University of
HUDDERSFIELD

University of Huddersfield Repository

Wade, Steve, Smith, Martin and Wolstenholme, Mark

Applying Genetic Programming to the Problem of Term Weight Algorithms

Original Citation

Wade, Steve, Smith, Martin and Wolstenholme, Mark (2000) Applying Genetic Programming to the Problem of Term Weight Algorithms. The New Review of Document and Text Management , Vol. 1. pp. 101-111. ISSN 1361-4584

This version is available at http://eprints.hud.ac.uk/7627/

The University Repository is a digital collection of the research output of the University, available on Open Access. Copyright and Moral Rights for the items on this site are retained by the individual author and/or other copyright owners. Users may access full items free of charge; copies of full text items generally can be reproduced, displayed or performed and given to third parties in any format or medium for personal research or study, educational or not-for-profit purposes without prior permission or charge, provided:

- The authors, title and full bibliographic details is credited in any copy;
- A hyperlink and/or URL is included for the original metadata page; and
- The content is not changed in any way.

For more information, including our policy and submission procedure, please contact the Repository Team at: E.mailbox@hud.ac.uk.

http://eprints.hud.ac.uk/

Application of a Genetic Algorithm to the Production of Text Signatures

Steve Wade, Martin Smith and Mark Wolstenholme, *C^{e}DAR - Centre for Database Access Research, The School of Computing and Mathematics, The University of Huddersfield, Queensgate Huddersfield, West Yorkshire. (S.J.Wade@HUD.AC.UK)*

This paper describes the results of a preliminary attempt to use a genetic algorithm to divide an inverted file into a specified number of partitions such that the total number of documents indexed by a particular partition is approximately equal to the total number of documents indexed by each of the other partitions. The purpose of identifying such equifrequent partitions is to assist in the generation of text signature representations of documents which are more discriminating than those created using more traditional techniques.

The paper is divided into six sections. Following the introduction, the second of these describes the main idea behind the signature approach The third section introduces the idea of genetic algorithms and briefly reviews earlier work on the application of this technique to information retrieval problems. The fourth section describes how we have used a genetic algorithm to partition a section of the inverted file used to index the LISA document test collection and how we have then used the partitioned file in the production of text signatures to represent documents in the collection. We might say that the signature representations produced in this way have been customised to the vocabulary of the LISA document collection and we would therefore expect them to be more discriminating than text signatures produced using more traditional techniques. The fifth section of the paper compares the results of searches conducted using signatures with results obtained from searches of the full inverted file. It is concluded that the signatures produced with the assistance of the genetic algorithm are more discriminating than those produced using simpler techniques.

INTRODUCTION

Genetic algorithms represent a new approach to solving a range of computationally-expensive problems that cannot easily be solved using conventional deterministic approaches. The approach is based on the identification of an initial set of possible solutions to a problem which are then iteratively improved using techniques analagous to those found in natural evolution. A recent paper by Jones, Robertson *and* Willett [3] outlines a number of information

retrieval problems that might be addressed in this way. One of these problems concerns the use of a genetic algorithm to divide an inverted file into equifrequent order-free groups of indexing terms. The work described here addresses a related problem. Our concern has been to partition an inverted file whilst preserving the order of the resulting groups in order to create a partition file which might be used in the production of text signatures.

TEXT SIGNATURES

The use of an inverted file structure is almost universal in large scale document retrieval systems. This approach permits rapid searches to be carried out but suffers from the large storage and processing overheads associated with updating and maintaining the online indexes which are needed to provide an interactive response. In many cases, the inverted indexes take as much space as the main record file itself and for large systems further overheads are incurred because it becomes necessary to maintain smaller secondary indexes to gain access to the main index. No such overheads are incurred in serial searching but the approach is inherently slow and cannot be expected to support interactive retrieval from large document collections; this is true in spite of the number of highly efficient string searching algorithms that have been suggested and the use that has been made of parallel hardware for serial searching.

The idea of using signature representations of text to speed up serial searching was first suggested by Harrison[2] in connection with the development of an efficient "Find X" facility for text editing, where X would be some user defined string of characters. A text string is a fixed length bit string representation in which individual bits are set if certain character strings, (which might be stems, words or phrases) are present in the text being characterised. The bit string is typically created by applying a hashing algorithm to each of these strings, this type of algorithm uses the characters in a particular string to compute a value which can be taken as a bit position in the text signature. Thus, the effectiveness of signature representations is

critically dependant upon the discriminating power of the hashing algorithm used to determine which bits are set.

Even the most discriminating algorithm will have to make use of the same bit position to represent a number of different terms. (This problem can only be avoided in situations where the signature length is at least the same size as the indexing vocabulary). A consequence of this is that a signature search may produce matches between query terms and documents which have not been indexed under those terms but which have been indexed under terms that hash to the same bit position. Such a mismatch is known as a "false drop". To avoid this problem text signatures are often used as the basis for a fast approximate search which serves to eliminate large sections of text from a more computationally expensive exact search, this implies a two-stage retrieval strategy. In the first stage a text signature representation of the query string is compared with corresponding representations of the documents in the database, in this way a small subset of the documents in the collection can be rapidly identified as worthy of a more detailed pattern matching search. Such an approach can dramatically reduce the time needed for a serial search.

GENETIC ALGORITHMS

Genetic algorithms offer a way to randomly search a problem space for the best answers to problems having the following characteristics:

- No obvious algorithmic solution
- Information structures that encode potential solutions
- A way of evaluating these structures
- Formal means of chopping and re-splicing these structures without creating nonsense rules.

The task of partitioning an inverted file meets all of these criteria. It would be difficult to specify a deterministic algorithm capable of finding the best possible solution for a range of inverted indexes. Our potential solutions are simple fixed length array structures containing the number of indexing terms in each partition. These are easy to

evaluate and can easily be chopped and respliced. They are our analogues to biological chromosomes.

From a computational point of view, genetic algorithms involve the selection of a number of possible solutions to a problem. Initially this selection will be completely at random. Each solution will exhibit a level of “fitness” to the problem which we must be able to measure. Possible solutions are assessed according to their fitness and put into a ranked order. The next stage is to breed the individual solutions with each other at random, with a greater probability of being chosen as a parent being assigned to the higher ranked solutions. This produces a new generation of solutions which share some characteristics of the parents in different combinations. (The inherited characteristics will in our case be simply sections taken from the parent array structure - they are therefore analagous to biological genes). The children are then assessed and put into ranked order and the process is repeated.

The operators that are acting in producing new offspring are known as “crossover” “mutation” and “inversion”. These operators have analogues in natural evolution, further discussion of them is deferred until the next section where an explanation of the precise way in which they have been implemented is presented.

METHOD OF THE EXPERIMENT

We are concerned with the use of genetic algorithms in placing partitions in an equifrequent way. We are therefore interested in using the evolutionary process to select *optimal* solutions from the many *possible* solutions. The input to this process will be a postings file (i.e. one that contains all of the indexing terms that would occur in a full inverted file along with their frequency of occurrence). Thus a section of this file might look like this:

ABSTRACT	2568
AGGLOMERATIVE	25
AUTOMATIC	12000
BIBLIOGRAPHIC	5000
BINARY	89

The input file is likely to be large (say) 10,000 terms. The signature representation will reduce this to (say) 100 bits. In this case, the data structure taken as input to the genetic algorithm is a partition array of length 99. Each location in the array is an integer representing the number of indexing terms in that particular partition.
The following might be a section of the array structure:

126	251	93	150	12	34	400

This data structure represents a situation where there are 126 terms in the first partition, 251 indexing terms in the second partition etc. Our genetic algorithm is used to produce and modify structures like this until the optimal set of partitions is obtained
First the signature length should be decided (the number of partitions of the index file). The total frequency, F_t of words on the index file can then be divided by the signature length M to give the ideal frequency of each partition F_s.

$$\frac{F_t}{M} = F_s$$

The genetic algorithm will pick partitions completely at random to produce the first generation of solutions. These are then evaluated with the following formulae:

$$E_t = \sum_{j=1}^{M} (F_s - F_j)$$ where E_t = total error.

A perfect partitioning of the input document will result in a value of zero for E_t. Each individual solution in the population can be assessed using the above equation, and put into ranked order. For the next generation a probability is assigned to each individual, depending on its position in the ordered list. The higher up the list, the greater the probability of being selected as a parent.
In the remainder of this section we discuss the specific way in which the genetic operators crossover, mutation and inversion are carried out on the partition array structures.

Crossover
Crossover involves the random selection of cut points in the bit string and swapping the chunks of material in between the cutpoints with another individual, this is analogous to asexual reproduction.

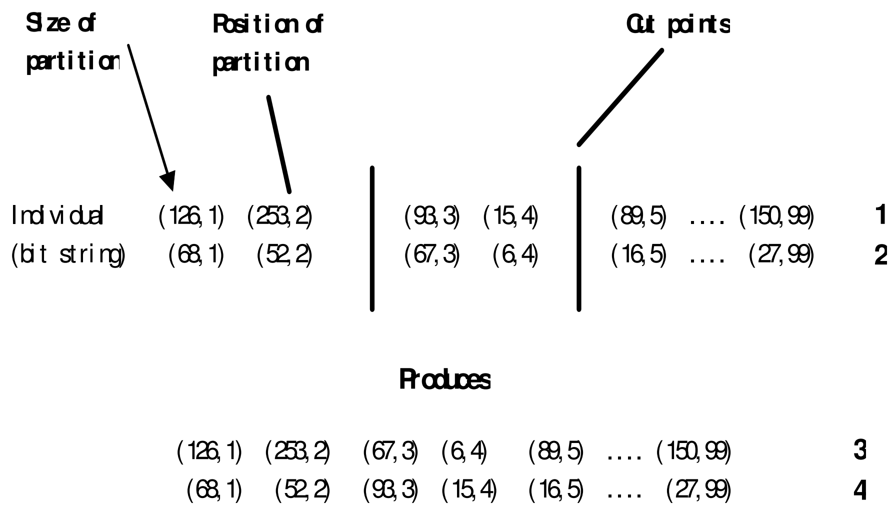
- Two structures $a_1..a_m$ and $b_1..b_m$ are selected at random from the current population.
- A crossover point x, in the range 1 to m-1 is selected again at random.
- Two new structures:

  $a_1a_2...a_xb_{x+1}b_{x+2}....b_m$

  $b_1b_2...b_xa_{x+1}a_{x+2}....a_m$

  are formed.

This situation is summarised in the following diagram:



It should be noted that the sum of the terms in solutions (3) and (4) above will not add up to the total number of terms in the original postings file (T_T) and so will need to be normalised. This involves multiplying each element in the two solutions by a common factor. (P_T / T_T) where P_T is the number of terms in the solution being considered. In each case the result will be a real number which will be rounded. At the end of this process there will still be leftovers, these are distributed at random throughout the solution.
Normalisation will also apply to mutation and inversion.

Mutation
Mutation has a small probability of happening and is the random change of any particular element in the data structure. For example:

(126,1) (253,2) (93,3) (15,4) ....(150, 99)

mutated becomes:

(126,1) (25***4***,2) (93,3) (15,4) ....(150, 99)

The purpose of mutation is to ensure that the solution does not get stuck at a local optimum.

Inversion
Inversion plays an ancilliary role, but is very important in bringing together elements that were far apart and separating ones that were close together. This promotes close linkage between the successful elements and is important in later crossovers.
The way in which inversion was applied might be summarised as follows:

(126,1) | (253,2) (93,3) (15,4) | ....(150, 99)

inverted becomes:

(126,1) (15,4) (93,3) (253,2)....(150, 99)

Cut points are randomly selected within an individual, and then the elements within the cut points are swapped around. Where inversion is to be followed by a crossover operation it is necessary to perform equivalent inversion operations on both mates.

Results with Test Data

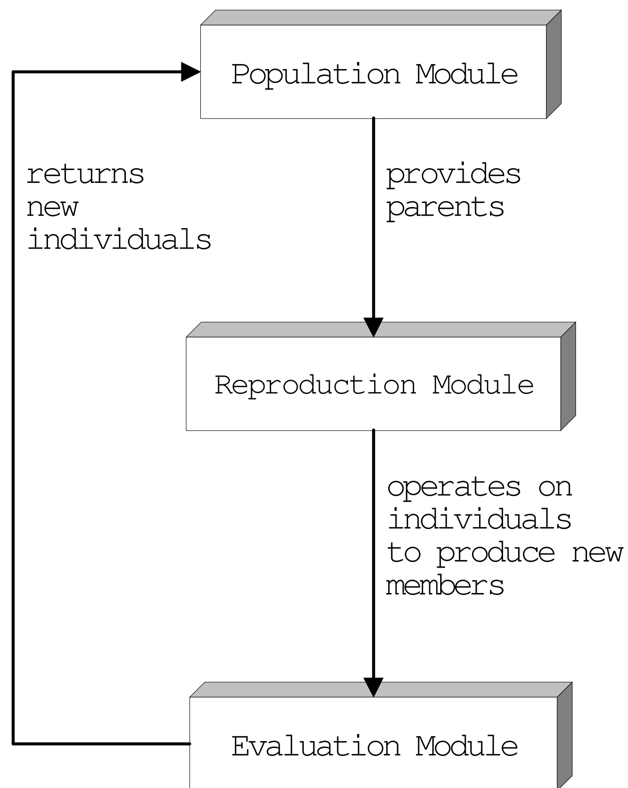
The algorithm was tested using a postings file that could be perfectly partitioned. The following represents an extract from the file:

3 4 5 1 7 3 4 5 1 7 3 4 5 1 7 3 4 5 1 7

Each of these numbers would be taken as the posting for a term in the index. In the test file the repeating pattern (3 4 5 1 7) is repeated 20 times. Thus the sum of all the postings is 400. If we wish to create 20 partitions of this data (for a signature length of 20) we would expect each partition to index 20 documents (400 / 20). Which is, of course, the sum of the frequencies in the repeating group. In order to create this perfect set of partitions the genetic algorithm would have to partition the file after every five terms. Thus the best solution would be:

5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

How the Software Operates
The software consists of four main modules:



As the program runs, an initial population of solutions is generated at random. Next, each individual solution in the population is evaluated by the fitness function. The program then begins a series of cycles known as generations. This involves a "parent" being selected from the population which is then passed to the reproduction module. The reproduction module will apply a genetic operator (crossover, mutation or inversion) to the parent to produce a new solution, the

“child”. If the operation is crossover or inversion, two parents are involved and two children are produced.

The operator may cause the children to differ from their parents in that they are closer to the ideal solution (having a lower error score than the parents) or further way (having a higher error score than the parents). The fitness of the children is measured in the evaluation module and one generation passes. The cycle is then repeated until the ideal solution is found (an error rate of 0) or the generation limit is reached.

Managing the Population

The initial population is created by randomly partitioning the postings file a number of times to produce a population of solutions. Common practice among GA practitioners is to have an initial population size of around 100. This population will grow with each new generation created. There are however limits to population size imposed by the technical constraints of software and hardware. This means that the population size can only increase until a fixed limit is reached; thereafter we have replaced the worst individual in the population with the new solution. This replacement strategy adds a selection pressure to the search since we would expect most of the new solutions to be better than the ones they replace.

In experiments with the test data, we have changed the initial and maximum population sizes and the rates for mutation, crossover and inversion. The following graphs summarise some of the results which focussed on:

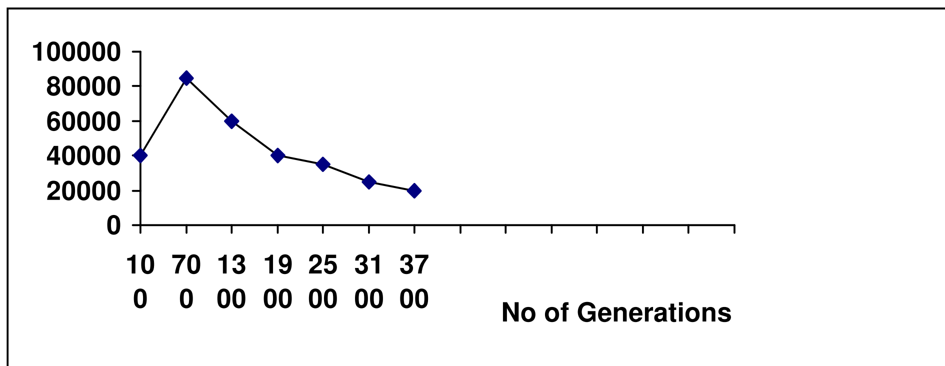
- the total error of the population
- the error of the best solution found after every 100 generations
- the number of generations taken to find the solution
- the time taken to reach the solution.

Initial Graphs

This graph refers to a run of the program using the test data described above with key parameters set as follows:

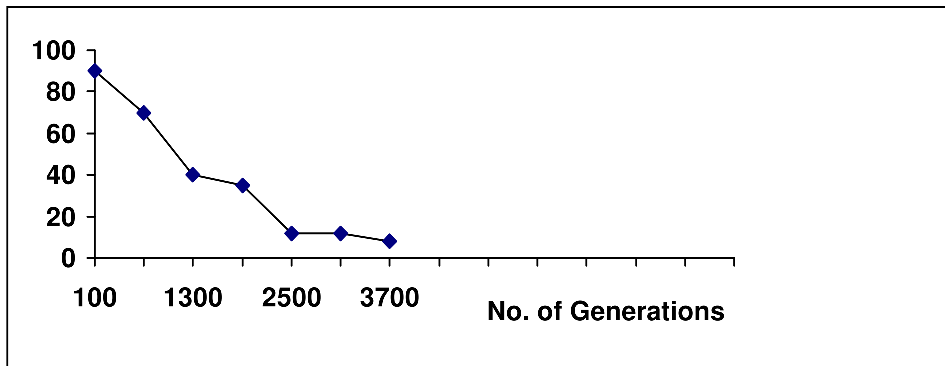
Initial Population	***30***
Max. Population Size	***700***
Mutation Rate (%)	***5***
Crossover Rate (%)	***75***
Inversion Rate (%)	***20***
Total Generations	***4000***
Time Taken (mins)	***10***

The following graph plots the total population error (Y axis) against the number of generations (X axis):



Notice how the total error of the population first goes up, reaches a peak at over 100,000 and then gradually falls down to 17000. This explained by the expanding population size for the first few hundred generations, until the maximum population size is reached. Then the total error falls as better solutions replace the weakest members of the population.

The next graph plots the results for the best individual solutions. Again the individual error is plotted on the Y axis against the number of generations on the X axis:

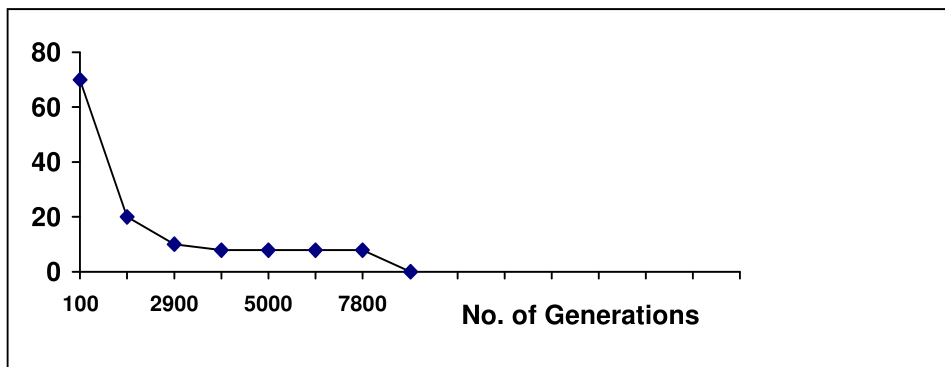


Notice how the error for the best individual falls sharply to 10 then bottoms out for a while before dropping to 6 and then to 0 the ideal solution. This is explained by the diverse initial population producing quite a high error for each individual. Then as the crossover operator works to bring the dispersed parts of the solution together, the individuals get better and better until they converge around 10. This is where the mutation operator is noticed by working in the background to create a new solution.

Results for a Small Population
This experiment shows that if the population is too small, then a solution is more difficult to find, because of premature convergeance, but program execution is much quicker.

Initial Population	***100***
Max. Population Size	***100***
Mutation Rate (%)	***5***
Crossover Rate (%)	***75***
Inversion Rate (%)	***20***
Total Generations	***10000***
Time Taken (mins)	***4***

Once again the graph plots the error rate for the best individual solution against the number of generations:



What appears to be happening is that initially only a few members of the population contain partitions with the ideal solution of 5 and these are scattered throughout the population. The crossover operation brings them together, because the search space is small but there are not enough 5's spread throughout the available strings. Because the population is small, individuals very quickly look alike.

It takes a while for the breakthrough to be made through the operations of inversion and mutation.
The next section will describe a comparison between the results of searches conducted using a full inverted file and text signatures developed using one of three techniques. One of these techniques, partition-based hashing, uses the partitions created by the genetic algortihm as the basis for creating the signature. The manner in which this has been done is discussed in the appropriate part of the next section.

Evaluation of Indexing Effectiveness
In this section we discuss a further series of experiments conducted with the LISA document test collection. This collection comprises 6004 documents and 35 queries and answers. In the comparisons that follow, the results obtained from a full inverted file search of the LISA database have been taken as a baseline against which the performance of different signature generation methods have been evaluated. The evaluation is based on the value of the Dice coefficient of similarity between references in the rankings produced by the signature-based and inverted-file-based searches averaged over the 35 queries in the collection.

The Dice coefficient is defined as $SIM = \frac{2C}{(A+B)}$ where C represents the number of documents common to the two sets, A represents the total number of documents in one of the sets and B the total number of documents in the set with which A is being compared; in these experiments the sizes of the retrieved sets of documents, A and B were either 10 or 20. The value of the function therfore varies from 0 to 1, with 1 representing identical sets and 0 representing sets which have no documents in common. A value of 1 for SIM would therefore imply that exactly the same documents had been found in the rankings produced by matching signature representations as were found in those obtained by searches based on the full inverted file,

although the order of these documents in the two rankings might be different.

The Hashing Techniques Used

The results recorded in the tables presented in this section were obtained using a text signature representation of the query created by stemming and then hashing each substantive query term to single position in the signature. Thus, "results for simple hashing techniques" would be reduced to the stem list "result simpl hash techniq" by eliminating stopwords and applying Porter's stemming algorithm to the remaining terms. Each of these stems would then be hashed to a single bit position in the signature. Signature representations of documents in the database were also created in this way, with a single signature representing the title and abstract of each document in the test collection. The resulting signatures were then stored as columns as an $N \times M$ table where N is the number of documents in the database and M is the bit-length of each signature, so that, in effect, the signatures are stored vertically. This data structure can be represented by the following table:

	Document Numbers				
	1	2	3	....	N
Bit Position					
1	0	0	0	....	1
2	1	1	0	...	0
3	0	0	1	...	0
:	:	:	:	:	:
M	0	1	1	...	1

For the purpose of searching the LISA database, this file structure offered the following advantages:

1. Each row in the table is a fixed length, fixed format record which is easy to manipulate.

2. Organisation by means of bit positions, rather than by means of the document numbers, allows ready access to the bits characterising a query; this in turn results in a comparatively rapid calculation of the requisite query-document similarities.

The Search

Each bit in a signature represents only the presence or absence of stems in a document, there is therefore no way of calculating the type of IDF (Inverse Document Frequency) weights commonly associated with best match searching in inverted file systems. The way in which the searches were conducted can be represented by the following algorithm:

```
Initialise total_SIM_so_far;

FOR I := 1 TO no_of_queries DO
        initialise docweight;
        FOR J := 1 TO QuerySize DO
                Calculate hash_value of Q[J];
                FOR K := 1 TO no_of_docs_in_database DO
                        IF K'th bit set in hash_value'th row
                        THEN docweight[K] := docweight[K] + 1;
                ENFOR
        ENDFOR;
        sort and rank documents in order of docweight;
        calculate SIM_for_this_query
        add SIM_for_this_query to total_SIM_so_far;
ENDFOR;
```

$$\text{SIM} := \frac{total_sim_so_far}{no_of_queries}$$

The results reported in the tables that follow were obtained by taking three different ways of calculating *hash_value* in the algorithm above. One of these was based on the partitioned file created by the genetic algorithm described above, the other two are briefly described below.

Division Hashing

Of the numerous hashing algorithms that have been suggested (Martin[5]) the one that consistently performs best under most conditions is the Division method (Knuth[4]).

The approach is very simple. A hash function h is chosen which can take on at most M different values where, in this case, M is the signature length, so that $0 < h(k) < M$, where K is the stem to be hashed or, more accurately, a number derived from some information in the stem. The Division method used in this evaluation meets these constraints by taking the remainder modulo M to give a value in the range $0 <= K \ MOD \ M < (M-1)$; this value is then incremented by one to give the hash value, so that $h(K) = (K \ MOD \ M) + 1$. In practice better performance has been observed if M is a prime number (Knuth[4]). Accordingly, the signatures considered in this section are all a prime number of bits long.

The version of the algorithm used looks like this:

```
division_hash_code := 0;
FOR I := TO length_of_stem DO
        get ordinate_value for I'th character in the stem;
        division_hash_code := division_hash_code + ordinate_value;
ENDFOR
division_hash_code := (division_hash_code MOD signature_length);
```

Concatenated Division Hashing

The Concatenated hashing algorithm used in these experiments is a derivation of the division hashing algorithm described above. In this approach the ordinal value of the characters in the stem is not simply added together; instead they are appended to one another. So that for example, the ordinal values 65 (representing "A") and 66 representing "B" would be joined to give the value 6566. The number generated in this way is truncated if it becomes too large for the machine to handle

The modified algorithm looks like this:

```
concatenated_hash_code := 0;
FOR I := TO length_of_stem DO
        get ordinate_value for I'th character in the stem;
        find no_of_digits_in_ord_value;
        IF concatenated_hash_code > 0 THEN
                multiply concatenated_hash_code by
                10^no_of_digits_in_ord_val
        ENDIF
        add ordinate_value to concatenated_hash_code
ENDFOR
concatenated_hash_code := (concatenated_hash_code MOD
signature_length);
```

Partition-Based Hashing

This approach makes use of the partitions identified by the genetic algorithm described in the previous section. Once the genetic algortihm has identified the equifrequent partitions the stems at the head of each partition can be written to a file. This having been done, the signature representations can be produced by taking a stem, searching the dictionary to identify the partition that the stem belongs in and then setting the bit position corresponding to that partition. So that, for example, the stem "Comput" would be identified as belonging in the partition headed by "Comprehens" if that entry in the partition file immediately preceded a stem alphabetically greater than "Comput" like "Concept".

The partition file created in this way is not very large (containing only as many stems as there are bits in the signature) it can also be created fairly easily and, given the known frequency distribution characteristics of vocabulary in free text (Cooper *et al* [1]) it need not be updated even in applications where the database is very dynamic.

The results presented in the next section are not based on a partitioning of the full inverted file structure associated with the LISA document test collection. The current version of our software has been written in Turbo Pascal for the PC, so we were constrained

by hardware and sofware limitations. Instead of the full inverted file we have used the genetic algorithm to partition a subset of the inverted file containing all of the terms occuring in the 35 queries provided by the test collection. This subset contains 350 indexing terms. We are currently porting our software to an environment where memory constraints are less restrictive, once this has been achieved we will be able to repeat these experiments with the full inverted file and correspondingly longer signatures. It does not seem unreasonable to expect that the results reported here will be broadly similar to those we will obtain from larger scale experiments.

The Results

When the top 10 references in the ranking are considered:

Hashing Technique	*Signature Length* **31**	**51**	**127**
Division	0.09	0.14	0.20
Concatenated	0.06	0.12	0.18
Partitioned	0.11	0.21	0.32

When the top 20 references in the ranking are considered:

Hashing Technique	*Signature Length* 31	51	127
Division	0.09	0.16	0.22
Concatenated	0.07	0.13	0.20
Partitioned	0.11	0.23	0.36

It can be seen from these results that the signature representations produced from the partitioned inverted file are more discriminating than signature representations produced using either of the alternative, commonly-used hashing algorithms.

CONCLUSION

In this paper we have discussed how a genetic algorithm might be used in the partitioning of an inverted file. These partitions are then used in the conversion of indexing terms to bit positions in text signatures, an operation normally carried out by a hashing algorithm. In this context, the ideal hashing algorithm would be one that makes use of the full length of the signature, to make each bit as discriminating as possible. This presents a problem, it is impossible to define a hash function that creates random data from the non-random data in actual files. The genetic algorithm approach seeks to compensate for the non-random nature of the text in a particular database by tailoring the conversion procedure to the occurrence characteristics of that text. An evaluation has been presented demonstrating that the signature representations produced with the genetic algorithm are more discriminating than those produced using simpler, more widely used hashing techniques.

REFERENCES

1. *Cooper, D. Dicker M.E. and Lynch M.F. -1980-* Sorting of textual databases: a variety generation approach to distribution sorting.- *Information Processing and Management 16 (1) pp. 59-66.*

2. *Harrison M.C. - 1971 -*. Implementation of substring search by hashing. *Communications of the ACM 14, pp. 777-779.*

3. *Jones G.; Roberston, A.M. and Willett, P. -1994 -* An introduction to genetic algorithms and to their use in information retrieval - *Online and CDROM Review Vol. 18. No. 1 pp. 3-12.*

4. *Knuth, D.E. - 1973 - The art of programming Vol. 1 Fundamental algorithms 2d. ed.* - Addison-Wesley.

5. *Martin, J. - 1977-* Computer Database Organisation. - Englewood Cliffs. Prentice Hall.

6. *Porter, M.F. - 1980 -* An algorithm for suffix stripping. - *Program 14 (3) pp. 130-137.*