Sentence	0.11747	0.26807	0.39809
Project	0.12902	0.28112	0.39458

Table 2: Project system running with redundancy elimination (algorithm 1) and centroid value algorithm 1

System	2% Summary	5% Summary	10% Summary
Lead Sentence	0.18273	0.31275	0.38454
Random Sentence	0.11044	0.27811	0.39408
Project	0.15361	0.31978	0.42219

Table 3: Project system running with redundancy elimination (algorithm 2) centroid value algorithm 1

System	2% Summary	5% Summary	10% Summary
Lead Sentence	0.10793	0.26004	0.37550
Random Sentence	0.14106	0.27962	0.36998
Project	0.15763	0.30371	0.38604

Table 4: Project system running without redundancy elimination (centroid value algorithm 2)

System	2% Summary	5% Summary	10% Summary
Lead Sentence	0.10793	0.26004	0.37550
Random Sentence	0.09187	0.26355	0.41014
Project	0.16466	0.33434	0.40713

Table 5: Project system running with redundancy elimination (algorithm 1) and centroid value algorithm 2

System	2% Summary	5% Summary	10% Summary
Lead Sentence	0.10793	0.26004	0.37550
Random Sentence	0.11847	0.26857	0.39709
Project	0.16717	0.32179	0.40562

Table 6: Project system running with redundancy elimination (algorithm 2) and centroid value algorithm 2

5.2 Findings

The main thing to notice from the results above is that the main system ("Project") performs better than both baseline systems in five out of six evaluations. In one evaluation it scores lower than the lead sentence system — see Table 2. Another thing to mention is the tight interval of the scores. The main system has scores which are within 0.1 of the baseline scores in most cases. This is an indication that the summaries generated by the main system are almost as good as those produced by the lead sentence system.

The results above are not surprising because extracting first sentences from news articles is guaranteed to produce a very good summary. This is because such documents begin with the most relevant information first. This fact makes it very hard for the main system to outperform the baseline.

It can be seen that the first redundancy algorithm gives poor results and the second algorithm by Saggion and Gaizauskas (2004) has almost identical score to the lead sentence baseline. The best results are given by the system running without the redundancy checker and using the first algorithm for computing the centroid value (see Table 1). The second best result was produce by the system in Table 5. When compared to the result from Table 3 where the only difference is the centroid value algorithm, we can see that the algorithm described in Section 4.5.3 performs better. Note that the scores for the lead sentence baseline system are the same throughout Tables 1 to 3 and Table 4 to 6, because redundancy elimination and sentence features are only applied to the main system.

In comparison to other multi–document summarization systems, our system gives very satisfactory results. Table 7 below shows a comparison between the summarizer produced here and another system by Saggion and Gaizauskas (2004). The results shown below are for the 5% summary with redundancy elimination taken from Table 5.

System	**Project**	**Saggion and Gaizauskas (2004)**
Baseline	0.26004	0.3459
Main	0.33434	0.3744

Table 7: System comparison

Probably the most important finding from the results above is that for short summaries, less than 5%, it is better to use the lead sentence summarizer and for longer summaries it is more appropriate to use the main system. In terms of speed and efficiency the system that performed best was the lead sentence summarizer and the main system running without redundancy checking.

6 Conclusions

The summarization system produced as part of this project has met the requirements described in the introduction, and also the goal of this project has been achieved. The function of the summarization system is to take a set of documents in electronic form, organize them by topic (if it has not been done already) and automatically summarize them. Conclusions as well as description of the features of the system are mentioned below.

6.1 Summarization system

The main goal of the project was to produce a multi–document extractive summarizer. This was completed and the development process was described in previous sections. A multi–document summarizer is a system that takes a set of documents on the same topic — a cluster — as input and generates a summary as output. This process is automatic and is controlled by a number of parameters and features. The features of the system in this project were chosen to be the *location value, centroid value* and *first sentence overlap value.* The values of these features are calculated for each sentence and their sum constitutes the total score for a given sentence in a document. Every sentence in the cluster has to be scored first before it is selected for extraction.

The summarizer in this project extracts information from documents and the result is a *generic, variable length, indicative* summary. These parameters are used to describe the type of summary and its purpose. A generic summary, as opposed to *user–focused* summary, is used when the output of the system is targeted at the general public and not a specific user with specific interests. A summary of variable length means that system users are allowed to specify the *compression rate* or, in other words, how much information from the source text will be included. Finally, the summary is indicative because its purpose is to help the user decide whether to read the full source text, or not. *Informative* summaries are the opposite of indicative summaries and their purpose is to provide as much information to the user as possible.

Additional features of the system were also implemented, although they were not required. These include *anaphora resolution* and *redundancy removal.* The anaphor resolver is a component of the system which is able to resolve pronominal anaphora like “he”, “she”, “they”, etc. This component uses a simplistic shallow approach to the problem and produced satisfactory results. The redundancy checker, on the other hand, involves some complex algorithms, but the improvements to the quality of the final summary were insignificant.

6.2 Performance

In five out of six evaluations the main system performed better than both lead sentence and random sentence baseline systems (see Section 5.1). At the beginning of development, when the first working version of the summarizer was evaluated, the results were equal to, or worse than the results from the baseline systems. These final results were achieved through constant evaluations and modifications of the system. Each time a new feature or algorithm was implemented, the system was tested and evaluated. Then the system was modified, parameters were optimized and the process was repeated.

System performance was measured using the ROUGE evaluation system. This is a program for automatic evaluation of computer generated summaries and is used in the Document Understanding Conference (See Sections 2.3.1, 2.3.2). ROUGE measures how similar two summaries are — a model and a generated summary. There are several different types of evaluation that are available from ROUGE, but the one that was used in the project is ROUGE–L. This measures the number of words that appear in a particular order in both summaries, also known as the longest common subsequence (LCS).

The results from the evaluation of the system showed that for short summaries, less than 5%, the leading sentence baseline system gives better ROUGE scores than the full featured system. This fact is not surprising because the most relevant information is usually contained in the first sentence(s) of the document, and thus the lead sentence system is guaranteed to give good results.

The additional components of the system — redundancy checker and anaphor resolver — did not greatly improve ROUGE scores. On the contrary, with both components working, the main system's scores were a bit lower than when these components were not used. These findings, although unexpected, suggested that these additional components can be turned off for the sake of efficiency.

6.3 Goals Achieved

The main goal set at the beginning of this document was to produce an automatic multi–document summarization system that performs better than baseline systems. This goal was clearly achieved. Although the system produced works better for summaries larger than 5% the results proved that the output from it is of reasonable quality.

Apart from completing the main objective of the project, I had the opportunity to experiment with different algorithms and approaches to the problem of redundancy elimination. Two different algorithms were implemented successfully and comparison between the two was made to decide which gives better results.

Another important achievement is the addition of the anaphora resolution feature to

the system. This feature has not been implemented in the previous project on the same subject. Although a very quick and simplistic implementation, the anaphor resolver has been extensively tested and works reasonably well for most named entities. The method correctly resolves about 40% of the encountered anaphora which is fairly good performance for a shallow approach like this.

6.4 Further Work

Finally I would like to point out that there were features that were desirable but were left out due to time restrictions. These include the use of machine learning techniques for automatic weight adjustments as well as extrinsic evaluations of all summaries produced. By using machine learning the system will be able to automatically find the optimal values for its weights. This would optimize the weights used in Equation 8 and give more accurate scores for sentences. This will affect the selection of sentences in a positive way. Hence the quality of the final summary will be improved. And that can be evaluated better using extrinsic evaluation system.

The summarizer developed here is far from being a useful application for the end user. At the moment the system can only handle DUC data files and the user interface is text based. It would be nice to have additional parsers developed for different file formats like RSS for example. Combined with Internet connectivity and automatic clustering the system will then be able to retrieve news from the Web, group them together by topic, automatically summarize them and present them to the user. A graphical user interface is also a nice feature but at the moment it is just unnecessary.

6.5 Final Remarks

Automatic summarization is very practical and interesting discipline. It requires understanding of text processing, statistics and artificial intelligence (Mani, 2001). The idea of having a machine that produces summaries from natural language is simple but the realization of it is not straightforward. The overload of information is becoming a real problem nowadays and this is the reason why further advances in the automatic summarization are required.

7 References

Aone, C., Okurowski, M. E., Gorlinsky, J., and Larsen, B. (1999). A Trainable Summarizer with Knowledge Acquired from Robust NLP Techniques. In Mani, I. and Maybury, M. T., editors, *Advances in Automatic Text Summarization*, pages 71–80. The MIT Press.

Barzilay, R. and Elhadad, M. (1999). Using Lexical Chains for Text Summarization. In Mani, I. and Maybury, M. T., editors, *Advances in Automatic Text Summarization*, pages 111–121. The MIT Press.

Beeferman, D. and Berger, A. (2000). Agglomerative clustering of a search engine query log. In *Knowledge Discovery and Data Mining*, pages 407–416.

Boguraev, B. and Kennedy, C. (1997). Salience-Based Content Characterization of Text Documents. In Mani, I. and Maybury, M. T., editors, *Advances in Automatic Text Summarization*, pages 99–110. The MIT Press.

Carbonell, J. G. and Goldstein, J. (1998). The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries. In Moffat, A. and Zobel, J., editors, *SIGIR98*, pages 335–336, Melbourne, Australia.

DUC (2001-2004). The document understanding conference
http : *//duc.nist.gov*.

Edmundson, H. P. (1969). New Methods in Automatic Extracting. *Journal of the Association for Computing Machinery*, 16(2):264–285.

Hovy, E. and Lin, C. Y. (1999). Automated Text Summarization in SUMMARIST. In Mani, I. and Maybury, M. T., editors, *Advances in Automatic Text Summarization*, pages 81–94. The MIT Press.

Kupiec, J., Pedersen, J. O., and Chen, F. (1995). A Trainable Document Summarizer. In Mani, I. and Maybury, M. T., editors, *Advances in Automatic Text Summarization*, pages 55–70. The MIT Press.

Lehnert, W. G. (1981). Plot Units and Narrative Summarization. In Mani, I. and Maybury, M. T., editors, *Advances in Automatic Text Summarization*, pages 178–214. The MIT Press.

Luhn, H. P. (1958). The Automatic Creation of Literature Abstracts. *IBM Journal of Research Development*, 2(2):159–165.

Mani, I. (2001). *Automatic Summarization*. John Benjamins Publishing Company, Amsterdam/Philadephia.

Mani, I. and Bloedorn, E. (2000). Summarizing Similarities and Differences Among Related Documents. In Mani, I. and Maybury, M. T., editors, *Advances in Automatic Text Summarization*, pages 357–379. MIT Press.

Mani, I. and Maybury, M. T., editors (1999). *Advances in Automatic Text Summarization*. MIT Press, Cambridge, MA.

Marcu, D. (1995). Discourse Trees Are Good Indicators of Importance in Text. In Mani, I. and Maybury, M. T., editors, *Advances in Automatic Text Summarization*, pages 123–136. MIT Press, Cambridge, MA.

McKeown, K. R. and Radev, D. R. (1995). Generating Summaries of Multiple News Articles. In Mani, I. and Maybury, M. T., editors, *Advances in Automatic Text Summarization*, pages 381–389. MIT Press.

McKeown, K. R., Robin, J., and Kukich, K. (1995). Generating Concise Natural Language Summaries. *Information Processing & Management*, 31(5):702–733.

Pollock, J. and Zamora, A. (1975). Automatic Abstracting Research at Chemical Abstracts Service. *Journal of Chemical Information and Computer Sciences*, 15(4).

Radev, D. R., Hatzivassiloglou, V., and McKeown, K. R. (1999). A description of the CIDR system as used for TDT-2. In *Proceedings, DARPA Broadcast News Workshop*, Herndon, VA.

Radev, D. R., Jing, H., and Malgorzata Stys, D. T. (2003). Centroid-Based Summarization of Multiple Documents. *Information Processing and Management*.

Saggion, H. and Gaizauskas, R. (2004). Multi-Document summarization by cluster/profile relevance and redundancy removal. In *DUC 2004*.

Wikipedia (2006). Data clustering — wikipedia, the free encyclopedia. [Online; accessed 24-April-2006].