Computing and Informatics, Vol. 28, 2009, 581–597

INCORPORATING STRUCTURED TEXT RETRIEVAL INTO THE EXTENDED BOOLEAN MODEL

Mathys C. du Plessis, Gideon de V. de Kock

Department of Computer Science and Information Systems
P. O. Box 77000
Nelson Mandela Metropolitan University
Port Elizabeth 6013
South Africa
e-mail: mc.duplessis@nmmu.ac.za

Manuscript received 21 July 2006; revised 30 November 2006
Communicated by Ján Paralič

Abstract. Conventional information retrieval models are inappropriate for use in databases containing semi-structured biographical data. A hybrid algorithm that effectively addresses many of the problems in searching biographical databases is presented in this article.

An overview of applicable structured text retrieval algorithms is given, with focus specifically on the tree matching model. Small adaptations to the Extended Boolean Model, to make it more applicable to biographical databases, are described. The adaptation of tree matching models to the hierarchical nature of data in a person record is described and a distance function between query and record is defined. A hybrid model between the Extended Boolean Model and the adapted Tree Matching Model is then presented. A fast ranking algorithm appropriate for general searches and a more effective (but more resource intensive) algorithm for more advanced searches is given. It is shown how dates can be incorporated in the hybrid model to create a more powerful search algorithm.

The hybrid algorithm can be used to rank records in descending order of relevance to a user's query.

Keywords: Hybrid search algorithms, biographical database, structured text retrieval, extended Boolean model, searches on dates

1 INTRODUCTION

Traditional information retrieval (IR) models do not accommodate the searching of relational databases where fields contain semi-structured text. In general, information retrieval models focus on retrieving text documents relevant to a user's query from a document set. Structured information is normally stored in a relational database and retrieved through means of primary or compound primary keys. This method can not be used in situations where no unique primary key exists, primary keys are unknown to users, or fields can contain general text. Work has been done on IR in XML documents. This article describes a search algorithm on structured and unstructured data in a large biographical database, but could, of course, be extended to any semi-structured document store with a non-changing structure. Specifically, the algorithm focuses on databases that store information on individuals and their relatives. Examples of biographical databases include genealogical databases, burial or cemetery records, military records, voters rolls, databases storing information relating to hereditary diseases and pedigrees of livestock, and hospital records.

To a certain extent the records considered are structured in fields as one would expect in a relational database. This structure, however, is not always useful for retrieval purposes, because of inaccuracies in the data and the limited amount of information available to the user when searching for an individual. For example, the user may know that an individual was baptized in the town *Queenstown.* If, for any reason, the stored record of the individual contains *Queenstown* as the individual's birth place but contains no information on the individual's baptism place, a search considering only the baptism place field would not retrieve the desired record. Furthermore, many of the fields contain natural language text and can thus not be seen as a field in the same sense as a field in the relational model. By taking into account the structure of the records and the relationships between fields, the above problems may be overcome.

Given a biographical database as described above, the problem addressed in this article is as follows: It is required to search for an individual's record using information on any of the database fields, supplied by the user. The retrieved records should be ranked in order of most relevant to the user's query. The search data in the query and the retrieved records need not be correct or complete. The algorithm should also take into account uncertainties in dates.

The case study used in this study is the Port Elizabeth Genealogical Information System (PEGIS) which contains the details of a large number of individuals (currently almost 600 000). The following sections contain an outline of the models used in the hybrid search algorithm, a discussion on how these models were integrated and how searches on dates can be provided.

2 BIOGRAPHICAL DATA STRUCTURES

It is assumed that all information relevant to an individual is stored using the following data structures:

Person: *Surname, first names, father, mother, gender, nicknames, general information.*

Event: *Event type, place, start date, end date, details.* Provision is made for the following events: *Birth, Baptism, Death, Burial, Residence, Will and Estate, Adoption, Military, Medical, Education, Occupation* and *Other.*

Relationship: *Man, Woman, Relationship type, place, start date, end date, details.* Provision is made for the following types of relationships: *Marriage, Common Law, Engagement* and *Extramarital.*

Zero or more Events and Relationships are associated with each person. Most of the fields contain general information, Place and Details usually contain general textual information, for example, the field *Birth place* could contain the string **Martjie Venter Hospital, Hopetown** or **The farm Doornkloof, district Cradock**. Fields may be left empty for any given individual.

3 STRUCTURED TEXT RETRIEVAL MODELS

Normally the field of information retrieval is concerned with retrieving textual records. These records are not structured in the same way as records in a relational database would be, where information is already formatted and is meant to be retrieved by means of a key attribute [1]. Some structure can, however, be found in a textual database, for example: chapters, sections, titles, etc. The user may find it useful to be able to specify information on the structure of the record to be retrieved. Models that allow the partial matching of a query to a record and the ranking of the result set, are referred to as Structured Text Retrieval Models.

According to Navarro and Baeza-Yates [2], the models proposed are not as mature as some classical retrieval models like the Boolean or Vector Space models. Several structured text retrieval models were compared by Beaza-Yates and Navarro [3], and it is interesting to note that only three of the compared models have $O(n)$ efficiency.

Considering several models the Tree Matching Model proposed by Kilpelinen and Mannila [4] was chosen as it is one of the more versatile models and would be best suited for searches in biographical databases. This model is similar to (but more powerful than) the model proposed by Burkowski [5].

The Tree Matching Model makes use of the fact that a document that can be broken up into sections, paragraphs, sentences and words can be represented by a tree structure (see Figure 1).

The tree hierarchy makes it possible to formulate queries as trees, where the sub-nodes for each node are given in brackets, for example:

$$\textbf{Paragraph}(\textit{button}, (\textbf{Sentence}(\textit{print}, \textit{paper})))$$

The model allows the query to be located in the document by taking into account the structure of the query and the document. This is called Tree Inclusion.

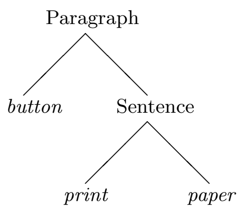


Fig. 1. The tree hierarchy of a query

The goal of the model is identifying ancestorship and labels (i.e. leaf nodes) rather than a direct hierarchial match. The model seeks minimal subtrees of the target. Formally, $\mathcal{T}$ is said to be an *included tree* of tree T if the nodes in $\mathcal{T}$ appear with similar hierarchial relationships in T. The trees considered here are ordered. We can say that if the left-to-right order of the nodes in $\mathcal{T}$ is the same as in T, the included tree is an *ordered included tree* of T. The difference between *ordered tree inclusion* and *unordered tree inclusion* can be seen graphically in Figures 2 and 3. Note that *ordered tree inclusion* is a stricter form of inclusion than *unordered tree inclusion.*

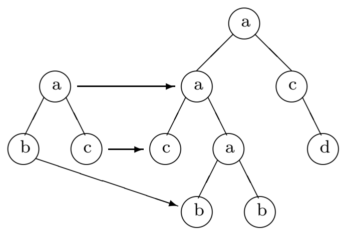


Fig. 2. Unordered tree inclusion. From [4].

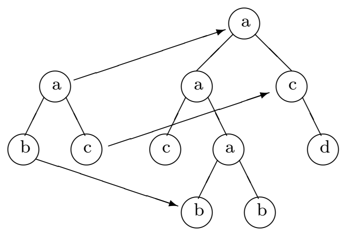


Fig. 3. Ordered tree inclusion. From [4].

Unfortunately, unordered tree inclusion is a NP-complete problem, while tree inclusion can be calculated in polynomial time [4]. Tree inclusion is thus a very expensive retrieval model. However, by appropriately indexing the database, and requiring that the database does not contain recursive hierarchies (i.e. no structure contains itself as a substructure), it has been shown that inclusion queries can be solved in O(n) time [6].

4 XML RETRIEVAL MODELS

In the last few years, a large body of work has been published on XML retrieval. XML essentially defines the structure of a document, thus making it very similar to the records described in this paper. A query language called XIRQL for IR in XML documents is described by Fuhr and Grosjohann [7]. XIRQL provides many of the traditional IR features that are missing in structured text retrieval, i.e. term weighting, relevance-orientated search, data types and vague predicates.

In XML retrieval it is necessary not only to index the terms in the documents, but also the structure of the document [8]. The structure index is then used to find documents that match the structure specified in the query. The major difference between the IR model described in this paper and XML retrieval is that not all documents in a XML document collection have the same XML tags or fields. Knowing the set of field in which data can occur simplifies the search algorithm considerably. Furthermore, XML retrieval models often aim to retrieve only relevant sections of documents, which would not make sense in a biographical information system where the focus is always on locating a specific individual.

Although XML retrieval models should not be applied directly to search algorithms in biographical databases, it is important to provide the relevant functionality found in XML retieval.

5 ADAPTATION OF EXTENDED BOOLEAN MODEL

The Extended Boolean Retrieval Model, developed by Salton et al. [9] is one of the more versatile retrieval models in that it can be seen as a unification of the Boolean [10], Vector Space [11] and Fuzzy Set [12] models. This versatility motivated the use of this model as the basis for a hybrid model that incorporates structured text retrieval techniques.

The ranking formula of the Extended Boolean Model in its general form is given by:

$$\begin{aligned} \text{sim}(d_j, q) = & \sqrt[p]{\frac{\sum\limits_{x=1}^{t} w_{x,q}^p . (w_{x,j})^p}{\sum\limits_{x=1}^{t} w_{x,q}^p}} \\ \text{for } q = & [(w_{1,q}, k_1) \vee^p \dots \vee^p (w_{t,q}, k_t)] \end{aligned} \tag{1}$$

$$\begin{aligned} \mathrm{sim}(d_j, q) = &\quad 1 - \sqrt[p]{\frac{\sum_{x=1}^{t} w_{x,q}^p.(1 - w_{x,j})^p}{\sum_{x=1}^{t} w_{x,q}^p}} \\ \text{for } q = &\quad [(w_{1,q}, k_1) \wedge^p \ldots \wedge^p (w_{t,q}, k_t)] \end{aligned} \tag{2}$$

where $w_{i,j}$ is a weight associated with term k_i in record d_j and $w_{i,q}$ is a weight associated with term k_i in the query q. The notation $\wedge^p$ is used to indicate that value of p can be varied to simulate different retrieval models.

Salton and Buckley [13] provided several heuristics by which appropriate query and record term weights can be selected. Since short query vectors can be expected an appropriate query term weight is:

$$w_{i,q} = \log \frac{N}{f_i} \tag{3}$$

where N is the total number of records and f_i is the number of records that contain term k_i.

The varied vocabulary and the fact that, in the case study, record vectors are comparatively short and are of homogeneous length, suggests the following as an appropriate record term weight:

$$w_{i,j} = \frac{F_{i,j}. \log \frac{N}{f_i}}{\sqrt{\sum_{l=1}^{t} \left(F_{l,j}. \log \frac{N}{f_l} \right)^2}} \tag{4}$$

where $F_{i,j}$ is the frequency with which index term k_i occurs in record d_j. It was decided not to make use of the normalization term (the denominator in Equation (4)) for the following reasons:

1. Including the term implies an extra pass of all term vectors after the $F_{i,j}$ components of the weights have been calculated. Apart from the computing time involved in this process, the memory requirements would be significant, especially for long query vectors.

2. The following sections will describe how the term weight will also be used as a measure of how well the structure of a target record matches the structure specified by the user in the query. Normalization with respect to other terms in the query would be pointless since we are interested in how well the record matches the structure with respect to other records.

For the same reason as point 2 above, it was decided to normalize each term weight by dividing by the maximum weight found for that term in all records. The weight of index term k_i in record d_j is thus:

$$w_{i,j} = \frac{F_{i,j}.\log\dfrac{N}{f_i}}{\max_r\left(F_{i,r}.\log\dfrac{N}{f_i}\right)}. \qquad (5)$$

The significance of the normalization term in Equation (5) will only become clear after the following sections. This is a different approach, but should not negatively affect the final ranking since terms will still be ranked in the same order relative to each other.

6 ADAPTATION OF STRUCTURED TEXT RETRIEVAL

The motivation behind the use of structured text retrieval models, discussed in Section 3, is such that the user's queries can specify where in a record a certain term must be found. A possible drawback to taking a strict structured view of the information is that some terms specified in a query may not appear in the corresponding field in the database. For example, a query that attempts to locate a person with the first name *John* may not return a record where *John* appears in the nicknames field. To overcome the problem, the hierarchical nature of the information should be taken into account.

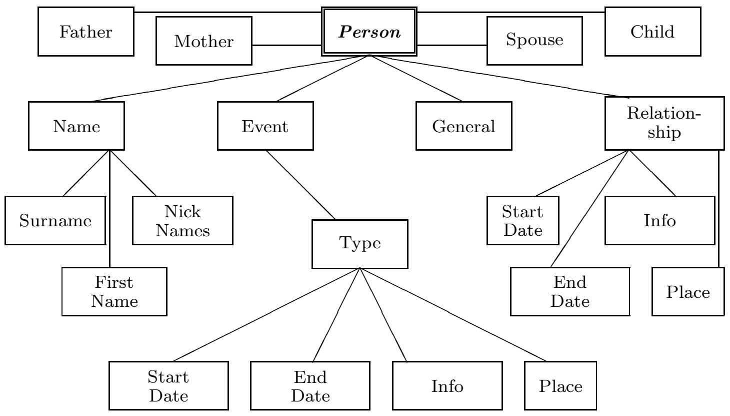


Fig. 4. Structure of a Person record

In Figure 4 the information of one Person record is shown structured as a tree. Note that the *conceptual* structure of a Person record is depicted and that the structure does not necessarily bear any resemblance to how this information is physically structured and stored in a database. Although Mother, Father, Spouse and Child records are Person records in their own right, they are depicted in Figure 4 to show their conceptual relationship to a person.

Queries specify the leaf or subtree where a term is expected. Consider the following 3 queries:

```
Person(John) AND Person(Hopetown)

Firstname(John) AND Birthplace(Hopetown)

Name(John) AND Event(Hopetown)
```

The first query retrieves records with the terms *John* and *Hopetown* anywhere in a Person record. The second query retrieves records with the term *John* specifically in a first name field and the term *Hopetown* specifically in a birth place field. The third query only requires the term *John* to be in a name subtree (first name, surname or nicknames) and the term *Hopetown* to be in an event subtree.

Note that it is not necessary to allow for recursive type queries for a single person, for example:

```
Person(Birth(Place(Hopetown)), Name(John))
```

The following query will suffice:

```
Birthplace(Hopetown) AND Name(John)
```

This makes it possible that the Tree Matching models discussed in Section 3 could be implemented efficiently.

Although tree matching provides powerful searching functionality, a broader approach is necessary for biographical databases. Consider the following query:

```
Birthplace(Hopetown)
```

If a pure Tree Matching algorithm was employed, only records that contain the term *Hopetown* in the Birth Place field would be returned. Records where the term appeared in the Baptism Place field would not be returned. There is a strong relationship between these two fields. If a child was baptized in a certain town, it is very likely that that child was also born there. Therefore records containing term *Hopetown* in the Baptism field should also be returned to the user, but with a lower rank depending on the how strongly the fields are related. Of course, it should be taken into account that the relationship between some fields (e.g. Birth Place and Death Place) would be very weak.

The following section describes a distance function that can be used to obtain a measure of how well structure specified in a query is matched in a given document.

6.1 Correlation Function

Let $H = \{H_1, H_2, \ldots, H_n\}$ be the set of all nodes (for example, Surname) and subtrees (for example, Name) within a Person record. Let $L = \{H_{L_1}, H_{L_2}, \ldots, H_{L_n}\}$ be the set of all leaf nodes in the tree hierarchy and let $N = \{H_{N_1}, H_{N_2}, \ldots, H_{N_n}\}$ be the set of all internal nodes (with their respective subtrees) in the tree hierarchy. Thus $H = L \cup N$.

Let $\preceq$ denote the subnode relationship in a tree, i.e. $H_j \preceq H_i$ means that $H_j \in H$ is in the subtree with root $H_i \in N$. We define a correlation function $\mathcal{D} : (H \times H) \longrightarrow [0, 1]$ indicating the relevance between any two nodes. Let the first parameter of $\mathcal{D}$ be the node as specified in the query, and the second parameter be the node where a term was found in a record. Note that $\mathcal{D}(H_i, H_i) = 1$ and that $\mathcal{D}(H_i, H_j) = 1$ if $H_j \preceq H_i$. Refer to Figure 4 in the following example:

- $\mathcal{D}(\text{Surname}, \text{Surname}) = 1$ – The fields are the same.
- $\mathcal{D}(\text{Name}, \text{Surname}) = 1$ – The second field is a subfield of the first.
- $\mathcal{D}(\text{FirstName}, \text{NickName}) = 0.8$ – The two fields are related.

$\mathcal{D}$ is not symmetric, i.e. in general $\mathcal{D}(H_i, H_j) \neq \mathcal{D}(H_j, H_i)$. Obviously this is the case where H_j is a node in the subtree denoted by H_i (i.e. $H_j \preceq H_i$). If the user specified that a term should fall within a name node and the term was found in the surname node, $\mathcal{D}$ should return a much higher correlation value than if the user specified a term should be found in the surname node but it was found in some other name node. For situations where neither $H_j \preceq H_i$ or $H_j \succeq H_i$, $\mathcal{D}$ is also not symmetric. For example, if the user specified in a query that a term should be found in a residence place node, and the term is then found in a baptism place node, it is likely that that person lived in that place in his or her early years. $\mathcal{D}$ should thus return a reasonably high value. Conversely, if the user specified that a term should be located in a birth place node, but the term was found in a residence node, it does not really imply that that person was born there. $\mathcal{D}$ should thus return a reasonably low value.

$\mathcal{D}$ makes use of a manually drawn up correlation table, from which the correlation of any two leaf fields can be read. For databases that are sufficiently large and complete, a program could be written that automatically determines the correlation between any two fields by seeing how often terms appear in both fields in the database.

Often the only information known about an individual is details of that person's parents, spouses, relationships or children. It is thus very useful to allow queries as follows:

```
Person(John) AND Mother(Hopetown)

Father(Firstname(John)) AND Mother(Birthplace(Hopetown))

Child(Name(John)) AND Spouse(Event(Hopetown))
```

For the sake of efficiency it is necessary to restrict the user to queries within one generation from the person to be searched for, i.e. not to allow recursive generational queries, like:

```
    Mother(Father(Surname(John)))
```

For the same reason we define $\mathcal{D}(H_i, H_j) = 0$ if H_i and H_j are not from the same person, for example, if a term to be found in a field in a Spouse record is found in a field in a Child record then there is no correlation between the query and the record.

Terms can occur several times in a query or a record. We define $\mathcal{H}_{x,y}$ as the set of nodes $\{H_{x,y_1}, H_{x,y_2}, \ldots, H_{x,y_n}\}$ where term x occurs in record or query y.

7 INCORPORATION OF STRUCTURED TEXT RETRIEVAL

Two approaches to incorporating structured text retrieval into Equations (1) and (2) will be discussed here. The first approach was designed specifically with an efficient implementation in mind and will be referred to as the *Fast Matching* approach. A second approach, though it may not be as fast as the first, should yield better retrieval results and will be referred to as the *Complete Matching* approach.

The major difference between the two ranking approaches is how multiple occurrences of terms in the same record are dealt with. Records in which a specific term only appears once, will be ranked exactly the same by the Fast and the Complete Matching algorithms.

7.1 Fast Matching

This approach assumes that the ranking depends only on the best structure match found for the term in the record. To incorporate structured text retrieval into Equations (1) and (2) we simply multiply the record term weight (given in Equation (5)) by the correlation function

$$w_{i,j} = \frac{F_{i,j}.\log\frac{N}{f_i}.\max_{H_s \in \mathcal{H}_{i,q}, H_v \in \mathcal{H}_{i,j}} (\mathcal{D}(H_s, H_v))}{\max_r \left(F_{i,r}.\log\frac{N}{f_i}.\max_{H_s \in \mathcal{H}_{i,q}, H_x \in \mathcal{H}_{i,r}} (\mathcal{D}(H_s, H_x))\right)} \tag{6}$$

where $\mathcal{H}_{i,q}$ is the query specified location of term k_i and $\mathcal{H}_{i,j}$ is the field where term k_i was found in record d_j.

Thus, if the term specified in the query is found in the correct field in a record, the correlation function will return 1 and the record will be ranked solely on the normal Extended Boolean Model formulas. If, however, the term is not found in the correct field, the correlation function will return a low number and $w_{i,j}$ will thus be smaller and, ultimately, the record will be ranked lower.

7.2 Complete Matching

The drawback to using Equation (6) is that, for situations where a term occurs several times in a record but few of those times in the desired field, the term will be assigned a weight as if all its occurrences were in the correct field. For example, consider two records in which a specified term occurs 5 times. In the first record all 5 occurrences are in the desired field. In the second record only one of the occurrences is in the desired field. The correlation function will return 1 for both records since the desired structure did occur. The two records would thus be ranked the same. Even worse, if the term occurred 6 times in the second but still only once in the desired location, the second record would actually be ranked higher than the first by Equation (6).

A more accurate approach would be to look at the value given by the correlation function for each time the term appears in a different field. Equation (5) is adapted by replacing the term frequency component, $F_{i,j}$, by the sum of the correlation between the field specified in the query and each occurrence in the record:

$$w_{i,j} = \frac{\left(\sum_{H_k \in \mathcal{H}_{i,q}} \sum_{H_l \in \mathcal{H}_{i,j}} \mathcal{D}(H_k, H_l) \right) .\log \frac{N}{f_i}}{\max_r \left[\left(\sum_{H_k \in \mathcal{H}_{i,q}} \sum_{H_l \in \mathcal{H}_{i,r}} \mathcal{D}(H_k, H_l) \right) .\log \frac{N}{f_i} \right]} \tag{7}$$

where $H_l \in L$.

In effect, the Complete Matching approach assigns a weight according to how the multiple occurring term is distributed in the record with respect to the query.

The role of the denominator in Equations (6) and (7) can now be better explained. Consider a situation where a query consists of several terms. Assume the record d_j contains only one of the terms, but it occurs several times and in the desired location. If we were to normalize the weight assigned to that term in d_j with respect to the Euclidean length of the weight vector for d_j then record d_j would be ranked too high (i.e. it may outrank records that contain d_j the same amount of times and also in the correct field, merely because the other records contain some of the other terms as well). The denominator used is an attempt to ensure that no single term ever dominates the ranking of a multiple term query.

For more information on the actual implementation of the algorithms and more detail on why the second algorithm can not be implemented as efficiently as the first, see [14].

7.3 Dates

The search algorithm can be greatly improved by allowing the queries of the following form:

```
Firstname(John) AND Birthdate(1980.01.12)

Name(John) AND Birthdate(1980-1981)
```

Note that a partial date or even a period can be specified when an exact date is associated with an event.

Unfortunately, it is impossible to treat dates in the same way as normal index terms because dates are often merely incorrect estimates or a period between two dates. Expecting an exact match with a query date is thus unrealistic. An approach that provides a measure of how well a date in a specified field in the query matches a date in a record is needed. It is important to note here that searching on a date alone would not be very useful in locating an individual because of the large number of people who have dates associated with them in any given period.

The field matching can be achieved using function $\mathcal{D}$ defined above. The general weight of a date should not be calculated as in Equation (5), rather, weights applicable to dates should take into account how close the relevant dates are to the query date. An approach similar to that followed by De Kock [15] will be followed. The selected approach will be given first and motivated below.

Let a date be represented as the number of days that have passed since 1 January 1 A. D. Define a period p as the tuple (*start date*, *end date*). The distance between two periods $p_1 = (s_1, e_1)$ and $p_2 = (s_2, e_2)$, $g(p_1, p_2)$ is given by

$$g(p_1,p_2) = \begin{cases} 999999: & \text{either } s_1 = 0 \text{ or } s_2 = 0; \\ \mid s_1 - s_2 \mid: & e_1 = 0 \text{ and } e_2 = 0; \\ 0.5: & e_1 \neq 0,\ e_2 = 0 \ \text{ and } s_2 \in [s_1, e_1]; \\ 0.5: & e_2 \neq 0,\ e_1 = 0 \ \text{ and } s_1 \in [s_2, e_2]; \\ 0: & e_1, e_2 \neq 0 \text{ and } [s_1, e_1] \cap [s_2, e_2] \neq \emptyset; \\ \min(\mid e_1 - s_2 \mid, \mid e_2 - s_1 \mid): & \text{otherwise.} \end{cases} \tag{8}$$

A large distance between two dates is given if either date is zero. A value of 0.5 is given if one is a date and the other is a period and the date lies within the period. This is to ensure that, when a query only specified a start date, occurrences of single dates that match the query date exactly will rank very slightly higher than records that merely contain a period in which the query date falls.

The similarity $0 \leq s(g(p_1, p_2)) \leq 1$ between two periods p_1 and p_2 is calculated as follows:

$$s(x) = e^{-4x^2/a^2}.$$

As can be seen from Figure 5, $s(x)$ is a bell shaped curve that decreases to almost zero for $x > a = 3\,650$. This function ensures that a non-zero relevance will be given for any two dates (rather than assigning the similarity value to zero for large differences). Small differences in dates (a few days) will give a value of almost 1, which will decrease steeply as the differences become greater. The value of $3\,650$

was chosen since it corresponds to about 10 years which is a good approximation of the period in which a date can be deemed relevant.

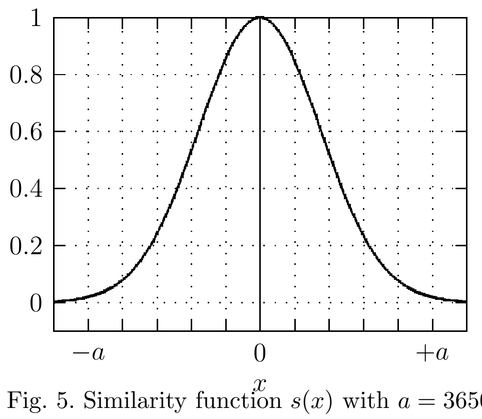


Fig. 5. Similarity function $s(x)$ with $a = 3650$

Given a query, q, that contains a period, p_i, we define the weight of a period term, $p_{i,j}$, in record d_j as

$$w_{i,j} = s(g(p_{i,q}, p_{i,j})). \max_{H_s \in \mathcal{H}_{i,q}, H_v \in \mathcal{H}_{i,j}} \left(\mathcal{D}(H_s, H_v)\right). \tag{9}$$

To be consistent with the weights calculated for normal terms, the weight for each period term is divided by the maximum weight found for the period in all records:

$$w_{i,j} = \frac{s(g(p_q, p_{i_j})). \max_{H_s \in \mathcal{H}_{i,q}, H_v \in \mathcal{H}_{i,j}} \left(\mathcal{D}(H_s, H_v)\right)}{\max_r \left(s(g(p_q, p_{i_r})). \max_{H_s \in \mathcal{H}_{i,q}, H_x \in \mathcal{H}_{i,r}} \left(\mathcal{D}(H_s, H_x)\right)\right)} \tag{10}$$

This weight can be used in Equations (1) and (2) in the same way as the weight for normal terms, given by Equations (6) or (7), is used.

8 SEARCH PROCESS

Define the set of individuals that can be used in queries to locate a person to be $\Delta = \{$`Person`, `Mother`, `Father`, `Spouse`, `Child`$\}$. Let Λ be the set of all fields that can be used in a query, for example `Birth Place` or `Surname` (see Figure 4). Formally, queries have the following form:

$$\bigvee_{i=1}^{m} \bigwedge_{j=1}^{n_i} \delta_{i,j}(\lambda_{i,j}(k_{i,j})) \tag{11}$$

where $\delta_{i,j} \in \Delta$, $\lambda_{i,j} \in \Lambda$ and $k_{i,j}$ is a term or a period. With the information in the query, Equations (1) and (2) can be evaluated for each record using Equations (6) or (7) for general terms and Equation (10) for dates. Note that for searches using relatives the weights of the terms must be calculated in the record of the relevant relative of the current individual's record.

Once a measure of similarity to the query has been assigned for each record, these records can be sorted in descending order of similarity and presented to the user.

9 SPELLING VARIATIONS

Spelling variations (especially of names and place names) are a complex problem which is compounded if a database contains words from more than one language. For example, the city *Port Elizabeth* is referred to as *Die Baai* in Afrikaans and *iBhayi* in Xhosa. De Kock [16] proposed a model where names are grouped into Equivalence and Similarity classes to allow for names that are alike. Two names are equivalent if both are equivalent to the same set of names, i.e. the relation is symmetric and transitive. Two names are similar if the relation is not necessarily symmetric or transitive, for example, *Pieter* is similar to *Peter*; *Peter* is similar to *Peet*, but *Pieter* is not similar to *Peet.*

These ideas were extended to all terms in the database. Equivalence classes provide a neat way to deal with terms from more than one language. When a search is performed, the query is automatically extended to include all equivalent and similar terms. For more details see [14].

10 IMPLEMENTATION AND PERFORMANCE

The algorithms were implemented using the data in PEGIS for the experiment. The database contains 589814 records and comprises a total of 189MB of data. Efficient B-tree indexes point to postings containing all information to evaluate Equations (6), (7) or (10) for any term. These postings (sometimes as many as 80 000) are repacked on the hard drive after the indexing process to decrease the amount of time it takes to load into memory and thus lowering the time it takes to do a merge. A correlation table was manually created. Further indexes were created to facilitate searches for an individual based on information on relations, for example a child or parents.

Performance experiments were done on a Pentium III computer with 512 MB of RAM. Queries that return a large number of hits (in excess of 50 000) yield response times of between 0.5 and 1.6 seconds. For smaller number of hits (less than 15 000) response times of less than 0.5 seconds can be expected.

Queries that contain dates take considerably longer, mostly about 8 seconds. The reason why these queries take so much longer than searches that contain only terms is that Equation (10) is evaluated for *all* dates in the database. Equations (6) and (7) are only evaluated for each of the terms in the query. Search times on dates were originally improved by limiting the search to records that contain the normal terms only. Even after this improvement the number of date evaluations in a date search is much more than that of normal terms. For example, the term

Farmer with an occurrence frequency of 16 148 is the most frequently occurring word in the database. If a query consists of the term *Farmer* and a date, the ±15 000 records that contain the term will contain about 30 000 dates. The real bottleneck occurs when these dates have to be retrieved from disk and evaluated. A solution to the problem is to store the date index in memory, or to make sure that it is cached before a search is done. The same queries that take about 8 seconds take between 0.5 and 1.5 seconds when cached.

11 CONCLUSIONS

The proposed algorithm makes use of the well established Extended Boolean Model and incorporates Structured Text Retrieval Models to provide a hybrid approach to searching for individuals in the biographical database. A correlation function is used to determine the relevance of fields specified in the query and the fields where terms are located in each record. The fact that certain restrictions are placed on the searches ensures the tree matching algorithms can be implemented efficiently. The structural match of records to queries is integrated in the term weights of the Extended Boolean Model, hence retaining the advanced search features of the Extended Boolean Model. Uncertainties in dates are catered for and an elegant solution is suggested for difficulties that arise from searches on periods between dates. The provision for searches on dates ensures a powerful final search algorithm. Since the inaccuracies in the Fast Matching algorithm will infrequently affect the search results, the Fast Matching algorithm can be used for most searches and the use of the Complete Matching algorithm can be reserved only for searches where further refinement on the record ranking is necessary. The algorithm allows for searches based on information on the relatives of an individual.

Many of the complications of searching biographical databases consisting of both structured and unstructured data have been successfully addressed.

REFERENCES

[1] Navarro, G.—Baeza-Yates, R.: A Language for Queries on Structure and Contents of Textual Databases. In Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development inIinformation Retrieval, ACM Press 1995, pp. 93–101.

[2] Navarro, G.—Baeza-Yates, R.: Proximal Nodes: a Model to Query Document Databases by Content and Structure. ACM Transactions on Information Systems (TOIS), Vol. 15, 1997, No. 4, pp. 400–435.

[3] Baeza-Yates, R.—Navarro, G.: Integrating Contents and Structure in Text Retrieval. ACM SIGMOD Record, Vol. 25, 1996, No. 1, pp. 67–79.

[4] Kilpelinen, P.—Mannila, H.: Retrieval from Hierarchical Texts by Partial Patterns. In Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Iinformation Retrieval, ACM Press, 1993, pp. 214–222.

[5] Burkowski, F. J.: Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Text. In Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM Press 1992, pp. 112–125.

[6] Kilpelinen, P.—Mannila, H.: Grammatical Tree Matching. In Proceedings of the Third Annual Symposiom on Combinatorial Pattern Matching, Springer-Verlag 1992, pp. 162–174.

[7] Fuhr, N.—Grosjohann, K.: XIRQL: A Query Language for Information Retrieval in XML Documents. Research and Development in Information Retrieval, 2001, pp. 172–180.

[8] Kotsakis, E.: Structured Information Retrieval in XML Documents. 17th ACM Symposium on Applied Computing, 2002, pp. 663–667.

[9] Salton, G.—Fox, E. A.Wu, H.: Extended Boolean Information Retrieval. Communications of the ACM, Vol. 26, 1983, No. 11, pp. 1022–1036.

[10] Ribeiro-Neto, B.—Baetza-Yates, R.: Modern Information Retrieval. 1st edition, Addison-Wesley 1999.

[11] Salton, G.—Lesk, M. E.: Computer Evaluation of Indexing and Text Processing. Journal of the ACM (JACM), Vol. 15, 1968, No. 1, pp. 8–36.

[12] Durkin, J.: Expert Systems, Design and Development. 1st edition, Macmillan Publishing Company 1994.

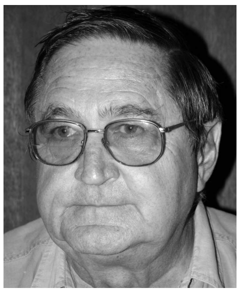
[13] Salton, G.—Buckley, C.: Term-Weighting Approaches in Automatic Text Retrieval. Information Processing and Management, Vol. 24, 1988, No. 5, pp. 513–523.

[14] du Plessis, M. C.: Search Algorithms on Structured and Unstructured Data in a Large Database. M. Sc. dissertation, University of Port Elizabeth, Port Elizabeth, December 2004. Supervisor: G. de V. de Kock.

[15] de Kock, G. de V.–de Kock, M. M..: A Search Algorithm for an Archive Database Using Person and Place Names. In Proceedings of the 2nd International Conference on Computer Science and its Applications, US Education Service, LCC, 2004, pp. 216–223.

[16] de Kock, G. de V.: Searching on Full Name Providing for Spelling Variations. In Proceedings SAICSIT 2002 : Enablement through Technology, Annual Research Conference of SAICSIT, Abstract, ACM International Conference Proceedings, 2002.

Mathys C. du Plessis received his M. Sc. in computer science and information systems in 2005 at the Nelson Mandela Metropolitan University (formerly the University of Port Elizabeth). He is currently enrolled for a Ph. D. in computer science at the University of Pretoria and holds a Junior Lectureship at the Nelson Mandela Metropolitan University.



Gideon de V. de Kock studied mathematics at the University of Stellenbosch and afterwards worked at the Numerical Analysis Section of the Council for Science and Industrial Research. He then held teaching positions at the Universities of Stellenbosch, South Africa, The North and Port Elizabeth, where he was Head of Department from 1974. He retired at the end of 2004 and still pursues his interest in genealogical research and genealogical information systems.