Adaptive Information Filtering as a means to overcome Information Overload

Daniel Tauritz
M.Sc. thesis

Department of Computer Science
Leiden University, The Netherlands

Last revised September 11th, 1996

Abstract

Information Filtering is concerned with filtering data streams in such a way as to leave only pertinent data (information) to be perused. When the data streams are produced in a changing environment (as most if not all are) the filtering has to adapt too in order to remain effective. Adaptive Information Filtering (AIF) is concerned with filtering in changing environments. The changes may occur both on the transmission side (the nature of the streams can change), and on the reception side (the interest of a user can change).

The thesis research described in this paper combines trigram analysis, clustering, and evolutionary computation, in an effort to create an AIF system with such useful properties as domain independence, spelling error insensitivity, adaptability, and optimal use of user feedback while minimizing the amount of user feedback required to function properly.

Acknowledgements

I owe a debt of gratitude to many people who have supported me during my thesis research. First of all I would like to thank Ida Sprinkhuizen-Kuyper for being my primary thesis advisor, for inspiring me to boldly pursue my research interests from the very beginning, for always having time to listen and encourage, for helping me find that one unfindable bug which is the nightmare of anyone with a deadline approaching, and for keeping me in high spirits with her unwavering good humor. Secondly I would like to thank Joost Kok for being my secondary thesis advisor, for finding time to listen and encourage, and for proof reading my thesis and suggesting several additions. Thirdly I would like to thank the NATO C3 Agency (formerly SHAPE Technical Center) for supporting my research with a traineeship, Gerlof Oudega for supervising me, my room mate David Clarke for his great humor and technical assistance, and all the other people at NC3A who were so supportive. In no particular order I would also like to thank Guszti Bartfai for being a friend and discussing various aspects of clustering and neural networks with me, Chris Paice for supplying me with the source code of the Paice/Husk stemmer, Egbert Boers for always smiling, assisting in getting my code running on the computers of the CS department of Leiden University, and introducing me to GNUPLOT, Ray Dassen for sharing his arcane knowledge of Unix, and for having convinced me some years ago that TCL/TK is the best approach to writing portable user interfaces, and so many others that I give up trying to list them all before even having begun. A special thanks is due to my family for putting up with me during my research, and particularly to my mother for correcting my spelling and grammar, and beautifying my proze. Lastly I would like to thank Sharon Meigs, who has a very special place in my heart. I could thank her for so much, but it will suffice just to thank her for being.

Contents

Preface

The research described in this thesis was prompted by a personal need of the author to deal with an overflow of data. Unsatisfied with the available Information Systems, and having been inspired by courses on neural networks for clustering and evolutionary computation for parameter optimization, the idea took form to combine these techniques to build a system with desirable properties not available in other such systems. Once an appropriate document representation method was found, namely trigram analysis, a first feasibility study was conducted, the results of which were reported in [Tauritz96a]. These results were encouraging, and the outline of how such a system could be constructed was presented in [Tauritz96b]. The logical next step, namely the working out of the outline, the implementing of a first prototype of such a system, and an experimental analysis of the system, is extensively documented in this thesis.
First an easily accessible introduction to the field of Information Filtering is given, and a comparison to Information Retrieval provided. Then the basic concept of the proposed system is sketched, followed by chapters on document representation, clustering, and evolutionary computation. After this introductory material a case study is presented, giving the technical details of the design of the system (including the approaches which did not work), and providing the results of various experiments with the system. The chapter on case studies is concluded with a brief comparison of this system with the one described in [Sheth94]. After the conclusion in which the strong as well as the weak points of the current design are detailed, a lengthy to do list is provided, describing how this research could be continued. The appendices contain a sample of the code produced during the implementation providing additional implementations details to those who venture to follow up on this research.

1. Introduction

Synopsis

In this chapter the research described in this thesis will be motivated, the area of research demarcated, and a proposed method of approach described which will be the red thread for the rest of this paper.

1.1. Information Overload

We live in what is often termed the “information age”. It might more appropriately be called the “data age”, for only relevant data is information, and finding relevant data among the ever-faster growing heaps of available data is becoming increasingly more difficult. This problem has received much attention the last few years, and is known under various labels such as “Infoglut” [BYTE92] and “Information Overload” [Maes94]. The problem is twofold. First of all the amount of available data is extremely large and searching through it requires advanced algorithms. The field of research concerned with this issue is called *Information Retrieval* (further abbreviated to IR), and will be discussed in paragraph 1.2. Secondly the amount of newly added data is growing rapidly, and processing this data also requires ever more advanced methods. The field of research concerned with this issue is called *Information Filtering* (further abbreviated to IF), and will be discussed in paragraph 1.3. There are two distinct ways in which IF can be employed. One is to process new data for storage in an IR system (see figure 1.1a). The other is to completely replace IR systems and instead provide all users with a personal informaton delivery system specially tailored to their personal information needs (see figure 1.1b).

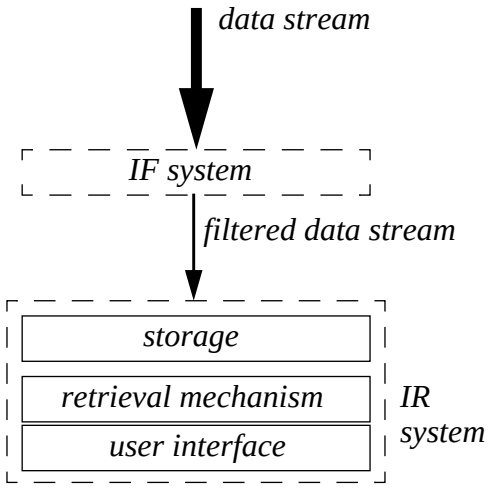

Fig.1.1a. Process for storage

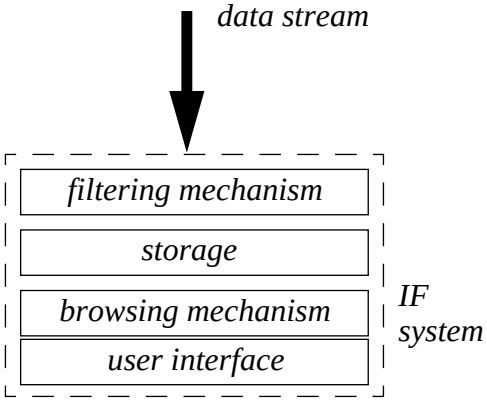

Fig.1.1b. Personal delivery system

In this thesis the emphasis will be on the second type of application, though most material covered is applicable to both types. A third field which is receiving a lot of attention of late is Data Mining (further abbreviated to DM). DM is closely associated with IR, though

there are some distinct differences. Like IR it produces information on request of the user by processing data stored in databases. Unlike IR it does not search for documents satisfying a query, but rather tries to detect hidden patterns in the data which might be of interest to the user (for example, a correlation between social status and spending habits). DM requires structured data to work on, and in [HPUser96] the following statement was made: "*Global 2000 companies struggling to get their corporate information into decision-making processes find that 80 per cent of business intelligence is still in documents, not SQL databases, claims Massachusetts-based Forrester Research in its report, Raising the Corporate IQ.*" Clearly there is a need for IF systems which can handle all this unstructured data stored in documents.

1.2. Information Retrieval

What is information retrieval?

There is no commonly accepted definition of IR, and over the years it has come to encompass an increasingly broad field. A few decades ago researchers in this field, such as Lancaster and Rijsbergen, adopted the view that IR systems satisfied queries, put to them by users with a specific information need, by returning references to material likely of interest. For instance, in [Rijsbergen79] we find the following definition for IR as stated by Lancaster: "*Information Retrieval is the term conventionally, though somewhat inaccurately, applied to the type of activity discussed in this volume. An information retrieval system does not inform (i.e. change the knowledge of) the user on the subject of his inquiry. It merely informs on the existence (or non-existence) and whereabouts of documents relating to his request.*" Now-a-days, however, users expect their information needs to be satisfied by immediately being supplied with the material likely to be of interest. In other words, today's users expect an information disclosure system, a view adopted for instance in [Bruza93]. As the term IR is more widely accepted, we will employ it throughout this thesis, although we will mean an information disclosure system.

What is an information retrieval system?

An IR system is the complete system necessary to perform the process of IR. One example is a librarian. Someone (the user) makes a request for information which the librarian then tries to satisfy. An IR system consists basically of two components. A user interface to make the requests (called queries in IR terminology) and present the retrieved information, and a retrieval system which tries to satisfy the query by returning relevant references. The queries can be formulated either in natural language, or in a so-called query language. The way the retrieval system works and the method for formulating the queries are interdependent, and can be described by a retrieval model.

Retrieval models

Here we will describe two classic retrieval models, namely the Boolean retrieval model [Rijsbergen79] and Salton's Vector Space Model (VSM) [Salton89]. A more recent model which is receiving attention is the Probabilistic model, see [Belkin92] and [Cooper94]. In the Boolean retrieval model the user has to formulate queries based on boolean logic. Those documents which are "true" for the query are retrieved. The queries

have to be expressed in terms of index terms and combined by the usual logical connectives and, or, and negation. The following example will illustrate the Boolean retrieval model.

Example

Assume the following query: $Q = (I_1 \text{AND} I_2) \text{ OR } ((\text{NOT} I_3) \text{ AND} I_4)$
This will retrieve all documents indexed by I_1 *and* I_2 *, as well as all documents indexed by* I_4 *but NOT by* I_3.

The great advantage of the Boolean retrieval model is that searching can be done in a very efficient way by using an inverted index, a matrix indicating which documents contain which terms. There are, however, two disadvantages. The first is that boolean retrieval is an all-or-nothing decision, which does not correspond to the intuitive notion that relevance of a document to a query is a matter of degree. This makes it difficult to deal with the often vague intentions of users, and makes the system very sensitive to spelling errors in both the queries and the stored documents. The second disadvantage is that humans, especially untrained ones, often find it very difficult to formulate large Boolean queries. In the VSM both documents and queries are represented as vectors in a common vector space. The dimensions of this space are the indexing terms. Similarity of meaning is assumed proportional to the distances between the vectors in the space. As vector spaces are only properly spanned by orthogonal vectors [Beauregard90] the term vectors must be assumed to be uncorrelated. This is, however, not usually the case. This issue is discussed in depth in [Salton89]. How documents can be represented as vectors, and similarity between documents determined by measuring the distances between their vector representations, is the topic of chapter 2. The advantages of the VSM are that the queries can be formulated in natural language, and that retrieval yields a ranking of the available documents according to relevance. The disadvantage is, however, that the search is not efficient, making the use of very large document collections and large numbers of indexing terms impractical. A possible solution is to cluster all the documents, and during retrieval find, for instance, the closest cluster and then compute only the distances between the query and the documents within that cluster. This is called cluster based retrieval, and is discussed in both [Rijsbergen79] and [Salton89]. The clustering of documents based on their vector representatives is the topic of chapter 3.

Relevance feedback

A powerful extension to the user interface presented so far is to allow the user to provide feedback to the system by indicating how relevant the retrieved documents were. This information can then be used by the system in various ways. The classic way is to use this information to refine the search [Salton89], but it can also be used to optimize certain parameters associated with the search process, creating a *learning* system. A particular optimization technique called evolutionary computation will be discussed in depth in chapter 4.

Information disclosure systems

In the case of an information disclosure system there is, in addition to the previously mentioned components, also a full-document retrieval system. One example of the sort of information disclosure systems which are currently receiving a lot of attention, are the so-

called Internet search engines. The most popular variant of these search engines maintains huge keyword indexes to many millions of WWW (World Wide Web) pages.

Evaluation

The classic way of evaluating the effectiveness of an IR system is through the use of the concepts of recall and precision [Zavrel95]. Precision is the ratio of the number of relevant documents retrieved to the total number of documents retrieved. Recall is the ratio of the number of relevant documens retrieved to the total number of relevant documents. So for both zero is the worst, and one is the best. The goal of an IR system is thus, to maximize both recall and precision. However, there is an inverse proportionality between these two measures. To increase recall one can relax the conditions to be met for a document to be retrieved, but this will inevitably lead to a decrease in precision, with more irrelevant documents being retrieved. To express the effectiveness of an IR system by one number, recall (r) and precision (p) can be combined in the E-measure [Rijsbergen79]:

$$E = 1 - \frac{\left(1 + B^2\right) \cdot p \cdot r}{B^2 \cdot p + r}$$

The parameter B reflects the relative importance attached to recall and precision. A value of 0.5 (or 2.0) for B corresponds to attaching twice (or half) as much importance to precision as to recall. With a value of 1.0 for B, the two factors are weighted equally. The range of E is between 0 and 1. Lower values of E indicate higher performance. 1 is worst, 0 is perfect.

1.3. Information Filtering

What is information filtering?

IF is the process of filtering incoming data streams in such a way that only particular data is preserved, depending on certain information needs. In the case of IF being employed as preprocessor for an IR system, these needs are typically predefined and rather static. When IF is being used as a replacement for IR to satisfy the needs of a particular person, these needs are, generally speaking, apt to change considerably over time. The rest of this paragraph will discuss IF systems in general, the aspects specific to the second type of IF system will be discussed in paragraph 1.4.

What is an information filtering system?

An IF system is the complete system necessary to perform the process of IF. One example is a censure department. The incoming data stream is filtered according to certain guidelines, leaving only approved documents in the output data stream. This can be done manually, or automatically through the use of a software system. The type of system discussed in this paper will be a software system. There are two main components common to all IF systems, namely the data stream, and the filtering transformation. In the case of a censure department the data stream might consist of movies, and the filtering transformation be performed by a computer program, censuring certain films based on a content analysis.

The data stream

In the case of a software system the data stream can consist of any object which can be digitally represented. Besides the traditional medium of textual documents this would include such modern media as audio, video, and in general multimedia. Some examples are:

- E-mail E-mail messages are typically textual documents with little structure. Sender and subject can be indicated, but the body of the message is free form text. New extensions are appearing at the moment which allow the inclusion of multimedia in the messages.
- Internet newsgroups Messages posted to Internet newsgroups are partially structured textual documents. While the body of the messages is free form text, just like E-mail messages, they typically include information such as sender, subject, keywords, and implicit information with regards to the topic of the message. This implicit information can be derived by examining to which particular newsgroup the message was posted. For a more detailed description of Internet news, including an example, see paragraph 5.1.
- News from news agencies such as Reuters More and more news is becoming available via the Internet, for example from Reuters (see `http://www.reuters.com./`). This usually involves textual documents ordered by topic.
- Radio The traditional medium of the radio is (still) an important supplier of data, and by digitizing the (often analog) broadcasts it can be processed by software systems. A new phenomena is the direct broadcasting of radio via the Internet, see for example `http://www.cyberville.net/index.htm`.
- Television Television, arguably the most important medium of the present day, can also be digitized and then processed by software systems. However, this is even more complicated than processing audio, both because of the much higher bandwidth, and the necessity of having to deal with video, for which no obvious transformation to text exists. Television broadcasting corporations such as CNN are opening WWW sites offering multimedia enhanced news reports, see for instance `http://www.cnn.com./`.

The filtering transformation

The basic concept here is that any object which differs “too much” from the stated information need will be filtered out, thus leaving only relevant objects in the output stream. There are however a number of reasons for desiring a more sophisticated approach to the filtering process. First of all, the distinction between relevant and non-relevant data is not a sharp one; rather the degree to which an object corresponds to an information need gradually declines as the “distance” between the two increases. And, secondly, the actual information needs are often not well understood, so the danger exists that relevant data will be filtered out because of an incorrect formulation of the information need. One way to deal with these problems is to cluster the incoming data stream and represent the information need as a set of points in the space of clusters. This method will be discussed in depth in chapter 3.

Differences between IR and IF

There are two major differences between IR and IF. First of all, in IR the information need is a short-term need, to be satisfied immediately, while in IF the information need is

typically a medium to long-term need. This has various implications. For instance, in IR a query can be refined, as opposed to IF, where user interests change over time. And secondly, in IR a static data storage is assumed during the processing of a query, while in IF the objective is the filtering of dynamic data streams. For an in depth review of the differences between IR and IF, see [Belkin92].

Collaborative filtering

An interesting extension to IF is collaborative IF, as proposed in [Goldberg92] in which it is described as follows: "*Collaborative filtering simply means that people collaborate to help one another perform filtering by recording their reactions to documents they read. Such reactions may be that a document was particularly interesting (or particularly uninteresting). These reactions, more generally called annotations, can be accessed by others' filters.*"

1.4. Adaptive Information Filtering

What is Adaptive Information Filtering?

Adaptive Information Filtering (AIF) is the process of filtering incoming data streams in such a way that only relevant data (information) is preserved. The relevancy of the data is dependent on the changing (adaptive) needs of a particular person or group of persons with a shared interest. Think for example of a newspaper. From all the news in the world available to the reporters a selection is made based on what is deemed to be of interest (relevant) to the readers of the newspaper, the rest is filtered out. As the information needs of the readers change the reporters must adapt their selection criteria correspondingly or the newspaper will fail.

What is an AIF system?

An AIF system is the complete system necessary to perform the process of AIF. One example is the system employed by our newspaper; radio and television news reports employ similar systems. The type of system discussed in this paper will be a software system. An AIF system consists of three main components. The data stream, the transformation from data to information through filtering, and the adaptive behavior. In the case of a software system the data stream might for example consist of Internet news, the transformation be performed by a computer program, and the adaptive behavior by the interaction between the user and the computer program.

The data stream

In the case of a software system with as incoming data stream Internet news, the data stream consists of textual documents with a specific classification. For example, if a document was posted to newsgroup x, then one may assume that the document can be classified corresponding to the topic of newsgroup x. This is of course not always the case, but except for groups with a very low S/N (signal to noise) ratio it will hold for the majority of the posted documents. Another common incoming data stream is E-mail which also consists of textual documents but which lacks a specific classification. In the

rest of this paper textual documents will be assumed. To deal with the data stream it is necessary to be able to compare the documents with the interests of the user at a given time. This implies the necessity of storing the interests of the user and the need for a transformation of documents and user interests to a common space where they can be compared. Such transformations will be discussed in chapter 2.

The filtering transformation

The basic idea is that any object which differs “too much” from the user’s interests will be filtered out, thus leaving only relevant objects for the user to peruse. There are, however, a number of reasons for desiring a more sophisticated approach to the filtering process. In addition to the reasons mentioned in paragraph 1.3, there is the fact that users will often have varied interests and in this case will prefer to have objects pertaining to each particular interest grouped together. Again the solution is to cluster the incoming data stream and represent the user interests as points in the space of clusters. This will be further discussed in chapter 3.

The adaptive behavior

In order to adapt to the changing information needs of the user, interaction with that user is necessary. Input from the user is needed with respect to two important issues. First of all, input is required concerning the user’s interests, and secondly, the user must indicate which objects have been clustered correctly and which incorrectly. One possibility is that the clusters themselves represent the user’s interests. A mechanism particularly suited to facilitating this adaptive behavior is evolutionary computation, which will be discussed in chapter 4.

2. Representation

Synopsis

In this chapter the necessity for transforming the incoming data stream and the user interests to a common space will be motivated, the transformation to a representation in the common space discussed in depth, and a number of preprocessing techniques described.

2.1. Syntactical versus semantical representations

As mentioned in paragraph 1.4, a basic requirement for AIF is to be able to determine if an object differs "too much" from an user interest. This implies the necessity to be able to measure the distance between the objects and the user interests. A distance measure is also necessary for the clustering process. To be able to measure the distance we need to transform the objects and the user interests to a common space and define a metric for that space. A space in which both objects and user interests are expressed in terms of their semantics would be ideal. The distance measure would then indicate, for instance, the degree to which an object and a user interest correspond semantically, precisely the comparison one would hope for. Transforming objects to such a space would require a semantical analysis of those objects, and as even the semantical analysis of a "simple" object such as written text is still an open problem in computer science [Scholtes93], this is currently not feasible. An alternative is to choose as common space a syntactical space. This would greatly simplify the transformation because no understanding of the objects is necessary. A practical choice for common space is a vector space where the vectors describe certain syntactical characteristics of the objects and the user interests. A vector space is a practical choice because many well known distance measures exist for it, such as:

Euclidean distance: $d(\bar{x},\bar{y}) = \|\bar{x} - \bar{y}\|$

Cosine correlation: $d(\bar{x},\bar{y}) = \frac{\bar{x} \bullet \bar{y}}{\|\bar{x}\| \cdot \|\bar{y}\|}$

Dice coefficient: $d(\bar{x},\bar{y}) = \frac{2\bar{x} \bullet \bar{y}}{\|\bar{x}\| + \|\bar{y}\|}$

where $\bar{x}$ and $\bar{y}$ denote vectors, $\bar{x} \bullet \bar{y}$ their inner product, and $\|\bar{x}\|$ is the norm of $\bar{x}$. For more information on measurements in the information sciences see [Boyce94].

The difficulty with syntactical transformations lies in the problem of choosing one for which the distance measure in the vector space reflects the semantical distance as accurately as possible. Two possible transformations for written text will be discussed in the following paragraphs.

2.2. Keyword analysis

An often used transformation is one based on keywords [Rijsbergen79]. In the simplest form a predetermined set of keywords is used. The transformation projects documents on vectors consisting of elements representing the frequency with which the keywords occur in the document.

Example

keyword set = {tree, house, sun, winter}
document = {In the summer the house gets hot from the sun. In the winter however the sun does not have this effect.}
vector = (0, 1, 2, 1)

The problem with using a predetermined set of keywords is twofold. First of all it is necessary to know beforehand which keywords should be included in the set. This requires domain specific knowledge. Secondly it precludes adapting to changes in the domain which would require the addition of new keywords to the set. To overcome both problems one could take the set of all words as keyword set. Unfortunately, this would require vectors too large for practical use. Also, new words are constantly being created, adding still further to the problem.

2.3. N-gram analysis

A completely different approach is the so-called *n*-gram analysis [DeHeer82]. The *n* stands for a positive integer. In 1-gram analysis the occurrence of single letters is determined, in 2-gram analysis that of pairs of letters, in 3-gram analysis that of triplets (in this context called trigrams), etc.

Example

document = {In the summer the house gets hot from the sun. In the winter however the sun does not have this effect.}

Table 1: 1-gram vector

a	b	c	d	e	...
1	0	0	1	14	

Table 2: 2-gram vector

...	er	es	et	...
	3	1	1	

Table 3: 3-gram vector

...	the	thf	thg	...
	5	0	0	

The larger *n*, the more accurately the distances between *n*-gram vectors correspond with the semantical distances of the original documents [Scholtes93]. The price paid for larger values of *n* is an exponentially increasing demand for memory, namely proportional to 26^n (or 27^n if the space is included), see figure 2.3.

n	# elements
1	26
2	676
3	17576
4	456976
5	11881376

Figure 2.3. N-gram analysis memory requirements for various values of *n*.

For $n \leq 3$ the memory demands are still moderate enough for practical use, $n = 4$ is for the moment only interesting for research purposes, and $n > 4$ is currently not of any use. In practice this means only $n = 3$ is normally used as the results for $n < 3$ are not accurate enough to be of any use [Schmidt88]. The great advantage of trigram analysis is that it is domain independent and the set is small enough to be used entirely so that the problem of having to expand the set because of domain changes is circumvented.

For an example of an experiment demonstrating the discriminatory power of trigram analysis, see [Tauritz96a].

2.4. Weighted vector representations

Trigram analysis alone does not provide sufficient discriminating power to accurately cluster textual documents, as shown in [Tauritz96a]. In that same paper it was, on the other hand, demonstrated that weighted trigram analysis does have sufficient discriminating power. In weighted trigram analysis each trigram is assigned a weight. The discriminating power can be substantially increased by choosing the right weights. The following example will demonstrate how weighted trigram analysis works.

Example

Suppose that the following three vectors were found:

vector	*aaa*	*bbb*	*ccc*
1	*1*	*0*	*1*
2	*1*	*1*	*0*
3	*0*	*1*	*1*

Note: All other trigrams are zero for all three vectors.

The distance between vectors 1 and 2 is the same as the distance between vectors 1 and 3. However, if vectors 1 and 2 represent objects which are semantically similar, and vector 3 represents an object which is semantically dissimilar to objects 1 and 3, it would be preferable if the distance between vectors 1 and 2 was smaller than the difference between vectors 1 and 3.

With weighted trigram analysis we can accomplish this by assigning a larger weight to a difference in frequency of trigram ‘aaa’ than to the other two trigrams. With weight 2 for ‘aaa’, and 1 for the other two trigrams, we find the following distances:

vectors 1 & 2: 1.4
vectors 1 & 3: 2.2
vectors 2 & 3: 2.2

Now the distance between the semantically similar objects is smaller than the distance between the semantically dissimilar objects, as one would hope for.

2.5. Normalized vector representations

It is to be hoped when comparing two textual documents of the same topic, that they will be considered similar enough to be grouped together, independent of the length of each document. However, using trigram analysis as described earlier, the degree of similarity is not at all independent of the length of each document, as the following example will show.

Example

Suppose that the following three vectors were found:

vector	*aaa*	*bbb*	*ccc*
1	*1*	*0*	*1*
2	*3*	*0*	*3*
3	*0*	*1*	*0*

Note: All other trigrams are zero for all three vectors.

Assume that vectors 1 & 2 represent objects concerning topic A, and vector 3 represents an object concerning topic B. The Euclidean distances between the vectors are as follows:

distance vector 1 & 2: 4

distance vector 1 & 3: 1.7

distance vector 2 & 3: 4.4

While one would want the distance between vectors 1 and 2 to be smaller than the distances between vectors 1 and 3 and between vectors 2 and 3, this is clearly not the case. The problem is that we are measuring the difference in frequency, which is clearly length dependent, instead of the difference in frequency distribution. We can easily solve this problem by normalizing the vectors using the cumulative frequency as normalizer. This is done by dividing all the elements of the original vector by the normalizer. The results of normalization applied to the above example are as follows:

cumulative frequency vector 1: 2

cumulative frequency vector 2: 6

cumulative frequency vector 3: 1

normalized vector	aaa	bbb	ccc
1	0.5	0	0.5
2	0.5	0	0.5
3	0	1	0

distance normalized vector 1 & 2: 0

distance normalized vector 1 & 3: 1.2

distance normalized vector 2 & 3: 1.2

Now the distance between vectors 1 and 2 is smaller than the distances between vectors 1 and 3, and between vectors 2 and 3, as we would expect.

2.6. Preprocessing techniques

While both keyword analysis and trigram analysis work to a certain degree, they are far from perfect (as one would expect of any syntactical transformation). Techniques which are able to help increase the discriminatory power of transformations from the space of textual documents to syntactical spaces are thus of great utility. According to [Scholtes93] such techniques are in fact of such importance that without them trigram analysis is not effective. In this paragraph a number of such preprocessing techniques will be described all of which can be applied before either keyword analysis or trigram analysis. All the techniques can be applied during preprocessing, the best sequence being the order in which they are discussed.

2.6.1 - Stop lists

Filtering out common words (called ‘stop’ or ‘fluff’ words in [Rijsbergen79]) such as ‘the’, ‘a’, ‘it’, ‘from’, etc. is computationally non-intensive and has two great advantages. First of all, it typically reduces the length of documents by between 30 and 50 per cent [Rijsbergen79]. And secondly, it increases the discrimination between documents as the following example will demonstrate.

Example

document 1 = {Where has the house gone off to?}
document 2 = {Where has the mouse gone off to?}
stop list = {where,has,the,gone,off,to}

Without filtering out the common words contained in the stop list, we obtain the following normalized Euclidean distance between the documents, by dividing the keyword frequencies through the cumulative document keyword frequency. The keywords are all the words not filtered out.

cumulative document 1 keyword frequency = 7
cumulative document 2 keyword frequency = 7

vector	where	has	the	house	mouse	gone	off	to
1	1/7	1/7	1/7	1/7	0	1/7	1/7	1/7
2	1/7	1/7	1/7	0	1/7	1/7	1/7	1/7

The normalized Euclidean distance of these two vectors is sqrt(2/49) = 0.2020...

However, when we filter out the common words we find the following:

cumulative document 1 keyword frequency = 1
cumulative document 2 keyword frequency = 1

vector	house	mouse
1	1	0
2	0	1

Now the normalized Euclidean distance is equal to sqrt(2) = 1.4142...

So in this (admittedly extreme) case the discriminatory power is multiplied by seven by employing a stop list. See appendix B for the stop list used in all the experiments described in paragraph 5.1.

2.6.2 - Stemming

Stemming (also called base form reduction) is the process of reducing words to their stems, typically accomplished through suffix stripping [Paice90]. For instance, the words “mathematics”, “mathematical”, and “mathematician” can all be reduced to the common stem “math”. Stemming improves the discriminatory power, see [Lennon81].
A standard approach is to remove the longest possible suffix from a list of suffixes [Rijsbergen79]. The Lovins stemmer [Lovins68] employs a list of 294 endings, the Dawson stemmer [Dawson74] a much more comprehensive list of around 1,200 suffixes, while the Porter stemmer [Porter80] proceeds through a fixed succession of five stages, with a different lookup table being used each time. In the case of the Lovins stemmer the truncated form is dealt with according to a set of recoding rules which deals with double consonants and carries out various other adjustments. An algorithm which circumvents the recoding stage of the Lovins stemmer is, for instance, the Paice/Husk stemmer [Paice90]. To illustrate how stemming algorithms work the Paice/Husk stemmer will now be described by reproducing the algorithm, acceptability conditions, and rule examples, as given in [Paice90].
In the algorithm shown in figure 2.6.2 the term *form* refers to any word or part-word which is currently being considered for stemming. The original word before any changes have been made to it is said to be intact.

Algorithm: Stemmer

1. Select relevant section:
   Inspect the final letter of the form;
   if there is no section corresponding to that letter, then terminate;
   otherwise, consider the first rule in the relevant section.
2. Check applicability of rule:
   If the final letters of the form do not match the reversed ending in the rule, then goto 4;
   if the ending matches, and the intact flag is set, and the form is not intact, then goto 4;
   if the acceptability conditions (see below) are not satisfied, then goto 4.
3. Apply rule:
   Delete from the right end of the form the number of characters specified by the remove total;
   if there is an append string, then append it to the form;
   if the continuation symbol is “.” then terminate;
   otherwise (if the continuation symbol is “>” then) goto 1.
4. Look for another rule:
   Move to the next rule in the table;
   if the section letter has changed then terminate;
   otherwise goto 2.

Figure 2.6.2. The Paice/Husk stemming algorithm

Acceptability Conditions

If these conditions were not present, the words “rent”, “rant”, “rice”, “rage”, “rise”, “rate”,
“ration” and “river” would all be reduced to “r” by the rules as laid down. The conditions used are:

a) if the form starts with a vowel then at least two letters must remain after stemming (e.g., “owed”/”owing” → “ow”, but not “ear” → “e”).
b) if the form starts with a consonant then at least three letters must remain after

stemming and at least one of these must be a vowel or "y" (e.g., "saying" → "say" and "crying" → "cry", but not "string" → "str", "meant" → "me" or "cement" → "ce").

These conditions wrongly prevent the stemming of various short-rooted words (e.g., "doing", "dying", "being"); it is probably best to deal with these separately by lexical lookup.

Rule examples

1. The rule "sei3y>" means: if the word ends in "ies" then replace the last three letters by "y" and then apply the stemmer again to the truncated form.
2. The rule "mu*2." means: if the word ends in "um" and if the word is intact, then remove the last two letters and terminate. This converts "maximum" to "maxim" but leaves "presum" (from "presumably" etc.) unchanged.
3. The rule "ylp0." means: if the word ends in "ply" then leave it unchanged and terminate. This ensures that the subsequent rule "yl2>" does not remove the "ly" from "multiply".
4. The rule "nois4j>" causes "sion" endings to be replaced by "j". This acts as a dummy, causing activation of the "j" section of the rules. Hence "provision" is converted first to "provij" and then to "provid".

2.6.3 - Conflation

Conflation is the process of grouping together non-identical words which refer to the same principal concept [Paice90]. These may be words of a completely different form (e.g. "autumn" and "fall"), or words with the same common root (e.g. "mathematics" and "mathematician"). In fact, stemming can be seen as a kind of conflation. The use of conflation for increasing the discriminatory power is universally recommended, see for example [Rijsbergen79] and [Paice90]. The following example will demonstrate the effectiveness of conflation.

Example

Consider the following two documents which have a high semantical correlation:

document 1 = {In the autumn she gives lessons in mathematics.}
document 2 = {In the fall the mathematician gives lessons.}

Using the following stop list:

stop list = {in,the,she}

We find the following cumulative document keyword frequencies:

cumulative document 1 keyword frequency = 4
cumulative document 2 keyword frequency = 4

And the normalized vectors are as follows:

vector	*autumn*	*gives*	*lessons*	*mathematics*	*fall*	*mathematician*
1	1/4	1/4	1/4	1/4	0	0
2	0	1/4	1/4	0	1/4	1/4

Which results in the following normalized Euclidean distance:

normalized Euclidean distance = sqrt(1/4) = 1/2

Now let us consider the case where we apply stemming and conflation in addition to the use of the stop list. After preprocessing we have the following documents:

preprocessed document 1 = {autumn give lesson math.}
preprocessed document 2 = {autumn math give lesson.}

The cumulative document keyword frequencies remain unchanged, but the normalized vectors are now as follows:

vector	*autumn*	*give*	*lesson*	*math*
1	1/4	1/4	1/4	1/4
2	1/4	1/4	1/4	1/4

With the vectors being identical the normalized Euclidean distance is of course zero, confirming the notion that the documents are semantically identical (at least to the degree one could hope a syntactical transformation to establish).

3. Clustering

Synopsis

An introduction to clustering algorithms will be presented in this chapter, a classic problem with many clustering algorithms, called the stability- plasticity dilemma, will be discussed, and a number of interesting developments in the area of neural networks will be described.

3.1. Introduction

What is clustering?

Clustering is the process of dividing a set of objects into multiple sets of objects, called clusters. The following example will illustrate this concept.

Example

original set = {apple, cake, strawberry, lemon, pie, cookie}
set no.1 of clusters = {[apple,strawberry,lemon], [cake,pie,cookie]}
set no.2 of clusters = {[apple,cake,lemon,pie], [strawberry,cookie]}

In this example, the first set of clusters divides the original set into two distinct sets where the clustering criterion was, is it fruit or something you bake? Note that this requires knowledge about the objects to be clustered. The second set of clusters also divides the original set into two distinct sets, this time with as clustering criterion, is it a word of more than 5 letters, or not? Note that this requires no knowledge about the objects to be clustered, it is purely syntactical.

What is a clustering algorithm?

A clustering algorithm is an algorithm which takes as input a set of objects and produces as output a set of clusters and a mapping of each input object to a cluster. There are many clustering algorithms which all differ with respect to complexity, performance, and clustering properties. An important distinction can be made between clustering algorithms which require the whole set of objects to cluster in advance and those which can process one object at a time and present the results of the clustering after each newly processed object. The first class of algorithms will further be referred to as batch clustering algorithms, the second as incremental clustering algorithms. As a typical AIF system has a perpetual incoming data stream, the rest of this paper will be restricted to discussing algorithms of the second class.

A simple clustering algorithm shell

To illustrate how a clustering algorithm works a simple algorithm shell (shell indicates that the structure of the algorithm is described, but not precisely defined) will be

described, after which a possible instantiation of that algorithm shell will be presented. For an overview of clustering algorithms see for example [Backer95].

In this particular algorithm shell a common vector space has been defined to which all the objects to be clustered are transformed. The clusters are represented by hyperglobes in the common vector space. A hyperglobe is described in terms of its center, represented by a vector in the common vector space, called a prototype vector, and a parameter *r* indicating its radius, which is the same for all clusters.

The algorithm shell works as follows. Initially no objects have been clustered, thus no clusters exist yet. Then the first object is presented, which is transformed to the common vector space. As this is the first object it defines the center of the first cluster. Then a second object is presented and transformed. Now a choice has to be made. Should this object be clustered as belonging to the first cluster, or should a new cluster be formed? This depends on whether or not the new object falls within the hyperglobe representing the first cluster. If it does, then the center of the first cluster will be moved a bit in the direction of the second object. Otherwise a new cluster is created with its center being defined by the second object. Let us assume the latter was the case. There are now two clusters. To determine how the third object should be clustered it is first necessary to determine which cluster is closest. If it is close enough, meaning that the object falls within the cluster's hyperglobe, the cluster is moved in the direction of the third object. Otherwise again a new cluster is created, its center being defined by the third object. This process continues for all further objects presented. The algorithm described by the above is given in figure 3.1a.

Algorithm: Cluster Generic

Step 1 - Initialization
- Start with no clusters

Step 2 - Present and transform new object
- Let *v* := transformation (new object)

Step 3 - Find the closest cluster (if any exist)
- Let *c* := prototype vector of closest cluster (*v*)

Step 4 - Is the closest cluster close enough?
- If *c* is too far from *v*, or if there are no clusters yet, then create a new cluster with prototype vector equal to *v*; goto step 2

Step 5 - Update a matched cluster
- Update the matched cluster by moving it closer to *v*; goto step 2

Figure 3.1a. Clustering algorithm shell

To obtain an actual algorithm it is necessary to create an instantiation of the algorithm shell by defining what an object is and specifying procedures for carrying out the transformation, finding the center of the closest cluster, determining if *c* is too far from *v*, and moving the matched cluster closer to *v*.

A possible instantiation of the clustering algorithm shell

As this paper is principally concerned with the clustering of textual documents, defining an object as a textual document is an obvious choice. It is then necessary to define what a textual document precisely is. In the rest of this paper a textual document will be considered to be a list of one or more lines, each line in its turn being composed of words

separated by exactly one space (acting as separator symbol) and each word consisting of one or more lower-case letters of the Latin alphabet. We further restrict this simple class of textual documents by requiring the documents to be syntactical and semantically correct English non-fiction. The transformation is the in paragraph 2.3 discussed trigram analysis. To find the closest cluster and to determine if *c* is too far from *v* we require a distance metric. For distance metric we will use the Euclidean metric. Moving the matched cluster closer to *v* will be accomplished by taking a linear combination of *v* and *c*. The algorithm is then as shown in figure 3.1b.

Algorithm: Cluster Euclidean

Step 1 - Initialization
- Start with no cluster prototype vectors

Step 2 - Present and transform new object
- Let V = $(v_1, v_2, \ldots, v_n)$:= trigram analysis (new object)

Step 3 - Find the closest cluster (if any exist)
- Find the C = $(c_1, c_2, \ldots, c_n)$ to minimize d(P,I) = $\sqrt{\sum_{x=1}^{n} \left(C_x - V_x \right)^2}$

Step 4 - Is the closest cluster close enough?
- If d(C,V) > *r*, or if there are no cluster prototype vectors yet, then create a new cluster, with prototype vector equal to V; goto step 2

Step 5 - Update a matched cluster
- Let C := $(1 - \lambda) \cdot C + \lambda \cdot V$; goto step 2

Figure 3.1b. Instantiation of Cluster Generic

The following example will illustrate the algorithm.

Example

In this example we consider a sequence of the following three objects:

object 1: {apples and cake}
object 2: {apples with cake}
object 3: {applesauce}

The trigram analysis of the objects results in the following vectors:

vector	ake	and	app	auc	cak	esa	ith	les	ple	ppl	sau	uce	wit
1	1	1	1	0	1	0	0	1	1	1	0	0	0
2	1	0	1	0	1	0	1	1	1	1	0	0	1
3	0	0	1	1	0	1	0	1	1	1	1	1	0

Note: All other trigrams are zero for all three vectors.

The clustering algorithm is executed with radius r=2.0 and adaptation parameter λ*=0.2.*

cycle	*prototype vectors*
start	-
object 1	*(1,1,1,0,1,0,0,1,1,1,0,0,0)*
object 2	*(1,0.8,1,0,1,0,0.2,1,1,1,0,0,0.2)*
object 3	*(1,0.8,1,0,1,0,0.2,1,1,1,0,0,0.2), (0,0,1,1,0,1,0,1,1,1,1,1,0)*

Figure 3.1c. Clustering cycles

The cycle starts off with no clusters having been formed. Then the first object is presented. Its vector transformation defines the first prototype vector (see second row of figure 3.1c) which represents the first cluster. Next the second object is presented. The distance between vectors 1 & 2 is $\sqrt{3}$*, which is smaller than the radius. Thus the second object is assigned to the first cluster which moves a bit in its direction (see third row of figure 3.1c). Last the third object is presented. It does not fall within the first cluster, thus a second cluster is formed, its prototype vector being defined by the vector transformation of the third object. So now two clusters exist with the prototype vectors given in the last row of figure 3.1c.*

3.2. The stability-plasticity dilemma

Cluster Euclidean has some nice properties. One is that the clusters continue to adapt to new input vectors indefinitely. Algorithms which feature this property — to forever retain the potential to adapt to new input vectors — are said to be “plastic”. Such algorithms are the ones of choice when operating in changing environments such as AIF, where user interests are expected to slowly alter over time. Cluster Euclidean is also fast, and simple to understand. However, it also has some less positive properties, which might, in certain situations, cause it to be less effective. This would depend on the particular sequence of input vectors. That the particular sequence of input vectors influences the clustering process is in itself a questionable property, as obviously the topic one would expect a particular object to be clustered under is independent of the sequence in which the objects are presented. This is, however, a general property of incremental clustering algorithms,

according to [Moore89: paragraph 5, page 181, lemma 3]: "*Different orders of presentation of input vectors during learning can result in different clusters. (The proof of this lemma is by example. This property is a general characteristic of any incremental clustering algorithm.)*"
While I agree that this holds for most clustering algorithms, it does not hold for all, as the following counter example will demonstrate. Assume you have a batch clustering algorithm called BATCH. An incremental clustering algorithm can then very easily be constructed as shown in figure 3.2.

Algorithm: INCREMENTAL

Step 1 - Initialization
- `Start with empty set of objects to cluster`

Step 2 - Presentation of new object
- `Add the new object to the set of objects to cluster`

Step 3 - Cluster
- `Execute BATCH with set of objects to cluster as`

Figure 3.2. An incremental clustering algorithm that does not depend on the order of presentation

This is an incremental clustering algorithm as described in paragraph 3.1, but as the sequence in which the objects to cluster were presented is not preserved in the set, that sequence cannot influence the clustering process.
Even more serious is the case in which presenting identical objects does not produce mappings to the same cluster. This can occur because after the first presentation the cluster has changed so much in the process of adapting to the objects presented since, that the original object does not lie any longer within the radius of the cluster. This is a form of instability. Another, even more serious, problem is that clusters can cycle indefinitely during repetitive presentation of a finite sequence of objects, see [Heins95] and [Moore89]. An in depth study of the problem of instability with Cluster Euclidean is to be found in [Moore89]. One possibility to at least prevent the occurrence of clusters cycling indefinitely is to decrease the adaptation parameter after each adaptation. The consequence is, however, that over time the algorithm looses more and more of its plasticity. This tension between stability and plasticity is called the stability-plasticity dilemma [Carpenter87] and can be posed as follows [Carpenter88]: "How does the system know how to switch between its stable and its plastic modes to achieve stability without rigidity and plasticity without chaos?"[1] A family of algorithms specifically designed to overcome this dilemma are those based on Adaptive Resonance Theory and will be discussed in subparagraph 3.3.2.

1. when to switch would seem the more appropriate question

3.3. Neural networks

Synopsis

As the promise of parallel computing is being fulfilled with ever more powerful parallel computers being built, parallel clustering algorithms become increasingly important. Neural networks offer an elegant model to define such algorithms. In this paragraph an introduction to neural networks will be given, and a number of promising types of neural networks described.

3.3.1 - Introduction

What is a neural network?

A neural network is a system of processing units, connections, and weights associated with the connections, which propogates activation patterns from its input units to its output units, augmented by a learning rule. A simple neural network is shown in figure 3.3.1 (after [Hertz91]).

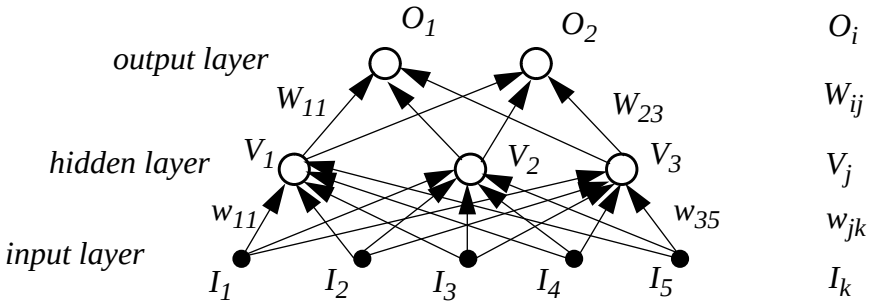

Figure 3.3.1. Schematic representation of a basic neural network

To perform a computation the units of the input layer are given a certain activation. The net input of the units of the hidden layer and the output layer is a weighted sum of the input signals (the output of the units of the previous layer). So, for example, the net input of the units of the hidden layer are computated as follows:

$$net_j = \sum_k w_{jk} \cdot I_k$$

The activation of a unit is a function of its net input and its previous activation, though in most neural networks its previous activation is disregarded and the activation equal to its net input. The output of a unit is determined by applying an output function, standard choices being identity, threshold function, and sigmoid function. The behavior of the network is determined by the combination of its architecture (defined by the processing units and the connections) and its set of weights. In traditional neural networks “learning” can occur through adaptation of the weights by a so called “learning rule”. More recently,

neural networks have been designed in which the architecture itself can also adapt. A few of such networks will be described in subparagraph 3.3.3.

Learning rules

Learning rules are usually classified as either supervised or unsupervised. With supervised learning two different modes of operation can be distinguished, namely the *training* mode, and the *production* mode. In the training mode input vectors are presented to the network and feedback from the environment is given in response to the output. This feedback can be an exact specification of how much the output differs from the optimal value. This is called learning with a teacher. Alternatively the response can simply indicate either correct or incorrect output. This is called learning with a critic, or reinforcement learning. When sufficient training has occurred it is hoped that the network will have generalized from the training examples, and will, with high accuracy, produce responses to novel input vectors in the production mode. In contrast, unsupervised learning assumes no feedback from the environment and thus does not distinguish between any such modes of operation. The network must discover for itself regularities, patterns, or categories, by exploiting the redundancy in the input data. The network must thus display some degree of self-organization.

Unsupervised competitive learning

One specific type of unsupervised learning is competitive learning in which only a single output unit is activated at any particular time. The output units "compete" with each other, the unit with the highest activation, also called the Best Matching Unit (BMU), wins the competition. The winning unit then adapts its weight vector to match the input vector even better (the learning phase). Such networks cluster or categorize the input data, making them obvious candidates for use as clustering algorithms in AIF systems. In the following subparagraphs two interesting families of unsupervised competitive learning neural networks will be discussed.
For a more in depth introduction to neural networks see for example [Freeman91] and [Hertz91].

3.3.2 - Adaptive Resonance Theory

Adaptive Resonance Theory (ART) was specifically developed by Grossberg to overcome the stability-plasticity dilemma. Neural networks belonging to the family of ART networks solve the dilemma by only matching (and thus adapting) stored prototype vectors representing the clusters when the input vectors are "sufficiently" similar (compare with step 4 of Cluster Generic). If they are sufficiently similar they are said to *resonate* and adaptation occurs, therefore the name Adaptive Resonance Theory. If not a new cluster is formed, represented by a prototype vector equal to the novel input vector, using a previously uncommitted output unit. In absence of uncommitted output units, a novel input gives no response. Just like Cluster Generic includes a parameter on which the meaning of sufficiently similar is dependent, namely the radius, ART also has such a parameter, called the vigilance level ρ, with $0 < \rho \leq 1$. The larger ρ, the more exact the match has to be, resulting in more and smaller clusters. According to [Hertz91] "*The vigilance level can be changed during learning; increasing it can prompt subdivision of existing categories.*" For an easy introduction to ART see [Heins95], which describes

ART1, the simplest of the ART family of neural networks, both in terms of its clustering capabilities and its network dynamics, and also provides further references. An interesting recent development in the field of ART is the design of ART networks capable of hierarchical clustering, see for example [Bartfai95].

3.3.3 - Self Organizing Topological Maps

Introduction

Self Organizing Topological Maps, or simply Self Organizing Maps, abbreviated to SOMs, are networks which perform a topology preserving mapping from some input space to an output space. This means that nearby outputs correspond to nearby inputs. The most common type of SOM maps some *n*-dimensional space onto a 2-dimensional grid [see figure 3.3.3], making it easy to visualize similarity relationships. This does assume a sufficient degree of redundancy to allow for the dimensionality reduction from *n* to 2. The most popular type of SOM is the Kohonen map, which will be discussed next, together with a few variations.

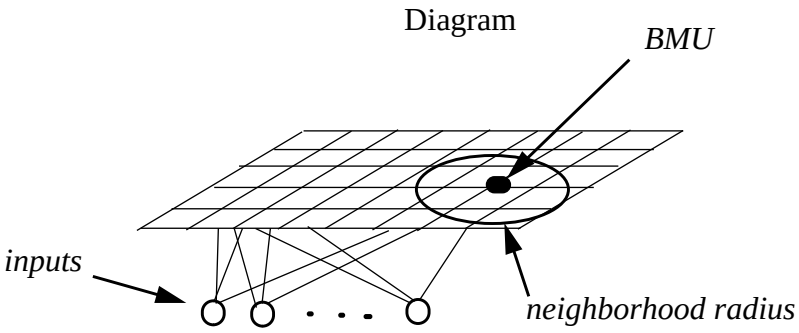

Figure 3.3.3. A common type of SOM

Kohonen maps

Kohonen maps are SOMs based on unsupervised competitive learning, but with a modified learning rule taking neighborhood relationships into account. This is done by adapting not only the BMU to better match the current input vector, but all the output units, the degree of adaptation depending on the distance of the output unit to the BMU. So the closer an output unit is to the BMU, the stronger the effect of adaptation, which results in a topology preserving mapping. Kohonen maps suffer, however, from the stability-plasticity dilemma. In order to guarantee stabilization the learning parameter is decreased in time, and thus it gradually looses its plasticity. Also, depending on the particular input data and initial parameters, the map may get tangled up, or collapse into a single point. Another potential problem is that when mapping from a *n*-dimensional space to a 2-dimensional grid, the network may have great difficulty properly adjusting to input spaces with a non-uniform distribution. Two variations on Kohonen maps which try to address these issues to varying degrees will now be discussed. For a good overview of the Kohonen map and its applications see [Kohonen90].

Fritzke’s growing cell structures

Fritzke’s growing cell structures differ from Kohonen maps in that they have a stable learning parameter and grow in response to the input data. As the network adjusts to the input space it approaches a state in which the density of the units correspond to the probability density of the input space. For further information and references see [Zavrel95].

Blackmore and Miikkulainen’s incremental growing grid

This network is incrementally grown on a fixed rectangular grid, using the same learning rule as the Kohonen map. Units can never be deleted, but the connections between the units can to indicate cluster boundaries. For further information and references see [Zavrel95] in which comparisons of Kohonen maps, Fritzke’s growing cell structures, and Blackmore and Miikulainen’s incremental growing grids, as applied to IR, are also made.

WEBSOM

The rapid growth of the World Wide Web is making it increasingly important to have sophisticated browsing capabilities for free-form textual document collections. A novel approach is using the two dimensional map generating capabilities of the Kohonen map to present users with a general view of document collections. Three main processing steps can be distinguished in the WEBSOM method. In the first step the documents are preprocessed, removing non-textual information and filtering out common words. The second step consists of forming a word category map using the SOM algorithm. In the third step a map of the documents is formed by mapping their text onto the word category map using again the SOM algorithm. For further information see [Honkela96].

4. Evolutionary Computation

Synopsis

In this chapter the use of evolutionary computation in AIF systems will be motivated, and an introduction to evolutionary computation given.

4.1. Adaptive behavior

In paragraph 2.4 weighted vector representations were introduced by taking an in depth look at weighted trigram analysis. Generally speaking any AIF system using a weighted vector representation has to deal with the problem of finding the optimal values for the weights. Other parameters, such as the radius and the adaptation parameter of Cluster Generic (see paragraph 3.1), also require optimization. The particular set of weight values greatly influences the clustering process. As mentioned in paragraph 1.4, the user provides the system with feedback indicating satisfaction or dissatisfaction with the clustering of a document. This feedback is essential for the system to be able to "learn" to predict in the future with higher accuracy how the user wants the documents to be clustered. It is important to note that because of the changing needs of users, the system will always need to have the ability to adapt to those changes. In other words, we need a learning system capable of optimization in a perpetually changing environment. The required adaptive behavior of such a system can be facilitated by evolutionary computation. In the next paragraph an introduction to evolutionary computation will be given, and in chapter 5 two case studies demonstrating this approach to complex dynamic parameter optimization will be presented.

4.2. Introduction to evolutionary computation

What is evolutionary computation?

Evolutionary computation is the process of finding (near) optimal solutions for computational problems using methods inspired by evolution theory. The general idea is that a population of trial solutions is maintained, this population evolves, and some members of the population are replaced, based on some kind of selection process, by new members which are the offspring of a selected group of members of the old population. Both selecting the members to be replaced and the members to reproduce can be done on the basis of the quality of the members. This quality is determined by an external objective function that operates on the behavior of the system resulting from the application of a particular trial solution, rather than on the coding of the trial solution itself. Before discussing the various components of this process in more detail, a humoristic example will be given to illustrate how evolutionary computation works.

Example

Let us postulate that via the SETI (Search for Extra Terrestial Intelligence) project NASA has come across some kind of radio signal from outer space. The problem is, however, that the frequency of the signal changes over time according to some kind of periodic function. The technician charged with modifying a radio to compensate for these changes is alas not at all good at math, but does manage to get the radio hooked up to a computer. Being very pleased with himself for getting it hooked up he decides to ask a friend of his, who happens to be a computer scientist, to program the computer in such a way that the radio will work as desired. His friend agrees to come and have a look. After having studied the situation for a few minutes she says that there is an elegant way of solving the problem which will be easy to extend if the signal turns out to be much more complex than is now thought, and a simple way which is barely worth her time. The technician thinks for a moment and decides that an elegant extendable solution will surely impress his superiors much more than a simplistic one, so he asks her to go for the first option. After a couple of hours she calls him and says the program is ready but they will need lots of radios, all hooked up to the computer, if he thinks he can manage that. Well, you are a technician or you aren't, so he goes off and gets a hundred radios and hooks them all up to the computer. Then she starts the program and within an hour she announces that the problem has been solved. He is of course very pleased and asks her if she would explain how she did it. She explains that the periodic function they are looking for is of the form:

*frequency = a + b * sin (c * (time + d))*

with a, b, c, and d being the unknown parameters. The hundred radios start off at random frequencies, depending on the randomly selected values for a, b, c, and d stored for each radio separately in the computer. Every ten seconds the computer checks how clear the signal is for each radio. The five worst tuned radios have their values for a, b, c, and d replaced by slightly altered values of the five best tuned radios. Clearly the chances are good that these new combinations will be better as they are close to combinations already performing well. The slight alterations insure that new combinations are continually being tried. As the population of combinations evolves the radios get better and better tuned, until a (near) perfect solution is found. The technician has listened in amazement and declares her to be a genius. Before she leaves he asks her what the simple solution would have been. "Oh" she says, "because it is such a simple function we could of course have done a few measurements with one radio and simply calculated the right values. But this was much more fun, wasn't it?"

Besides demonstrating how evolutionary computation works, and showing that it is a bit of an overkill for problems which can be easily solved analytically, it is worth mentioning that this example is also illustrative of how evolutionary computation can facilitate adaptive behavior. Imagine for instance that the periodic function of the radio signal very slowly changed over time in a completely random way. Then there would be no analytical function which could describe it, but the system of the hundred radios would have no problem adapting to the changes by following the same procedure as it did to fine tune initially.

Discussion of the components

Before the evolutionary process can commence, the initial members of the population have to be specified. This can be done either by using predetermined values, or

completely at random (as in the radio example). The next step is to choose which members of the population will be replaced. This can be done at random, but, applying Darwin's principal of "survival of the fittest", it is in general better to select the weakest members (or at least let the weakest members have a greater chance of being selected). To determine which of the members of the population are the weakest an external objective function is required, called the fitness function (in the example this function returned higher fitness values for better tuned radios). The fitness function operates on the outcome of applying a trial solution (phenotype), rather than on the coding of the trial solution (genotype) itself. This is comparable to evolution in nature where the fitness of an individual is determined by its phenotype rather than its genotype. The actual fitness of a particular trial solution "at a particular time" is thus dependent on both the genotype and the environment at that time (and perhaps for a certain period of time before that as well). Once the members to be replaced have been determined, the parents of the new members have to be selected. While this could be done at random, applying Darwin's principal of "mating of the fittest" it is in general better to select the fittest members (or at least let the fittest members have a greater chance of being selected). Selecting parents based on their fitness causes an effect called selective pressure. Certain trial solutions get reproduced more often than others, and this causes the population to converge to those high fitness solutions. The stronger the preference for fitter parents is, the higher the selective pressure. The danger of too high a selective pressure is premature convergence in a local optimum due to loss of genetic diversity in the population. Note also that if both selecting the members to be replaced and selecting the parents of the new members are done at random, the whole process boils down to a very complex way of performing simple random search. This is obviously not advisable.

There are many methods for selecting parents, two of which will now be briefly described. The first is roulette wheel selection, the second tournament selection. Suppose we want to select *n* parents. With roulette wheel selection first the roulette wheel gets divided proportional to the fitness of the members, then *n* individuals are selected by either spinning a 1-slot wheel *n* times, or spinning an *n*-slot wheel one time. With tournament selection a predetermined number of individuals are uniform randomly chosen from the population and the best placed into the mating pool. This is repeated until the mating pool has been filled.

Having selected the parents the next step is to create offspring. The simplest method is to take a copy of a parent and mutate it slightly. Another very popular method is to apply cross-over to two parents, creating a new member by taking some genes from the one parent and some from the other. This completes the circle and can continue until the perfect solution has been found or the process is terminated accepting the best found solution so far. There are a number of parameters influencing the process. One is, as already mentioned, selective pressure. Others include the population size, the number of members to be replaced in each generation, the size of the mating pool, the mutation rate, etc. Finding the optimal values for these parameters is extremely difficult; one method is to include the parameters in the evolutionary process.

Approaches to evolutionary computation

There are three traditonal lines of investigation in evolutionary computation, namely, genetic algorithms, evolution strategies, and evolutionary programming [Fogel95: paragraph 3.7, page 103]. While these are all evolutionary algorithms consisting essentially of the components previously described, there are also some striking differences, both in their technical implementation and the philosophies behind them. For

example, the traditional method of coding in genetic algorithms is binary, while evolution strategies and evolutionary programming traditionally use real valued representations. In genetic algorithms there is a great deal of emphasis on genetic operators such as cross-over and inversion, while mutation is only a background operator. This is in shrill contrast to evolution strategies, where mutation is the main operator, and evolutionary programming, where mutation is the only operator. From a philosophical standpoint one can argue that genetic algorithms work at the level of chromosomes, evolution strategies at the level of the individual, and evolutionary programming at the level of the species [Fogel95: paragraph 3.7, pages 103-104].

While these three approaches to evolutionary computation have long been developing independently from each other, one sees increasingly ideas of the one being applied in the other. Also, the variation in implementation methods is growing rapidly. For example, genetic algorithms using floating point representations have become common, and evolution strategies are now also being applied to discrete parameter optimization. The blurring of these three approaches should help focus attention on the unifying theory behind evolutionary computation. Hopefully this paper can contribute in a small way by describing a case study (in paragraph 5.1) where evolutionary computation was used without regard for any specific approach, but rather tailoring a problem specific method using components from various evolutionary algorithms.

For an in depth introduction to evolutionary computation, see for instance [Michalewicz94]. In [Fogel95] it is argued that evolutionary computation is a necessary requirement for any intelligent system: "*Every system that incorporates the evolutionary processes of reproduction, mutation, competition, and selection, by whatever means, inherently evokes inductive learning. Intelligence is not the end result of evolution; rather, it is woven into the process itself. Intelligence and evolution cannot be separated.*"

5. Case studies

5.1. A trigram based Adaptive Information Filtering system

Synopsis

In this paragraph a case study undertaken by the author will be discussed in which a system as proposed in paragraph 1.4 was implemented using several methods described in chapters 2, 3, and 4. The experiments were conducted in a real world environment, namely the filtering of Internet news.

5.1.1 - Introduction

What is Internet news?

Internet news is a hierarchical system of discussion groups, each concerned with a specific topic. One can participate by sending a message to any particular group, either in response to an already posted message or to introduce a new topic (which should of course be a sub-topic of the general topic of the group one is sending to). That message will then be distributed to all the computers in the world which are set up to receive that newsgroup. In turn others then have the chance to react to the new message.

Example

```
Group: comp.lang.java.announce
        |     |     |         |
        +-- top level hierarchy for computer related items
              |     |         |
              +-- hierarchy for items related to a specific prog. language
                    |         |
                    +-- hierarchy for items related to the Java prog. language
                              |
                              +-- specific topic: announcements
```

So this group has as topic, announcements relating to the Java programming language[1].

There are a number of problems associated with Internet news. First of all one has to find the appropriate newsgroup. If your topic does not have a dedicated newsgroup it may be split over a large number of groups. Another problem is that the more popular groups have such a high traffic rate that finding the information you are interested in can be very time consuming. To deal with these problems the ideal system would be one which can rearrange messages according to the specific topic interests of the user and filter out all non-relevant data. This is precisely what an AIF system attempts to do.

1. for another example see [Tauritz96b], page 12

Overview of the case study

The goal of the case study was to ascertain if constructing the AIF system as described in paragraph 1.4, using the proposed combination of weighted trigram analysis, incremental clustering, and evolutionary computation for parameter optimization, is viable and worth further research.
In [Tauritz96a] it was shown that weighted trigram analysis has sufficient discriminating power to separate a small static set of documents belonging to two topics. In this case study the more ambitious aim of demonstrating that it has sufficient discriminating power to separate a dynamic stream of documents belonging to multiple topics was assumed. To this end it was necessary to design and implement a system capable of incrementally clustering a dynamic stream of documents with the capability of dynamic parameter optimization in a non-stationary environment. For such a system to be successful two important properties are further necessary, namely, scalability of the number of topics to be separated and sufficient generalizing power of the weighted trigram analysis not to degrade the accuracy of the clustering too much when new documents are presented with which the system was not trained. And last but not least, the system must also be able to incorporate user feedback.
The case study consisted of two parts. The first without user feedback, the second with. There were two main reasons for this. First of all, to be able to measure the performance of the system many thousands of documents needed to be presented which is not feasible if user feedback is required for all of them. And secondly, the system first needs to be trained to reach an adequate level of performance before it becomes operational. In the following three sub-paragraphs first the basic architecture of the system will be discussed in depth followed by the specifics of each part, including results.

5.1.2 - Basic architecture

Retrieval

The Internet news articles were acquired in this experiment via a TCP/IP (Transmission Control Protocol/Internet Protocol) connection, employing a TCL (Tool Command Language) script[1] to retrieve them from a NNTP (Network News Transfer Protocol) server. The portability of the experiment is greatly enhanced by the fact that TCL interpreters are available for many platforms, including PC's, MAC's, and various flavors of UNIX. The source code of the TCL script employed is included as Appendix A.

Preprocessing

To facilitate the processing of the retrieved documents, which are in ASCII (American Standard Code for Information Interchange) format, they are first converted to the set of all lower-case letters and the space denominator. This is done by converting all upper-case letters to lower-case, and all non-letters between two letters to one single space denominator. So for instance the sentence "Hello * world!" is converted to "hello world". Only the actual text of the news articles was considered, not the article headers. As last preprocessing step all the words included in the stop list employed are filtered out, thus enhancing the discriminatory power of the clustering algorithm (see subparagraph 2.6.1).

1. for more information on TCL see http://www.smli.com./research/tcl/

The stop list employed is included as Appendix B. After the preprocessing trigram analysis is performed upon the preprocessed files creating document vector files representing the articles.

Datastructures

The system employs many data structures. The most important ones will now be described. In this study the incoming data stream is stored in document vector files after preprocessing upon retrieval. One file is created for each newsgroup article. The first element of such a document file indicates the number of trigrams indexed in the rest of the file. In this case study the number is always 19683 (27^3). The rest of the file holds the absolute trigram occurrence frequencies. Once loaded the document vectors are represented by an array, the first element holds the so-called normalizer, which is simply the sum of all the trigram frequencies, the rest of the elements are the trigram frequencies themselves.
The clustering process is based on Cluster Euclidean (see paragraph 3.1), employing normalized weighted trigram analysis (see paragraphs 2.3, 2.4, and 2.5). The cluster prototype vectors are stored in files quite similar to the document vector files, the only difference being that the trigram occurrence frequencies are not integers indicating the absolute frequencies, but reals indicating the normalized relative frequencies (they are normalized such that the sum of the elements is always equal to one). Once loaded the prototype vectors are stored in two-dimensional arrays, indexed first by cluster, then by trigram. The number of trigrams is stored in a separate variable.
Long term storage of the trial solutions is accomplished by storing them in weight vector files. The weight vector files hold the score, the age, the radius, the learning parameter, the number of weights (equal of course to the number of trigrams), and the weights themselves (integers in this case study). Once loaded the weight vectors are represented by an array which holds the scores, ages, and weights, and two separate arrays for the radius and the learning parameter. These parameters will be discussed in detail in the section on evolutionary computation.
The execution of the program can, in part, be controlled through the use of a parameter file which permits specification of the range of the weights, the size of the population, the number of members to be replaced in each generation, the age at which members become electable for replacement, and a parameter indicating the selective pressure which ranges from zero to one. These parameters will be discussed in detail in the section on evolutionary computation.

Evolutionary computation

Selective pressure was implemented through the use of exponential ranking, as employed in the experiments described in [Tauritz96a]. First the member with the highest fitness gets a chance to be selected. Selection is performed by choosing a random number between zero and one and comparing it with the selective pressure parameter. If it is smaller the current member is selected, otherwise the member with the next highest fitness becomes the current member and the process repeats by again choosing a random number between zero and one. The selective pressure parameter was set at 0.1 for the experiments conducted in this case study as the results of [Tauritz96a] indicated that this was a good choice.
The only genetic operator used was mutation. Crossover was not used because [Tauritz96a] indicated no benefits, a view supported by [Fogel95, subparagraph 4.4.3].

To keep track of how the population of trial solutions was doing, the lowest, average, and highest fitness values of the population were tracked. After a batch of new news articles was retrieved, they were processed one by one. For each article all the trial solutions were evaluated. The first problem encountered was deciding how to assign fitness values to the population members. The first approach tried was assigning a value of 100 both to the members of the initial population and to all newly created members. For a correct clustering the value of a member was increased by 1, for an incorrect clustering it was decreased by 1. Using this model there are several ways to implement evolution, two of which are:

- After each article has been processed by all population members replace a fixed percentage, for instance those with the lowest fitness values, by mutations of members of the population, for instance those with the highest fitness values.
- After each article has been processed by all population members use a cut-off value, for example 90, below which all members are replaced by, for instance, those with the highest fitness values.

Both have however major drawbacks, namely:

- If over time the fitness values associated with the population members increase, as one would of course hope, the new members will always be immediately replaced, bringing evolution to a dead end.
- A new member could be unlucky and score badly to begin with and get replaced on that account, while actually it was a good solution. And, if all members are scoring better than 50% on average, the chance exists that none will be replaced because none will drop under the cut-off value.

Aging

To overcome these problems a slight modification was introduced. The members of the initial population were still initialized at fitness 100, but newly created members were assigned the average fitness. This did solve the problem of the new members being replaced immediately, but also introduced new problems. One was that this caused lucky new members to replace older members when this was not desirable. For example, if the average fitness is 130, and the highest fitness is 140, then a new member might get lucky and by scoring a few times success replace a much older member which was temporarily unlucky. The other problem has to do with the artificial upward pressure caused by replacing underachievers by new members starting off at the average fitness. Solving these new problems was more complicated, as no obvious solution was apparent. The key to the solution was in realizing that, to properly take into account the higher statistical reliability of the solutions represented by the older members, one has to separate score and age, and define the fitness as a function of both. An obvious function is:

fitness = score / age

All newly created members are initialized with score and age both set at zero. For each clustered document the age gets increased by one, and the score either with one if the clustering was correct, otherwise with zero. This means that the score is never larger than the age, which causes the nice property that the range of the fitness values is limited to the interval [0,1]. The value is 0 when the member in question has incorrectly clustered all documents processed so far, and 1 when the member has made no mistake in clustering as yet. This function also has the nice property that the fitness of members of various ages can be meaningfully compared.

Two pool strategy

Still not all is well, as the following example will demonstrate. Imagine you have two softball teams. One old and proven with a ratio of 90% of all games played won and with over a hundred games played so far, the other a new and unproven team, with the same ratio of 90%, where only ten games have been played so far. Clearly the proven team is much more reliable, the new team might simply have been lucky a couple of times. This is, however, not reflected in the fitness function, and it is not obvious how this should be taken into account. The solution adopted was splitting the population into two pools, one for the new unproven members with ages below a certain value, and the other for the adults. The lowest, average, and highest fitness values of the population are now defined as the lowest, average, and highest fitness values of the adult pool. This strategy seems to work quite well, although the question remains what the connection is between the age distribution of the population and the statistical reliability of the fitness values of the members. While high age certainly indicates high reliability, it is certainly not so that this can simply be reversed. This is because the children are slight mutations of the better performing adults and, therefore, can be expected to, in general, perform quite well themselves.

5.1.3 - Dynamic topic separation experiments

In the experimental setup without user a substitute was necessary to determine if a specific clustering was correct or not. This was done by creating a file ‘batch.ind’ when retrieving a batch of news articles, in which, for each article, the newsgroup from which it had been retrieved was indicated. All the trial solutions were evaluated for each article by using them to cluster that article. The new score was determined by comparing the assigned cluster with the correct one. Two important assumptions were made. First of all, that the articles retrieved from a specific newsgroup showed greater similarity to each other than to articles from the other newsgroups. And secondly, that all the articles belonging to a particular newsgroup were suited to be clustered together. To fulfill these assumptions as accurately as possible newsgroups were chosen for the experiments which were moderated. Moderated newsgroups are newsgroups for which articles cannot be posted freely, but must be approved by a human moderator who tries to filter out non-relevant postings. One experiment was done including an unmoderated newsgroup and the effect was dramatic as will be shown later in this paragraph.
The experiments were initialized by creating the start clusters through computing document vector averages of the first *n* documents (values tried for *n* ranged from 10 to 30), and the weight vectors at random but within ranges equal to those used in the parameter file. The basic algorithm of these experiments is shown in fig. 5.1.3a.

Algorithm

```
For each article {
   For each member {
      Apply clustering algorithm with current member to the current
         article to update the fitness of the current member, but
         without adaptation of the prototype vectors
   }
   Apply clustering algorithm with the fittest member to the current
      article to adapt the prototype vectors
}
```

Figure 5.1.3a. Dynamic topic separation algorithm

Figures 5.1.3b1 to 5.1.3f2 show the graphs of all the experiments conducted. The graphs are offered in sets showing the average fitness and the highest fitness of the population. The x-axis indicates the number of the article being presented, the y-axis the average or highest population fitness.

In the first set the results of the *two newsgroup experiment* are shown. The task was to split the newsgroups misc.news.bosnia (moderated news about Bosnia) and misc.news.southasia (moderated news about south Asia). After presenting the training set (200 articles) about ten times the system seems to converge to an average accuracy rate in excess of 90% (fig. 5.1.3b1), the highest fitness even to around 100% (fig.5.1.3b2).

The *three newsgroup experiment* had as task to split the newsgroups misc.news.bosnia, sci.military.moderated (moderated discussions of military warfare), and comp.os.os2.announce (moderated announcements concerning the operating system OS/2). After presenting the training set (200 articles) about ten times the system seems to converge to an average accuracy rate in excess of 90% (fig. 5.1.3.c1), the highest fitness to around 95% (fig. 5.1.3c2).

The next set shows the results of the *four newsgroup experiment*. The task was to split the newsgroups misc.news.bosnia, misc.news.southasia, comp.os.os2.announce, and comp.lang.java.announce (moderated announcements concerning the programming language Java). After presenting the training set (200 articles) a large number of times, the system seems to converge to accuracy rates in excess of 90% (fig. 5.1.3d1 & fig. 5.1.3d2), with the exception of dips from time to time which will be discussed in paragraph 6.1. Notice that the time to converge is quite a bit higher than with the previous experiments with fewer newsgroups.

To see what would happen when an unmoderated newsgroup with a very low S/N ratio is presented to the system, the *unmoderated newsgroup experiment* was conducted (400 articles). The task was to split the newsgroups misc.news.bosnia, misc.news.southasia, and comp.os.ms-windows.nt.setup.hardware (unmoderated news about setup problems with Microsoft's Windows NT). As expected the addition of an unmoderated newsgroup drastically lowered the accuracy rate (fig. 5.1.3e1 & fig. 5.1.3e2).

The final experiment was to determine if the system is capable of generalizing from what it has learned during the processing of a training set, and applying that knowledge to separating a test set. To this end the weight vector files produced by the *three newsgroup experiment* were taken as starting point, and then a test set (200 articles) offered, consisting almost entirely of new documents retrieved from the same newsgroups as used in the *three newsgroup experiment*. The results (fig. 5.1.3f1 & fig.5.1.3f2) indicate accuracy comparable to the training set.

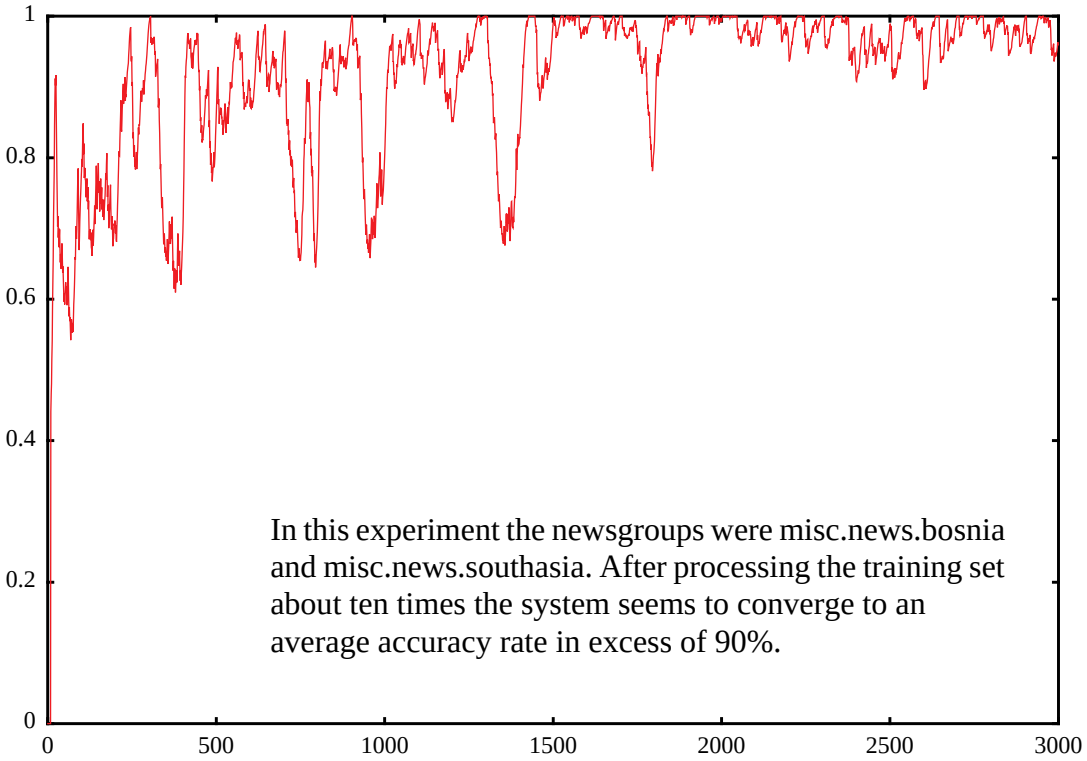

Figure 5.1.3b1. Two newsgroup experiment: Average fitness values

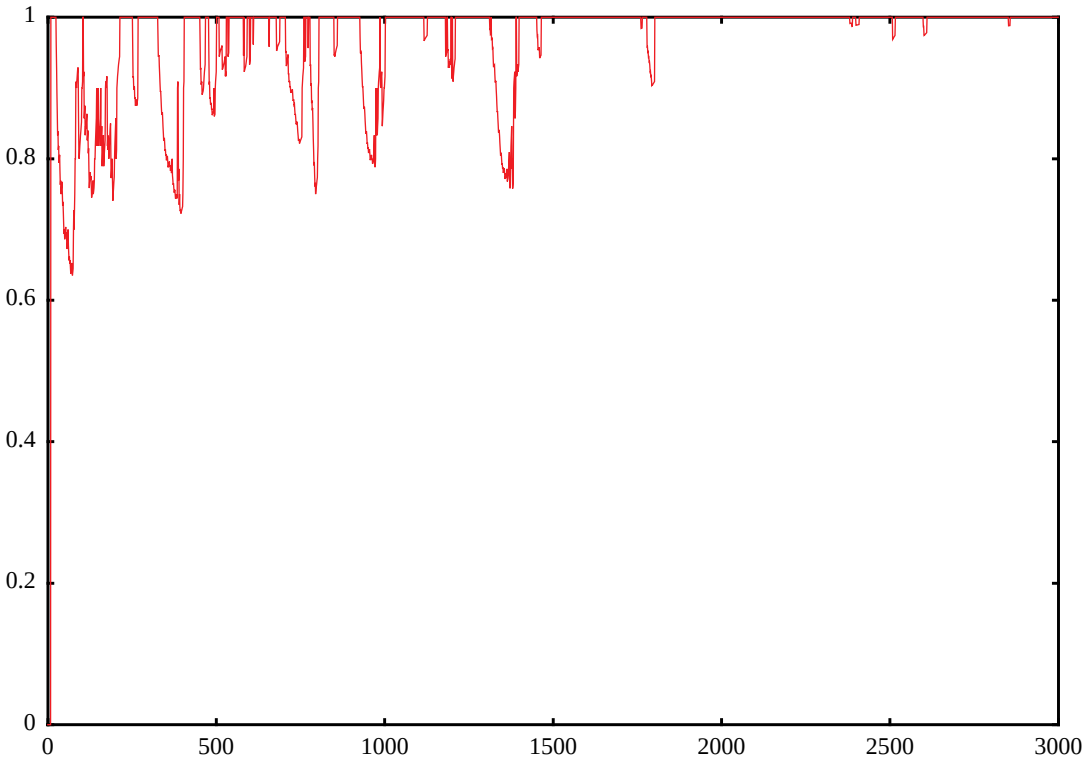

Figure 5.1.3b2. Two newsgroup experiment: Highest fitness values

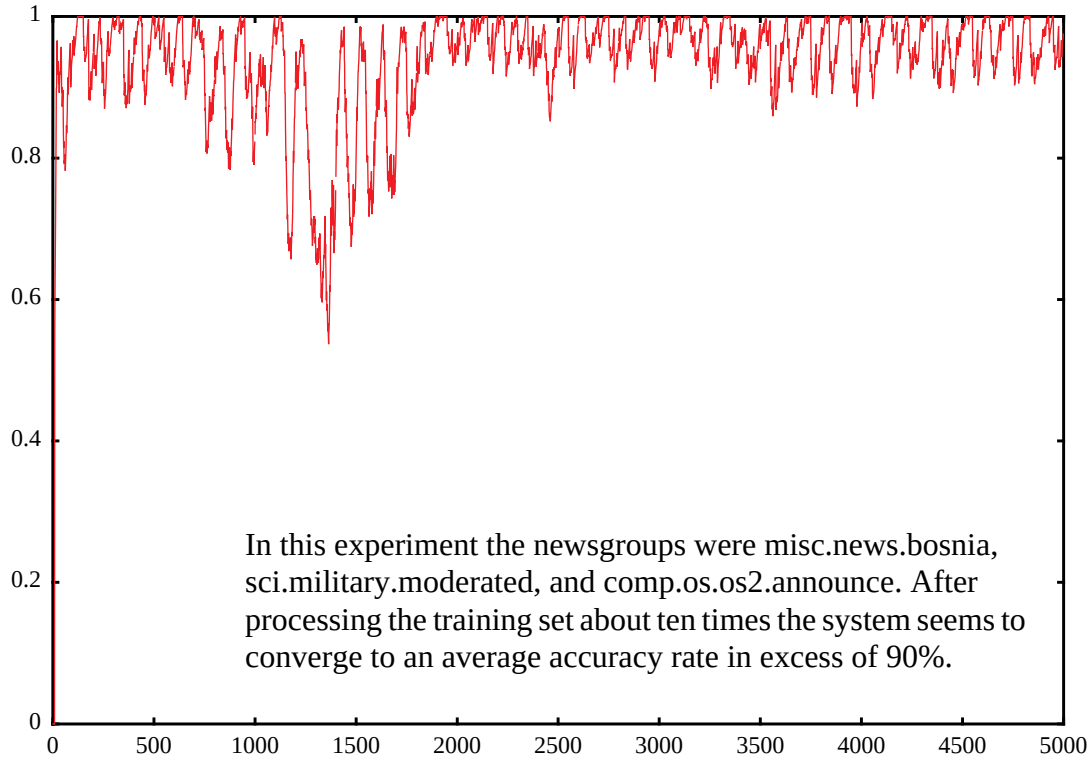

Figure 5.1.3c1. Three newsgroup experiment: Average fitness values

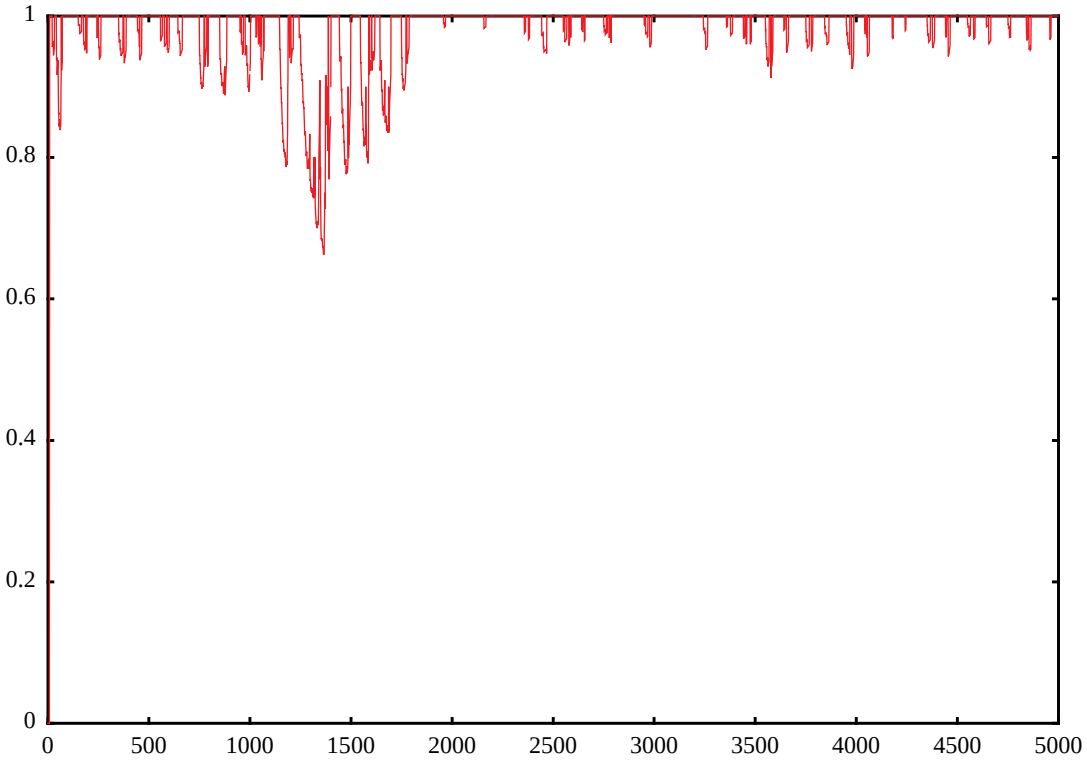

Figure 5.1.3c2. Three newsgroup experiment: Highest fitness values

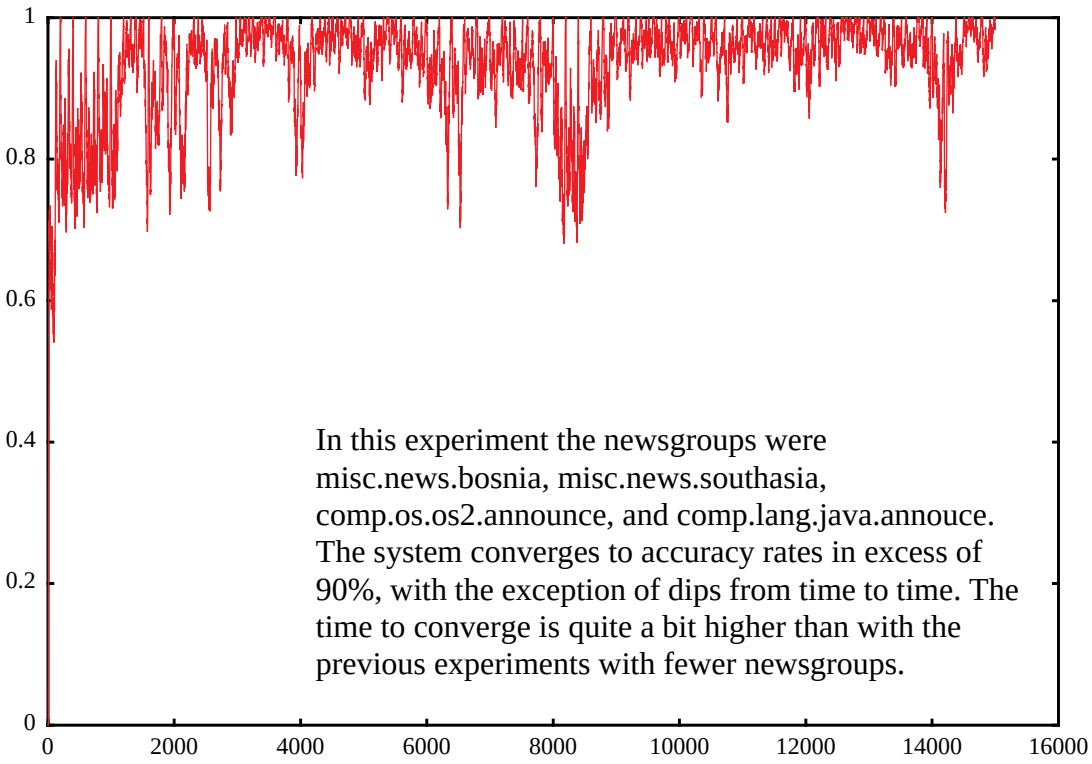

Figure 5.1.3d1. Four newsgroup experiment: Average fitness values

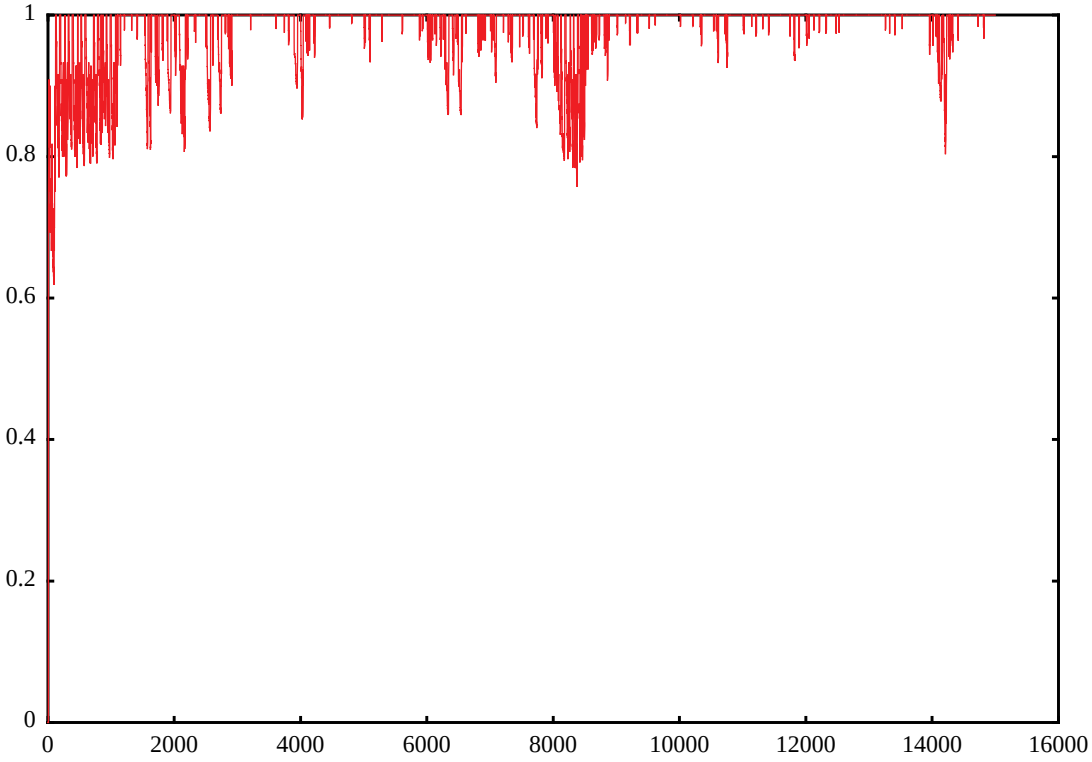

Figure 5.1.3d2. Four newsgroup experiment: Highest fitness values

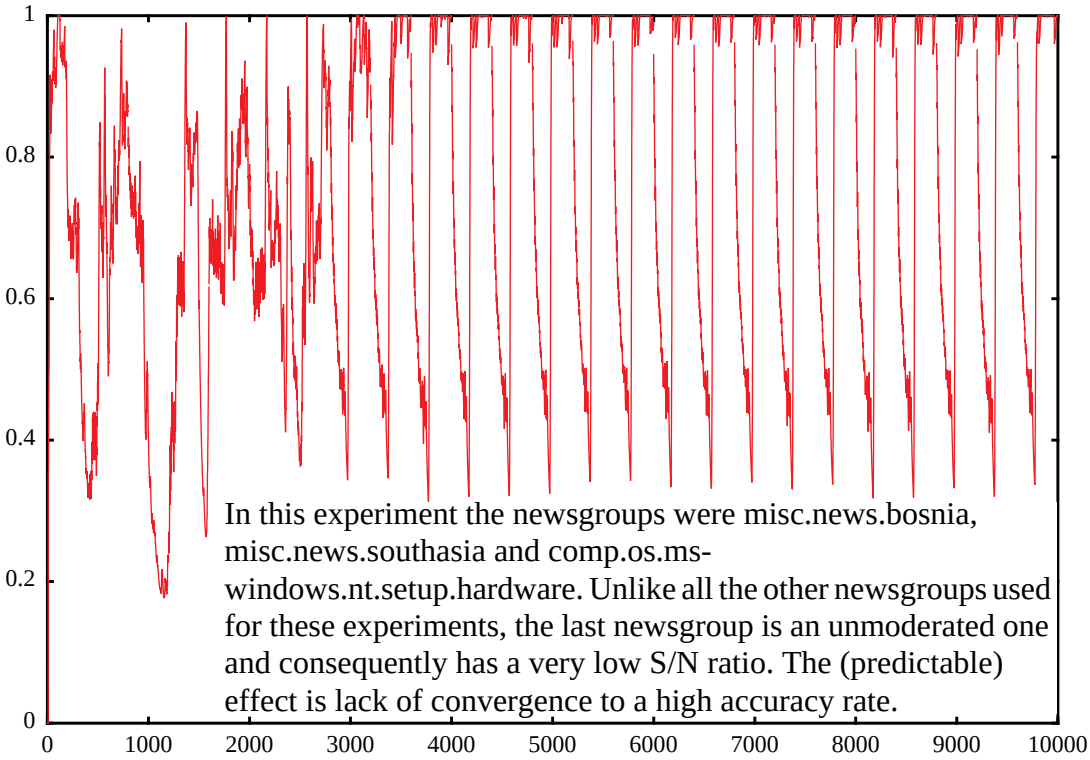

Figure 5.1.3e1. Unmoderated newsgroup experiment: Average fitness values

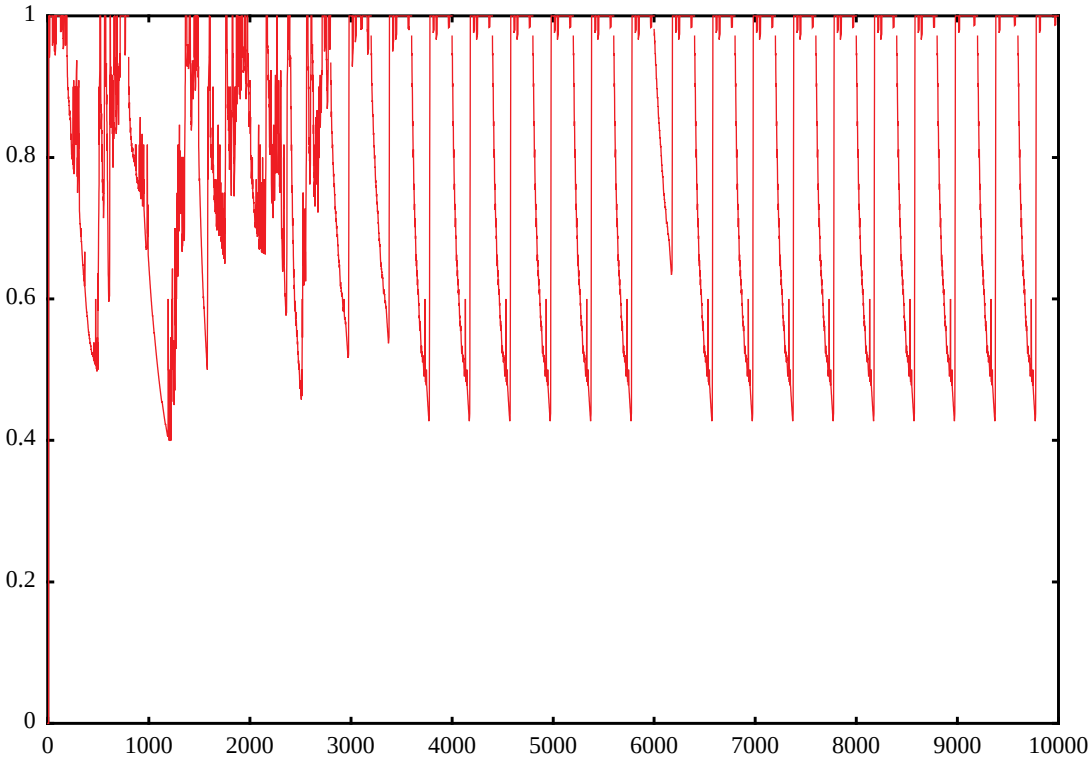

Figure 5.1.3e2. Unmoderated newsgroup experiment: Highest fitness values

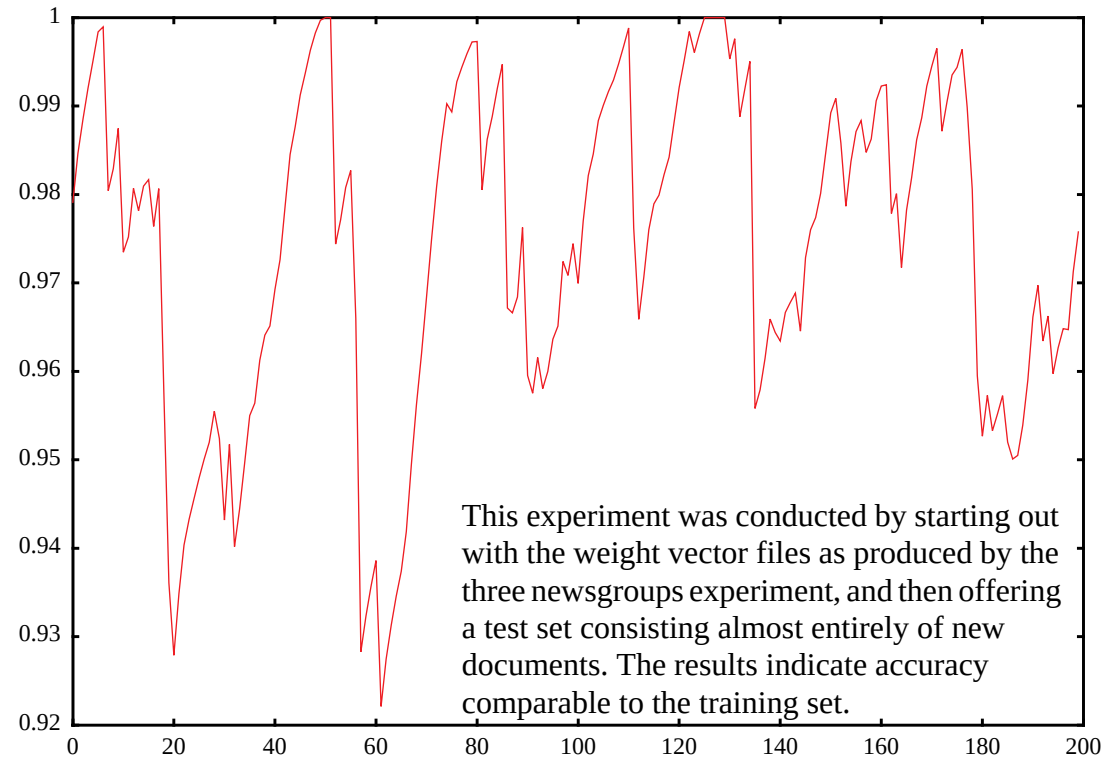

Figure 5.1.3f1. Test set experiment: Average fitness values

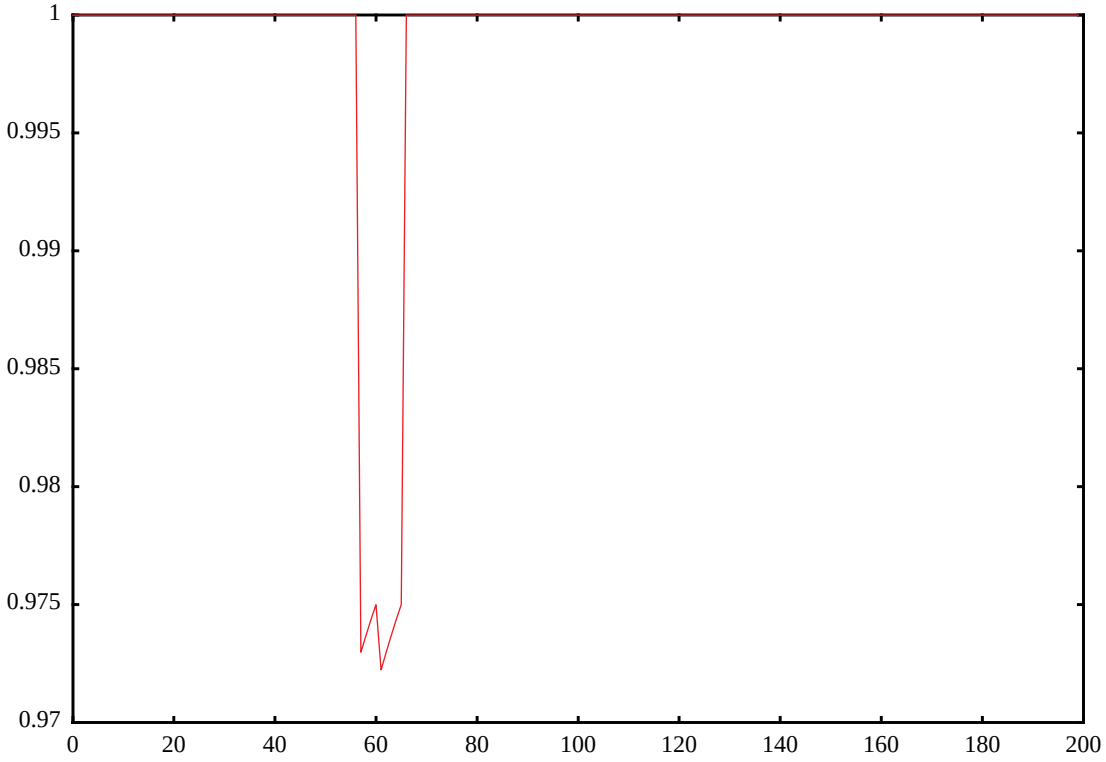

Figure 5.1.3f2. Test set experiment: Highest fitness values

5.1.4 - Demonstrator

The demonstrator provides a very simple user interface with the minimal functionality necessary to demonstrate the principle of incorporating user feedback. It was created using TK, a graphical toolkit based on TCL, which is also portable from PCs to MACs to various UNIX platforms. The TK script which constitutes the user interface of the demonstrator is included as appendix D. The user can choose between two actions, namely *fetch* and *view*. Fetch causes the system to retrieve all the new articles of the predetermined newsgroups available in a particular experimental setup of the demonstrator, and then cluster them using the member with the highest fitness at that moment. Selecting view causes a listbox to appear showing the available clusters to choose from. Double clicking with the left mouse button on one of the cluster descriptions brings up a listbox with the articles to choose from. Selecting an article brings up a viewer with scrollbar displaying that article, and offering two buttons to exit again, one being *approve*, the other *reject*. This provides the user with a very simple way of giving feedback to the system. If the user thinks the article has been clustered appropriately then (s)he exits by clicking the approve button, if not (s)he exits by clicking the reject button. This information is then used to evaluate the population members. The basic algorithm is given in fig.5.1.4a.

Algorithm

```
Action: fetch news
    Retrieve batch of new news articles
    For each article  {
        Cluster using fittest member
    }

Action: view article
    User indicates approval or rejection of clustering
    For each member {
        Apply clustering algorithm with current member to the article
        being viewed to update the fitness of the current member. Score
        is increased if one of the following two possibilities holds:
        1. With this member the article is clustered the same as shown
            to the user, and the user indicated approval
        2. With this member the article is clustered different then what
            is shown to the user, and the user indicated rejection
    }
```

Figure 5.1.4a. Basic demonstrator algorithm

The initialization necessary before the demonstrator is operational is quite complex because the user will expect “reasonable” performance immediately, not after many thousands of evaluations. To this end the experimental setup of the demonstrator was initialized by supplying it with the population resulting from one of the dynamic topic separation experiments.

Fig.5.1.4b contains a screenshot of the demonstrator in operation.

Demonstrator

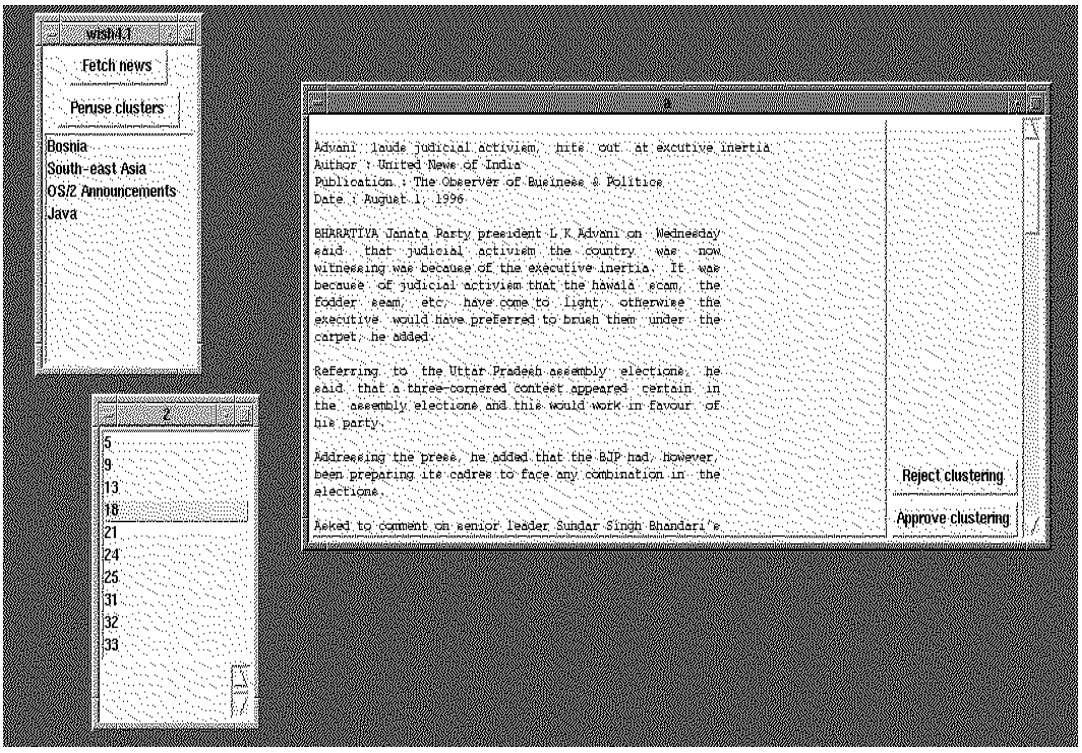

Figure 5.1.4b. Screenshot of demonstrator in operation

5.2. A keyword based Adaptive Information Filtering system

A different approach to designing an AIF system was taken in [Sheth94]. While that system cannot really be compared to the one proposed in paragraph 5.1, because it was specifically designed to filter Internet news and not incoming textual data streams in general, there are enough similarities to make it worth investigating.
The basic idea of Sheth's system is as follows. A dynamic set of information filtering interface agents is maintained. Each agent is responsible for satisfying a specific information need of the user. An agent consists of a population of profile individuals. When processing the incoming information stream documents are matched against the profiles, and top scoring documents are presented to the user. The scores are based on how well the documents matched a profile, and on that profile's fitness. The user can provide positive or negative feedback for a document. The profile which presented the document gets modified based on the relevance feedback of the user. Also, the fitness of that profile increases or decreases for positive and negative feedback respectively. The populations of profile individuals are all evolved separately.
There are a number of striking similarities and differences between the two systems. They both try to specialize to the current interests of the user and adapt to the user's changing information needs over time, minimizing the amount of feedback the user has to give for the system to perform adequately. They also both use evolutionary computation for parameter optimization. The representation of the documents is, however, completely

different. Sheth's system uses keywords as opposed to trigrams. Another major difference is in the way separate topics are treated. Instead of using a clustering algorithm, Sheth's system maintains a set of topics with for each topic a set of profiles which are matched with the documents in the newsgroups associated with that topic during the filtering process. Each set of profiles is a separate population and is separately evolved. Each profile is allowed to contribute articles to the set presented to the user, the number each may contribute is dependent on how well it matched with each article and its fitness. While the system described in paragraph 5.1 is in principle more powerful than Sheth's system because it does not have the drawbacks of a keyword based system (see paragraph 2.2) and because it is not restricted to use with newsgroups, both systems seem to share as weakness the problem of dealing with newsgroups with low S/N ratios. Sheth's system is, however, not welded to a keyword based approach and could thus probably be combined with a trigram based approach. Both systems illustrate interesting ideas which should be further explored.

6. Concluding

Synopsis

In this chapter conclusions will be drawn as to the effectiveness of the proposed approach of using trigram analysis, clustering, and evolutionary computation to create an AIF system. Also, suggestions for future research based on the insights gleaned during the undertaken research will be given.

6.1. Conclusions

In [Tauritz96a] it was demonstrated that the combination of trigram analysis, clustering, and evolutionary computation was sufficient to separate a static set of textual documents. The results as presented in paragraph 5.1 of this thesis show that an AIF system based on this approach is also capable of accurately separating a dynamic stream of documents. Furthermore, the experiments with a varying number of clusters indicate that increasing the number of clusters only effects the time needed to converge, not the accuracy rate, which would mean that the system is scalable. Also, the experiment with a test set as described in paragraph 5.1 seems to indicate that after sufficient training the system is capable of processing untrained documents with an accuracy rate comparable to that of processing the trained documents. This means that the system successfully generalizes.

The system in its present state does have a number of weak points. First of all, it is very sensitive to the quality of the presented documents, as the experiment with the unmoderated newsgroup demonstrated. One has to take into consideration, however, that in the dynamic topic separation experiments the system was not allowed to add new clusters. In an experiment where this was allowed it might very well be that the “junk” documents would be clustered separately and the accuracy rate not or only slightly degraded. A second problem is that interpreting the performance statistics is extremely difficult because the evolutionary process is so dynamic. A possible solution might be to track the system accuracy rate as opposed to the accuracy rates of the separate population members. Another problem that needs investigating is the occurrence of “dips” in the accuracy level. These are probably in part due to the varying quality of the documents being processed, but randomness also seems to play a role, possibly indicating an instability due to the population getting stuck in local optima from time to time. A possible solution might be to decrease the amount of mutation for members with a high fitness.

Further research is needed to conclusively demonstrate the scalability and generalizing power of this system, and to ascertain if the weak points identified above can indeed be overcome as suggested. Many possible improvements will be suggested in the next paragraph. To conclude, the initial incarnation of this AIF system shows great promise. Clearly further research is warranted.

6.2. Future research

There are a number of areas which could benefit from further research. These are in the nature of the document representation, the clustering algorithm, the evolutionary computation model, larger experiments to further validate the scalability and generalizing power of the system, and the user interface. Some proposals for each area will now be suggested.

Representation

- In paragraph 2.6 various preprocessing techniques were introduced, of which only the use of stop lists has currently been incorporated into the system. The incorporation of conflation and stemming is a logical next step which is unanimously recommended in the literature as having a positive effect on the discriminating power of syntactical transformations.
- It might be possible to increase the discriminatory power by switching to quadgram analysis and compensating for the huge increase in memory requirements by using only a subset of all possible quadgrams. Two ways one could restrict the subset are: 1. Only a minority of all quadgrams are used in a specific language, so determining the relevant quadgrams could be quite beneficial. 2. In [Rijsbergen79] some ideas of Luhn were discussed which come down to eliminating all words in a document which occur more often than a certain number of times and all words occurring less often than a certain number of times. Extrapolating Luhn's ideas from words to quadgrams would allow a significant reduction; certainly eliminating the upper-limit quadgrams should increase the discriminatory power
- If very large documents take too long to process one might consider speeding up the process by only performing trigram analysis upon a sample of the document.

Clustering

- In the current implementation of the system a very simple clustering algorithm is employed. A more advanced variation which allows each cluster to have its own radius would probably improve the performance of the system because topics generally have different degrees of specialization which cannot currently be addressed because of the uniform cluster radius.
- If an algorithm is adopted where each cluster has its own radius, it might be worth investigating what the effect would be of allowing the clusters to change size, and even split when a certain limit has been reached. Also, two clusters could perhaps be allowed to merge if they come close enough to each other.
- The use of neural networks (see paragraph 3.3), especially in combination with parallel hardware, could be quite beneficial.
- The current algorithm does not allow hierarchical clustering, while that would better reflect the different degrees of specialization within a particular topic hierarchy (as opposed to each cluster having its own radius which pertains to different degrees of specialization of completely different topics).
- It might be beneficial to allow documents to be assinged to multiple clusters.

Evolutionary computation

- To allow a more refined search of the solution space the weight representation could be changed from integers to reals.
- As mentioned in paragraph 4.2 finding the right system parameters is extremely difficult. One solution is to allow self-adaptation of the system parameters by encoding them in the trial solutions. For more information on self-adaptation see for example [Fogel95].
- In the evolutionary computation model currently being used a number of population members get replaced after each document for which all the trial solutions have been evaluated. The children which are added in their place start with score and age zero, thus loosing the fitness information of their parent. This might negatively effect the evolution because greater age is the only guarantee of statistical reliability of the fitness value of a member. An alternative evolutionary computation model could overcome this problem by, for instance, mutating all the members of the population based on their fitness (the lower the fitness the more mutation should occur) and replacing the weakest few members with copies of the best few members.

Validation

- To be better able to measure the performance of the system the accuracy of the system as a whole should be tracked as opposed to a population based accuracy measure.
- To conclusively demonstrate scalability larger experiments with many more clusters must be performed and evaluated.
- To conclusively demonstrate the generalizing power of the system this experiment should be repeated many times under various conditions and preferably with a larger test set.

User interface

The current demonstrator obviously needs to be greatly enhanced in order to serve as a decent user interface. Some functions which it would be beneficial to incorporate are:

- User control of the selected sources for the incoming data stream.
- Ranking the documents within the clusters based on how relevant they were to the user. This implies that some method must be provided so that the user can specify this.
- Allowing the user to provide additional feedback by specifying the correct cluster for documents where the clustering was rejected, or to add new clusters.

Bibliography

Backer, Eric (1995). *Computer-assisted reasoning in cluster analysis*, Prentice Hall.

Bartfai, Guszti (1995). *An ART-based Modular Architecture for Learning Hierarchical Clusterings*, Technical Report CS-TR-95/3, Department of Computer Science, Victoria University of Wellington, New Zealand, Feb.'95.

Beauregard, Raymond A., Fraleigh, John B. (1990). *Linear Algebra*, 2nd edition, Addison-Wesley.

Belkin, Nicholas J., Croft, W. Bruce (1992). "Information Filtering and Information Retrieval: Two Sides of the Same Coin?", *Communications of the ACM*, December 1992, Vol.35, No.12, pp.29-37.

Boyce, Bert R., Meadow, Charles T., Kraft, Donald H. (1994). *Measurement in Information Science*, Academic Press.

BYTE (1992). "Managing Infoglut", Volume 17, Number 6, June 1992, McGraw-Hill.

Carpenter, Gail, Grossberg, Stephen (1987). "A Massively Parallel Architecture for a Self Organizing Neural Pattern Recognition Machine", *Computer Vision, Graphics, and Image Processing*, 1987, Vol.37, pp.54-115.

Carpenter, Gail, Grossberg, Stephen (1988). "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network", *IEEE Computer*, March 1988

Cooper, Wm.S. (1994). "The Formalism of Probability Theory in IR: A Foundation or an Encumbrance?", *Proc. of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Dublin, Ireland, pp.242-247.

Dawson,J.L. (1974). "Suffix removal and word conflation", *ALLC Bulletin*, Vol.2, No.3, pp.33-46.

De Heer, T. (1982). "The Application of the Concept of Homeosemy to Natural Language Information Retrieval", *Information Processing and Management*, Vol.18, No.5, pp.229-236.

Fogel, David B. (1995). *Evolutionary computation: toward a new philosophy of machine intelligence*, IEEE Press.

Freeman, James A., Skapura, David M. (1991). *Neural Networks: Algorithms, Applications, and Programming Techniques*, Addison-Wesley.

Goldberg, David, Nichols, David, Oki, Brian M., Terry, Douglas (1992). "Using Collaborative Filtering to Weave an Information Tapestry", *Communications of the ACM*, December 1992, Vol.35, No.12, pp.61-70.

Heins, Lucien G., Tauritz, Daniel R. (1995). *Adaptive Resonance Theory (ART).: An Introduction*, Internal Report 95-35, Department of Computer Science, Leiden University, The Netherlands, ftp://ftp.wi.LeidenUniv.nl/pub/CS/StudentReports/ir95-35.ps.gz.

Hertz, John, Krogh, Anders, Palmer, Richard G. (1991). *Introduction to the theory of neural computation*, Addison-Wesley.

Honkela, Timo, Kaski, Samuel, Kohonen, Teuvo, Lagus, Krista (1996). *Newsgroup Exploration with WEBSOM Method and Browsing Interface*, Report A32,

Laboratory of Computer and Information Science, Faculty of Information Technology, Helsinki University of Technology, Finland.

HP User (1996). "News: Web To Dish Databases?", Vol.13,No.7, July/August 1996.

Kohonen, Teuvo (1990). "The Self-Organizing Map", *Proceedings of the IEEE*, Vol.78, pp.1464-1480.

Lennon, M., Pierce, D.S., Tarry, B.D., Willet, P. (1981). "An evaluation of some conflation algorithms for information retrieval", *Journal of Information Science*, Vol.3, pp.177-183.

Lovins, B.J. (1968). "Development of a stemming algorithm", *Mechanical Translation and Computational Linguistics*, Vol.11, pp.22-31.

Lovins, B.J. (1971). "Error evaluation for stemming algorithms as clustering algorithms", *Journal of the American Society for Information Science*, Vol.22, pp.28-40.

Maes, Pattie (1994). "Agents that Reduce Work and Information Overload", *Communications of the ACM*, Vol. 37, No.7, July 1994.

Michalewicz, Zbigniew (1994). *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd extended edition, Springer-Verlag.

Moore, Barbara (1989). *ART 1 and Pattern Clustering*, pp.174-185.

Paice, Chris D. (1990). "Another Stemmer", *SIGIR ???*.

Porter, M.F. (1980). "An algorithm for suffix stripping", *Program*, Vol.14, pp.130-137.

Rijsbergen, C.J. van (1979). *Information Retrieval*, 2nd edition, Butterworths, London.

Salton, G. (1989). *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley.

Schmidt, Stephanie, Teufel, Bernd (1988). "Full text retrieval based on syntactic similarities", *Information Systems*, Vol.13, No.1, pp.65-70.

Scholtes, Johannes Cornelius (1993). *Neural Networks in Natural Language Processing and Information Retrieval*, Ph.D. thesis, University of Amsterdam.

Sheth, Beerud Dilip (1994). *A Learning Approach to Personalized Information Filtering*, M.Sc. thesis, Massachusetts Institute of Technology, United States of America.

Tauritz, Daniel R., (1996a). *Optimization of the discriminatory power of a trigram based document clustering algorithm using evolutionary computation*, April 1996, Internal Report 96-5, Department of Computer Science, Leiden University, The Netherlands, ftp://ftp.wi.LeidenUniv.nl/pub/CS/StudentReports/ir96-05.ps.gz.

Tauritz, Daniel R., (1996b). *Concepts of Adaptive Information Filtering*, July 1996, Internal Report 96-19, Department of Computer Science, Leiden University, The Netherlands, ftp://ftp.wi.LeidenUniv.nl/pub/CS/StudentReports/ir96-19.ps.gz.

Zavrel, Jakub (1995). *Neural Information Retrieval: An Experimental Study of Clustering and Browsing of Document Collections with Neural Network*s, M.Sc. thesis, University of Amsterdam, The Netherlands.

Appendix A. TCL script for retrieving Internet news from NNTP server

 # NNTP commands - Last revised: March 29th, 1996

```
proc nntp_connect {server output} {
  set f [socket $server nntp]
  fconfigure $f -buffering line   ;# this causes a flush after each newline
  gets $f l
  puts $output $l
  return $f
}

proc nntp_group {f group} {
  puts $f “group $group”
  gets $f l
  return $l
}

proc nntp_subject {server} {
  puts $server “head”
  set l “ “
  while {[lindex $l 0] != “Subject:”} {
    gets $server l
  }
  set subject $l
  while {$l != “.”} {
    gets $server l
  }
  return $subject
}

proc nntp_next {f} {
  puts $f “next”
  gets $f l
  return $l
}

proc nntp_head {server output} {
  puts $server “head”
  set l “ “
  while {$l != “.”} {
    gets $server l
    puts $output $l
  }
}

proc nntp_body {server art_num output} {
  puts $server “body $art_num”
  gets $server l
  while {$l != “.”} {
    gets $server l
    puts $output $l
  }
}
```

```
# Fetch new news - Last revised: August 19th, 1996

# --- Support routines ---

# Load NNTP extensions
source /daniel/tcl/nntp.tcl

# --- Local initialisation ---

set serverdns news.uni-hohenheim.de.
#set serverdns news.uni-stuttgart.de.
set group(1) misc.news.bosnia
set group(2) misc.news.southasia
set group(3) comp.os.ms-windows.nt.setup.hardware

# Read group index files
for {set loop 1} {$loop <= 3} {incr loop} {
  if {[file exists /daniel/demo/group$loop.ind]} {
    # Read article number
    puts “Reading file ‘group$loop.ind’”
    set groupfile($loop) [open /daniel/demo/group$loop.ind r]
    gets $groupfile($loop) groupindex($loop)
    close $groupfile($loop)
  } else {
    # Create topic.art
    puts “The file ‘group$loop.ind’ will be created”
    set groupindex($loop) 0
  }
}

# --- Remote initialisation ---

# Set up connection to NNTP server
puts “Connecting...”
set server [nntp_connect $serverdns stdout]

# Retrieve group info for both newsgroups
set groupheader(1) [nntp_group $server $group(1)]
set groupheader(2) [nntp_group $server $group(2)]
set groupheader(3) [nntp_group $server $group(3)]

# Extract article number info from group headers
puts “Extracting group header information...”
set oldest_art(1) [lindex $groupheader(1) 2]
set newest_art(1) [lindex $groupheader(1) 3]
set oldest_art(2) [lindex $groupheader(2) 2]
set newest_art(2) [lindex $groupheader(2) 3]
set oldest_art(3) [lindex $groupheader(3) 2]
set newest_art(3) [lindex $groupheader(3) 3]

# Did any new articles arrive?
if {($groupindex(1) > $newest_art(1)) && ($groupindex(2) > $newest_art(2)) && ($groupindex(3) >
$newest_art(3))} {
  puts “No new articles have arrived!”
  close $server
  return
}
```

```
# Determine article number of first new articles
for {set loop 1} {$loop <= 3} {incr loop} {
  if {$groupindex($loop) >= $oldest_art($loop)} {
    # Previously retrieved articles are still available and need to be skipped
    set current_art($loop) [expr $groupindex($loop)+1]
  } else {
    # All available news should be retrieved
    puts “Warning! It is possible news has been lost due to expiration.”
    set current_art($loop) $oldest_art($loop)
  }
}

# Open batch index file
set index [open /daniel/demo/batch.ind w]

# Set counter to old number of articles plus 1
set f [open /daniel/demo/article.num r]
gets $f dummy
gets $f oldcounter
close $f
set counter oldcounter
incr oldcounter

# --- Retrieve new articles ---

while {($current_art(1) <= $newest_art(1)) || ($current_art(2) <= $newest_art(2)) || ($current_art(3) <=
$newest_art(3))} {
  incr counter

  # Determine which group to retrieve next article from
  set selection 0
  if {$current_art(1) > $newest_art(1)} {
    set selection 2
  }
  if {$current_art(2) > $newest_art(2)} {
    set selection 3
  }
  if {$current_art(3) > $newest_art(3)} {
    set selection 1
  }
  if {$selection == 0} {
    # Make a random choice between group1 and group2 and group3
    set selection [expr [clock seconds]%3+1]
  }

  # Retrieve article
  puts “Retrieving news from group: $group($selection)”
  set dummy [nntp_group $server $group($selection)]
  puts “Retrieving article: $current_art($selection)/$newest_art($selection)”
  set f [open /daniel/demo/$counter.art w]
  nntp_body $server $current_art($selection) $f
  close $f

  # Update group index files
  set groupfile($selection) [open /daniel/demo/group$selection.ind w]
  puts $groupfile($selection) $current_art($selection)
  close $groupfile($selection)
```

```
  # Update batch index file
  puts $index $selection
  flush $index
  incr current_art($selection)
}

# Update article.num
set f [open /daniel/demo/article.num w]
puts $f oldcounter
puts $f counter
close $f

for {set loop oldcounter} {$loop <= $counter} {incr loop} {
  # Filter article
  exec "/daniel/aif/tools/filter/filter $counter.art $counter.fil"

  # Vectorize article
  exec "/daniel/aif/tools/vector/vector $counter.fil $counter.vec"
}

# Close NNTP server connection
close $server

puts "Session completed"
```

Appendix B - Stop list used in the trigram based AIF system

about	after	again	all	also	always	am	an	and
any	are	as	at	be	been	before	between	beyond
both	but	by	can	did	do	does	each	first
for	from	get	go	had	has	have	here	how
if	in	into	is	it	its	last	let	like
made	many	may	me	mine	more	most	much	my
never	no	not	now	of	off	on	only	or
other	our	ours	out	over	same	should	so	some
such	than	that	the	their	them	then	there	these
they	this	to	too	try	until	use	very	was
we	went	were	what	when	where	which	who	whose
why	will	with	without	would	yes	yet	you	your
yours								

Appendix C. C++ code for dynamic topic separation experiment

```
// Title     : Stream
// Author : D.R. Tauritz
// Revised: September 5th, 1996
//
// Input: Batch index file, cluster definition files, weight vector files,
//            article vector files and parameter file
//
// Output: Cluster definition files, weight vector files and log file
//
// Note: Most array indexing starts at 1

#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <time.h>
#include <string.h>
#include “gausrand.h”

#define VECTOR_SIZE 19686                  // Set for stream experiment (2+27^3)
#define MAX_ARTICLES 1000                 // Set for stream experiment

#define SCORE 0
#define AGE 1

#define MAX_DEVIATION 0.00001

unsigned int num_of_clusters=3;                    // Set for stream experiment

unsigned long int vector[VECTOR_SIZE];       // Article vector being processed

// Population of associated weight vectors
// dimensions: pop_size x VECTOR_SIZE
// weight[][0] = SCORE; weight[][1] = AGE
unsigned long int (*weight)[VECTOR_SIZE];

// Weight vector rank list
// dimensions: pop_size x 2
// rank[i][0] = weight vector index with rank i; rank[i][1] = fitness of rank[i][0]
double (*rank)[2];

// Prototype vectors representing the clusters
// dimensions: num_of_clusters x VECTOR_SIZE
float (*cluster)[VECTOR_SIZE];
float (*backup_cluster)[VECTOR_SIZE];

double *max_radius;      // Max radius for each weight vector
double *labda;           // Labda for each weight vector
unsigned int *index;

unsigned long int num_of_trigrams;
unsigned int max_fitness;

long int *gasseed;               // Needed by gasdev (see gausrand.h)
long int *ranseed;               // Needed by ran1 (see gausrand.h)
```

```
unsigned int num_of_insertions=0;  // Number of elements in sorted list

double lowest_fitness;
double average_fitness=(double)0;
double highest_fitness;

unsigned long int evaluation;

// Variables defined in parameter file
unsigned int pop_size;
unsigned int num_of_replacements;
unsigned int age_limit;
double selective_pressure_rate;
unsigned int min_weight;
unsigned int max_weight;

// Functions ----------------------------------------------------------------

void reverse(char s[])
// From: Kernighan & Ritchie,"The C Programming Language", Prentice-Hall 1978
// Reverses characters in input string
{
  int c, i, j;
  for (i=0,j=strlen(s)-1; i<j; i++, j--) {
    c=s[i];
    s[i]=s[j];
    s[j]=c;
  }
}

void my_itoa(int n, char s[])
// From: Kernighan & Ritchie,"The C Programming Language", Prentice-Hall 1978
// Converts integers to strings
{
  int i, sign;

  if ((sign=n)<0)
    n=-n;
    i=0;
    do {
      s[i++]=n%10+'0';
    } while ((n/=10)>0);
  if (sign<0)
    s[i++]='-';
  s[i]='\0';
  reverse(s);
}

unsigned int select()
// Select fittest member with highest chance, chances of being selected decreasing
// exponentially with decreasing fitness, this causes selective pressure
{
  unsigned int selection=1;
  int selected=0;
  int i=num_of_insertions;
  while ((i >= 1) && (selected==0)) {
    if (ran1(ranseed)<selective_pressure_rate) {
      selection=(unsigned int)rank[i][0];
```

```
      selected=1;
    }
    i--;
  }
  if (i<1)
    selection=(unsigned int)rank[num_of_insertions][0];
  return selection;
}

double dist_to_cluster(unsigned long int v[],float c[],unsigned int weight_vector)
// Returns weighted Euclidean distance between article vector and cluster prototype vector
{
  double norm,div1,div2,sub,weighted_sub,sum;
  norm=(double)v[0];
  sum=0;
  for (unsigned int i=1;i<=num_of_trigrams;i++) {
    div1=((double)v[i])/norm;
    div2=(double)c[i];
    sub=div1-div2;
    weighted_sub=(double)weight[weight_vector][i+1]*sub;
    sum += weighted_sub*weighted_sub;
  }
  return sqrt(sum);
}

void copy_vector(unsigned long int v1[],unsigned long int v2[])
{
  for (unsigned long int i=0;i<=num_of_trigrams+1;i++)
    v2[i]=v1[i];
}

void adapt_cluster(unsigned int group,unsigned long int vector[],unsigned int member)
{
  // Reduce relative slow array lookups
  unsigned long int norm=vector[0];
  double current_labda=labda[member];
  double sum=0;

  // Update all elements
  for (unsigned long int i=1;i<=num_of_trigrams;i++) {
    cluster[group][i]=(float)((1-current_labda)*cluster[group][i]+current_labda*((long double)vector[i]/
(long double)norm));
    sum+=cluster[group][i];
  }

  // Normalize
  for (i=1;i<=num_of_trigrams;i++)
    cluster[group][i]=cluster[group][i]/(float)sum;
}

unsigned int clust (unsigned int member)
// Returns index of assigned cluster
{
  double closest_dist=num_of_trigrams*2500; // why?
  int closest=-1;
  // Find closest cluster
  double distance;
  for (unsigned int i=1;i<=num_of_clusters;i++) {
```

```
    distance=dist_to_cluster(vector,cluster[i],member);
    if (distance < closest_dist) {
      closest=i;
      closest_dist=distance;
    }
  }
  // Check if closest has been set
  if (closest==-1) {
    cout << "Error: closest not set in function 'clust'" << endl;
    cout << "Distance: " << distance << "   Closest cluster: " << closest << endl;
    exit(1);
  }

  // Is the closest cluster close enough?
  if (closest_dist <= max_radius[member]) {
    // Update prototype vector
    if (closest<1 || closest>(int)num_of_clusters) {
      cout << "Error: Trying to update incorrect cluster prototype vector" << endl;
      exit(1);
    }
    adapt_cluster(closest,vector,member);
  } else closest=(int)num_of_clusters+1;

  return (unsigned int)closest;
}


void mutate(unsigned int child)
// Gaussian mutation
{
  unsigned long int new_value;
  for (unsigned int i=2;i<=num_of_trigrams+1;i++) {
    new_value = (unsigned long int)(weight[child][i] + gasdev(gasseed));
    if ((new_value >= min_weight) && (new_value <= max_weight)) {
      weight[child][i] = new_value;
    }
  }
  double new_radius;
  new_radius = (double)(0.02 * gasdev(gasseed) + max_radius[child]);
  if ((new_radius > 0) && (new_radius <= 0.2)) {
    max_radius[child] = new_radius;
  }
  double new_labda;
  new_labda = (double)(0.05 * gasdev(gasseed) + labda[child]);
  if ((new_labda > 0) && (new_labda <= 0.5)) {
    labda[child] = new_labda;
  }
}


void insert(unsigned int member)
{
  int i=1,low,high,mid;
  double fitness=(double)weight[member][SCORE]/(double)weight[member][AGE];

  // Determine insertion point
  if (num_of_insertions == 0)
    i=1;
```

```
  else if (fitness > highest_fitness)
    i=num_of_insertions+1;
  else if (fitness < lowest_fitness)
    i=1;
  else {
    low=1;
    high=num_of_insertions;
    while (low <= high) {
      mid = (low+high)/2;
      if (fitness > rank[mid][1]) {
        if (fitness < rank[mid+1][1]) {
          i=mid+1;
          break;
        } else low=mid+1;
      } else if (fitness < rank[mid][1]) {
        if (fitness > rank[mid-1][1]) {
          i=mid;
          break;
        } else high=mid-1;
      } else {
          i=mid;
          break;
      }
    }
    // Check for incorrect termination of while-loop
    if (low > high) {
      cout << "Error in 'insert'" << endl;
      exit(1);
    }
  }

  // Add
  for (int j=num_of_insertions;j>=i;j--) {
    rank[j+1][0]=rank[j][0];
    rank[j+1][1]=rank[j][1];
  }

  rank[i][0]=(double)member;
  rank[i][1]=fitness;
  num_of_insertions++;
  cout << num_of_insertions << "... ";
  cout.flush();

  // Update statistics
  lowest_fitness=rank[1][1];
  average_fitness=((num_of_insertions-1)*average_fitness+fitness)/num_of_insertions;
  highest_fitness=rank[num_of_insertions][1];
}


void update_population()
{
  // Produce offspring if there are adults
  if (num_of_insertions==0) return;

  cout << "Create offspring" << endl;
  unsigned int num_of_children;
  if (num_of_insertions<num_of_replacements)
```

```
    num_of_children=num_of_insertions;
  else
    num_of_children=num_of_replacements;

  unsigned int i,weakest_member_index,parent;

  for (unsigned int loop=1; loop<=num_of_children; loop++) {

    // Select adult member of population for parenthood
    do parent=select(); while (weight[parent][AGE]<age_limit);
    cout << "Parent: " << parent << endl;
    cout << "Number of adults: " << num_of_insertions << endl;

    // Replace weakest member of adult population with new child
    weakest_member_index=(unsigned int)rank[1][0];
    cout << "Weakest member: " << weakest_member_index << endl;
    copy_vector(weight[parent],weight[weakest_member_index]);
    max_radius[weakest_member_index]=max_radius[parent];
    labda[weakest_member_index]=labda[parent];

    if (num_of_insertions>1)
      average_fitness=((average_fitness*num_of_insertions)-rank[1][1])/(double)(num_of_insertions-1);

     mutate(weakest_member_index);
    weight[weakest_member_index][0]=0;                    // Score
     weight[weakest_member_index][1]=0;                   // Age

    num_of_insertions--;

    if (num_of_insertions>0) {
      // Update rank list
      for (i=1;i<=num_of_insertions;i++) {
        rank[i][0]=rank[i+1][0];
        rank[i][1]=rank[i+1][1];
      }

      // Update statistics
      lowest_fitness=rank[1][1];
      highest_fitness=rank[num_of_insertions][1];
    }
  }
}


// Main =============================================================

int main(int argc, char *argv[])
{
  // Display notice
  cout << "\nStream by D.R. Tauritz\n" << endl;

  // Log time
  time_t time_of_day;
  time_of_day = time (NULL);

  ofstream logfile ("stream.log",ios::app);
  if (!logfile) {
    cout << "Error: Unable to open file 'stream.log' for output" << endl;
```

```
  exit(1);
}

logfile << ctime (&time_of_day);
logfile.close();

// Read input files ----------------------------------------------------

// Read batch index file
ifstream indexfile ("batch.ind");
if (!indexfile) {
  cout << "Error: Unable to open 'batch.ind'" << endl;
  exit(1);
}
index = new unsigned int [MAX_ARTICLES];
if (!index) {
  cout << "Error allocating 'index'" << endl;
  exit(1);
}
unsigned int num_of_articles=1;
while (indexfile >> index[num_of_articles]) {
  cout << "Article: " << num_of_articles << "  Cluster: " << index[num_of_articles] << endl;
  num_of_articles++;
}
indexfile.close();
num_of_articles--; // compensate for last unwarranted addition

// Read parameter file
ifstream paramfile ("stream.par");
if (!paramfile) {
  cout << "Error: Unable to open parameter file" << endl;
  exit(1);
}
paramfile >> min_weight;
paramfile >> max_weight;
paramfile >> pop_size;
paramfile >> num_of_replacements;
paramfile >> age_limit;
paramfile >> selective_pressure_rate;   // range = [0,1]
paramfile.close();

weight = new unsigned long int [pop_size+1][VECTOR_SIZE];
if (!weight) {
  cout << "Error allocating 'weight'" << endl;
  return 1;
}

cluster = new float [num_of_clusters+1][VECTOR_SIZE];
if (!cluster) {
  cout << "Error allocating 'cluster'" << endl;
  return 1;
}

backup_cluster = new float [num_of_clusters+1][VECTOR_SIZE];
if (!backup_cluster) {
  cout << "Error allocating 'backup_cluster'" << endl;
  return 1;
}
```

```
// Read cluster definition files
char *filespec = new char[14];
char *helpstring = new char[14];
for (unsigned int group=1;group<=num_of_clusters;group++) {
  filespec = strcpy (filespec,"cluster");
  my_itoa(group,helpstring);
  filespec = strcat (filespec,helpstring);
  filespec = strcat (filespec, ".vec");
  ifstream clustdeffile (filespec);
  if (!clustdeffile) {
    cout << "Error: Unable to open cluster definition file '" << filespec << "'" << endl;
    exit(1);
  }
  clustdeffile >> num_of_trigrams;
  for (unsigned int loop=1;loop<=num_of_trigrams;loop++)
    clustdeffile >> cluster[group][loop];
    clustdeffile.close();
}

max_radius = new double [pop_size+1];
if (!max_radius) {
  cout << "Error allocating 'max_radius'" << endl;
  return 1;
}

labda = new double [pop_size+1];
if (!labda) {
  cout << "Error allocating 'labda'" << endl;
  return 1;
}

rank = new double [pop_size+1][2];
if (!rank) {
  cout << "Error allocating 'rank'" << endl;
  return 1;
}

// Read weight vector files
cout << "Inserting:" << endl;
for (unsigned int loop=1;loop<=pop_size; loop++) {
  filespec=strcpy (filespec, "w");
  my_itoa(loop,helpstring);
  filespec=strcat (filespec,helpstring);
  filespec=strcat(filespec,".vec");
  ifstream weightfile (filespec);
  if (!weightfile) {
    cout << "Error: Unable to open file 'w" << loop << ".vec'" << endl;
    exit(1);
  }
  weightfile >> weight[loop][0];              // Score
  weightfile >> weight[loop][1];              // Age
  weightfile >> max_radius[loop];
  weightfile >> labda[loop];
  weightfile >> num_of_trigrams;

   for (unsigned int i=2;i<=num_of_trigrams+1;i++) {
    weightfile >> weight[loop][i];
```

```
 }
 weightfile.close();
 // If member is adult it must be added to the ranking list
 if (weight[loop][1]>=age_limit)
   insert(loop);
}
cout << endl;

// Initialisation of dynamic datastructures

ranseed = new long int;
if (!ranseed) {
  cout << "Error allocating 'ranseed'" << endl;
  return 1;
}

gasseed = new long int;
if (!gasseed) {
  cout << "Error allocating 'gasseed'" << endl;
  return 1;
}

// Initialize random number generator ran1 (see gausrand.h)
// This generator will be used for selection & initial population
srand(time(NULL));
*ranseed = -1-rand();

// Initialize gaussian random number generator gasdev (see gausrand.h)
// This generator will be used for mutation
*gasseed = -1-rand();

// Main loop ------------------------------------------------------------

for (unsigned int article=1; article <= num_of_articles; article++) {

  // Read article vector file
 my_itoa(article,helpstring);
 filespec = strcpy (filespec,helpstring);
 filespec = strcat (filespec, ".vec");
  cout << "Reading article file: " << filespec << endl;
  ifstream articlefile (filespec);
  if (!articlefile) {
    cout << "Error: Unable to open '" << article << ".vec'" << endl;
    exit(1);
  }
  articlefile >> num_of_trigrams;
  vector[0]=0;
  for (loop=1;loop<=num_of_trigrams;loop++) {
    articlefile >> vector[loop];
    vector[0]+=vector[loop];
  }
  articlefile.close();

  // Process population ----------------------

  cout << "Process population" << endl;
  unsigned int i;
  for (unsigned int member=1;member<=pop_size;member++) {
```

```
  // Backup prototype vectors
  for (group=1;group<=num_of_clusters;group++) {
    for (i=1;i<=num_of_trigrams;i++) {
    backup_cluster[group][i]=cluster[group][i];
   }
  }

  // Check if score is smaller or equal to age (which it should be)
  if (weight[member][SCORE]>weight[member][AGE]) {
   cout << "Error: Score of member " << member << " is higher than age" << endl;
   cout << "Score: " << weight[member][SCORE] << "   Age: " << weight[member][AGE] << endl;
   exit(1);
  }

  if (weight[member][AGE]>=age_limit) {
     cout << "Lowest fitness : " << lowest_fitness << endl;
     cout << "Average fitness: " << average_fitness << endl;
     cout << "Highest fitness: " << highest_fitness << '\n' << endl;
  }

   // Add children of age to adult pool
  weight[member][AGE]++;
  if (weight[member][AGE]==age_limit)
  insert(member);

  // Determine initial rank index of weight[member]
  if (weight[member][AGE]>=age_limit)
  for (unsigned int loop=1;loop<=num_of_insertions;loop++)
  if (rank[loop][0]==member) i=loop;

  double temp_index, temp_fitness;
  if (clust(member)==index[article]) {
      weight[member][SCORE]++;
      if (weight[member][AGE]>=age_limit) {
        rank[i][1]=(double)weight[member][SCORE]/(double)weight[member][AGE];
        // Update rating list
        if (i<num_of_insertions) {
          while (rank[i+1][1]<rank[i][1] && i<num_of_insertions) {
            temp_index=rank[i+1][0];
            temp_fitness=rank[i+1][1];
            rank[i+1][0]=rank[i][0];
            rank[i+1][1]=rank[i][1];
            rank[i][0]=temp_index;
            rank[i][1]=temp_fitness;
            i++;
        }
      }
    }
  } else {
     if (weight[member][AGE]>=age_limit) {
       rank[i][1]=(double)weight[member][SCORE]/(double)weight[member][AGE];
       // Update rating list
       if (i>1) {
         while (rank[i-1][1]>rank[i][1] && i>1) {
           temp_index=rank[i-1][0];
           temp_fitness=rank[i-1][1];
           rank[i-1][0]=rank[i][0];
```

```
            rank[i-1][1]=rank[i][1];
            rank[i][0]=temp_index;
            rank[i][1]=temp_fitness;
            i--;
           }
          }
         }
        }
        cout << "Score: " << weight[member][SCORE] << "   Age: " << weight[member][AGE] << endl;

        if (weight[member][AGE]>=age_limit) {
          lowest_fitness=rank[1][1];
          highest_fitness=rank[num_of_insertions][1];
          double total=(double)0;
          for (unsigned int i=1; i<=num_of_insertions; i++)
            total+=rank[i][1];
          average_fitness=total/num_of_insertions;
        }

        // Restore prototype vectors
        for (group=1;group<=num_of_clusters;group++) {
          for (i=1;i<=num_of_trigrams;i++) {
            cluster[group][i]=backup_cluster[group][i];
          }
        }


      } // End of process population loop

      // Evolution
      update_population();

      // Apply clustering algorithm using fittest member
      unsigned int dummy = clust((unsigned int)rank[num_of_insertions][0]);

      ofstream logfile ("stream.log",ios::app);
      logfile << average_fitness << "          " << highest_fitness << endl;
      logfile.close();
    }

      // Write output files ----------------------------------------------------

      // Write weight vector files
      cout << "Writing weight vector files..." << endl;
      for (loop=1;loop<=pop_size; loop++) {
        filespec=strcpy (filespec,"w");
        my_itoa(loop,helpstring);
        filespec=strcat (filespec, helpstring);
        filespec=strcat(filespec,".vec");
        ofstream weightfile (filespec);
        if (!weightfile) {
          cout << "Error: Unable to create file " << filespec << endl;
          exit(1);
        }
        weightfile << weight[loop][SCORE] << '\n';                  // Score
        weightfile << weight[loop][AGE] << '\n';                    // Age
        weightfile << max_radius[loop] << '\n';
        weightfile << labda[loop] << '\n';
```

```
    weightfile << num_of_trigrams << '\n';
    for (unsigned int i=2;i<=num_of_trigrams+1;i++) {
      weightfile << weight[loop][i] << '\n';
    }
    weightfile.close();
  }

  // Write cluster definition file
  cout << "Writing cluster definition files..." << endl;
  for (group=1;group<=num_of_clusters;group++) {
    filespec=strcpy (filespec, "cluster");
    my_itoa(group,helpstring);
    filespec=strcat (filespec,helpstring);
    filespec=strcat(filespec,".vec");
    ofstream clustdeffile (filespec);
    if (!clustdeffile) {
      cout << "Error: Unable to create cluster definition file" << filespec << endl;
      exit(1);
    }
    clustdeffile << num_of_trigrams << '\n';
    for (unsigned int loop=1;loop<=num_of_trigrams;loop++)
      clustdeffile << cluster[group][loop] << '\n';
    clustdeffile.close();
  }

  // Close log file
  time_of_day = time (NULL);
  logfile << ctime (&time_of_day);
  logfile.close();

  return 0;
}
// End of file ============================================================
```

Appendix D. TCL/TK control script for the demonstrator

 # Demonstrator - Last revised: August 28th, 1996

```
# global variables
set article 0
set cluster 0

button .fetch -text “Fetch news” -command {fetch_news}
pack .fetch
button .peruse -text “Peruse clusters” -command {catch {list_clusters}}
pack .peruse

proc fetch_news {} {
  source stream.tcl
  exec “cluster”
}

proc list_clusters {} {
  global cluster
  listbox .clusters
  pack .clusters
  set f [open clusters.lst]
  while {[gets $f line] >= 0} {
    .clusters insert end $line
  }
  close $f
  bind .clusters <Double-Button-1> {
    set cluster [.clusters curselection]
    inc cluster
    catch {list_articles}
  }
}

proc list_articles {} {
  global article cluster
  toplevel .$cluster
  listbox .$cluster.articles -yscrollcommand “.$cluster.scroll set”
  pack .$cluster.articles
  scrollbar .$cluster.scroll -command “.$cluster.articles yview”
  pack .$cluster.scroll -side right -fill y
  set f [open cluster$cluster.ind]
  while {[gets $f line] >= 0} {
    .$cluster.articles insert end $line
  }
  close $f
  bind .$cluster.articles <Double-Button-1> {
    set article [selection get]
    catch {display_article}
  }
}

proc display_article {} {
  global article
  toplevel .a
  text .a.text -relief raised -bd 2 -yscrollcommand “.a.scroll set”
  scrollbar .a.scroll -command “.a.text yview”
  pack .a.scroll -side right -fill y
```

```
  pack .a.text -side left
  set f [open $article.art]
  while {![eof $f]} {
    .a.text insert end [read $f 1000]
  }
  close $f
  .a.text configure -state disabled
  button .a.approve -text “Approve clustering” -command {evolve 1}
  button .a.disprove -text “Reject clustering” -command {evolve 0}
  pack .a.approve -side bottom
  pack .a.disprove -side bottom
}

proc evolve {status} {
  global article cluster

  # write status file
  set f [open article.inf w]
  puts $f $article.vec
  puts $f $cluster
  puts $f $status
  close $f

  # execute evolution program
  exec “evolve”

  destroy .a
}
```