University of Alberta

PalmIRA: Information Retrieval in the Palm of Your Hand

by

Jeffery Antoniuk

Technical Report TR 02-16
July 2002

DEPARTMENT OF COMPUTING SCIENCE
University of Alberta
Edmonton, Alberta, Canada

PalmIRA: Information Retrieval in the Palm of Your Hand

Jeffery Antoniuk

July 29, 2002

Abstract

Personal Digital Assistants (PDAs) are becoming increasingly popular and with this increase in popularity comes the increased number of applications that store textual data within the PDA. The research area of information retrieval has developed a number of effective and efficient techniques for more powerful desktop computers which can not be directly applied to PDAs due to storage and CPU constraints. This thesis introduces PalmIRA, an information retrieval system containing a PDA based portion and a PC based portion specifically designed for the characteristics of a PDA, more specifically a PalmOS based PDA. The design attempts to create an efficient and effective information retrieval system. This thesis also introduces a new collection fusion technique and a few measures to evaluate the effectiveness of the proposed collection fusion procedure.

1 Introduction

Nowadays, Personal Digital Assistants (PDAs) are becoming increasingly popular. A PDA offers "carry anywhere" portability while retaining, although at much lower levels, storage and functional capability of Personal Computers (PCs) and laptops. The data storage capacity, dynamic memory, and processor speed of a PalmOS based PDA is very low when compared to a modern PC or laptop. On the positive side of the trade-off are the PDA characteristics of being lightweight (e.g., Handspring Visor Deluxe weighs 5.4 oz. including batteries) and having the very low power requirement necessary for extending battery life. These two features allow a PDA to extend the availability of computational device use beyond the reach of PCs or laptops. The portable extension is also exhibited by the symbiotic relationship that the PalmOS based PDA maintains with the PC via the HotSync synchronization of data between the PalmOS based PDA and the PC [34]. The PalmOS HotSync Manager acts as a communication layer between the PC and the PDA such that the data can be exchanged, updated, and synchronized between the PC and the PDA. If the PDA were to suffer a data loss event, the HotSync process stores on the PC the data required to restore the PDA data to the data present at the time of the last HotSync operation.

Given the fact that a PalmOS based PDA can store large amounts of textual data (e.g., Handspring Visor Deluxe's 8 MB), users require efficient and effective means to retrieve information from the stored textual data. The book by Baeza-Yates et al. [1] describes techniques (e.g., vector model, inverted index) to accomplish information retrieval. These techniques as described in the book have a resource demand that exceeds the constraints of PDAs. In order to implement an information retrieval system that uses the vector model and an inverted index within the constrained environment of a PalmOS based PDA, the information retrieval techniques must be engineered to provide efficient and effective retrieval. The PDA's relationship with a PC via the HotSync process can be exploited such that the PC completes the computationally intensive task of building the inverted file that is written onto the PDA and used by the PDA information retrieval application.

The textual data contained within the PDA resides in multiple collections or databases of documents since each PalmOS application (e.g., MemoPad, Mail, DataBook) maintains its own database. If a user wants to retrieve a set of documents regarding a specific topic, those documents may be spread throughout the multiple databases of documents. One approach is for the user to search each collection individually, evaluate which documents from which result lists are interesting and merge the documents into a global result list by hand. Result merging or in other words collection fusion [48] attempts to automate the process.

The goal of the thesis can be split into the following two parts. The first part is to engineer a proven information retrieval approach to efficiently execute in the constrained environment of a PDA. The second part is to efficiently fuse the result lists of multiple collections taking advantage of the PDA information retrieval engine and to propose new measures to evaluate the effectiveness of a result merging scheme.

1.1 Thesis Outline

This thesis starts out by describing the engineering required to efficiently implement an inverted index and vector model information retrieval scheme while attempting to maintain the retrieval effectiveness of this approach in the PDA (Chapter 2). The next chapter describes a new collection fusion technique and compares this technique to existing techniques using existing and newly proposed effectiveness measures using a simulation of the PDA information retrieval system (Chapter 3). The collection fusion techniques are chosen because they can be implemented efficiently within the PDA information retrieval system. Finally, conclusions are drawn and opportunities for future work are presented (Chapter 4).

2 Single Collection Information Retrieval in a PDA

2.1 Introduction

Single collection information retrieval focuses on a search engine finding and ranking interesting documents from one database of documents. The setup typically consists of an index (e.g., inverted index, signature files, suffix trees) used by the retrieval model (e.g., vector model, boolean model, probabilistic model) to retrieve and rank documents from a database or collection of documents [1].

This chapter focuses on how to efficiently engineer an information retrieval system for a PalmOS based PDA environment while maintaining retrieval effectiveness. This chapter begins with an introduction to the related work (Section 2.2) and continues with an overview of the PalmOS based PalmIRA information retrieval system (Section 2.3). Next described is the model used to predict which documents are relevant to the query (Section 2.4). The chapter then goes on to describe how the PC is utilized to build the inverted index during PC - PDA synchronization (Sections 2.5 and 2.6), and how the PalmOS based PDA information retrieval application functions (Section 2.7).

2.2 Related Work

The main components of an information retrieval system [1][16] involve an indexing structure containing key terms along with the documents the term appears within, an information retrieval model to determine the relevance between the query and the documents and a query language. Research produced a number of variations on the above for indexing (e.g., signature files, suffix files, inverted files etc.) and for retrieval models. (e.g., vector model, probabilistic model, boolean model, etc.) The research papers evaluate these information retrieval techniques on large computing systems. Since the goal of this chapter is to describe an efficient way to engineer a vector model, inverted index retrieval system for a PDA, the details of this approach are described in Section 2.4 and Section 2.5 respectively. Further information about other approaches can be found elsewhere (e.g., [1]).

At the time of writing this thesis, there are no known research oriented approaches for information retrieval of textual data residing on a PDA. However there are two closed source industry solutions not part of published research papers. Intelligentfind is a PDA application that uses a sequential search method to search through a PDA database. It does not use an indexing structure [33]. It suffers a high retrieval time from the low speed CPU of a PDA versus the amount of textual data that can be stored on the PDA and the number of terms in the query.

IBM has created an information retrieval tool that off-loads the index creation from the PDA to the PC during the PDA-PC synchronization (similar to our approach). The tool was known as IBM Palm Pirate[1] and has changed its name to Pirate Search [41]. The results are displayed in order of relevance by some secret ranking algorithm. This application creates a copy of the database being indexed and inserts an index into the copy. Hence, there is a (critical) space overhead associated with the technique.

2.3 PalmIRA System Overview

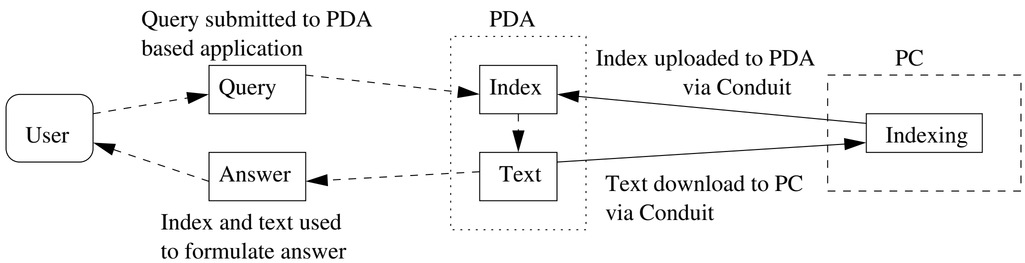


Figure 1: PalmIRA System Overview

There are two parts to the PalmIRA information retrieval system: a PalmOS based application for the PDA and a PalmOS Conduit (i.e., a plug-in module for the PC portion of the HotSync process).

The PalmOS application residing on the PDA behaves much like a normal search engine in that a user enters a query and a ranked list of resulting documents is displayed. In order to allow fast, efficient, and effective retrieval, an inverted index stores information required to calculate the degree of similarity based rank using the vector model [1] between the query and each indexed document.

A conduit is a plug-in module (dynamic link library) for the PC based Palm HotSync Manager. A conduit, once registered with the PalmOS HotSync manager, is executed during PDA synchronization time. An example of a conduit synchronization could involve the synchronization of a PalmOS address book with a PC address book application. If a new address is entered in to the PalmOS based PDA, a conduit is executed during the HotSync synchronization between the PC and the PalmOS based PDA that copies the entry from the PDA and places the entry into the PC based address book. If the conduit specifically designed for this task does not exist then the PC address book and PalmOS address book are not synchronized. The PalmOS Conduit API is capable of creating conduits with more sophisticated logic than described in the example and PalmIRA takes advantage of these features.

The ability to build conduits allows the inverted index to be built during the HotSync synchronization process by the PC. Once the inverted index is built, the inverted index is written back to the PDA.

2.3.1 PalmOS Memory Model

PalmOS uses a 32-bit architecture with basic data-types of 8, 16, 32 bits [51]. PalmOS uses one or more memory modules known as "cards" (all devices at the time of writing this thesis contain only one card) [27]. A card contains storage and dynamic memory.

Memory is allocated in chunks where a chunk is at least 1 byte to slightly less than 64 KB of contiguous memory (due to system overhead requirements). Chunks can be movable or non-movable. Non-movable chunks are referenced by pointers i.e., referenced by the fixed address of the memory location. Movable chunks are referenced by memory handles [34]. Memory handles store a reference to an entry in the "Master Pointer Table". This table in turn stores the address of the movable chunk and the table is updated during the compaction process

[1] Palm Pirate authors: M. Herscovice, D. Cohen and Y. Maarek

of the PalmOS memory manager with the new address of each chunk after the movable chunk is relocated in the storage or dynamic RAM to defragment the memory. To read the chunk when using a memory handle, the chunk is locked, becomes temporarily non-movable and a temporary (until unlocked) pointer (as described previously) is used to access the chunk. This is slightly slower than the pointer access method but allows defragmentation to occur which is important when considering the amount of available memory.

The memory model designed for PalmOS 3.5-4.0 differs from that used for PCs. PalmOS PDAs at the time of writing this thesis do not have hardware similar to that of a PC hard-disk and not do use a file system similar to that of a PC. A PDA contains dynamic RAM with similar functions to PC RAM (e.g., stack space, dynamic memory allocations) and storage RAM with a similar function to that of a hard-disk or other storage device for a PC [13].

Dynamic RAM is divided into areas for:

- system globals (approx. 2.5KB),
- TCP/IP stack (32 KB),
- system dynamic allocation (variable),
- application stack (4 KB default),
- dynamic allocation, application globals, application static variable (¡=36 KB)

Each is item in the list is dedicated a certain amount of physical memory and the amount is dependent on the version of PalmOS. The amounts described are for PalmOS 3.0-3.3, the version of PalmOS on the PDA used for experimentation described in Section 2.8. This version has a total of 96 KB of dynamic memory [51].

Storage RAM is where PalmOS stores data. Data is stored in databases as opposed to the notion of files on PCs. A database consists of a list of records and database header information. Each record is stored in a chunk. The chunks that make up a database do not have to be contiguous in memory and are rearranged when needed to defragment memory which merges free space fragments. Records are edited in place instead of reading records into dynamic memory [26]. Finding a record in a database can occur in three ways:

- Unique Record ID,
- “Index” value between 1 and the number of ordered records in the database and this access method is analogous to an array data structure access,
- Search of a list of records using the internal data of record (key) e.g. binary search

To combat the constraints of a PDA, small memory software design patterns described by Noble and Weir were used [34]. These include:

- Packed Data - data is not aligned on word boundaries in the PalmOS databases.
- Compaction - defragmentation of dynamic or storage RAM to reduce the fragmented free space by rearranging movable chunks. This compaction normally executes when a memory allocation does not find a large enough chunk of contiguous memory.

2.4 Information Retrieval Model

Information retrieval attempts to predict what documents are relevant to a given query. A specific model attempts to accomplish this prediction. The type of implemented model for this project is the vector model. The following description of the vector model is based largely on Baeza-Yates et al. [1].

The vector model is based on the idea that certain terms are more meaningful than others. If a term occurs a relatively high number of times in a document then the term is likely more important than other terms. The exception is stop-words[1][8] (articles, prepositions, and conjunctions) which occur very frequency and express little to no meaning. This idea of intra-document similarity is known as the term frequency (*tf*) described by Equation 1 where $freq_{i,j}$ is the frequency of term k_i in document d_j and the $max_l freq_{l,j}$ is maximum term frequency in document d_j.

$$tf_{i,j} = \frac{freq_{i,j}}{max_l \, freq_{l,j}} \tag{1}$$

Also, if a term occurs within a low number of documents, this term is likely more important than other terms. This type of term tends to show a larger inter-document dissimilarity. This idea is known as the inverse document frequency (*idf*) described by Equation 2 where N is the number of documents in the collection and n_i is the number of documents that term k_i appears in.

$$idf_i = \log \frac{N}{n_i} \tag{2}$$

How meaningful a document term is depends on the intra-document similarity and the inter-document dissimilarity measure known as the weight of term k_i in document d_j. The document term weight ($w_{i,j}$) is determined by Equation 3.

$$w_{i,j} = tf_{i,j} \times idf_i \tag{3}$$

How meaningful a query term is depends on the weight of term k_i. The query term weight ($w_{i,q}$) is determined by Equation 4.

$$w_{i,q} = (0.5 + \frac{0.5 \, freq_{i,q}}{max_l \, freq_{l,q}}) \times \log \frac{N}{n_i} \tag{4}$$

$$sim(d_j, q) = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}} \tag{5}$$

The vector model represents the query and the document as a vector. Each dimension of the vector represents one term in the set of terms defined by union of the terms within the collection of documents and the query terms. Each dimension contains a weight. The cosine of the angle between the document vector and the query vector determines the degree of similarity between the two and is given by Equation 5. In other words, the lower the angle between the two vectors, the higher the degree of similarity. The above are the information retrieval equations as per Baeza-Yates et al. [1].

If $w_{i,j}$ and $w_{i,q}$ are calculated at query time on the PDA, then predictably large execution times result from the constrained PDA to determine $max_l freq_{l,j}$ and to do the floating point arithmetic. To increase efficiency in the PDA environment, it is possible to only calculate the query dependent portion involving $w_{i,q}$ on the PDA. The document dependent portion $w_{i,j}$ can be calculated for all terms on the PC to increase efficiency. Towards this goal, instead of storing the frequency of term k_i in document d_j in the inverted list [1], another value θ (Equation 6) may be stored. Equations 6 and 7 are rearrangements of Equation 5 such that Equation 6 describes the portion calculated on the PC and Equation 7 describes the portion calculated on the PDA.

θ from Equation 6 is converted from a floating point number into a byte sized number by multiplying by 100 and rounding. Storing a byte representation reduces the inverted index storage required and is necessary given the constrained storage of a PDA.

$$\theta_{i,j} = 100 \times \frac{tf_{i,j} \times idf_i}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2}} \tag{6}$$

$$sim(d_j, q) = \frac{\sum_{i=1}^{t} \theta_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,q}^2}} \tag{7}$$

2.5 Inverted Index Construction

The goal of an inverted index is to index a text collection such that searching the inverted index is cheaper than searching the entire text collection. A inverted index consists of a vocabulary and a posting list for each term. The posting list identifies what documents that term occurs within [1]. All documents that contain a given term are easily determined by locating that term's posting list.

The PC - PDA synchronization feature of the PalmOS based PDA provides the opportunity to build a conduit as part of the PalmIRA information retrieval system. The purpose of this conduit is to off-load part of the processing required for the information retrieval process from the PDA to the PC. The processing off-loaded onto the PC involves: downloading database (Section 2.5.1), preprocessing steps (Section 2.5.2), building an efficient data structure (Section 2.5.3), calculating the weight $w_{i,j}$ of term k_i in document d_j (Section 2.5.4) and building the inverted index database that is stored on the PDA (Section 2.6).

2.5.1 Acquisition of PalmOS Based PDA Data

In the PalmOS environment, each application stores data items in a database. The PalmOS database is similar to a PC data file. The PalmOS database is composed of a number of records. For example the PalmOS MemoPad application stores each MemoPad memo in one record. Each record within a PalmOS database is identified using a unique record id.

The first step in building the inverted index is to download each record from the PDA to the PC during the synchronization. The PalmIRA conduit reads each record directly from the PDA database. Because another conduit may also download the same records from the PDA to the PC, this may cause extra downloading. PalmIRA cannot take advantage of the other conduits because the other (if present) conduit that accesses the same PalmOS records as PalmIRA may be any possible third party conduit that stores the data on the PC in any conceivable format. Not all of the possible third party formats can ever be interpreted by PalmIRA. Hence, the trade-off is the guarantee in reading the text for the extra downloading.

2.5.2 Document Preprocessing

Each record or textual item that is read from the PDA is parsed into individual lower-case terms. Using regular expressions [24], a term is defined as consecutive alpha numerical characters (a-z, 0-9) or (-, %, $) where each term is delimited by any character not "`valid`={a-z, 0-9, -, %, $}", "[¬`valid`]+[a-z,0-9]+[¬`valid`]+", and the term containing a - character must have a alphabetical character following and preceding the - character, "[¬`valid`]+[a-z,0-9]+[-][a-z,0-9]+[¬`valid`]+", and the term containing a % or $ must have a digit character following or preceding the % or $ character "[¬`valid`]+[0-9]+[%,$][¬`valid`]+" or "[¬`valid`]+[%,$][0-9]+[¬`valid`]+". For example "brother-in-law no.7 %100 $110" would contain five terms: brother-in-law, no, 7, %100, $100. The period between no and 7 is considered a delimiter because period is not in the set `valid`.

Stop-words are eliminated using a binary search against a static list of stop-words available from [8]. Stop-words are terms in language that carry little or no meaning such that they do not make good document discriminators [1] (e.g., "to","the"). Prepositions, articles, and conjunctions comprise a large portion of the stop-word list.

Besides eliminating words that are poor document discriminators, the stop-word elimination helps to decrease the size of the inverted index. Case sensitivity is ignored, i.e., all terms are converted to lower case.

A preference in the application allows for the choice of stemming the terms within the collection using Porter's stemming algorithm [35]. The stemming algorithm replaces the suffix of the term based on a number of rules. The idea is that the rules remove plurals, past tense suffixes and the like (e.g., zooms, zooming, zoomed) from the root of the word. Porter's stemming algorithm is simple and fast given the constraints of a PDA unlike other more complicated stemming algorithms (e.g., N-grams, table lookup, successor variety [1]). The resulting root of the terms (e.g., zoom) may increase precision and recall (query's effectiveness [1]) while decreasing the index size, synchronization time, and not increasing the time to execute a query.

Each term is then added to the sparse matrix data structure described in Section 2.5.3.

2.5.3 Sparse Matrix Data Structure

In-order to calculate the weight of term k_i in document d_j, Equation 3 must be calculated. Equation 3 requires the $freq_{i,j}$ of term k_i in document d_j be stored in order to calculate the weight $w_{i,j}$. The frequency information is best stored in a matrix such that accessing row i, column j returns the $freq_{i,j}$.

A sparse matrix data structure was chosen since the structure can store the frequency count of term k_i in document d_j (and later a information retrieval weight) and since many terms k_i do not exist in document d_j (i.e., sparse data). In addition, the information retrieval process (more specifically Equation 5) does not require the term k_i in document d_j frequency if the frequency is 0. Since documents do not contain an instance of each of the terms, this produces a matrix with entries with 0 frequencies if a non-sparse matrix is used. The result of using a non-sparse matrix is inefficiencies in traversing unneeded 0 frequency items and in memory requirements to store the 0 frequencies. The larger amount of memory required to store the non-sparse matrix produces larger amounts of swapping and a significant increase in execution time over the sparse matrix data structure.

Next, the sparse matrix data structure is defined. The sparse matrix is made up of 3 data structures: Term Header, Document Header, and Nodes as displayed in Figure 2.

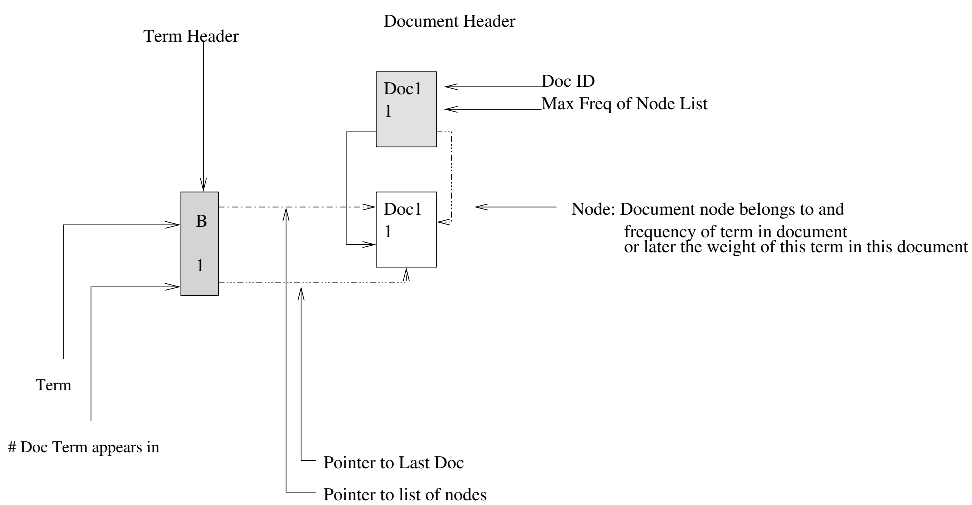


Figure 2: Sparse Matrix Overview

The nodes are the entities that make up the cells within the matrix. A node in column j in row i represents document d_j contains term k_i [2]. The Node item ($\texttt{Node}_{i,j}$) maintains the following elements:

[2]Each node is represented by a 16 bytes long structure.

- $\texttt{Node}^{nRow}$- pointer to the next node in a row-wise direction
- $\texttt{Node}^{nCol}$- pointer to the next node in a column-wise direction
- $\texttt{Node}^{Value}$- a floating point value (4 bytes). The value is used both for a frequency count (*freq*$_{,j}$) as the documents to parse are parsed and as a weight calculated by Equation 6 to save space.
- $\texttt{Node}^{Did_{d_j}}$ - unique id (4 bytes) of the document represented by the column the node exists in.

Each item of the Term Header (TH) represents a row in the sparse matrix. The TH is an array of TH items[3]. The array data type allows for fast binary searches. Each TH item maintains the following elements:

- $\texttt{TH}^{k_i}$- the term represented by the TH item k_i
- $\texttt{TH}^{LinkPtr}$- a pointer to a linked list of nodes
- $\texttt{TH}^{LastPtr}$- a pointer to the last node in the linked list
- $\texttt{TH}^{n_i}$- the number of documents that the term appears in (n_i which is used in Equation 2).
- $\texttt{TH}^{idf_i}$- the float value of inverse document frequency (*idf*) of term k_i (Equation 2)

Each item of the Document Header (DH) represents a column in the sparse matrix. The DH is made up of a linked list of items [4]. Each DH item maintains the following elements:

- $\texttt{DH}^{Did_{d_j}}$ - document id value by the DH item
- $\texttt{DH}^{nextPtr}$- a pointer to the next DH Item in the linked list of DH Items
- $\texttt{DH}^{LinkPtr}$- a pointer to a linked list of nodes
- $\texttt{DH}^{LastPtr}$- a pointer to the last node in the linked list
- $\texttt{DH}^{max_l freq_{l,j}}$ - the max frequency ($max_l\ freq_{l,j}$) of any single node within the column representing document d_j (i.e max frequency of any term in document d_j)
- $\texttt{DH}^{Denom}$- the float value of the denominator of Equation 6[5].

The reasoning behind the usage of each of the structure members for the the Document Header (DH), Term Header (TH), and Node structure will become clear as the following example progresses. The construction and efficient utilization of the sparse matrix given a very simplified example collection of documents is worked through to help explain the purpose of each structure member for the following four (very simplified) documents [6]:

Doc1: Contents: B C A C A A

[3] Each item is of size 20 bytes plus the memory required to represent the string value of the term.

[4] Each item in the DH is represented by a 24 bytes long structure.

[5] PalmIRA uses a float instead of double data type in the MS Visual C++ environment because floats are 4 bytes whereas doubles are 8 bytes. This reduces the amount of dynamic memory required by the sparse matrix structure. The float data type retains 7 digits of precision as opposed to 15 digits for a double data type. The extra precision is not required because of the technique to reduce storage costs described in Section 2.6 to create the PalmOS database representation of the inverted file. The 4 byte float data type values range from approximately 1.17549×10^{-38} to $3.40282 \times 10^{+38}$ which is large enough for the purposes described above in the Document Header, Term Header, and the Nodes.

[6] Notice the document order is based on the alphabetical contents of the documents. The textual documents are read from the PDA by PalmOS index order which has been specified in the PalmOS application (e.g., MemoPad) as alphabetical.

Doc4: Contents: F

Doc3: Contents: F

Doc2: Contents: F A C

First the PC based conduit reads the first record by index from the PalmOS based PDA database. Using the example, the first record is Doc1. Doc1 is first added as a column representative (i.e., DH) within the sparse matrix. Doc1 is preprocessed by the process described in 2.5.2. Each term, defined by the preprocessing step, is dealt with in the following manner depending whether or not it exists in the sparse matrix structure.

A binary search of the TH for the term in question determines if the term exists resulting in:

1. If the term is not found during the binary search of the TH items, then add the term to the TH array maintaining the sorted order via an insertion sort. Next, add a new Node item to the structure with frequency ($\mathtt{Node}^{Value}$) "1". In addition, update $\mathtt{TH}^{n_i}$ and update $\mathtt{TH}^{LastPtr}$ for term k_i. Also updated is the max frequency in the corresponding DH item for document d_j ($\mathtt{DH}^{max_l freq_{l,j}}$).

   Since a binary search precedes the insertion sort, the insertion sort can be optimized. The position to insert the term into the array can be approximated from position references obtained within the binary search algorithm when the term is not found. The technique involves using the mid-point information produced by the binary search algorithm to act as an approximation of the position where the new distinct term should be added. From the approximation, the exact position can be found quickly by moving backward or forward in the array from the approximated position extracted from the binary search. Once found, room is made for the term in the array [7] [8].

2. If the term k_i is found by the binary search to be in the array of TH items then:

   (a) if term k_i previously occurs in document d_j and the $\mathtt{TH}^{LastPtr}$ of term k_i points to a node that exists in the column represented by Doc d_j then increment the frequency count of the node pointed to by the $\mathtt{TH}^{LastPtr}$ ($\mathtt{Node}^{Value}$) and update the max frequency of the DH item for d_j ($\mathtt{DH}^{max_l freq_{l,j}}$)

   (b) else this is the first occurrence of term k_i in document d_j then add a new node, update $\mathtt{TH}^{LastPtr}$ of term k_i and $\mathtt{DH}^{LastPtr}$ of document d_j pointers to point to the new node and update $\mathtt{DH}^{max_l freq_{l,j}}$ for d_j

The motivation for $\mathtt{TH}^{LastPtr}$ is to reduce the time to traverse the linked list of nodes each time the last node is accessed (when a new node for that term has to be added to the end of the linked list or if that term occurs in the record multiple occurrences and the last node is updated by increasing the frequency count of the node). Nodes are never added to the beginning or middle of the linked list because a new column (DH item) is added to the end of DH list each time a record is read from the PDA data file. For example, nodes are always being added to the current DH item that represents the current record (document) being parsed.

The idea of $\mathtt{DH}^{LastPtr}$ is to reduce the time to find the last node in the linked list of nodes. This is used when a new node is added to find the last node and link the new node to the end of the linked list.

The document id element of the Node item ($\mathtt{Node}^{Did_{d_j}}$) is used to speed up the determination of what column (i.e., document id) the $\mathtt{TH}^{LastPtr}$ represents. Given a reference to a node, the $\mathtt{Node}^{Did_{d_j}}$ prevents a significant number of traversals required to determine what column the node is contained within. This information becomes useful to determine if the node referenced by $\mathtt{TH}^{LastPtr}$ is to be updated (if the document id of the current term

[7] Once the position within the array is located for where the term is to be placed, moving one chunk of memory should be more efficient than copying each individual TH item one array position.

[8] Although adding an item to a linked list would be easier, a linked list approach instead of the array approach to represent the set of TH items suffers from the inability to as efficiently search for an item within the linked list.

equals the $\texttt{Node}^{Did_{d_j}}$) or otherwise linked to a new node (if not equal). Also, the $\texttt{Node}^{Did_{d_j}}$ is useful during the PalmOS database inverted index file creation (Section 2.6).

The linked list of nodes is linked in a column-wise manner in order of first occurrence within the document because there is no need to be able to traverse the column in sorted by term order. The order of occurrence saves the time of the sorting and still provides all the necessary access required by the information retrieval calculations.

$\texttt{TH}^{n_i}$ is updated each time a new node is added to the row to save having to traverse the linked list of items that make up the row to determine the value which is required to calculate inverse document frequency (*idf*) (Equation 2).

$\texttt{DH}^{max_l freq_{l,j}}$ may be updated each time a node is updated (or added). This saves the time of having to traverse the linked list for each column to determine the "max frequency" of each column at a later time. Storing the $\texttt{DH}^{Denom}$ is useful in reducing the time to traverse the linked list similar to $\texttt{DH}^{max_l freq_{l,j}}$.

The next portion describes how the example documents introduced in section 2.5.3 are placed into the sparse matrix. The goal of the example is to help illustrate the motivation behind the sparse matrix data structure definition introduced previously in this section.

First Doc1 is read from the PalmOS based PDA database by the PC based conduit. This causes a new item to be appended to the DH list. The TH is searched for the first term within Doc1 ("B") which is not found. A new TH item is added to the array. A new $\texttt{Node}_{i,j}$ is added with frequency ($\texttt{Node}^{Value}$) = 1 and document id ($\texttt{Node}^{Did_{d_j}}$) of Doc1. The TH item for "B" is updated with $\texttt{TH}^{n_i} = 1$, last node ($\texttt{TH}^{LastPtr}$) and first node ($\texttt{TH}^{LinkPtr}$) pointers pointing to the new node. The DH item for Doc1 is updated with max frequency ($\texttt{DH}^{max_l freq_{l,j}}$) = 1, $\texttt{DH}^{LastPtr}$ and $\texttt{DH}^{LinkPtr}$ pointers pointing to the new $\texttt{Node}^{Did_{d_j}}$ (see Figure 3).

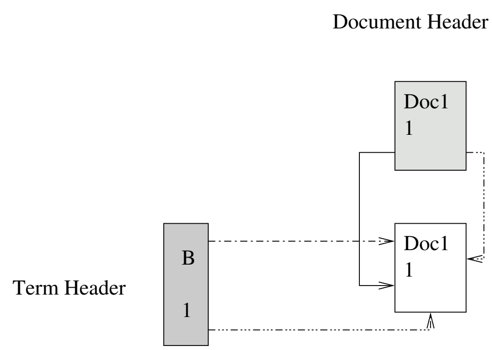


Figure 3: Sparse Matrix Example 1

The next term in Doc1 is "C". The term is searched for in the TH and not found. A new TH item is added to the array in sorted order. As a result, a new $\texttt{Node}_{i,j}$ is added to the internal portion of the sparse matrix by accessing a node via $\texttt{DH}^{LastPtr}$ item for Doc1 (in this case) and updating $\texttt{Node}^{nCol}$ pointer of $\texttt{DH}^{LastPtr}$ node to point to the new $\texttt{Node}_{i,j}$. This saves having to traverse the linked list of nodes starting from DH or TH. The $\texttt{DH}^{LastPtr}$ item for Doc1 is updated (see Figure 4).

The next term in Doc1 is "A" and the process is equivalent to the process for the prior instance of term "C".

The next term in Doc1 is the second instance of term "C". The term is searched for and found in the TH. Since the $\texttt{TH}^{LastPtr}$ points to a node with the $\texttt{Node}^{Did_{d_j}}$ of the current document then just update the frequency of that node ($\texttt{Node}^{Value}$) and $\texttt{DH}^{max_l freq_{l,j}}$ (if applicable). The repeating instances of the "A" term follow a similar process. See Figure 5.

After parsing all the terms in Doc1, the next document is Doc4 since the documents are sorted in alphabetical order of document contents, not in document id order. The new document (Doc4) results in a new item being added to the end of the DH list. The first and only term is term "F" that is processed like the first instance of term "B" in Doc1.

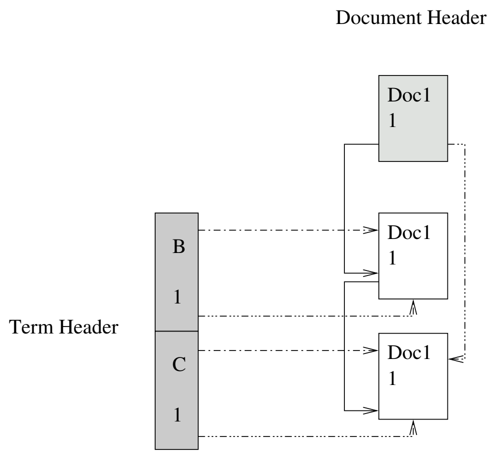


Figure 4: Sparse Matrix Example 2

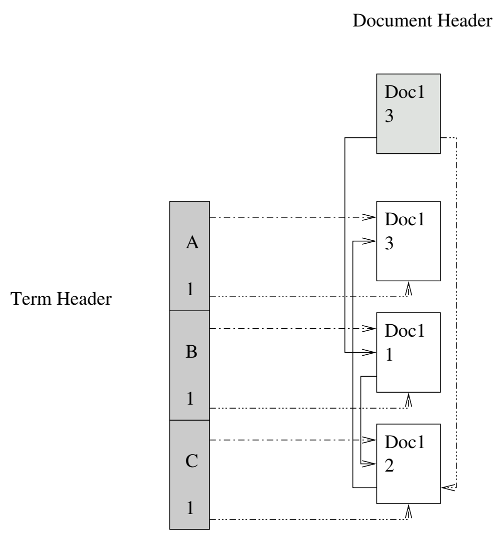


Figure 5: Sparse Matrix Example 3

After parsing all the terms in Doc4, the next document is Doc3. The new document (Doc3) results in a new item being added to the end of the DH list. The first and only term is term "F" which is searched for and found in the TH array. The $\texttt{Node}^{Did_{d_j}}$ pointed $\texttt{TH}^{LastPtr}$ (Doc1) is not equal to the current document id (Doc3) therefore add a new $\texttt{Node}_{i,j}$. See Figure 6.

This continues until all terms in all document have been read and the final result of the sparse matrix structure is displayed in Figure 7.

2.5.4 Weight Calculation

Once all of the documents have been read from the PalmOS database, the document weight (Equation 3) for each term k_i is calculated for each document d_j (i.e., for each $\texttt{Node}_{i,j}$ in the sparse matrix). The same sparse matrix structure is used for both the frequency counts and the weight calculation because by storing the number of documents (N), the max term frequency ($\texttt{DH}^{max_l\, freq_{l,j}}$) in the DH items ($max_l\ freq_{l,j}$) and the number of

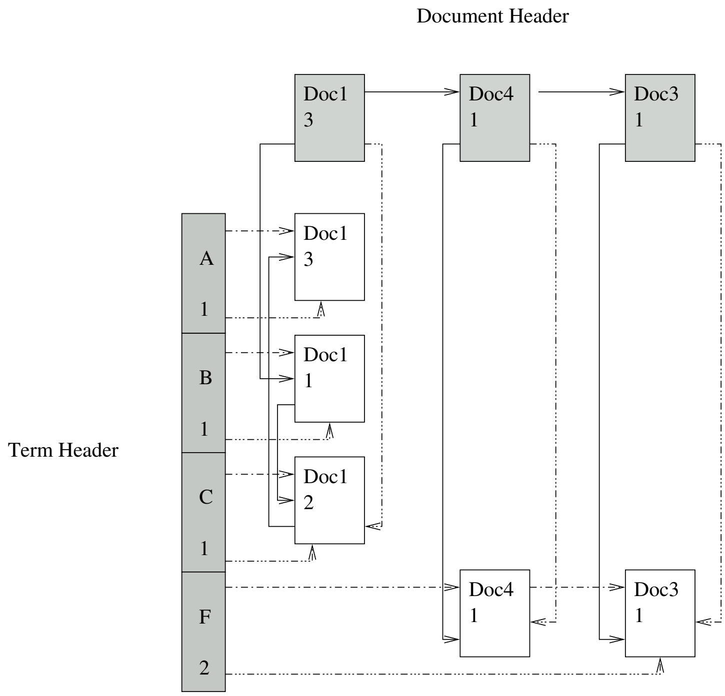


Figure 6: Sparse Matrix Example 4

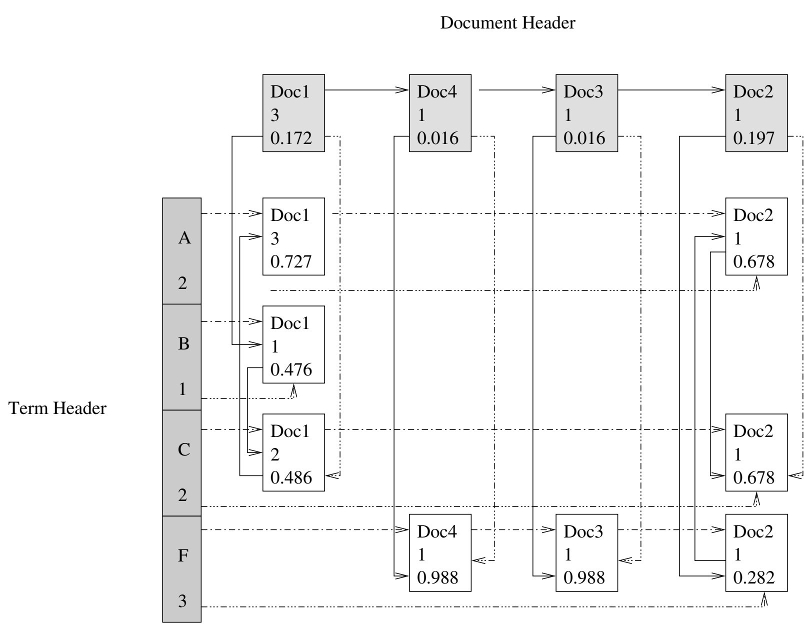


Figure 7: Sparse Matrix Final

documents that a term appears ($\mathtt{TH}^{n_i}$) in the $\mathtt{TH}$ items, the frequency value stored in each $\mathtt{Node}_{i,j}$ can be replaced without creating a second copy of the data structure.

The term weight calculations of Equation 6 are broken into two steps each employing one traversal of the sparse matrix. The first step calculates $w_{i,j} = tf_{i,j} \times idf_i$ for each matrix $\texttt{Node}_{i,j}$ and keeps track of $\sum_{i=1}^{t} w_{i,j}^2$. The second step calculates the square root and division calculation for each matrix $\texttt{Node}_{i,j}$.

The weight calculation is accomplished for each $\texttt{Node}_{i,j}$ by traversing each $\texttt{Node}_{i,j}$ in a row-wise manner. Because of the row-wise traversal, the *idf* (Equation 2) is consistent for each document and can be calculated once for the entire row using the information stored in that `TH` item before the weight calculation loop traverses the entire row. Storing $\texttt{TH}^{idf_i}$ saves the calculation of the *idf* (Equation 2) for each column ($\texttt{Node}_{i,j}$) in the row. Also required to calculate the term weight is the term frequency (*tf*) (Equation 1) portion of the term weight calculation (Equation 3). The max term frequency for document d_j is stored as part of the `DH` for document d_j. This saves time since only the `DH` item needs to be found to acquire the max term frequency for the current document instead of traversing the entire column of the sparse matrix to determine the value.

The `DH` item search is optimized by starting the search at the previously found `DH` item pointer from the previous iteration. The algorithm starts from this point and iterates through the `DH` items until the `DH` item associated with the current $\texttt{Node}_{i,j}$ is found (i.e., the $\texttt{DH}^{Did_{d_j}}$ equals the $\texttt{Node}^{Did_{d_j}}$). For example, after the weight for $\text{node}_{i,j}$ ($\texttt{Node}^{Value}$) is calculated, the weight for the next $\texttt{Node}_{i,j+x}$ is calculated by finding the corresponding `DH` for document d_{j+x} starting from the `DH` item d_j and iterating through the `DH` items until the `DH` item for document d_{j+x} is found. Without the $\texttt{Node}^{Did_{d_j}}$ stored as an element of a node, each node in each column of nodes would have to be iterated through to locate the column that contains the reference to the current node. Each time the weight $w_{i,j}$ for $\texttt{Node}_{i,j}$ is calculated, $\texttt{DH}^{Denom}$ is updated in the corresponding `DH` item d_j.

Since the result of Equation 6 is stored in the inverted index, the document weight value resulting from Equation 3 for each node ($\texttt{Node}^{Value}$) needs to be divided by the $\texttt{DH}^{Denom}$ value stored in the `DH` item for document d_j. This process iterates through the sparse matrix in a column-wise fashion. Using the calculated $\texttt{DH}^{Denom}$ value stored in the current `DH` item, all $\texttt{Node}_{i,j}$ in the column j are traversed and divided by $\texttt{DH}^{Denom}$ value. After each column is re-calculated, each $\texttt{Node}_{i,j}$ then contains the calculated resulting term weight of Equation 6. The column-wise traversal does not require that the current column (`DH` item) be looked up to obtain the $\texttt{DH}^{Denom}$ value as a row-wise traversal would.

The sparse matrix used is based on the sparse matrix data structure described within Horowitz and Sahni [25] with some previously described modifications in order to increase its efficiency. The structure is then used to create the packed inverted index stored on the PDA.

Other data structures could be used in place of a sparse matrix. One option could be to store the data as it is being processed directly into inverted index [12] [1] for the posting lists, a trie [1] [25] [12] for the vocabulary and use a lookup table to keep track of the data stored in the head nodes of the current sparse matrix data structure.

2.6 Inverted Index for a PalmOS based PDA

The inverted index used by the PalmIRA information retrieval system consists of two separate databases: `irTerms` and `irWeight`. The properties of a PalmOS database influence this decision. The limiting properties are: the usable data size of each record (64 KB), the number of records per database (64 K), and size of the overhead (i.e., non-data) segment of each record (8 Bytes) [9].

PalmIRA packs multiple term posting lists in one record saving space and allowing expandability beyond the PalmOS limit of 64 K terms (where one record stores one term). The record packing allows for the number of terms indexed to grow beyond 64 K [10].

[9] One option other than the one used could be to have one term posting list per record which would allow only 64 K terms to be indexed.

[10] Also, every PalmOS PDA record contains a header. Each PalmOS record contributes an additional 8 bytes to the size of the data being stored within the record. The header for each record includes a 4-byte local ID of the record, 8 attribute bits, and a 3-byte unique ID. To decrease the number of records and thus the total size of the database, multiple term posting lists are combined into one record and each posting list is separated by 5 hexadecimal 0x00 bytes (5 bytes is used to store each term posting list item). This saves *((“sizeof header”− “sizeof separator in* `irWeight` *and* `irTerms` *record formats”)* $\times$ *# distinct terms)* bytes (i.e., $(8 - 5 - 1) * 21,289 = 42,578$ bytes).

The result is an `irTerms` file that contains all terms that are indexed by the inverted index. The count of the number of terms from the first term to the term k_i signifies the ordinal of term k_i. This ordinal i is used to determine the position of term k_i posting list within the second file, `irWeight`. The ordinal i signifies that the i^{th} posting list in the `irWeight` database belongs to term k_i

2.6.1 `irTerms` Database

The purpose of the file is to determine whether or not a distinct term has been indexed (e.g., stop-words do not occur in this file) within the information retrieval index. The `irTerms` database contains all the terms within the inverted index and the terms appear in the same sorted order as the `TH`. The second purpose is to allow the determination of the ordinal value of the term. For example, given a list of terms "ab", "ac", ..., the term "ac" maintains the position of 1. In other words "ac" has a ordinal value of 1 (i.e., second term when starting to count from from 0). This term count is used to determine the position of the term's posting list in the `irWeight` file (Section 2.6.2). The third purpose is to store the value of Equation 2 (Inverse Document Frequency, *idf*) to avoid having to calculate the *idf* on the PDA at search time. The query weight calculation (Equation 4) requires the *idf*.

The first portion of the `irTerms` record exists to act as an index into the list of terms contained within the data segment of each `irTerms` record. The string value of the last term and the ordinal value of the last term along with a separator for each are stored at the beginning of each `irTerms` record. This allows the query algorithm (Section 2.7.2) to determine whether or not the query term may be in the current record since the terms in the `irTerms` database are in sorted order [11]. Figure 8 displays the file format of the `irTerms` database when the option of stemming in not used (Section 2.7.1).

Following the index portion of the each record, the internal portion consists of the term string and the float value of Equation 2 (*idf*) from each the sequentially read, sorted alphabetically `TH` items. Multiple terms are appended to a `irTerms` record until the PalmOS 64KB record size limit is reached then a new record is created.

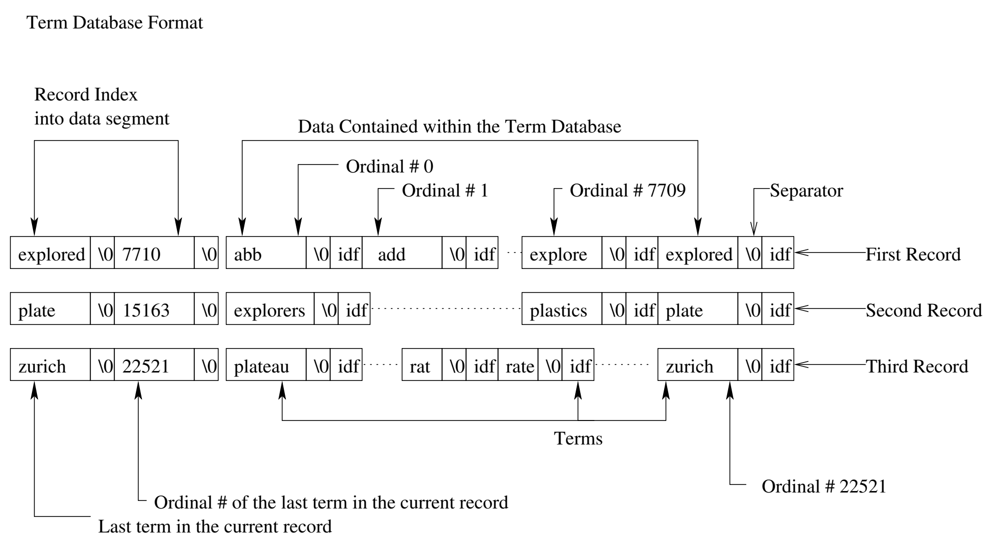


Figure 8: `irTerms`Example PalmOS Database

[11]by knowing that the previous records last term is less than the query term and the query term is less than the last term of the current record then the current record is the record that is the candidate to contain the query term. The correct record to search is determined by comparing only the first few index bytes (index of size: length of the last term in the record + 5 bytes) of the record. Once the candidate record is determined, then the algorithm sequentially searches the internal portion of the record.

2.6.2 `irWeight` Database

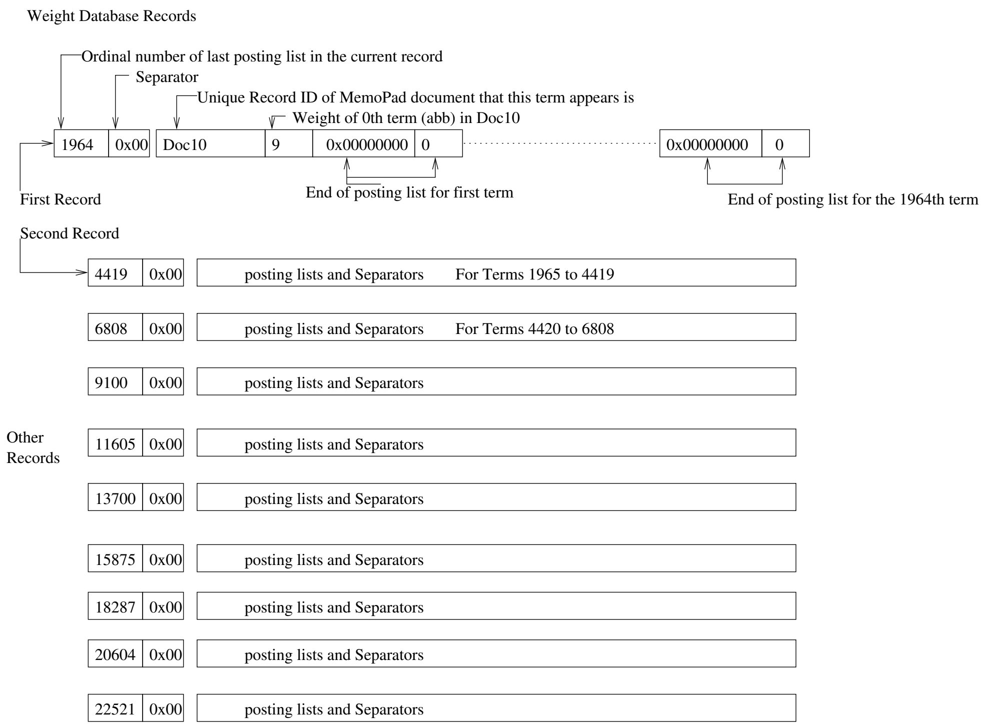


Figure 9: `irWeight`PalmOS Database Format

The goal of the file is to store a posting list for each term that appears in the `irTerms` database. The first term in the `irTerms` database corresponds to the first posting list that occurs in the first record of the `irWeight` database. The first 4 Bytes of each record specifies the ordinal (Term Count) from the start of the `irWeight` database to the last posting list in that record. This is used to prevent sequentially searching each record by acting as an index to determine if the record contains the ordinal that is being searched for. This index acts in a equivalent manner to the index (first part) of the `irTerms` record structure described previously in Section 2.6.1. The next byte acts as a separator and is followed by the internal data segment of the record (i.e., posting lists). The posting list for term k_i consists of a number of items where each item consists of the document id d_j (4 bytes) and the weight $w_{i,j}$ (1 byte) of term k_i in document d_j where j is the set of all document ids that contain term k_i. The weight is converted from a 4 byte floating point value to a 1 byte value by multiplying the float point value by 100 (Equation 6), rounding the value into an integer and storing the value as a byte. Storing document weight as a byte discards some of the significant figures of the floating point representation but hopefully the slight loss of accuracy does not outweigh the trade-off of the reduced storage space required to store the byte representation. In terms of accuracy, PalmIRA produces similar single collection retrieval accuracy to the results published in a paper by Viles and French [45]. They use the same retrieval model, a possibly different stemming algorithm and a possibly different rule for defining terms. Figure 9 displays the file format of the `irWeight` database that is stored on the PDA.

Each row in the sparse matrix contains the data required to create that term's posting list. To create the posting list for term k_i, each $\texttt{Node}_{i,j}$ in the linked list of nodes pointed to by the `TH` item representing term k_i is sequentially traversed. The document id ($\texttt{Node}^{Did_{d_j}}$) and the weight ($\texttt{Node}^{Value}$) stored in each $\texttt{Node}_{i,j}$

traversed is added as an item to the term posting list[12].

The items added to term k_i posting list are sorted in document id order. The motivation for the sort becomes evident during the degree of similarity calculation (Section 2.7.4)[13].

Because more than one term posting list is stored in each `irWeight` record, each posting list is separated by a separator. If term k_{i+1} posting list is small enough to append to the k_i and prior term posting lists already stored in the record, then the posting list is append along with the separator. Else, the current record is written to the PDA and a new record is created. In Figure 9, the first record contains the posting lists of 1965 terms where as other records contain a variable number of term posting lists. The first term (ordinal 0) has a posting list that contains only one item (weight 9, document id Doc10) (i.e., that term was found in only one document).

2.7 PDA Information Retrieval Engine

This section describes the portion of the PalmIRA information retrieval system that executes within the PalmOS based PDA environment and utilizes the inverted index created by the PC resident conduit portion described in Section 2.5. The application has a simple graphical user interface due to the 160x160 screen resolution of the PalmOS based PDA (Figure 10).

The GUI allows the user to enter a natural language query. Once the query has been entered, the first step of the query execution is to preprocess the query (Section 2.7.1). For each query term identified by the preprocessing step, it attempts to locate the query term within the `irTerms` PalmOS database and, if located, store the ordinal value of the term (section 2.7.2). Using the ordinal of each query term, it locates the posting list within the `irWeight` database. Also, it stores the record ID and offset from the start of the record to the posting list of each term (Section 2.7.3). Next, it calculates the degree of similarity between the query and the documents. The degree of similarity is used to rank the query results (Section 2.7.4). The last step of the PalmIRA information retrieval application is to display the ranked by degree of similarity results by writing to the screen the numeric value of the degree of similarity and a snippet of the first twenty characters of the document (Section 2.7.6). A user is able to browse the list of documents that are considered similar to the query. Another feature allows the user to select the query result and a new window opens to display the contents of the selected item.

2.7.1 Query Preprocessing

After the query is entered, the application parses the query string into a list of terms as in Section 2.5.2[14].

In order to increase efficiency, some information is stored for each query term. The information stored is called a `terminfo` structure and includes:

- $\texttt{k}_i$ - a string representing the query term
- $\texttt{freq}_{i,q}$ - frequency of term k_i within the query string (Equation 4)
- $\texttt{Ord}_{k_i}$ - ordinal value of term k_i in the `irTerms` database

[12] Document id and unique record id are equivalent values in the implementation.

[13] The items within each term posting list are sorted by unique document ID assigned by the PalmOS instead of record index (which is the easiest order to read the records from the PDA by the conduit). Note: PalmOS stores records in a list. This list can be sorted which physically changes the position of records in the list. Records in PalmOS can be accessed by a unique record ID or by an index into the sorted list. The index is the record's position in the list. Deleting a record changes the position and hence the index for records following the deleted record in the list. Therefore, deleting record with index 2 causes the record with index 3 to have index 2. It is unwise to store the index value because reading the record by index may yield a different record for that index (e.g., index 2 represents a different record before and after deletion).

[14] All characters that are not in the set {a-z, A-Z, 0-9, %, $, -, NULL} are converted to a blank character. Then the blank character is used to delimit individual terms. The motivation is to delimit queries for example "Canada\cr" (Canada ended with a carriage return) such that the query is interpreted properly.

Figure 10: PalmOS Screen Size

- `RWid`$_i$ - `irWeight` record id that contains term k_i's posting list
- `Off`$_i$ - offset of the posting list for term k_i from the beginning of the `RWid`$_i$ `irWeight` record
- `Did`$_{d_j}$ - document id value of the posting list item referenced by the `RWid`$_i$ of the record and the `Off`$_i$ offset from the beginning of that record
- `w`$_{i,q}$ - query term weight calculated by Equation 4

The array of `terminfo` structures as described above is sorted by ascending alphabetical order with respect to the query term represented.

2.7.2 Find Query Term Ordinal Value

After preprocessing the query, each query term is compared against the list of terms in the inverted index to determine if it has been indexed. All terms of the inverted index are stored in the `irTerms` database file. A query term may not be a member of the inverted index if that term does not exist in any of the documents of the indexed collection or if that term exists in the list of stop-words.

The first step in finding the query term is to determine a candidate `irTerms` record for containing the query term from the possibly many `irTerms` records within the `irTerms` database. A binary search using the index values at the start of each `irTerms` record finds the smallest index value that is greater than the query term that is being searched for. The index at the start of the `irTerms` records contains the string of the last term of the `irTerms` record. All terms within the data segment of the current record are less than or equal to the index at the start of the current record and greater then the index value (last term) of the previous record. The result of finding the smallest index that is greater than the query term is that this record is the only record that may contain the query term if the query term is part of the inverted index.

For each query term that exists in the inverted index, the $\texttt{Ord}_{k_i}$ of that query term within the `irTerms` file is stored as part of the `terminfo` structure (section 2.7.1). The $\texttt{Ord}_{k_i}$ is the number of terms (starting from the first term in the database) that precedes the term in the inverted index[15]. The purpose of $\texttt{Ord}_{k_i}$ is to find the location of the query term's posting list inside the `irWeight` database. Following each term in the `irTerms` database is a 4 bytes float value representing the *idf* value (Equation 2) for that term. The extracted *idf* and stored $\texttt{freq}_{i,q}$ are used to calculate the query weight (Equation 4) for query term k_i. The query term weight $w_{i,q}$ value is stored within the `terminfo` structure for the query term k_i.

If the query contains more than one term then the process can be optimized. For the second and subsequent query terms, the process of determining $\texttt{Ord}_{k_i}$ from the `irTerms` database can start the search where the search (either successful or unsuccessful) for the previous term left off. The effect is to narrow the search space for subsequent query terms.

Given that the query terms are in sorted order and the terms within the `irTerms` database are also in sorted order, the current record (from the search for the previous term) can be passed into the binary search as a lower bound for the search. If the index portion of a `irTerms` record is less than the current query term then that record is known not to be relevant for any query term with a higher alphabetical order than the current query term. This narrows the search space for each subsequent query term. This is in contrast to the search for each query term always executing the binary search over all `irTerms` records.

Another opportunity exists to help decrease the search space and thus the search time. At the time the query term k_i is found within the `irTerms` record, the ID of the record and a pointer to the term within that record are known. If the query term k_{i+1} is less than the index value of record R_r but greater than the query term k_i then the query term k_{i+1} must be between the position of query term k_i in the record and the end of the record. In this case the binary search is not required to find the candidate record because the current record is the candidate record. The search for the query term k_{i+1} begins at the pointer to the position within the `irTerms` record that contains the string equivalent to the query term k_i. Because the current record is being used, time is not wasted on a binary search that accesses records[16] [17].

2.7.3 Query Term Posting List Locator

After finding the $\texttt{Ord}_{k_i}$ for each query term, $\texttt{Ord}_{k_i}$ is used to locate the posting list of each term within the `irWeight` database. The `irWeight` may contain one to many records depending on the number of terms and the size of each term's posting list. As displayed in Figure 9, and described in Section 2.6.2 each `irWeight` record begins with an index describing the $\texttt{Ord}_{k_i}$ of the term whose posting list is the last posting list in that record.

The goal of this step is for each query term, locate the query term's posting list in the `irWeight` database and store the $\texttt{RWid}_i$ and $\texttt{Off}_i$ in the `terminfo` structure for that query term. The first step is to locate the record that contains, in the data portion of the record, the query term's posting list. A binary search to find the smallest index value (index value is the first value of each record, Section 2.6.2) that is larger than or equal to $\texttt{Ord}_i$ for the current query term locates the `irWeight` record. Because the index value is the ordinal of the term whose posting list is the last posting list in the record, and the query terms and posting list are in sorted by term

[15] $\texttt{Ord}_{k_i}$ is determined using the index value of the previous record, R_{r-1} (if r is not the first record) and then counting the number of terms from the start of the internal data portion (not index portion) of record R_i to the position of the query term to locate. The index portion contains the string representation of the last term of each `irTerms` record along with the last term's ordinal value.

[16] Because the current record is being used, the result is that the memory handle used to access the queried record does not have to be queried, locked, read, and unlocked. The PalmOS memory managers allows for the use of memory handles such that when a memory handle is unlocked, the PalmOS memory manager can automatically defragment the memory to increase storage [34], [13].

[17] The `irTerms` records separate the terms and the *idf* float values by a NULL character. Using the NULL character for a separator allows for an easy string comparison implementation.

order, the posting list for query term k_i must be in the located record. The record ID of the located record $\mathtt{RWid}_i$ is stored in the `terminfo` structure for that term. The next step is to find the offset within the located record of the query term k_i posting list $\mathtt{Off}_i$ [18].

For queries with more than one term, the two optimizations (described in Section 2.7.2) that help to decrease the search space are also used.

2.7.4 Degree of Similarity Calculation

Once the start of each query term's posting list is located, the next step is to calculate the degree of similarity of each document to the query. Documents that do not contain at least one of the non-stop-word query terms are considered to have a degree of similarity of 0. These irrelevant documents are indicated by the fact that none of the query terms contain that document in their posting list.

The challenge is to efficiently determine for each query term if document d_j is in query term k_i posting list. The idea of the algorithm is to sequentially traverse the documents starting with the smallest document id and ending with the largest document id in the set of documents defined by the union of the posting lists of the query terms.

Remember from Section 2.6.2 that all items that compose one term posting list are sorted by the unique record id (used interchangeably with document id) of the record that contains the corresponding document. Recapitulating, at the beginning of the degree of similarity calculation, the `terminfo` structure for each query term points (via $\mathtt{RWid}_i$ and $\mathtt{Off}_i$) to each query term's posting list item with the smallest document id in that posting list. Each `terminfo` structure item also contains the document id ($\mathtt{Did}_{d_j}$) value of the posting list item pointed to by $\mathtt{RWid}_i$ and $\mathtt{Off}_i$.

First the algorithm finds the smallest document id in the `terminfo` structure of all the query terms. Once the smallest document id is identified, for each query term that contains that document id:

- contribute the term's stored document weight ($\theta_{i,j}$) within the `irWeight`file posting list item to the degree of similarity equation (Equation 7).
- increment $\mathtt{Off}_i$ to point to the next posting list item
- update the current $\mathtt{Did}_{d_j}$ stored in the `terminfo` structure item for that term to the value pointed to by the $\mathtt{Off}_i$

After all query terms are compared against the current smallest document id and thus the running totals of $\sum_{i=0} w_{i,j}$ and $\sum_{i=0} w_{i,j}^2$ from Equation 7 are completely summed, then the degree of similarity calculation is completed as in Equation 7. The similarity and the document id are stored in an array of sorted by degree of similarity results.

The algorithm is setup to query and read the `irWeight` database only if the $w_{i,j}$ value is required. The number of queries and database reads are reduced because the document id of the posting list item pointed to by the offset is stored in the `terminfo` structure for each term. Caching the document id in `terminfo` increases efficiency by not have to lookup the document id from the index to determine if the smallest document id is pointed to by $\mathtt{RWid}_i$ and $\mathtt{Off}_i$.

2.7.5 Example

This next part introduces an example query using the index contained within the `irTerms` and `irWeight` PalmOS databases stored on the PDA and illustrated in Figure 8 and Figure 9 to help explain how the PalmOS

[18] To find the position of the posting list for query term k_i within the data portion of the `irWeight` record ($\mathtt{RWid}_i$), the algorithm starts by finding the index value (`Ord`) of the previous record $\mathtt{RWid}_{i-1}$. Then it iterates through the posting list items counting the posting list separators until this iterative value (number of separators + index value of the previous records) equals $\mathtt{Ord}_{k_i}$ of term k_i.

based information retrieval application satisfies a query.

For a query "rat rate explorers" given the `irTerms` database file displayed in Figure 8, the following would occur. First the query terms are sorted yielding "explorers rat rate" in the preprocessing step (Section 2.5.2). Next, the $\texttt{Ord}_{k_i}$ of each of the terms is determined (Section 2.6.1). Using the query term "explorers" and the index segment of the `irTerms` database records, a binary search is used to find the candidate record with the smallest index value that is greater than the query term "explorers". In this case, the second record with index "plate" is the candidate record. So at this point the query term "explorers" is either in the data segment of the second record or "explorers" is not in the inverted index. Next the internal data segment portion of the second record is searched. In this case "explores" is the found and the *idf* value following the term is used to calculate the query weight ($\texttt{w}_{i,q}$) (Equation 4) for query term k_i is stored in the `terminfo` structure item. The ordinal number (7711) for "explorers" is determined by the addition of the ordinal value of the last term of the previous record for the index portion of the first record (7710) plus the position of the term with in the data segment of the second record (1). For the second query term "rat" we know that it is not within the second record because the index of the last term within the second record is "plate" therefore a binary search is executed. Passed into the binary search as the lower bound is record 2 since record 2 is the record where the previous term (query term "explorers") was located so the binary search only searches records 2 and 3. The third record becomes the candidate record after the binary search and the term "rat" is found by the internal data segment search. After an internal data segment search, the `terminfo` structure is updated. For the third query term "rate", it is known that the query term may only be within the third record because the index of the last term within the third record is "zurich" which is greater than the query term "rate", therefore a binary search search is not required. Using the pointer pointing to the second query term, the third query term internal data segment search begins at that point. The third query term is easily found since it directly follows the second query term in the `irTerms` record. The `terminfo` structure is updated.

This next part of the example illustrates Section 2.7.3, i.e., how the PalmOS based portion of the system locates the posting lists of each of the query terms. Figure 9 displays an example of the `irWeight` database. For the first query term "explorers", the stored ordinal number of 7711 is searched for using a binary search. The binary search returns record number 4 because it is the record with the smallest index value (9100) that is greater than the ordinal number of the query term "explorers" (7711). The next step is to search the internal data segment of the `irWeight` record to find the offset of the posting list. Since each posting list is delimited by separators of 5 bytes of all zeros, the separators are counted. The posting list is located when the index value of the previous record (6808) plus the number of posting lists traversed is equivalent to the ordinal value of the query term (7711). This is the same idea as described in the `irTerms` traversal (Section 2.7.2). The binary search optimizations, described in Section 2.7.2, are used to decrease the search space for query terms "rat", "rate".

This next part of the example illustrates how the PalmOS based portion of the system computes the degree of similarity calculation for each of the query terms (Section 2.7.4). Figure 11 displays an example of the algorithm to calculate the degree of similarity. The example consists of 2 documents: Doc1 with terms "rat", "rate" and Doc2 with terms "explorers" and "rat".

At the start of the similarity calculation, pointers ($\texttt{RWid}_i$ and $\texttt{Off}_i$) point to the first item of each of the query term posting lists. Using the $\texttt{Did}_{d_j}$ stored in the `terminfo` structure (Doc2 "explorers", Doc1 "rat", Doc1 "rate"), the smallest of these values (Doc1) is chosen as the first document to have the degree of similarity calculated for it. Next considering only the query terms that currently point to document id Doc1 ("rate", "rat"), the degree of similarity of document id Doc1 is calculated. The $\texttt{RWid}_i$ and $\texttt{Off}_i$ for "explores" does not point to document id Doc1 and therefore considering the stored order of the posting list, this posting list does not contain document id Doc1 and "explores" does not contribute to the degree of similarity measure for document id Doc1. The first query term considered in the similarity calculation is "rat". The weight $w_{i,j}$ for term "rat" in document Doc1 is read by querying for the `irWeight` record id (obtained from the `terminfo` structure item), locking the record, traversing to the offset (obtained from the `terminfo` structure item) and reading the weight value from

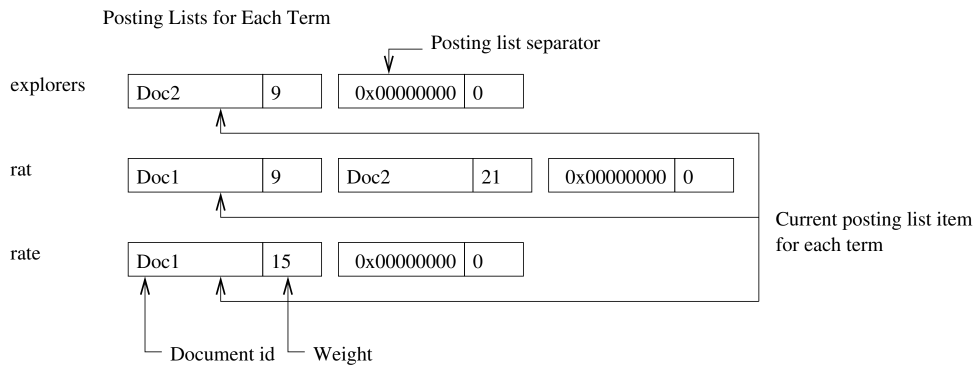


Figure 11: Example Degree of Similarity Calculation - Begin

the `irWeight` database record. This weight value $w_{i,j}$ of "9" is entered into the summation $\sum_{i=1}^{t} w_{i,j} \times w_{i,q}$ along with the query weight stored in the `terminfo` structure item $w_{i,q}$. In addition, the summation of the $\sum_{i=1}^{t} w_{i,q}^2$ is maintained locally. After the weight is read, $\mathtt{Off}_i$ for the term "rat" is incremented and the new $\mathtt{Did}_{d_j}$ referencing the new $\mathtt{Off}_i$ is stored in the `terminfo` structure. In this case, Doc2 for "rat" and the separator for "rate" are stored, e.g., Figure 12. After all terms are considered, the two summations, one for each query term "rat", "rate" are passed into the final calculation for Equation 7. The document id Doc1 and the degree of similarity are stored in a sorted array.

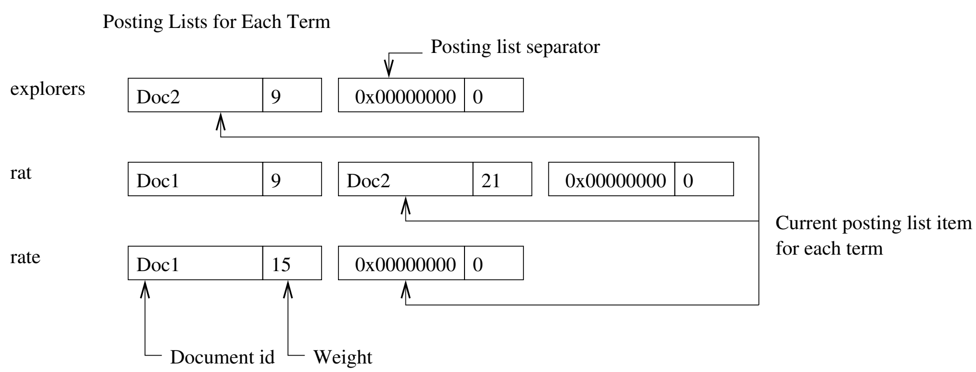


Figure 12: Example Degree of Similarity Calculation - After First Document

The same process is used for document id Doc2 for terms "explorers" and "rat". When $\mathtt{Off}_i$ within the `irWeight` record references a zero document id and a zero weight, then all posting list items for that query term have been processed. When this occurs for all query terms then the results are displayed, Figure 13.

2.7.6 Graphical User Interface (GUI)

A list-box component displays the results on the PalmOS device. The results consist of the degree of similarity, document id and a snippet of the text of the document[19]. The document id of each result array item provides the

[19]As the degree of similarity is calculated with respect to the query for each document in the collection, the document record id and similarity measure for that document are stored in a descending by similarity ordered array. This array stores the top 200 entries pushing out of the array the item with lowest degree of similarity each time a new document with a degree of similarity greater than the 200^{th} member of the array is added. A hard limit of 200 items is imposed. The storage of only the 200 most relevant (i.e., highest similarity

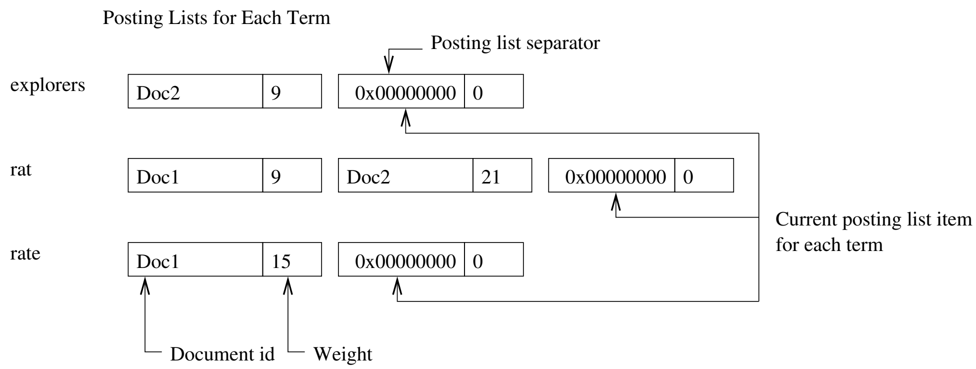


Figure 13: Example Degree of Similarity Calculation - After Last Document

means to query the original document from the collection and to extract a snippet of the document to display as part of the query result (Figure 14).

(a) (b)

Figure 14: PalmIRA: (a) Query; (b) Query Result

2.7.7 Graphical User Interface (GUI) Enhancements

The list-box GUI component that displays the results was changed from loading all query result items to loading only 10 query result items at one time into the list-box component to display. Scrolling allows access to any

measure) documents acts as a hard limit placed on the number of items the array can handle and thus an upper limit on the amount of memory dynamically allocated for the array. A query that returns the 200 most relevant documents usually provides more results than are browsed or utilized by the user.

remaining items (in groups of 10) that do not fit within the 10 items currently display within the list-box.

The consequences are faster query execution times and lower dynamic memory cost since only enough memory for 10 items must be allocated for display purposes. The decrease in time is also a result of the fact that only the snippets of 10 of the results must be queried from the collection database at one time instead of each of the possibly 200 query result documents. Since records must still be queried for each new screen, the query time savings is offset by a negligible increase in time to update the display as the user browses the query results.

2.8 Efficiency Experiments

The test database is the TIME database from Cornell University's [7] website. The data set includes world news articles from 1963. The collection contains 425 documents for a total size of 1513KB with 83 queries. The data set also contains a list of queries and their relevance document list.

This experiment converted the database into a format such that it could be loaded into the Palm platform. This involved modifying the data set so that each document would fit in a PalmOS record such that documents greater than the PalmOS record length limit of 64KB were split into multiple records.

The evaluations were completed using a Handspring Visor Deluxe PalmOS v3.1 PDA with a 600 MHZ Intel Pentium III Win98SE laptop with 64 MB of RAM and USB to synchronize with the PDA. Version 4.0 of the conduit development kit was used along with Metrowerks CodeWarrior for PalmOS 4.0.

Without stemming, the number of distinct terms is 22521. The total time to synchronize is 71 sec and can be broken down into:

- to download and parse textual data and create sparse matrix: 45 sec
- to calculate weight ($w_{i,j} = idf_i * tf_{i,j}$): $<$ 1 sec
- to build and transfer to PDA 5 packed term records: 8 sec
- to build and transfer to PDA 10 packed posting list records: 17 sec

A naive approach which did not use a sparse matrix required approximately 635 sec possibly due to large amounts of swapping required to access the non-sparse matrix.

The total size of the inverted index is 928,293 bytes and can be broken into:

- 285,693 byes for the `irTerms` index
- 642,600 bytes for the `irWeight` index

The non-stemmed inverted index is not compressed and the opportunity for reducing the size with compression is discussed in future work (Section 4.2). Stemming and increasing the size of the stop-word list such that some of the more popular terms are not indexed will reduce the index size but analysis of viability of these are left to future work. The idf_i for each term in the `irTerms` database comprises a total of 90,084 of the 285,693 bytes.

TIME Database query #55 is "suggestion made by president Kennedy for a NATO nuclear missile fleet manned by international crews". The above query #55 returns 379 documents where each of the 379 documents contains at least one of the query's non-stop-word terms.

The total time to retrieve the documents is 7 sec for query #55. The naive approach required over double the time to complete the query. This is the time breakdown for query #55:

- find the term location in `irTerms` database: 1 sec (2 sec before binary search enhancements Section 2.7.2)
- find the posting lists location in the `irWeight` file: 2 sec (3 sec before binary search enhancements Section 2.7.3)

- calculate the similarity and order based on similarity: 4 sec (9 sec before caching of document id Section 2.7.4)
- display results (first 20 char of Doc): <1 sec (2 sec before GUI enhancements Section 2.7.7)

This (simplistic) experiment served to show that the approach we chose to implement the information retrieval framework was a vast improvement over the straight forward approach, specifically in terms of index building time and storage overhead. Unfortunately, we were not able to perform an objective comparison with similar tools (e.g., Intelligentfind [33], Palm Pirate [41]) since their code is not open and no further details about their implementation is available. Nevertheless, we believe we have build a solid framework upon which we can improve by tackling the information fusion problem (next chapter).

3 Collection Fusion

3.1 Introduction

Large amounts of research have addressed the issue of attempting to retrieve relevant documents from a single text collection efficiently and effectively [1]. The previous chapter describes an implementation for a PDA. An extension to this research area is how to efficiently and effectively retrieve information from multiple, autonomous collections.

The basic multiple collection retrieval problem/task involves the following progression. Given a query, distribute the query to some of the multiple, autonomous collections and their associated search engine. Independent of the other collections, each of the multiple collections constructs a list of results for that query. The multiple results lists are merged using some technique into one combined, global result list.

The collection selection and collection fusion problems originate from this situation where there exists multiple, autonomous collections. On the one hand, the collection selection problem deals with issues involving how to choose which collections will best satisfy the information retrieval query. On the other hand, the collection fusion problem deals with issues involving how to merge or fuse the result list retrieved from each collection into one meaningful result list. The latter approach is explored in this chapter.

Collection selection evaluates the degree of relevance of each collection to the query. Collections that have a high degree of relevance to the query should contain documents that are highly relevant to the query. The value of the relevance of the collection can be utilized to help in the fusion of the result lists or to help to reduce the number of collections that need to be queried. This is an attempt to prune the search space and is based on the idea that searching a very large number of collections is not feasible in terms of resources, user time, and computation time.

Collection fusion (otherwise known as results merging) merges the results from multiple independent collections and their associated search engine. The goal is to create one ranked result list from multiple result lists such that the merged result maximizes the precision/recall[20] of documents relevant to the query. That is, relevant documents should be ranked highly and the non-relevant documents should be ranked lowly regardless of the document's originating collection. Each collection is autonomous in that each collection is associated with a set of documents and an information retrieval ranking engine. Given a query and a result list from each of the multiple collections, the "ideal" result is to merge the result lists into one global result list such that most highly relevant documents rank highly in the global (merged) result list. It is desirable to favor "good" collections while still allowing a "good" document from a "bad" collection the opportunity to achieve a high rank within a merged global result list [5].

[20] Precision is the percentage of retrieved documents that are relevant and recall is the percentage of all relevant documents that have been retrieved.

Data fusion differs from collection fusion in that data fusion utilizes multiple search engines with differing retrieval techniques on the same set of data whereas collection fusion may have multiple different sets of data [38] [31]. One approach is Democratic Data Fusion [44], where the number of search engines that find the same document helps to determine the merged rank of that document.

The situation where a document in one collection contains a counterpart that exists in another collection is not considered in the scope of this research. Duplicate detection and handling at result list merge time is left to the area of data fusion.

In the next section, similarities and differences between PalmIRA collection fusion and Meta-Search are discussed (Section 3.1.1). After that is a survey of related work (Section 3.2) including collection fusion approaches and a discussion about why certain types of approaches could be inappropriate for use in a PDA (Section 3.2.1) and a survey of some proposed effectiveness measures (Section 3.2.2). Following that is a description of the collection fusion techniques experimented with including the proposed technique (Section 3.3) along with a description of effectiveness measures including the proposed techniques (Section 3.4). Lastly, the experimental setup (Section 3.5), the experimental results using Cornell data (Section 3.6) and the TREC data (Section 3.7) are described.

3.1.1 Similarities and Differences with Meta-Search

Given a query, a Meta-Search engine distributes the query to multiple World Wide Web (WWW) search engines in an attempt to increase the search space since each WWW search engine does not have a complete view of the WWW [31]. The meta-search engine does not itself contain an index or view of the web but may contain statistics about the various search engines for search engine selection purposes. The main parts include:

- selection of a set of search engines (also known as databases or collections) to distribute the query to,
- collection or gathering of certain specific retrieved documents from the search engines (e.g., first 20 retrieved documents),
- the merging or fusion of results,
- presentation of results.

The PalmIRA retrieval system differs from meta-search in a number of ways.

- Retrieval model (e.g., vector model) between collections:
  - PalmIRA - consistent
  - Meta-Search - cannot guarantee consistency
- Data:
  - PalmIRA - internally stored on a PDA
  - Meta-Search - usually WWW data at multiple autonomous locations
- Platform characteristics:
  - PalmIRA - constrained (e.g., low memory, limited CPU)
  - Meta-Search - high scalability (e.g., distributed systems)
- Communication between collection and result merging algorithms:

– PalmIRA - search engines and associated database accessed directly by the result merging algorithm without communication since both exist on the PDA
– Meta-Search - collections accessed over the internet

- Search engine statistics (e.g., term frequency) availability to result merging algorithm:
  – PalmIRA - easily accessible since both exist on same PDA
  – Meta-Search - not easily accessible since they may exist on different computers and requires some level of cooperation (e.g., STARTS [20])

3.1.2 Example Introduction

In the running example used henceforth within this chapter, two independent collections of documents exist: X and Y. Each collection is composed of 10 documents: X1. . . X10 for collection X and Y1. . . Y10 for collection Y.

Given a query, the collections use a vector model based search engine to retrieve documents. The example uses the query containing terms A, B, C, D. For the query, the following documents have been judged to be relevant Y3, Y4, Y9. As can be seen from Fig. 15, Document X9 contains the same query terms as Y9 but is not considered relevant. Document X9 is used as an example of a document that contains query terms but the query terms are used in a context that does not relate to the meaning of the query (i.e., randomly mentioned terms). For example, given a query such as “mouse for an Apple computer”, a document containing “I saw a mouse on my computer while eating an apple” would be retrieved by the vector model but should not be considered relevant to the query. That is, the query terms “mouse”, “apple”, “computer” are randomly mentioned in the document such that the context of their use in the document differs from the context of their intended use in the query. Similar reasoning should be applied to the other irrelevant documents that contain query terms.

In order to create a reference or baseline for measuring the effectiveness of collection fusion strategies, a reference collection was created by concatenating the two independent collections together. A reference result list is created by retrieving documents from the reference collection (Fig. 15).

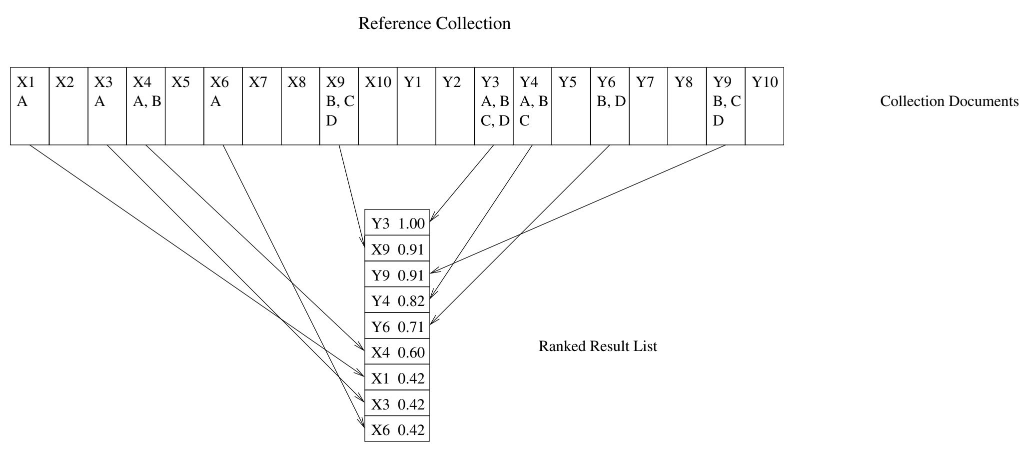


Figure 15: Reference collection query results example

The running example uses the vector model for document retrieval. For each independent collection, the collections are independently queried. The retrieval model uses no knowledge or parameters from other collections. This yields one result list for each collection (Fig. 16).

The vector model uses collection dependent features to compute the degree of similarity between the query and each document. An example of this are documents X9 and Y9 which both contain the same terms B, C, D. Notice in Fig 15, the reference collection, that documents X9 and Y9 have equivalent degrees of similarity.

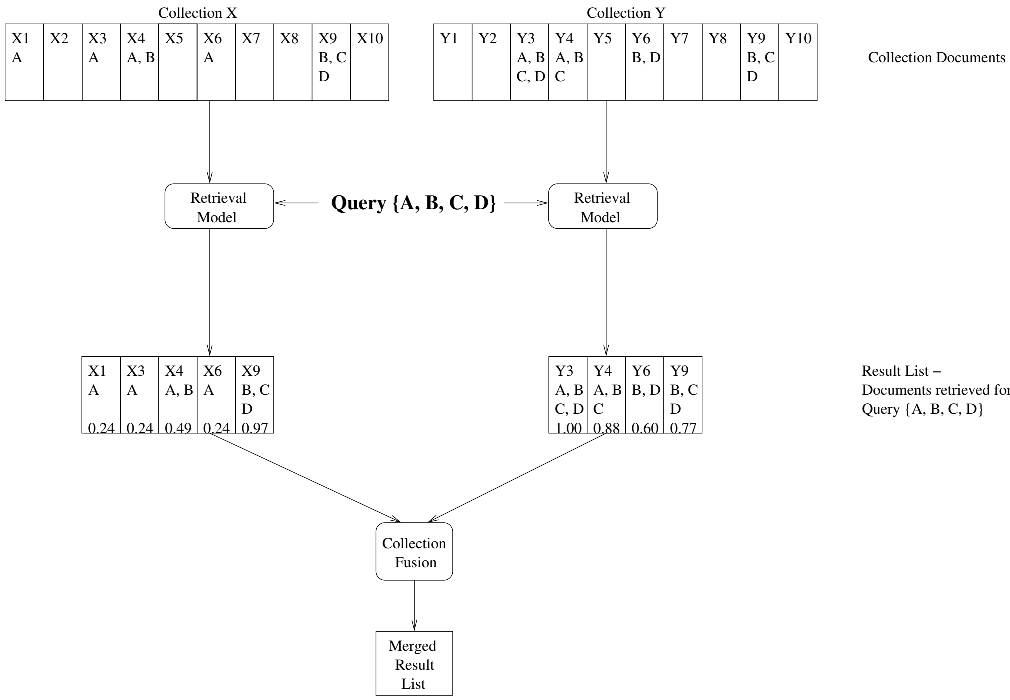


Figure 16: Multiple Collection Results Merging (Collection Fusion)

But when document X9 and Y9 reside in independent collections, X9 and Y9 do not have equivalent degrees of similarity (Fig. 16). The difference is due to the collection dependent feature known as the inverse document frequency (*idf*). The *idf* may differ for a given term between each collection and helps to produce the differing degree of similarity.

After each of the independent collections are queried, the next step is to produce one merged result list out of the two result lists. Collection fusion produces the merged result list. Section 3.3 describes examples of the round robin (RR), round robin random (RRR), original weights (raw scores), and Co-occurrence collection fusion approaches.

3.1.3 Challenges in Collection Fusion

If the query is submitted to search collections X and Y then 2 result lists are created. How can the two results lists be merged into one global result list maximizing effectiveness (i.e., maximizing precision and recall)? To answer these questions, a number of issues should be considered.

Each collection may be heterogeneous or homogeneous in reference to the topic(s) covered by the documents within the collection. A set of collections containing a mixture of the two types of collections is also possible. A set of heterogeneous collections occurs where each collection represents many topics. Each collection in the set may contain the same broad range of topics as the other collections. A set of heterogeneous collections (e.g., where each collection contains stories from a single different newspaper), may have relevant documents to a given query spread throughout the set of collections. A set of homogeneous collections occurs where each collection represents a more focused topic. Each collection in the set may contain a different focused topic when compared to the other collections (e.g., each collection contains papers from one specific conference such a SIGIR, SIGKDD, SIGMOD, etc.). In a set of homogeneous collections, the relevant documents may appear in only one collection. It is unlikely that all or most of the collections contain documents relevant to a given query.

Random mentions of query terms within a collection still contribute to the result when using vector model retrieval. These random mentions of query terms may or may not produce documents with large degrees of similarity depending on the term frequency (*tf*) and inverse document frequency (*idf*) of the query terms.

Each collection may utilize a different ranking model. Collections may tend to have terms with inverse document frequency *idf* or other collection dependent parameter values that vary widely from collection to collection. Even if the same ranking algorithm is used for each collection, the varying content of each collection may produce incomparable raw (i.e., unchanged from the retrieval model algorithm) document scores.

Collections may contain highly dynamic content and any collection fusion technique must be able to adapt.

How to order/rank the documents as they are merged from various collections? Round robin and raw scores fusion techniques have drawbacks which will be shown in Section 3.3. Another type of technique attempts to take the raw document score and modify the score by a value reflecting the collection the document is from. Section 3.2 explores the answer to this question by introducing various techniques.

3.2 Related Work

This section describes literature published related to the area known as collection fusion or in other words, result merging. This includes collection fusion methods (Section 3.2.1) and effectiveness measures that evaluate the collection fusion techniques (Section 3.2.2).

3.2.1 Collection Fusion

According to Voorhees et al. [47], there are two types of merging strategies: integrated and isolated.

Integrated merging strategies allow distributed collection search engines access to some amount of information from other collections or allow transfer of information between the search engine and merging engine. The goal of the information is to help maximize the merged result list effectiveness. This requires dissemination of some amount of information such as [45], involving a communications protocol such as STARTS [20].

Isolated merging strategies assume that the the merging strategy can not assume any more information from the collection than a result list [47]. Search engines for each collection have no communication with or knowledge of others. There is no communication between the multiple retrieval engines or use of global algorithm parameters by the search engines. The collection fusion algorithm must merge the result lists using only information returned by the collection or information that can be inferred or gathered based on that information.

Craswell et al. [10] and Meng et al. [31] describe four types of merging strategies that may be either isolated or integrated. These include:

- rank based - assumes no document scores are available,
- score based - where documents are assigned new scores based on the score assigned by the collection search engine. The new score may be based on a collection weighting.
- document content based - where the document's contents are parsed and analysed during result merging,
- knowledge discovery of search engine properties - e.g., stemming, retrieval model, etc.

In the paper by Steidinger [42], the author compared six collection fusion models. The author chose the six models based on the models requiring no more input (and sometimes less) then the following: ranked results list of documents with each document assigned a score, the length of each result list, the number of collections containing each query term (*CF*) and number of documents within each collection containing each query term (*DF*). Some of the models require only the result lists as inputs.

- **Round Robin (RR):** uses no prerequisite information and merges the result lists as the name suggests in an interleaving fashion [14].

- **Round Robin Random (RRR):** requires only the length of the result list. The merging randomly (biased toward longer result lists) chooses a collection result list to remove the head of the list and merge [48].

- **Round Robin Block (RRB):** requires only the length of the result lists. A block length for each result list is determined by dividing each result list by the smallest list length. Then in a round robin fashion, the block length number of result list items from each result list are merged into the global result list.

- **Raw Scores (RS):** requires similarity scores for each document. In order for document scores to be comparable between possibly different collection retrieval algorithms, the scores have to be normalized [11] [39] to be comparable. For example, one retrieval algorithm returns results between 0-1, another between 0-100. The model performs a merge-sort to produce the global result list. Powell et al. [36] use the minimum and maximum scores to normalize the raw score.

- **Normalized Inverse Document Frequency (NIDF):** requires document similarity scores and document frequencies (*DF*). For a given query term, the model determines the number of documents from each collection that contain the query term (DF). This collection fusion model uses the inverse of the document frequency to determine a normalization factor that attempts to approximate the effect of the collection dependent parameters used in the document retrieval model (e.g., vector model's inverse document frequency parameter *idf*). The technique modifies the original document scores with the normalization factor of the collection the document exists within.

- **Collection Weight (CW) or Weighted Scores [5]:** requires document similarity scores, document frequencies and collection frequencies. Based on a technique described in Callan et al. [5], the technique determines the conditional probability of term r_j in collection c_k and based on this, the weight (w_j^k) of term *j* of collection *k* is computed. The $\sum w_j^k$ over all query terms produces a collection weight that is multiplied against the original document score to re-weight each document.

Steidinger concluded, in order of decreasing effectiveness, the the six models ranked: RRR, RRB, RS, CW, RR, NIDF. The document retrieval involved the MG system [53] and an Oracle system.

In the paper by Callan et al. [5], the collection fusion algorithm first assigns a weight to each of the collections and then re-weights the scores of the documents depending on the collection weight. The idea is to give the new document score the value the document would have received if each of the separate collections were merged into one collection.

To rank the collections, the authors use the idea of a "collection retrieval inference network" or CORI net. The algorithm produces probabilities for the CORI net based on the INQUERY [4] document retrieval model but with small changes to some parameters to reflect its use as a collection weighting scheme instead of a document weighting scheme. The scheme contains two constants: default term frequency and default belief.

Callan et al. compared 4 collection fusion methods using precision and recall. These include interleaving (i.e., RR), raw scores, normalized scores, and weighted scores. The normalized scores method used statistics to normalize collection dependent parameters across all collections to create comparable document scores across all collections. Thus the retrieval engines are not autonomous. The weighed scores approach is the technique that Steidinger based the CW fusion technique [42] upon. The document retrieval was done using the INQUERY system. Callan et al. find that the weighted scores fusion technique produces the top results of the four techniques (slightly better than the raw score approach). This is in contrast to Steidinger [42] who found that the raw score approach slightly outperformed the weighted scores approach. This contradiction is possibly due to the default parameters not producing the same results since each author ([42] and [5]) uses differing portions of the TREC/TIPSTER data.

The Larkey et al. paper [28] investigates the effect of topically organized U.S. Patents data while using INQUERY's CORI algorithm [5]. The authors investigate normalizing the raw scores and a global *idf* [5] approach using topically organized data. The authors find that the global *idf* (as opposed to collection *idf*) better approaches the precision of the baseline, centralized collection.

The paper by Yager et al. [54] introduces a modification to the previously described round robin merging method by Voorhees et al. [48] to produce a deterministic result by removing the randomness. The two approaches are as follows. First, the collection to contribute its highest ranking un-merged document is the collection with the highest V_i value as calculated by $V_i = \alpha n_i - (1 - \alpha) g_i$ where n_i is the number of un-merged documents in collection *i* and g_i is the number of merged documents from collection *i*. Collection V_i score ties are resolved by a round robin collection fusion approach. The α parameter in techniques described in the paper is set to 0, 1, 0.5, or learned. The value of α describes what the importance of the number of un-merged documents is to the final ranking of a document. The second approach is a proportional approach. This approach calculates the V_i value as $V_i = \frac{n_i - 1}{N_i}$ where N_i is the number of documents originally in that collection. The collection to contribute its highest ranking un-merged document is the collection with the highest V_i value.

Yuwono et al. [56] measure the "goodness" of each collection and based on the "goodness", merge the result lists from multiple collections in their D-Wise search engine. This approach does not require document scores to be assigned by each collection. The "goodness" measure is based on the Cue-Validity Variance that indicates in which of the collections the query terms are concentrated. The Cue-Validity measures the inter-collection dissimilarity or by how much a query term distinguishes one collection from another using a document frequency statistic from each collection. The result merging algorithm uses the relative ranks of documents and assigns a score to each document based the ordinal of the rank and a distance measure between consecutively ranked documents in a collection result list. The distance between two consecutively ranked documents is inversely proportional to the "goodness" of the collection. Documents from "good" collections have a smaller distance between each consecutively ranked document than a "bad" collection and thus tend to be ranked higher in the merged ranking. The result merging assigns a score to each document in a deterministic manner and then combines result list documents by order of their new score.

In the paper by Calve et al. [6], the authors introduce a method using the ranks of the documents and the idea of logistical regression to merge the documents. Logistical regression is used to predict the probability of a document being relevant to a query based on an independent variable, the ordinal of the rank of a document. The documents are then sorted and merged based on the level of probability. The higher ranks (i.e., closer to 1) are given more importance than lower ranks based on a logarithmic scale. Higher ranking documents are considered to have a higher difference between them than the low ranking documents.

Voorhees's research group [48], [47], [49], [43], [50] introduces an idea to learn the distribution of retrieved documents from the results of past queries and uses this to determine the number of documents to select from each collection for un-seen queries. Training queries initialize the approaches. Unseen queries are matched against the training queries. Three approaches include: Modeling Relevant Document Distributions (MRDD), Query Clustering (QC) and Neural Networks.

The techniques presented by Voorhees et al. attempt to retrieve documents from multiple collections and then merge the result lists independent of each collection's: contents, retrieval model, document weighing scheme, and similarity measure.

In MRDD, the model predicts how many top ranked documents to select from each collection using the K nearest training queries. Training queries are represented as term frequency weighted vectors (queries may contain multiple instances of a term). The distribution of the judged relevant documents for each query in each collection is also stored. Given a query, the K-most similar queries (i.e., nearest neighbors) based on the cosine vector model similarity measure are found. Next, the average document distribution is found by calculating the average number of relevant documents returned from each collection for the K-most similar queries. Then, the number of documents to select from each collection is calculated based on the document distribution such that the number of relevant documents existing in the results selected from each collection is maximized. Finally, documents are merged based on the round robin random approach using a C-faced (C = # collections) die biased by the number of documents not yet merged.

In QC, training queries are represented as term frequency weighted vectors (queries may contain multiple instances of a term). The queries are clustered into topic areas for each collection via the Ward clustering method. The similarity measure is the number of documents retrieved in common between the two queries. This assumes that if two queries retrieve a high number of documents in common then the two queries are about the same topic. A centroid vector of the cluster represents the topic area of the cluster and is determined by averaging the query vectors of all queries in the cluster. Each cluster is also assigned a weight reflecting the average number of documents retrieved by the members of the cluster for each collection. For each collection, the weight of the cluster with the closest centroid vector to the given query is used to determine the number of documents to select from that collection. The documents are then merged using the round robin technique.

The MRDD and QC techniques approach the precision of the single collection baseline to within about 10% at low numbers of retrieved documents. The drawback to these 2 approaches is that the training data may not be sufficient to predict the number of documents that are relevant in each collection for queries about topics un-related to the training data.

In Towell's et al. paper [43], a neural network approach is compared to the QC and MRDD approaches above. A significant decrease in performance is found. The neural network uses the term frequency weighted vectors as input.

Baumgarten [2] introduces a model based on the extension of the probability ranking principle for non-multiple collection information retrieval. This extension includes the collection selection and collection fusion steps of multiple collection information retrieval. The approach is a non-heuristic framework. The idea is to probabilistically rank the documents from multiple collections. The documents are ranked in decreasing order of probability of being relevant to the query. The density of the probability distribution selects the collections. Separated from each collection search engine, a broker site ranks each collection and merges the result lists from the selected collections. The broker uses statistics from the query and each collection to rank the documents.

In [14], [15], [40], Fox et al. used document re-scoring schemes based on the "goodness" of the collection where the "goodness" is measured by an aggregation of the result list document scores from each collection and then merging the documents based on the new document scores. The aggregation represents the result list documents scores as a single value such that the value measures the "goodness" of the collection. The result list aggregations include:

- CombMAX - maximum document score in each result list
- CombMIN - minimum document score in each result list
- CombSUM - summation of document scores in each result list
- CombANZ - average of document scores in each result list
- CombMED - median document score in each result list

The CombSUM approach shows the best results.

In the paper by Rasolofo et al. [37], the authors introduce a merging strategy that is a product of the collection score and the document score. The collection score is based on the length of the result list from each collection.

Yu et al. [55] introduce a collection fusion approach that uses the collection assigned document score of the highest ranking un-merged document from each collection result list to determine the next document to merge. The collection to contribute the next document to merge is determined by estimating the similarity between the highest ranking document of each collection and the query. The similarity is based on the document frequency of each query term.

In the Profusion multiple collection retrieval engine [19], a query is submitted only to collections that have demonstrated the ability to produce good results from past queries of the same topic. This is accomplished using training data to create a taxonomy of topics, create a dictionary associating words with topic(s) and calculate a confidence factor for each search engine given a topic. Given a query, the query is broken into topic(s) via a dictionary and then the query is passed on to the best search engines based on the confidence factor or "goodness" with respect to the query topic(s). The result merging is based on a re-weighting scheme involving multiplying the document score with the confidence factor of the collection and then merging based on the new document scores.

The paper by Zhu et al. [57], involves re-scoring the retrieved HTML documents based on their current score and the quality of the document. The quality metrics experimented with individually or in combination include:

- currency - modification date
- availability - ratio of broken to total number of links
- information-to-noise ratio - ratio of the number of tokens to the size of the document
- authority - from Yahoo Internet Life reviews
- popularity - number of incoming links
- cohesiveness - internal document similarity

The conclusion of the paper stated that document scoring based on the document score and the popularity produced the highest precision.

Gravano and Garcia-Molina [22] introduce a method to compute new scores based on meta-data about the documents returned by the search engines. The merging algorithm does not use the document scores. The meta-data describes features or attributes of each document. Using the meta-data, the algorithm attempts to extract the best matches from the result lists instead of returning the entire contents from each result list to the user. The scoring algorithms are considered manageable if the original score and the new score based on the meta-data are reasonably close, then not all documents need to be presented to the user.

In Inquirus [29], the result merging is based on the content of the documents contained within the result lists of each collection. The broker in charge of the result merging downloads each document and analyses them based on the query terms. The downloading allows the context of the query terms to be discovered, analysed and displayed. This technique merges the multiple result lists by creating a new score for the documents. The new document score is based on: the number of query terms, the number of instances of each query term, and the minimum distance between the i^{th} and j^{th} query terms. The distance is measured in characters.

In the work by Craswell et al. [10] [9], the authors introduce the idea of using a sum of the feature distances to calculate a new document score. The features are terms. The calculation for each is based on: the distance from the beginning of the document to the term, the distance between current and previous terms and the document frequency of a term. Also, the authors experiment with the idea of using reference statistics or a sample of 10% of the documents in a collection instead of using statistics from the entire collection (e.g., document frequency of a term).

In the paper by Meng et al. [32], the authors develop a set of rules to detect properties of the underlying collection's search engine. The knowledge gained is then used to increase the effectiveness of collection selection, document selection and result merging steps. The technique [30] works by submitting strategically developed probe queries to each collection and analysing the returned documents. A knowledge-base is developed consisting of characteristics of various, documented in the literature, approaches involving: indexing methods (e.g., stemming), document term weighting functions, query term weighting functions, and similarity functions. The team developed strategies for determining what if any stemming is used, what if any stop-words are used, strategically designed queries to attempt to discover the functions utilized for query and document term weighting and degree of similarity calculation. With respect to result merging, any knowledge discovered may help to adjust local or compute new document scores to make document scores more comparable between collections.

The paper by Viles et al. [45] involves a distributed collection information retrieval system where each site knows some portion of all other sites' information (i.e., there does not exist a centralized meta-data repository as in [48]). Information is disseminated to other sites to improve retrieval effectiveness relative to the situation where no information is shared. The shared information is known as collection wide information (CWI). The purpose is to create a consistent, collection wide *idf*. The authors introduce two issues:

- How to circulate CWI? (e.g., STARTS [20])
- At what intensity should CWI be circulated? (main issue of this paper). E.g., A site knows about its own documents and 25% of the documents at other sites (i.e., "lazy dissemination")

With "lazy dissemination", the authors argue that the insertion of a document or group of documents may not change the CWI enough to influence overall effectiveness. An increased level of dissemination is required when the documents are allocated to sites based on content as opposed to random allocation. Almost no difference in precision vs. recall as dissemination levels change when using a random document allocation (i.e., heterogeneous collections) is displayed.

3.2.2 Effectiveness Measures

The most common effectiveness measures used to evaluate collection fusion are based on precision and recall. Precision is the percentage of the retrieved documents judged relevant to a query. Recall is the percentage of the judged relevant documents that have been retrieved.

The three main approaches as described in [1] are:

- average precision at the 11 standard levels of recall (i.e., 0%, 10%, ... 100% levels of recall).
- average precision at document cut-offs (i.e., after n non-relevant or relevant documents have been seen).
- average recall at document cut-offs.

In the paper by Yuwono et al. [56], the effectiveness is evaluated by comparing the the $tf \times idf$ scores of the documents in the merged result list with the scores of the documents in the single collection run.

Callan et al. [5] assumed that given a query, the collection with the most documents judged relevant is "ideally" ranked the highest. The measure used the mean square error between the collection rankings of their collection weighting algorithm and a ranking based on the number of judged relevant documents from each collection of the TREC data-set (i.e., "ideal" ranking).

Gravano et al. [23] compare their estimated database ranks with the "ideal ranks" (baseline). The 4 measures are:

- sum similarity measures of a given collection result set members above a given threshold
- number of documents of a given collection result set above a given threshold
- sum similarity measures of a given collection result set members that appear in the set of the top *K* highest similarity measure documents from all collections.
- number of documents of a given collection result set members that appear in the set of the top *K* highest similarity measure documents from all collections.

Methods of effectiveness measuring for collection selection are described in [18] and [17].

3.3 Reference Collection Fusion Techniques

This section describes in more detail the four collection fusion techniques used in the experiments of Section 3.6 and 3.7. Considering the retrieval algorithm described in Chapter 2, the characteristics of each collection fusion technique are described using input parameters/data, complexity, memory required, weaknesses, and strengths.

For a collection fusion technique to be appropriate for use in a PDA, it must be efficient given the PDA constraints. Techniques that examine the contents of the documents in the result list (e.g., [29] and [10]) incur a high processing cost. Techniques that utilize a learning approach using training data (e.g., [48], [47], [49], [43], [50], and [19]) suffer an increased storage requirement for the learned information and a more calculation intensive fusion process using the learned information. Other techniques require statistics to be gathered or stored (e.g., [6], [56] and [5]) which incur a processing cost and/or a storage cost on the PDA. And still other techniques require extra data about the documents that is not gathered by PalmIRA (e.g., HTML links [57], document meta-data other than the inverted index [22] and non-vector retrieval models [2]). The collection fusion approach must minimize the storage and processing cost. This leaves some of the simpler score and ranked based approaches (e.g., [42], [55], [37], [14] and [54]) of which some are experimented with in following sections.

3.3.1 Round Robin (RR) Fusion

This approach merges one result list document from each collection in one round and then merges the next un-merged documents in additional rounds ([14]). This interleaves the documents from each collection.

The algorithm characteristics include:

- Input: required input into the algorithm is a ranked result list from each collection.
- Complexity: linear O(N) where N is the sum of the size of result list from each of the $|C|$ collections. The $|C|$ is the number of collections.
- Memory: constant.
- Weaknesses: assumes relevant documents are spread approximately uniformly (i.e., set of heterogeneous collections) between the collections and thus the result lists. Although the technique is simple, each collection is modeled as contributing equal number and quality of results to the global result list. RR might boost the document ranking of a document from a irrelevant collection in a collection set.
- Strength: independent of the retrieval model since no parameters are required.

Example: In the round robin approach, the highest ranking but yet un-merged document is removed from one collection's result list (X9) and then from another collection (Y3) in a round robin fashion until all documents in the result lists are merged. Figure 17 displays the final ranked result list. Notice that this will produce a low precision if relevant documents are not spread evenly between each collection.

3.3.2 Round Robin Random (RRR) Fusion

The RRR technique proceeds very much like the RR technique in that the fusion operates by each round choosing a document to merge. The document to merge is chosen by rolling a $|C|$ faced die biased by the number of un-merged documents remaining in each result list ([42], [48]). $|C|$ is the number of collections. A random number is chosen between 1 and the total number of un-merged documents. If the documents are numbered between 1 and the number of un-merged documents, the collection which contains that un-merged document is chosen as the next collection to contribute its highest ranking document to the global result list. E.g., documents numbered 1 to x are the 1^{st} to x^{th} un-merged documents existing in collection 1 and documents numbered $x+1$ to $x+y$ are

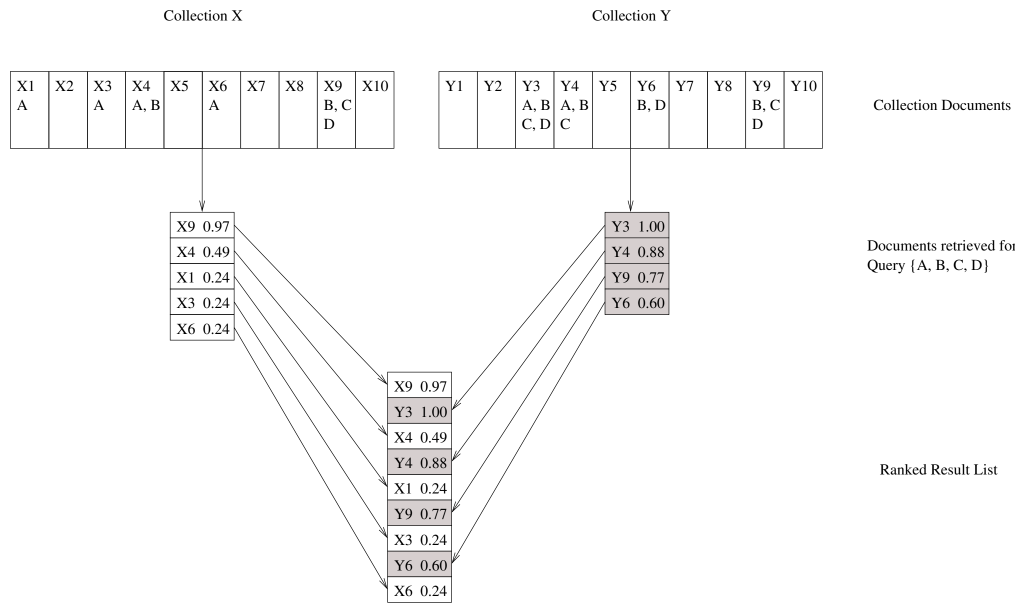


Figure 17: Collection Fusion - Round Robin

the $(x+1)^{th}$ to $(x+y)^{th}$ un-merged documents existing in collection 2 , and so on. If random number $x+3$ is generated, the highest ranking un-merged document from collection 2 is the next to be merged. For the following experiments, the random seed is set to the number of documents retrieved.

The algorithm characteristics include:

- Input: required input into the algorithm is a result list from each collection and the size of each result list.
- Complexity: linear O(N) where N is the sum of the size of result list from each of the $|C|$ collections. The $|C|$ is the number of collections.
- Memory: integer data types to store number of un-merged documents in each result list.
- Weakness: assumes a random distribution of relevant documents between the collections and thus the result lists. Relevant documents assumed to have a high probability of being in a larger result list.
- Strength: independent of the retrieval model since only the length of the result lists are required.

Example: In the round robin random approach, the probability of a given collection being chosen to contribute its currently highest ranking, un-merged document is biased toward the number of un-merged documents in each collection. In the first round (Fig 18), X9 is chosen to be merged since collection X contains the largest list of un-merged documents and thus has the highest probability of being merged. In the second round, each collection has the same number of un-merged documents (e.g., 4) and therefore the equivalent probability. In this round by chance collection X is chosen and the highest ranking un-merged document in collection X (X4) is merged. In the third round, document Y3 is merged. There are other orderings possible do to randomness and the changing probability as the number of un-merged documents in each result list changes. There is bias toward randomly choosing the the larger list of un-merged documents and this example illustrates the bias. This does not always lead to the optimum merging since in this case only documents from collection Y are considered relevant to the query.

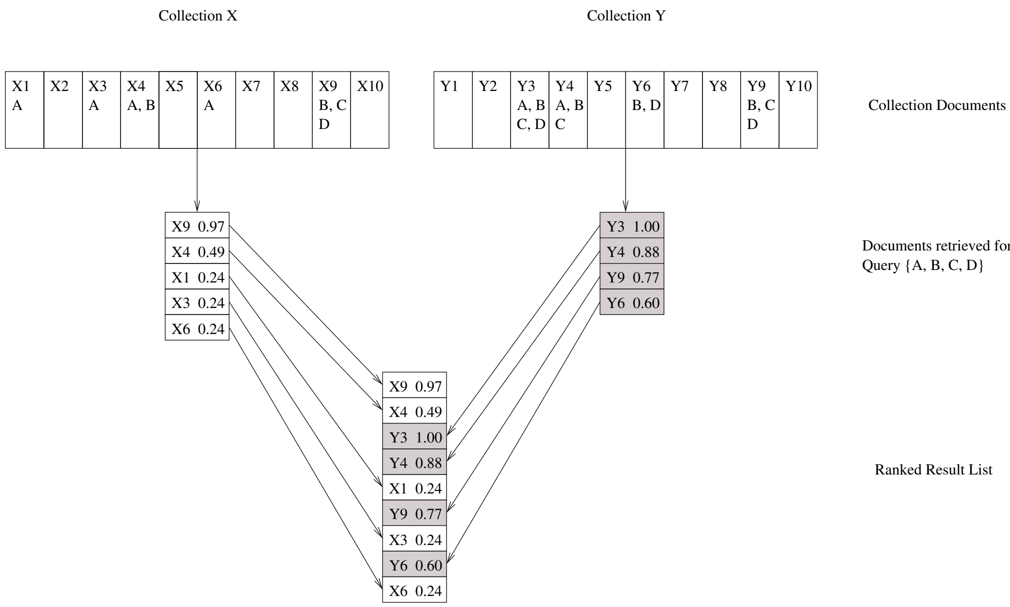


Figure 18: Collection Fusion - Round Robin Random

3.3.3 Original Weights (Raw Score) Fusion

This approach merges the collection result lists into one global result list sorted based on the score of the documents ([42], [11]). The merging works like a merge-sort based on the score of the document.

The algorithm characteristics include:

- Input: required input into the algorithm is a result list from each collection where the result list contains a score for each document.
- Complexity: to merge result lists already sorted by score: $O(N \times |C|)$ where $|C|$ is the number of collections. This is required to find the largest score not yet merged into the global result list from the sorted result list of each collection.
- Memory: Constant.
- Weakness: requires comparability of document scores between each collection result list. Some retrieval models do not assign scores to documents, others assign scores over different ranges (e.g., 0.0 - 1.0 or 0 - 100%) or assign scores that need to be normalized to be comparable. The retrieval models may differ for each collection and may use collection dependent parameters causing different documents to receive differing rankings based on the contents of the collection (e.g., inverse document frequency).
- Strength: simple as in requires a score for each document in the result list from each collection.

Example: In the original weights (raw score) approach, the documents are merged using a merge-sort like approach based on the score (i.e., degree of similarity) of the document (Fig 19). As a result, document Y3 is merged first since it has the highest score, followed by X9, Y4, and so on. A random mention of terms C and D in collection X cause document X9 to have a high degree of similarity even though X9 is not in the list of relevant documents. i.e., the fused rankings become skewed by collection dependent features in the retrieval model and the skew causes X9 to be ranked higher than the other relevant documents (Y4, Y9, and Y6). Since X9 and

Y9 contain the same terms (B, C, and D) then they should be ranked next to each other in the context of this approach. In this case, X9 is ranked second while Y9 is ranked fourth.

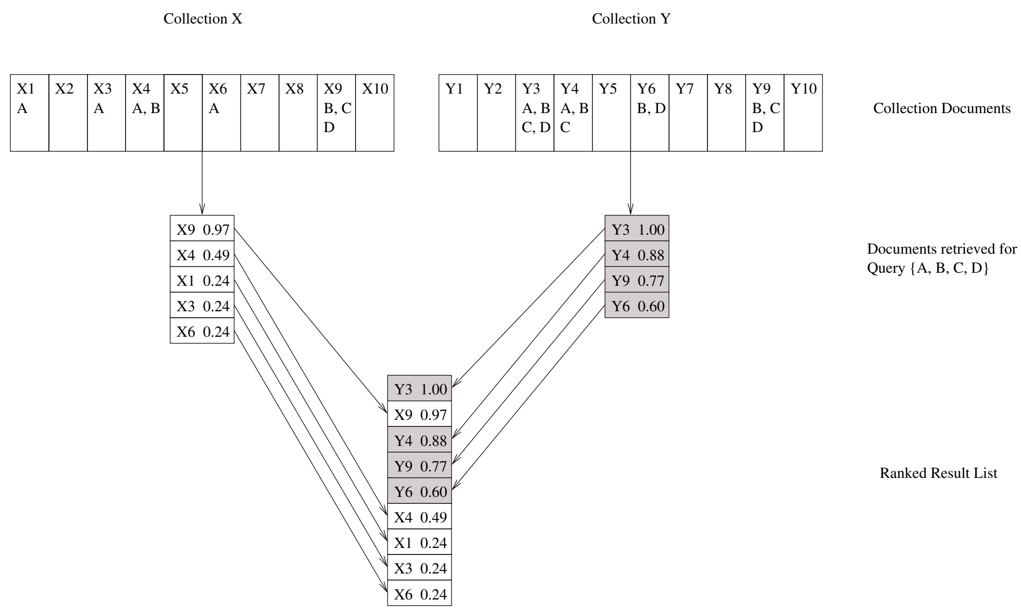


Figure 19: Collection Fusion - Original Weights (Raw scores)

3.3.4 Co-occurrence Collection Fusion - Our Contribution

Co-occurrence fusion attempts to accomplish collection fusion by re-weighting the score of each result list document based on the "goodness" of the collection and then merge and sort them by the new scores to create a merged result list. This "goodness" of the collection may be useful in a collection selection approach but this is not the focus of this research.

The heuristic that governs the "goodness" of a collection is based on the level of co-occurrence of query terms within documents d_j of collection C_k. A collection that contains a large number of documents containing a large portion of the query terms is considered better than a collection that contains a small number of documents with a small portion of the query terms.

The vector model represents document d_j and query q as vectors $\vec{d_j}$ and $\vec{q}$. Each dimension in the vector represents one of t indexed terms. Each dimension i is non-zero in the document/query vector if term i is contained within the document/query. The number of terms from query q that co-occur in document d_j within collection C_k, is the number of times that term i represented by dimension i in both $\vec{d_j}$ and $\vec{q}$ is non-zero for $i \in \{1 \dots t\}$.

More formally, given t terms and N document vectors from collection C_k, $\vec{q}$ is represented as a matrix $[q]_{1 \times t}$ and all N documents in collection C_k are represented as a matrix $[w_{i,j}]_{N \times t}$. Each row represents the weight $w_{i,j}$ of term k_i in document d_j for $i = \{1 \dots t\}$. The co-occurrence degree of C_k with respect to query q is:

$$|q^t \times b(C_k)|_{L1}$$

where

$$b(C_{k(i,j)}) = \begin{cases} 1 \; if \; w_{(i,j)} \neq 0 \\ 0 \; otherwise \end{cases}$$

For example if C_k is collection Y from Figure 20:

$$C_k = \begin{pmatrix} 0.70 & 0.40 & 0.52 & 0.52 \\ 0.70 & 0.40 & 0.52 & 0.00 \\ 0.00 & 0.40 & 0.00 & 0.52 \\ 0.00 & 0.40 & 0.52 & 0.52 \end{pmatrix}$$

$$b(C_k) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

$$q = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$$

$$q^t \times b(C_k) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \\ 2 \\ 3 \end{pmatrix}$$

$$|q^t \times b(C_k)|_{L1} = 12$$

Equation 8 calculates the "goodness" ($W_{(C_k,q)}$) of collection C_k given query q and the result list of N documents produced by the query. The value is normalized by the summation of the degree of co-occurrence over all the collections such that $\sum_{k=1}^{|C|} W_{(C_k,q)}$ equals 1 (where $W_{(C_k,q)}$ is the weight of collection C_k).

$$W_{(C_k,q)} = \frac{|q^t \times b(C_k)|_{L1}}{\sum_{k=1}^{|C|} |q^t \times b(C_k)|_{L1}} \tag{8}$$

If the query contains a single term, then the "goodness" is based on the number of documents in the result list. If only one collection is being queried, then the weight of the collection is considered to be 1 and the document scores remain unchanged.

After the collection weight ("goodness") is calculated for each collection, the result lists for each collection are re-weighted. The re-weighting of the degree of similarity of document d_j in collection C_k to query q is a product of the original document score ($sim_{(d_j,C_k,q)}$) and the collection weight ($W_{(C_k,q)}$)

$$sim^+_{(d_j,C_k,q)} = sim_{(d_j,C_k,q)} \times W_{(C_k,q)} \tag{9}$$

The new degree of similarity ($sim^+_{(d_j,C_k,q)}$) of each document reflects the "goodness" of the collection that the document was from. The degree of similarity can also be considered the score or weight of the document. The documents from each collection result list are then merge-sorted into one merged/fused result list based on the new degree of similarity.

The algorithm characteristics include:

- Input: required input into the algorithm is a result list from each collection where the result list contains a score for each document and the level of co-occurrence of the query terms within each collection.

- Complexity:

  - to calculate the sum of degree of the query term co-occurrence, the calculation uses elements as they are accessed by the vector model calculation. In the algorithm described in Section 2.7, a small addition is made to keep track of the sum of the level of co-occurrence of query terms as the vector model is being calculated for each document. This step is a simple summation that occurs during the calculation of the vector model for each document, therefore no additional complexity.
  - to re-weight documents: O(N).
  - to merge results lists sorted by weight: $O(N \times |C|)$ where $|C|$ is the number of collections. This is required to find the largest weight not yet merged into the global result list from the sorted result lists of each collection.

- Memory: One double data type for each collection to hold the sum of the degree of query term co-occurrence.

- Weakness: requires access to some form of information to determine the query term co-occurrence in each document or a single value of query term co-occurrence for the entire collection from the retrieval model. However, in the case of PalmIRA the inverted index for each collection is stored locally on the PDA. The number of query terms present in a document is determined during the inverted list access by the vector model calculation. The sum of query term co-occurrence for each collection is easily transferred as input into the collection fusion model.

- Strength: more importance is given to collections consisting of documents containing many of the query terms. The paper by Gravano [21] shows that using phrase information as part of a collection selection index using inference networks increases effectiveness of result merging. The difference between these techniques is that in this technique the term must co-occur in the document while Gravano suggested consecutive co-occurrence of terms in the format of a phrase.

Example: In the Co-occurrence fusion approach (Fig. 20), the document scores are re-weighted based on a weight assigned to the collection in which the document exists. After the query ("A B C D") operation on a collection, a result list and sum of the level of co-occurrence is passed on to the collection fusion algorithm. For collection X, the level of co-occurrence is 8 based on 3 documents containing 1 query term, 1 document containing 2 query terms and 1 document containing 3 query terms ($3 \times 1 + 1 \times 2 + 1 \times 3 = 8$). For collection Y, the level of co-occurrence is 12 based on 1 documents containing 2 query term, 2 document containing 3 query terms and 1 document containing 4 query terms ($1 \times 2 + 2 \times 3 + 1 \times 4 = 12$). So collection X passes a result list and the value 8 while collection Y passes a result list and the value 12 to the collection fusion algorithm. The Co-occurrence fusion algorithm produces the denominator or normalization factor as the sum of the numerators (8+12=20). The collection weight of collection X is 0.4 ($\frac{3\times1+1\times2+1\times3}{20} = \frac{8}{20}$). For collection Y, the weight is 0.6 or ($\frac{1\times2+2\times3+1\times4}{20} = \frac{12}{20}$). The documents from the result lists are then re-weighted depending on the "goodness" of the collection. The document score of Y9 is originally 0.77 and after re-weighting becomes 0.46 (0.6×0.77). The two result lists are then merged, sorted by the new document score.

This approach gives a higher weight to documents from collection Y and thus the documents from collection Y are ranked higher than documents from collection X. For X9, the document is ranked fourth as opposed to the previous approach that ranked the document second (Figure 19). The previous technique ranked X9 higher (second) and since X9 is not considered relevant, Co-occurrence fusion shows an improvement by ranking the document fourth. Co-occurrence fusion shows an improvement over previous techniques since it ranks the relevant documents higher in the result list.

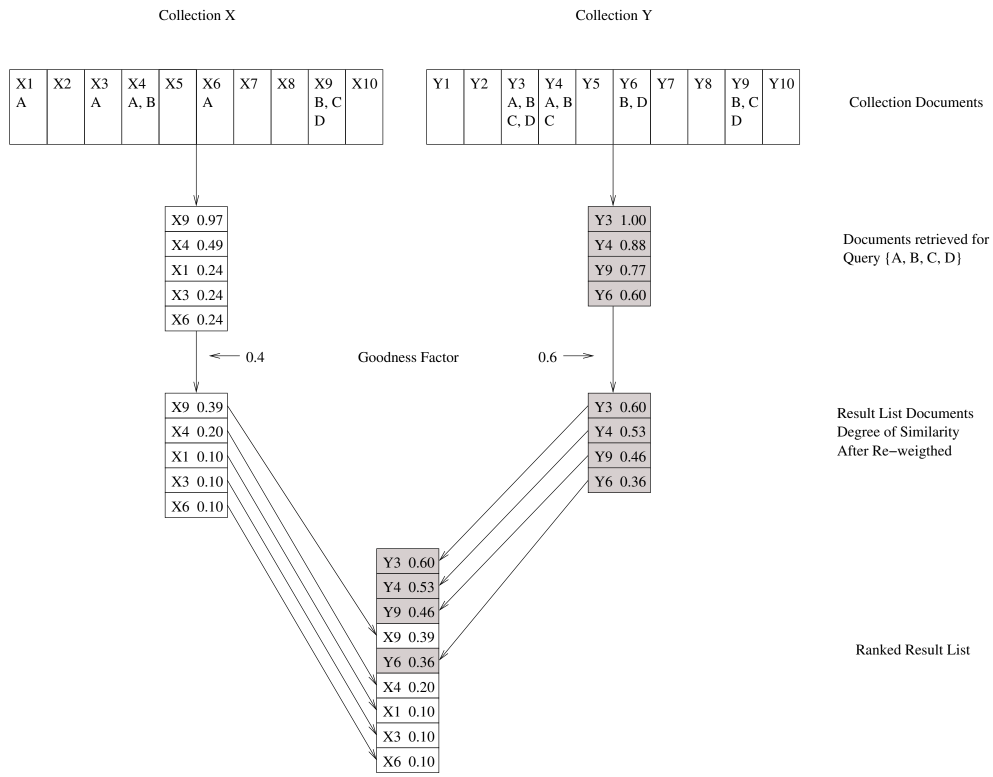


Figure 20: Collection Fusion - Co-occurrence Fusion

3.4 Proposed Effectiveness Measures

Section 3.2.2 describes attempts to measure the effectiveness of merging techniques in the surveyed literature. The difference in the proposed effectiveness measures is that they attempt to show how on average the rank of the documents differs between the baseline reference and the collection fusion method. The effectiveness measures based on precision show only the tendency of relevant documents to rank above irrelevant documents. These measures lack the ability to show whether or not a high ranking document in the reference result list is handicapped by the collection fusion technique and as a result receives a low rank. The following proposed measures attempt to provide insight into how the ranks of documents after collection fusion compare to a baseline.

This section proposes 4 new measures that attempt to determine effectiveness of a collection fusion technique. Given a query, the measures are based on the difference in document rank between the result list of a collection fusion technique and that of the reference collection. The difference measures are calculated relative to the reference collection. The reference collection consists of all of the individual collections concatenated together to form one collection and then indexed (Section 3.1.2). The assumption is that the result list produced from the reference collection is the best answer to that query. The following measures attempt to show how much each collection fusion technique differs from the reference collection.

None of the difference based measures include an ordering penalty. Intra-ordering does not penalize for the various possible orderings of a subset of consecutively ranked documents with equal weights in the reference collection result list. I.e., documents are moved around in a block of documents with the same weight to minimize the difference of various orderings.

3.4.1 Rank Difference (dR)

The rank difference (dR) measure describes by how much a given collection fusion result list document differs in rank compared to that document in the reference collection result list. For a given query, this calculates the difference in rank position between the result list of a collection fusion technique and the result list of the reference collection of a given document. Equation 10 is calculated over N documents in the result list of the fusion technique and an un-normalized average is produced for each query q. No normalization occurs thus allowing analysis of how far out the rank ordinals are on average for the retrieved documents. A value of $dR(q) = 0.0$ indicates that on average the documents are ranked at equivalent ordinals when comparing the reference and the fusion result lists, i.e., the smaller $dR(q)$, the better. The $dR(q)$ is averaged over all Q queries (Equation 11) to produce a single value for the fusion technique. Formally we have:

$$dR(q) = \sum_{d=1}^{N} \frac{|P^r_{(d,q)} - P^c_{(d,q)}|}{N} \tag{10}$$

$$dR = \sum_{q=1}^{Q} \frac{dR(q)}{Q} \tag{11}$$

where given a query q, $P^r_{(d,q)}$ is the position of document d in the result list of the reference collection r, $P^c_{(d,q)}$ is the position of document d in the result list of the collection fusion technique c and N_q is the size of the result list.

Example: In the dR measure, for each document in the collection fusion result list (Fig. 21(a)), calculate the absolute value of the difference in position of the corresponding document in the reference collection result list. Document Y3 is first in both result lists yielding a difference of 0. Document Y4 is second compared to fourth yielding a difference of 2. Continuing to document X9, documents X9 and Y9 have the same score in the reference result list. The measure assumes that the order of X9 and Y9 could be reversed and therefore the measure yields a difference of 1 (instead of the absolute value of -2). This ordering independence holds for the other three measures presented next as well. Averaging of the resulting 9 values yields (3/9) 0.33 for the collection given the query. Therefore, on average documents are 0.33 of an ordinal out of position. In the case of Fig 21(b), a larger value shows that the RRR fusion is not as effective.

3.4.2 Rank Difference of only relevant documents (dRR)

Equation 12 is a slight modification of Equation 10. Only the relevant documents contribute to the value for a query and the absolute value of the difference is not used. The measure expresses how the relevant documents in the collection fusion technique result list are ranked with respect to the reference result list. This measure captures how the collection fusion technique treats the relevant documents by tracking the rank of relevant documents with respect to the baseline. In Equation 12 a positive value indicates that the relevant document (d) is ranked higher (i.e., closer to first) in the collection fusion result list than that document in the reference collection result list. A negative value indicates the opposite. Hence the larger (positive) the $dRR(q)$ value, the better the answer.

Equation 12 is averaged over all N^{rel} relevant documents d_r in the collection fusion result list. No normalization occurs thus allowing analysis of how far out the rank ordinals are on average for the retrieved relevant documents. The $dRR(q)$ is averaged over all Q queries (Equation 13) to produce a single value for the fusion technique. That is:

$$dRR(q) = \sum_{d=1}^{N^{rel}} \frac{(\rho^r_{(d_r,q)} - \rho^c_{(d_r,q)})}{N^{rel}} \tag{12}$$

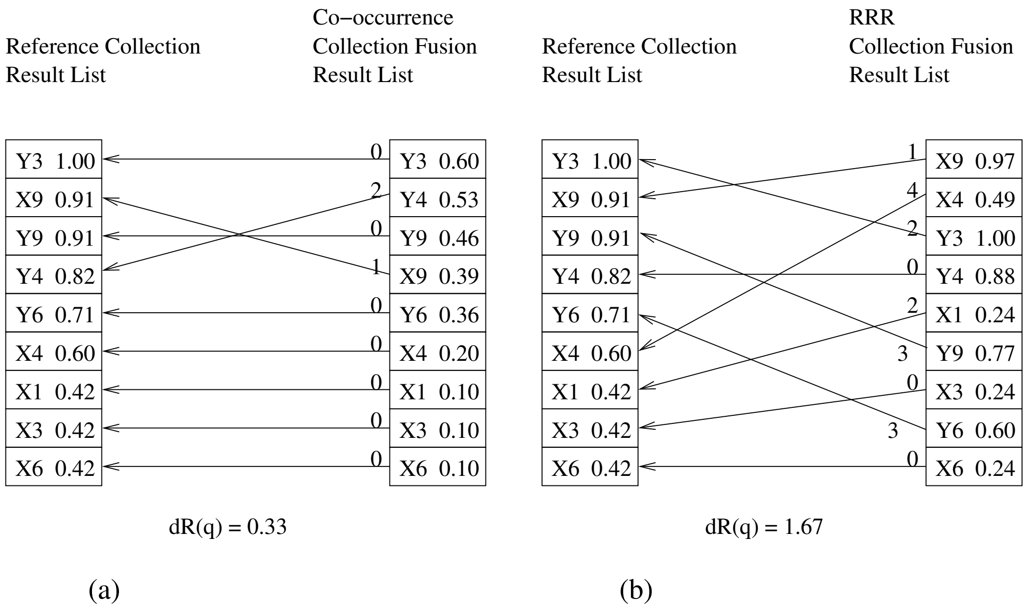


Figure 21: Rank Difference Measure (dR) (a) Co-occurrence (b) RRR

$$dRR = \sum_{q=1}^{Q} \frac{dRR(q)}{Q} \tag{13}$$

where $\rho^{r}_{(d_r,q)}$ is the position of a relevant document d_r in the result list of the reference collection and $\rho^{c}_{(d_r,q)}$ is the position of that relevant document d_r in the result list of the collection fusion technique.

Example: In the dRR measure, only the relevant documents in the collection fusion result list are considered. In the case of Fig 22(a), these documents are Y3, Y4, and Y9 (as specified earlier). For each of the relevant documents, the difference in position is calculated. For Y3 and Y9 this is 0. For Y4, this is +2. The measure retains the sign of the value (i.e., no absolute value). Averaging over the 3 values yields +0.67 for the collection given the query. The positive value shows that the relevant documents are ranked 0.67 ordinals higher on average in the collection fusion result list than in the reference result list. In the case of Fig 22(b), a negative value shows that on average relevant documents are ranked lower. Therefore, the RRR fusion is not as effective.

3.4.3 Weighted Rank Difference (dWR)

Another measure is the weighted rank difference (dWR) measure which shows by how much a given collection fusion result list document differs in rank compared to that document in the reference collection result list. This measure also considers that if the document has a large weight in the reference collection result list then it is more important for the collection fusion technique to rank the document in the same position since the absolute value is used. Equation 14 is calculated over N documents in the result list of the fusion technique for a given query and an average is produced, $dWR(q)$ (Equation 15). A value of 0.0 indicates that on average the documents are ranked at equivalent ordinals when comparing the reference and the fusion result lists. Hence the larger the $dWR(q)$ value, the better the answer. The $dWR(q)$ is averaged over all queries (Equation 16) to produce a single

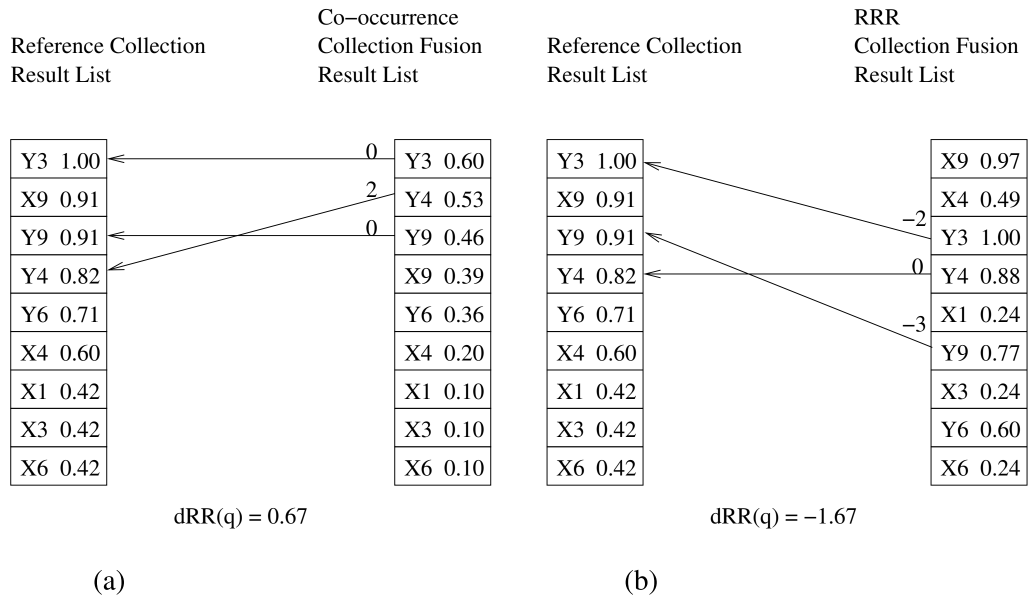


Figure 22: Rank Difference Measure - Relevant Documents Only (dRR) (a) Co-occurrence (b) RRR

value for the fusion technique. More formally:

$$dWR(d,q) = \frac{|P^r_{(d,q)} - P^c_{(d,q)}| \times W^r_{(d,q)}}{\max_q(|P^r_{(d,q)} - P^c_{(d,q)}| \times W^r_{(d,q)})} \tag{14}$$

$$dWR(q) = \frac{\sum_{d=1}^{N} dWR(d,q)}{N} \tag{15}$$

$$dWR = \sum_{q=1}^{Q} \frac{dWR(q)}{Q} \tag{16}$$

where $W^r_{(d,q)}$ is the degree of similarity of a document d in the reference collection result list and $\max_q$ is the maximum value over the result list for query q. The result is normalized to be between 0.0 and 1.0.

Example: In the dWR measure (Fig. 23(a)), the difference in rank and the weight of that document in the reference result list are considered. The measure assigns a value of 0 to Y3 since there is no positional difference in rank (e.g., $(1-1) \times 1.00$). For Y4, the positional difference is 2 and a weight of 0.82 producing 1.64. For X9, the positional difference is 1 (no order penalty, absolute value) and a weight of 0.91 producing 0.91. Each value is then normalized by the maximum value in the set of documents (1.64 in this case) producing 1.0 for Y4 and 0.55 for X9. After averaging over the 9 values, 0.17 is the value for the collection given the query. In the case of Fig 23(b), a larger value shows that the reference documents with high scores tend to be more out of position than in Co-occurrence fusion. Therefore, RRR fusion is not as effective.

3.4.4 Weighted Rank Difference of only relevant documents ($dWRR$)

Equation 17 is a slight modification of Equation 14. Only the relevant documents contribute to the value for a query and the absolute value of the difference is not used. The measure expresses how the relevant documents

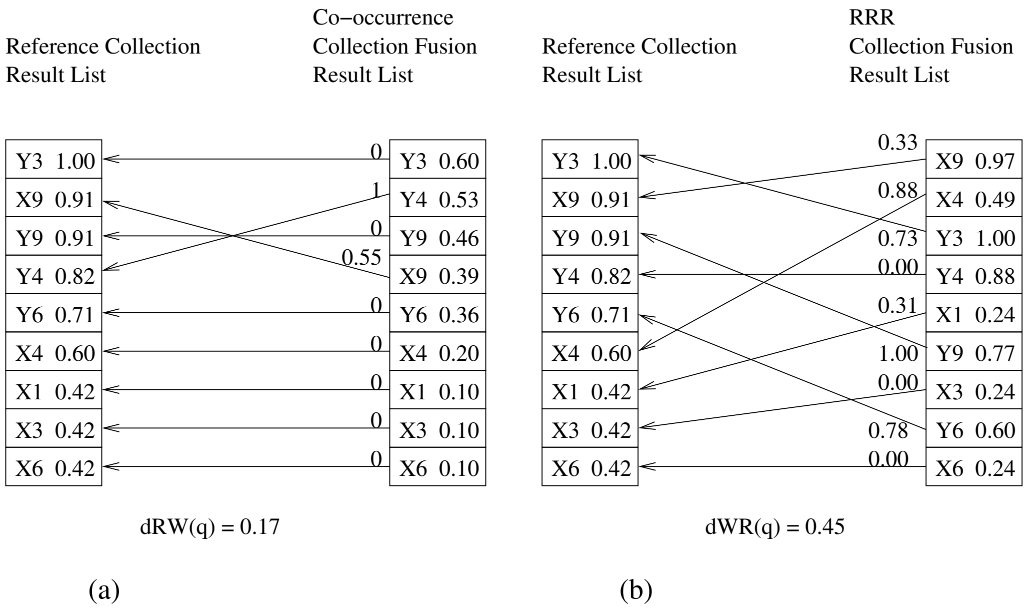


Figure 23: Weighted Rank Difference Measure (dWR) (a) Co-occurrence (b) RRR

are ranked with respect to the reference result list by giving more weight to documents with a higher score when they are out of order in a positive or negative direction. In Equation 17 a positive value indicates that the relevant document (d_r) is ranked higher (i.e., closer to first) in the collection fusion result list. A negative value indicates the opposite. Hence the larger (positive) the $dWRR(q)$ value, the better the answer.

Equation 17 is averaged over all N^{rel} of the relevant documents d_r in the collection fusion result list for a given query producing $dWRR(q)$ (Equation 18). The $dWR(q)$ is averaged over all Q queries (Equation 19) to produce a single value for the fusion technique. More formally:

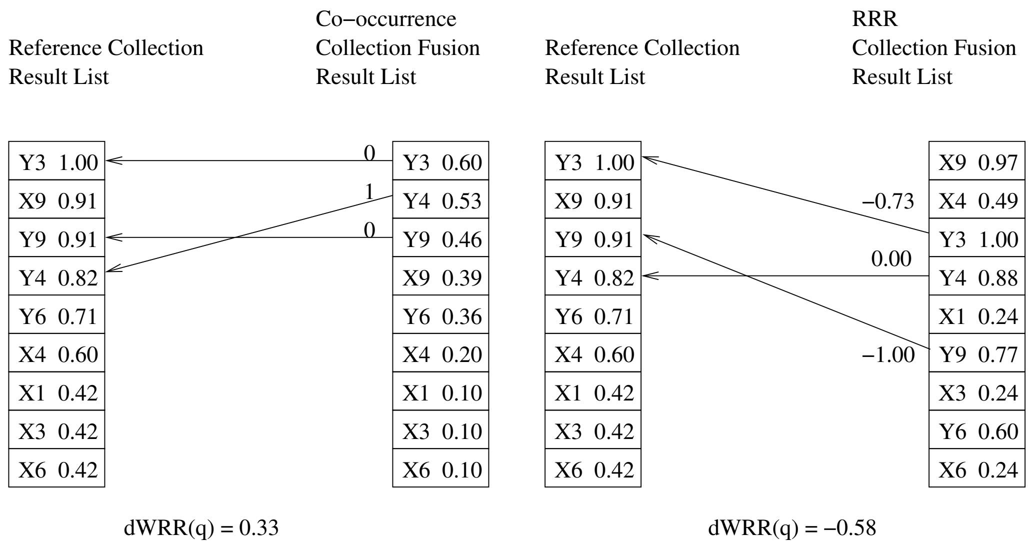
$$dWRR(d_r, q) = \frac{(\rho^r_{(d_r,q)} - \rho^c_{(d_r,q)}) \times \omega^r_{(d_r,q)}}{\max_q(|\rho^r_{(d_r,q)} - \rho^c_{(d_r,q)}| \times \omega^r_{(d_r,q)})} \tag{17}$$

$$dWRR(q) = \frac{\sum_{d=1}^{N^{rel}} dWR(d_r, q)}{N^{rel}} \tag{18}$$

$$dWRR = \sum_{q=1}^{Q} \frac{dWR(q)}{Q} \tag{19}$$

where $\omega^r_{(d_r,q)}$ is the degree of similarity of a relevant document d_r in the reference collection result list and $\max_q$ is the maximum value over the result list for query q. The result is normalized to be between -1.0 and 1.0.

Example: In the $dWRR$, only the relevant documents in the collection fusion result list are considered (e.x. Y3, Y4, and Y9) in Fig. 24(a). The value is 0 for Y3 and Y9 since the positional difference is 0. For Y4, the position difference is 2 (no absolute value) and a weight of 0.82 yielding 1.64. This value is normalized by the maximum of the absolute value of the value contributed by each document (0, 1.64, 0). This produces a value of 1 (e.g $\frac{1.64}{1.64}$=1) from Y4. After averaging over the 3 values, 0.33 is the value for the collection given the query. This yields a high positive average value if on average highly ranked relevant documents with large scores are ranked higher in the merged result list as opposed to the reference baseline. In this case the merged result list produces a higher ranking for the relevant documents than the baseline. In the case of Fig 24(b), a negative value shows that the relevant documents with large weights are ranked lower and thus RRR fusion is not as effective.



(a) (b)

Figure 24: Weighted Rank Difference Measure - Relevant Documents Only ($dWRR$) (a) Co-occurrence (b) RRR

3.5 Experimental Setup

Testing the effectiveness of a collection fusion technique requires multiple collections where each collection contains a large number of documents. Because of the limitations of storage, CPU speed, etc. of a PDA, a simulation of the PalmIRA application was created to run within a Linux environment. The simulation simulated PalmOS Data Management by utilizing a flat file for record storage.

Two different data sources were used in the following experiments. The sources are:

- CACM, CISI, CRAN, and MED data
- TREC data

3.6 Experiments and Analysis: CACM, CISI, CRAN, and MED Data

3.6.1 Data Description and Preprocessing

For the first set of experiments, we used freely available data-sets from Cornell University [7]. These data-sets were originally used for single collection, stand-alone information retrieval experiments. Each data-set (e.g., CACM) contains a collection of documents, a collection of queries and a set of relevance judgments for the documents within only that data-set. These relevance judgments are used to determine the effectiveness of the retrieval method in terms of precision/recall measures.

In a multi-collection retrieval, each data source is considered to be a separate collection. A given query is submitted to each collection producing a result list for each collection. The result lists are then merged to produce one single result list. An assumption is made for the Cornell data that only the documents within the data-set that the query originated are relevant to the query. This is due to the lack of relevance judgments for the documents within the other 3 collections of documents.

	CACM	CISI	CRAN	MED	TOTAL
# queries with relevance judgments	52	76	225	30	383
# documents	3204	1460	1400	1033	7097

Table 1: Data-sets

In order for the CACM, CISI, CRAN, and MED data-sets to become usable for the multiple collection information retrieval experiments some data cleaning steps were required. To fix the problem of duplicate document ids occurring within the concatenated reference collection or merged result lists, document ids for three of the four collections were changed to include a prefix. For the CACM and CISI data-sets, not all of the queries have relevance judgments. There are 431 queries when all four (CACM, CISI, CRAN, MED) data-sets are combined. There are 381 queries that have relevance judgments. All queries that did not have a relevance judgment associated with the query were removed. For documents in CACM and CISI, the .X section (key to citation information) was removed. For queries in CACM and CISI, the .N section (describing the origin of the query) was removed.

For a baseline, a reference collection is built by concatenating the documents from the CACM, CISI, CRAN, and MED data-sets. The reference collection is indexed and queried as a single collection (See Section 3.1.2). The resulting result lists act as a baseline for the collection fusion strategies.

After data cleaning, Table 1 represents the number of documents and number of queries with relevance judgments. These values are used in the following experiments.

All tests were completed without using stemming (described in Section 2.5.2) unless otherwise stated. Stemming reduces terms to their grammatical root. By doing this, the meaning of the term may be lost. For example, the term “fishing”. Porter’s stemming algorithm [35] reduces “fishing” to the root “fish”. In essence, by querying for the verb “fishing”, the results now contain documents for the verb “fishing” and the noun “fish” which is not ideal. Witten et al. [52] argue that extraneous material may be retrieved as a result of the stemmed version of the query such that:

> “. . . , a search for the work by “Cleary and Witten” turns into a quest for “clear and wit”.”

Conducting an experiment using stemming increased precision by less than 3% at each of the 11 levels of recall for both the reference collection and Co-occurrence fusion. Indeed, some web search engines do not use stemming (e.g., Google [21]).

For non-stemming experiments, the average number of relevant documents retrieved was 14.5, maximum 133, minimum 0, median 9, and standard deviation 18.207.

Table 2 tracks the number of times that Co-occurrence fusion assigns the collection that the query originated from, the highest, second highest, and so on, collection weight. That is, the collection with the highest weight tends to have its documents increased in rank relative to the other collections.

For the 383 queries in 4 collections, the results in order from highest to lowest collection weights are: 317, 56, 9, and 1 respectively (Table 2). I.e., 83% of the time, the collection that the query originated from is assigned the highest collection weight. Table 2 describes how often the collection the query originated from was assigned the largest weight (e.g., queries from the CACM collection caused Co-occurrence fusion to assign the CACM collection the highest weight 43 out of 52 occasions).

If this is considered before the data cleaning in Section 3.6.1 then for the 431 queries, the results are: 330, 84, 16, and 1 respectively.

[21] www.google.com

	CACM	CISI	CRAN	MED	TOTAL
Highest	43 (83%)	47 (62%)	208 (92%)	19 (63%)	317 (83%)
	8 (15%)	28 (37%)	14 (6%)	6 (20%)	56 (15%)
	1 (2%)	1 (1%)	3 (1%)	4 (13%)	9 (2%)
Lowest	0 (0%)	0 (0%)	0 (0%)	1 (3%)	1 (0%)

Table 2: Collection weight of the collection in which query originated

3.6.2 Precision and Recall Measure Analysis

Precision is one approach to measure effectiveness (Section 3.2.2). Precision is the percentage of the retrieved documents judged relevant to a query. Recall is the percentage of the judged relevant documents that have been retrieved. Three precision based techniques are:

- average precision at the 11 standard levels of recall (0%, 10%, ... 100% recall).
- average precision at document cut-offs (i.e., after n non-relevant or relevant documents have been seen).
- average recall at document cut-offs.

The precision at each level of recall is averaged over all queries (e.g., 383 queries for the CACM, CISI, CRAN, MED combination data-set). The curves represent results lists created by queries posed to the reference collection along with result lists created by collection fusion techniques: Round Robin, Round Robin Random, Original Weights (Raw Scores) and Co-occurrence collection fusion. The reference collection is considered to be the baseline ranking. Another possibility which is used is to use only the queries from one collection (e.g., CACM) and compute the precision measures. The CACM, CISI, CRAN, MED collections each have relevance judgments for the queries and documents in its own collection (i.e., no relevance judgments for queries from one collection for another collection's documents).

A second possible baseline involves comparing the precision curves already mentioned to the curve created when querying a single, independent, stand-alone collection (i.e., no collection fusion). E.g., run the set of CACM queries against only the CACM documents and compute the measures. This is as opposed to querying the reference collection or each independent collection (e.g., CACM, CISI, CRAN, MED) and merging the results. This is the baseline precision if we assume that only documents from the collection that the query originated (e.g., CACM) are relevant to that query (i.e., the only relevance judgments available for CACM, CISI, CRAN, MED data-sets are for documents internal to only that data-set). Based on this assumption, querying multiple collections and fusing the results should "ideally" approach the effectiveness of the stand-alone data-set. This assumption may be false since it is possible for documents from other collections to be relevant but un-judged. The difference between the stand-alone approach and the reference collection precision is considered noise. There are two possible sources of noise. First, some documents documents belonging to data-sets (e.g., MED) other than the collection that the query originated (e.g., CACM) are actually relevant to the query. Second, the reference collection producing concatenation dilutes/concentrates the inter-document dissimilarity of the collection dependent parameters. The "Only C Data-set" is the curve of the precision/recall measure for the stand-alone data-set where collection C is searched independent of the other collections. I.e., in "Only C Data-set" there is no collection fusion or collection concatenation as in the reference collection, just the search of the stand-alone collection C. The motivation is to see how searching one collection independently of other collections compares to searching multiple collections concatenated (as in the reference collection) or multiple collections with fused result lists.

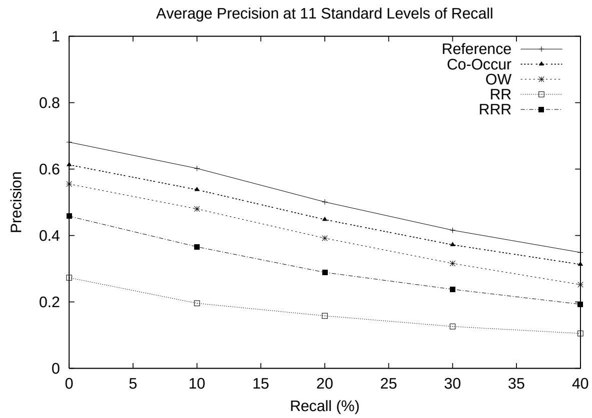


Figure 25: Average Precision at 11 Standard Levels of Recall - All Queries

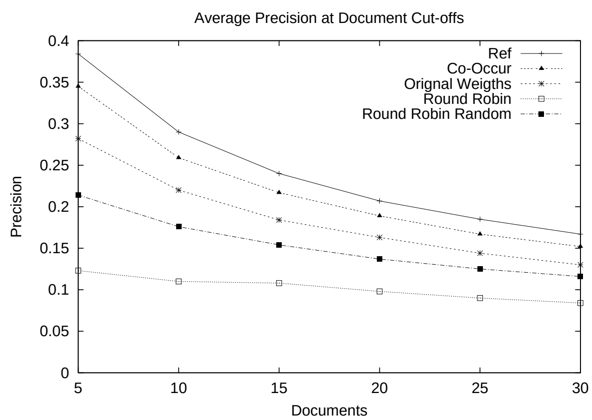


Figure 26: Average Precision at Document Cut-offs - All Queries

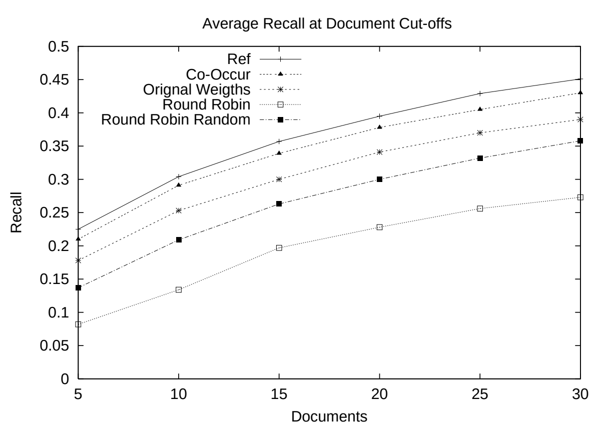
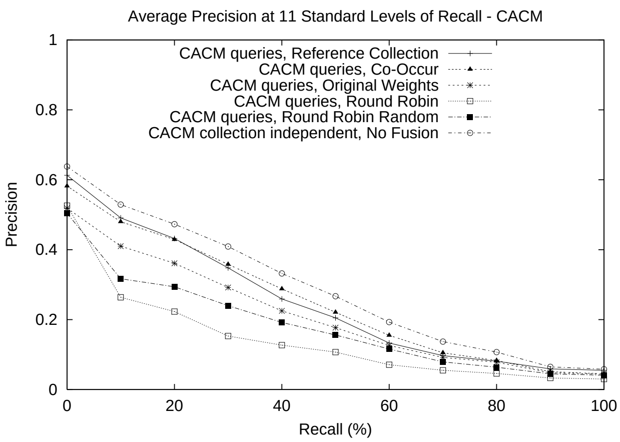
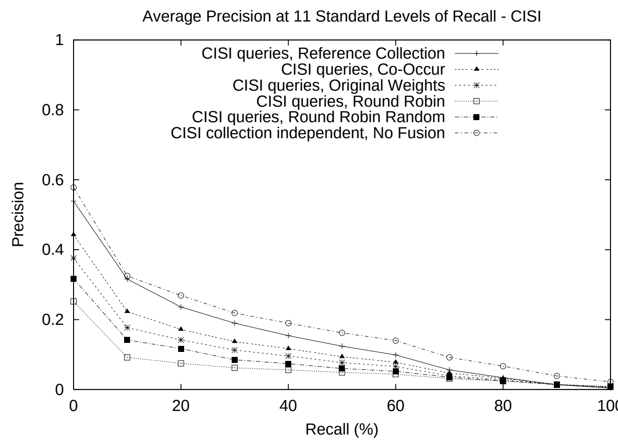


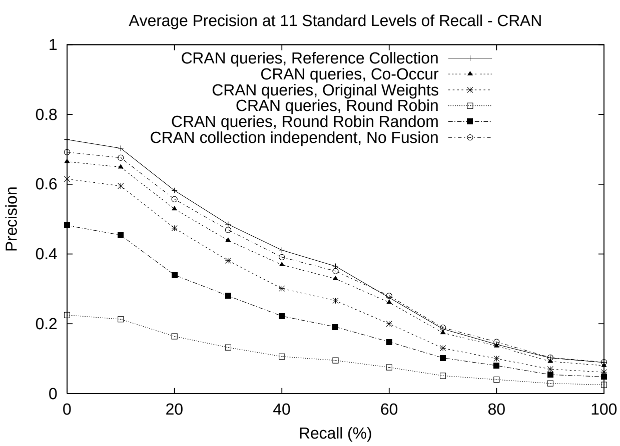
Figure 27: Average Recall at Document Cut-offs - All Queries



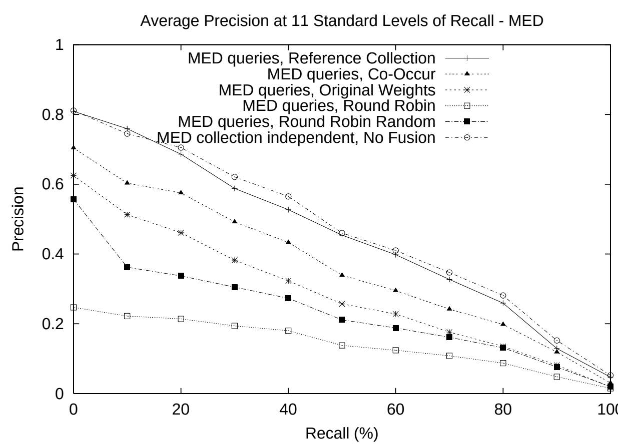
(a) CACM Queries



(b) CISI Queries

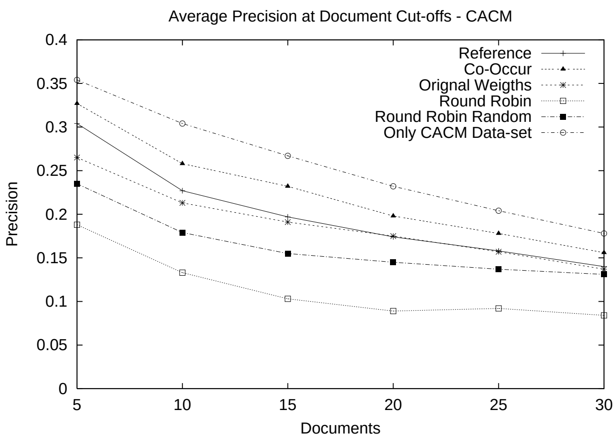


(c) CRAN Queries

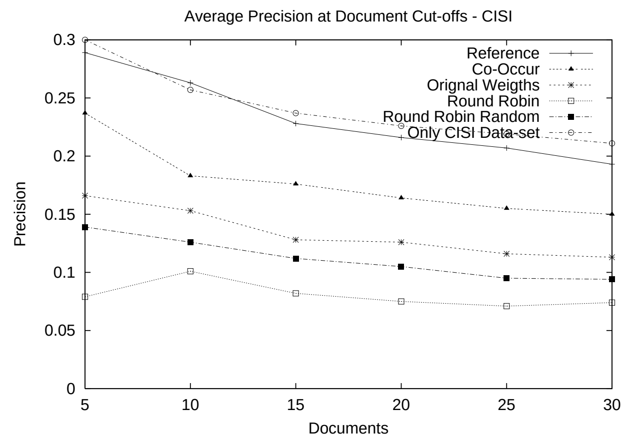


(d) MED Queries

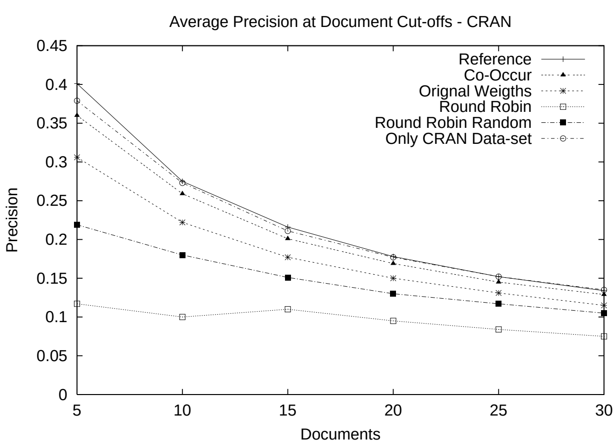
Figure 28: Average Precision at 11 Standard Levels of Recall



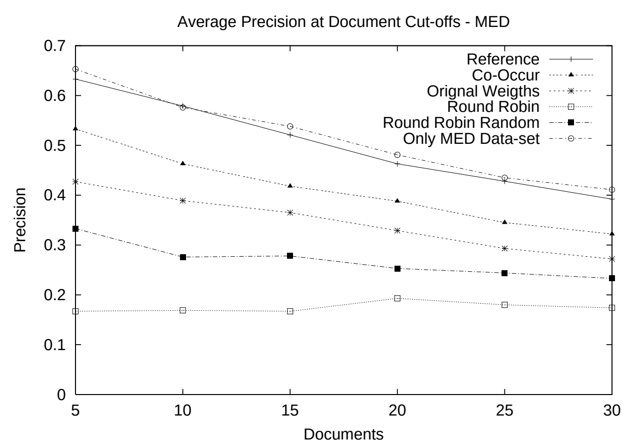
(a) CACM Queries



(b) CISI Queries

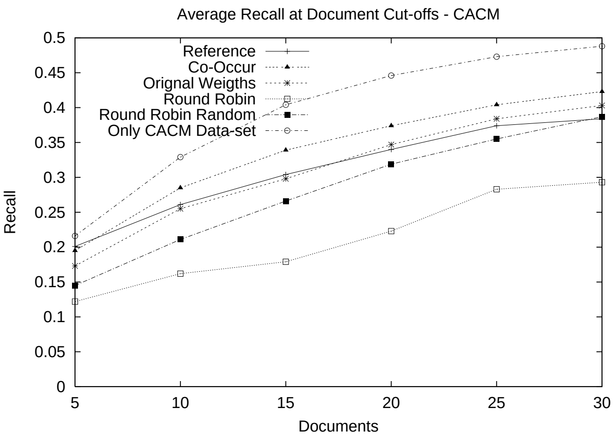


(c) CRAN Queries

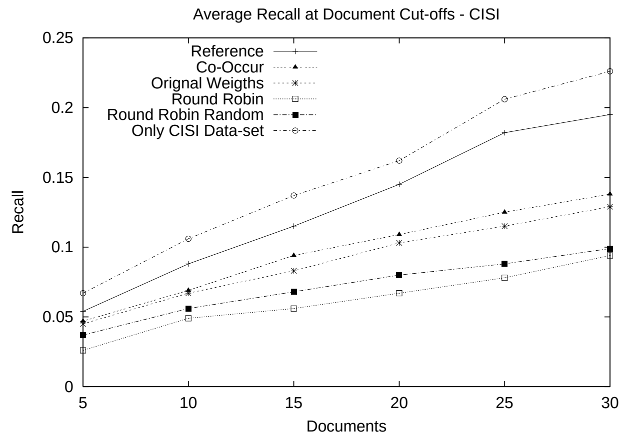


(d) MED Queries

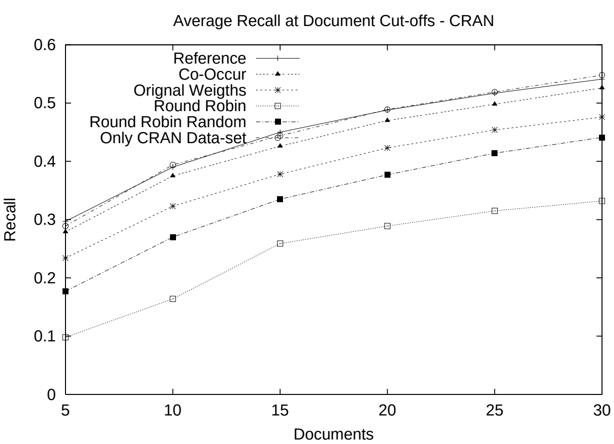
Figure 29: Average Precision at Document Cut-offs



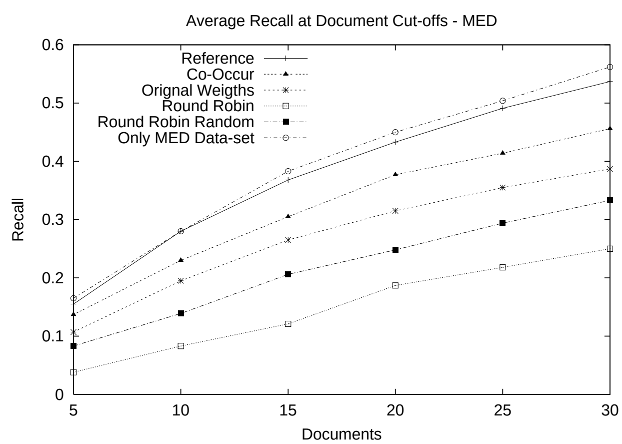
(a) CACM Queries



(b) CISI Queries



(c) CRAN Queries



(d) MED Queries

Figure 30: Average Recall at Document Cut-offs

In all precision/recall based effectiveness measures, the Co-occurrence fusion approach produces better results than the other collection fusion approaches. In the precision at 11 standard levels of recall graph (Figure 25), there is an approximately 7% difference between the baseline reference collection and Co-occurrence fusion at low levels of recall and the difference narrows as the level of recall increases. The baseline is better than Co-occurrence fusion. To determine if this trend would continue when using queries from only one data-set (e.g., CACM), graphs were created using queries from each data-set. Using queries from the CISI, CRAN, and MED data-sets, these query sets follow this trend (CISI Figures 28(b), 29(b), 30(b), CRAN Figures 28(c), 29(c), 30(c), MED Figures 28(d), 29(d), 30(d)). On the other hand, the queries from CACM produce Co-occurrence results that differ from the trend such that the results are close to or slightly higher than the reference collection (Figures 28(a), 29(a), 30(a)).

With respect to second baseline, the curve for the CISI, CRAN, and MED queries follow very closely the curve of the reference collection (CISI Figures 28(b), 29(b), 30(b), CRAN Figures 28(c), 29(c), 30(c), MED Figures 28(d), 29(d), 30(d)). For the CACM queries, this changes such that reference collection is relatively noticeably lower than the single collection, CACM data-set without collection fusion results (Figures 28(a), 29(a), 30(a)). This suggests that the effectiveness of CACM queries are reduced when run against the concatenated reference collection. The concatenation that created the reference collection must change the *idf* value of some/all terms such that the terms used to indicate the relevant documents are assigned a lower *idf* and thus a lower degree of similarity. The result is that the precision and recall for the CACM queries is adversely affected when the collections are concatenated together into the reference collection.

Based on the above graphs for this data, Co-occurrence fusion is shown to produce the best precision/recall results when compared to the other fusion techniques. Co-occurrence also demonstrates that it can produce better results than the reference collection under certain circumstances, in this case when the queries from the CACM collection are used.

3.6.3 Rank Difference Measures Analysis

The rank difference measures produce a value for each query based on how the result list of a collection fusion technique differs from that of the reference collection. The measure is calculated after considering n documents of the fusion technique's result list. The effect is to visualize how out of position in terms of rank the documents are when compared to the reference collection. The results are produced after a set of different numbers of documents have been seen in the result list (e.g., 10, 20, 30, etc.).

A curve is created for each collection fusion technique by averaging the value over all queries (e.g., 383 for the CACM, CISI, CRAN, MED combination data-set).

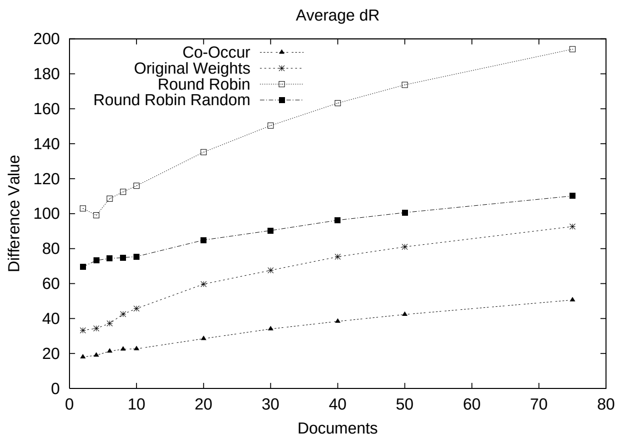
The Co-occurrence fusion techniques tends to produces result lists with documents ranked such that the difference in rank ordinal is closer to 0 than the difference rank of documents of the other fusion techniques. When considering relevant and non-relevant documents in the dR measure, it produces results where the Co-occurrence fusion result list on average more closely matches the reference results at all levels after n documents are considered in the measure (Figure 31(a)). If only the relevant documents are averaged over the levels of n documents as in the dRR measure (Figure 31(b)), on average the relevant documents in the result list of the fusion technique are ranked higher than in the reference collection as indicated by the positive value.

The dWR and $dWRR$ measures show on average that Co-occurrence fusion tends to assign ranking ordinals to documents with higher weights closer to ordinals of the reference collection than the other collection fusion methods (Figures 31(c), 31(d)). If only the relevant documents are considered, $dWRR$, Co-occurrence fusion on average ranks the relevant documents with a high weight close to but not quite as high as in the reference collection as shown by the slight negative value in Figure 31(d). The queries have differing numbers of relevant documents where the average is 14.5. This suggests that even though relevant documents on averaged ranked 0 to 4 ordinals higher (Figure 31(b)) than in the reference collection, relevant documents with a high weight are not necessarily ranked higher (Figure 31(d)). Figure 25 shows that the precision curves of the reference collection start out being higher and then converge with the Co-occurrence fusion curve as the percentage of recall increases. The converging precision suggests that the relevant documents with a lower weight and thus appearing lower in the result list and being seen later are being ranked higher by Co-occurrence fusion than in the reference collection. If the considered relevant documents in the $dWRR$ measure after 8 result list documents are considered (middle ranked documents), the relevant documents are on average lower ranked than then the reference collection. The documents with the middle ranking and weights appear to be reducing the effectiveness of Co-occurrence fusion.

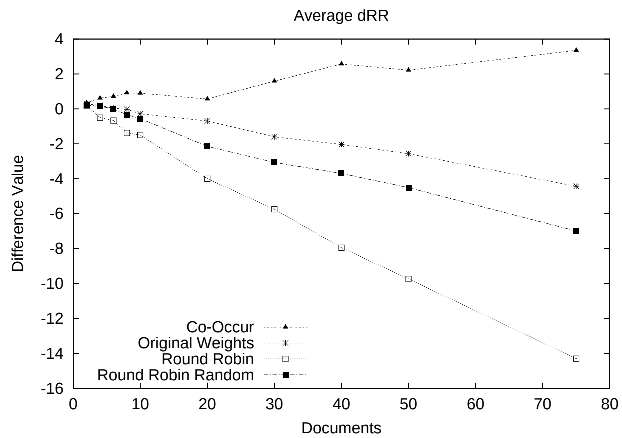
In order to determine if the queries from one data-set (e.g., CACM) are adversely or positively affecting the results of the concatenated set of queries from all data-sets, a second set of graphs use only the queries from one collection (e.g., CACM). The motivation is to determine if the queries from one data-set tend to yield results dissimilar to the result of queries for each of the other data-sets or the the concatenated set of queries from all data-sets.

For the CISI, CRAN, and MED queries separately, they tend to follow the trend of concatenated query case for all four measures (similar to the precision/recall results above) (Figures 31(b), 31(d), 31(a), and 31(c)). For the CACM queries, the results show that the relevant documents tend to be ranked higher than in the reference collection (Figure 32, 33). This supports the higher precision curve for the CACM queries in the previous (precision/recall measures) section.

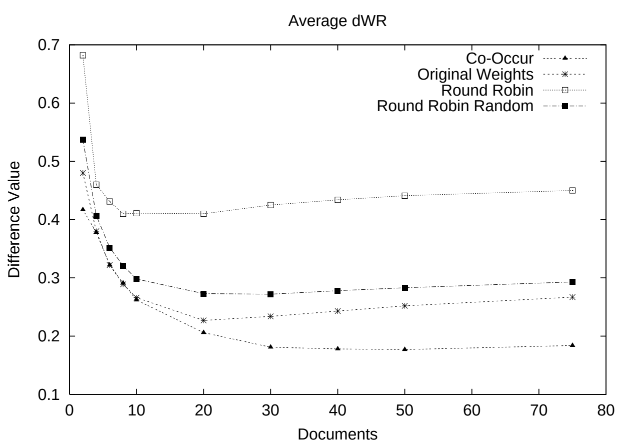
Based on the above graphs for this data, Co-occurrence fusion is shown to produce the best rank measure results when compared to the other fusion techniques. Co-occurrence also demonstrates that it can produce better results than the reference collection under certain circumstances, in this case when the queries from the CACM collection are used.



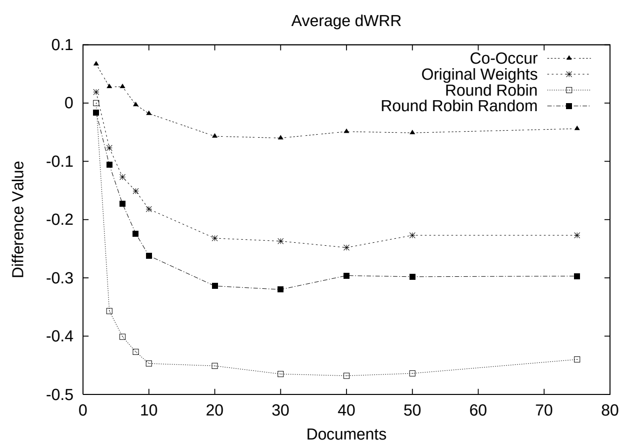
(a) Average Rank Difference (dR) at Document Cut-offs (smaller is better)



(b) Average Rank Difference - Rel (dRR) at Document Cut-offs (larger is better)



(c) Average Weighted Rank Difference (dWR) at Document Cut-offs (smaller is better)



(d) Average Weighted Rank Difference - Rel ($dWRR$) at Document Cut-offs (larger is better)

Figure 31: Rank Difference Measures

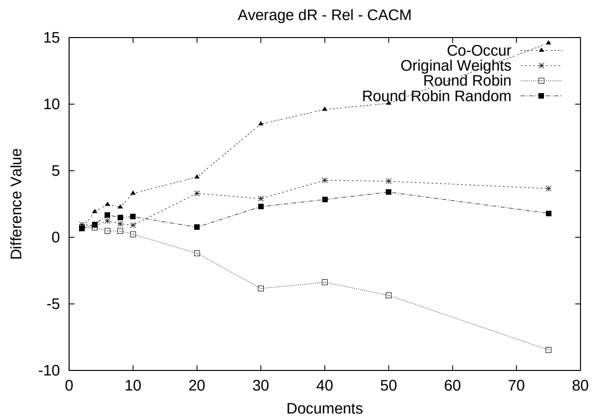


Figure 32: Average Rank Difference - Rel (dRR) at Document Cut-offs

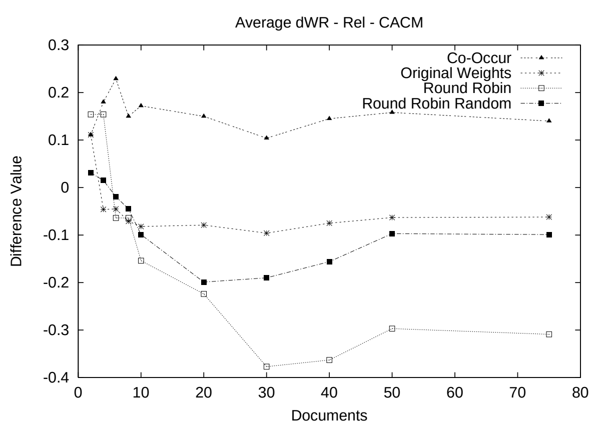


Figure 33: Average Weighted Rank Difference - Rel ($dWRR$) at Document Cut-offs

3.7 Experiments and Analysis: TREC/TIPSTER Data-set

3.7.1 Data Description and Preprocessing

The TREC/TIPSTER data is a collection designed for use in information retrieval experiments. The data originated from the Text REtrieval Conferences held annually since 1992 [46]. The main goal of the conference is to foster research in information retrieval by providing data to act as universal test-bed/benchmark and fostering communication.

Data-sets designed for information retrieval research usually consist of a collection of documents, a set of queries and a a set of relevance judgments describing which documents are relevant to each query. The TREC data used in the following experiments consists of 11 collections of documents including:

- Text Research Collection Volume 1
  - Wall Street Journal (WSJ) (1987, 1988, 1989)
  - Associated Press (AP) (1989)
  - Department of Energy abstracts (DOE)
  - Computer Select disks copyrighted by Ziff-Davis (ZF).
- Text Research Collection Volume 2
  - Wall Street Journal (WSJ) (1990, 1991, 1992)
  - Associated Press (AP) (1988)
  - Computer Select disks copyrighted by Ziff-Davis (ZF)

These collections are cleaned such that only the SGML <TEXT> segments are used as documents and all SGML tags are removed.

The TREC data also consists of a set of topics or information requests that can be used as queries and a sub-set of the documents that have been judged by humans to be relevant or not relevant to a topic. The queries used in the following sections are the SGML <TEXT> portions of topics 51-200. These topics have relevance judgments for each of the 11 collections. The difference between TREC data and the data used in Section 3.6 is that the TREC topics have relevance judgments for all of the collections of documents described above.

For a baseline, a reference collection is built by concatenating the documents from the 11 collections. The reference collection is indexed and queried as a single collection (See Section 3.1.2). The resulting result lists act as a baseline for the collection fusion strategies.

There are approximately 700,000 documents and 1 million unique terms. Each query returns the number of documents that have a degree of similarity greater than 0 up to a maximum of 32767 documents.

3.7.2 Precision and Recall Measures Graph Analysis

For the TREC collections, the Original Weights fusion approach produces the results closest to that of the reference collection (Figure 34). Eight of the eleven collections are newspaper articles (Wall Street Journal and Associated Press) with overlapping time periods. It is highly likely that the contents overlap in that the content is reflective of many topics (i.e., heterogeneous). Therefore, the collection dependent parameter (*idf*) value would be comparable between these collections and thus does not affect the ranking of the documents as a non-comparable (*idf*) would. For topics 51-200, on average the distribution of relevant documents is mainly over the WSJ and AP collections. The Original Weights approach is less effective than using the previous data (Section 3.6).

One other possible reason for the difference between the two sets of data is that the length of the result list does not correspond to the number of relevant documents, therefore skewing Co-occurrence. E.g., the result list contains a very large number documents with few query terms. This might also be shown by the RRR approach when considering precision since it is also based on the length of the result list. This may be fixed by normalizing Co-occurrence by the result list length to reduce the skew caused by large differences in result list length. But some collection fusion techniques do take into consideration the length of the result as part of the "goodness" of the collection [37] so this is left to future work.

For RR fusion at less than 5% recall, it approaches the precision of the Original Weights fusion. This seems the show that the relevant documents are distributed throughout most of 11 collections (unlike in Section 3.6) and highly ranked in those collections.

The Average Precision at Document cut-offs (Figure 37(b)) show that when considering 5-30 retrieved documents, the Original Weights approach is closest to the baseline and Co-occurrence closely follows. When considering the first 5 documents of the result list, RR approached the precision of Co-occurrence fusion leading to the belief that on average 2 out of the first 5 documents are relevant from the top ranked document of each of the first 5 collections (AP88, AP89, DOE, WSJ87, and WSJ89) is relevant.

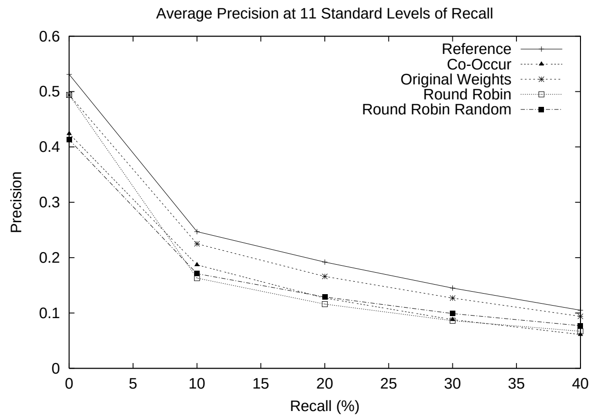


Figure 34: Average Precision at 11 Standard Levels of Recall - All TREC Queries

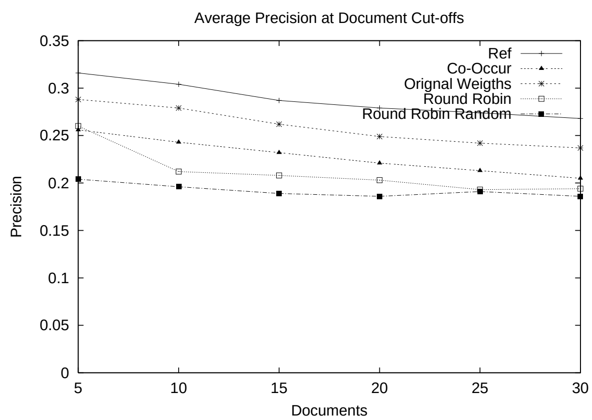


Figure 35: Average Precision at Document Cut-offs - All TREC Queries

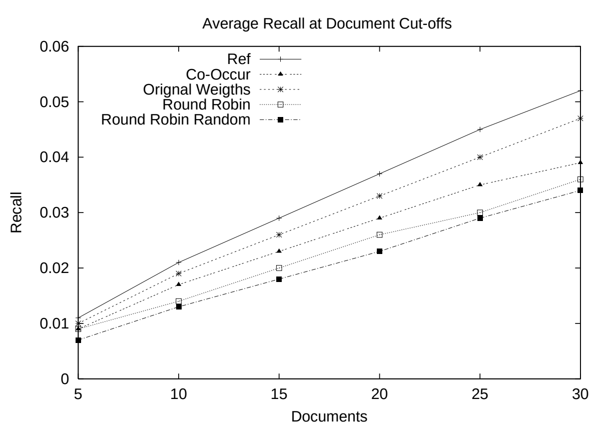


Figure 36: Average Recall at Document Cut-offs - All TREC Queries

3.7.3 Rank Difference Measures Graph Analysis

In Figure 37(a), Co-occurrence shows the least amount of positional difference of the documents (relevant or non-relevant) in a negative or positive direction. Figure 37(b) shows with the positive value of the curve that the relevant documents retrieved by Co-occurrence fusion are ranked higher than in the reference collection. Therefore, at least part of the value of the Co-occurrence curve produced by dR (Figure 37(a)) is due to these higher ranked relevant documents. When the weight of the document in the reference collection is considered, dWR (Figure 37(c)) shows that the Co-occurrence fusion tends to rank documents (relevant or non-relevant) on average more out of position in a positive or negative direction than the other approaches when compared to the reference collection. Part of this value is due to the relevant documents. $dWRR$ (Figure 37(d)) indicates that the relevant documents in the Co-occurrence result list tend to be ranked higher than in the reference result list.

On average, Co-occurrence fusion (and other methods) tend to rank the relevant documents higher than the reference collection as indicated by the positive values of Figure 37(b) and 37(d). This is not reflected in the precision/recall graphs for the following two reasons. First, using the average (e.g., Equation 18) can sometimes produce misleading statistics so the median values must also be considered since a few extreme values may influence the average. The median calculated over all queries is less than the average calculated over all queries for each rank measure. This indicates that some (i.e., less than half) of the relevant documents are being ranked much higher in Co-occurrence fusion. The medians are less than the averages but they follow the same trends and this does not fully explain the apparent contradiction between the precision/recall graphs and rank difference measure graphs. Secondly, only the relevant documents in the collection fusion technique are considered and no penalty is given if more relevant documents exist in the first n documents of the reference collection than in the first n documents of the fusion result list. Since precision is the ratio of number of relevant documents to the number of documents retrieved, the fact that some relevant documents are ranked much higher does not necessarily mean that most/all relevant documents are ranked higher thus producing a higher precision.

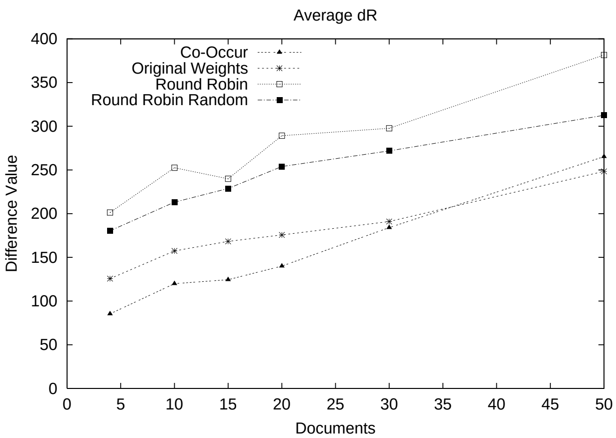
The low dRR and $dWRR$ scores for Original Weights fusion shows that the relevant documents within the first n result list documents are ranked very close to the ranks of the of the reference collection. This may help to show that on average the collection dependent parameter for the query terms are close to those of the reference collection.

4 Conclusions and Future Work

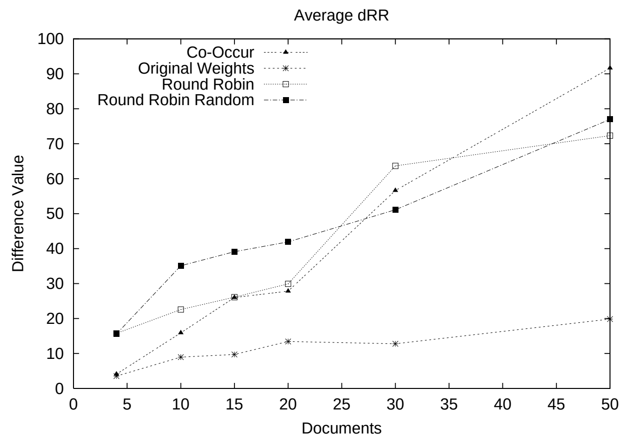
4.1 Conclusions

The first goal of the thesis was to engineer a proven information retrieval approach to efficiently and effectively execute in the constrained environment of a PDA. Towards this goal, it was found that an efficient information retrieval system (known as PalmIRA) could be built. This involved exploiting the symbiotic relationship between the PDA and the PC at synchronization time to use the PC to build the index. An optimized for speed sparse matrix was used by the PC to handle the information required to build the inverted index. The inverted index was formatted such that a small amount of storage space on the PDA is required and such that the information within the inverted index could be accessed with a small CPU and dynamic memory cost. The result is a information retrieval system executing on a PDA that executes queries quickly enough that it does not stretch the patience of a user.

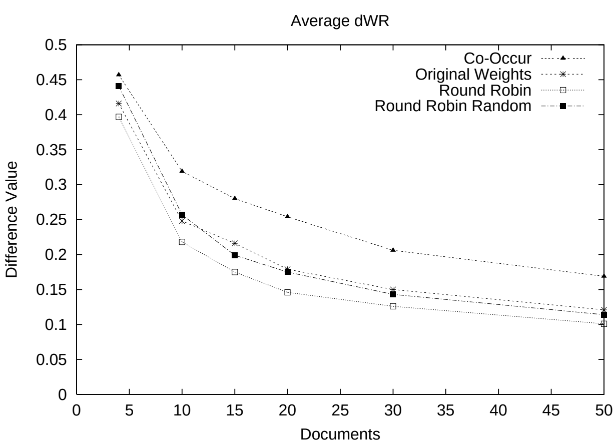
Another goal was to efficiently fuse the result lists of multiple collections taking advantage of the PDA information retrieval engine and existing within the PDA constraints. Towards this goal, Co-occurrence collection fusion is proposed. Co-occurrence fusion assigns a “goodness” to each collection based on the level of co-existence of query terms in the result list documents. Co-occurrence fusion tends to rank the relevant documents higher on average then the RR and RRR approaches for both data sources experimented with. Co-occurrence fusion shows the ability to out-perform the reference collection in at least one circumstance. Co-occurrence fusion



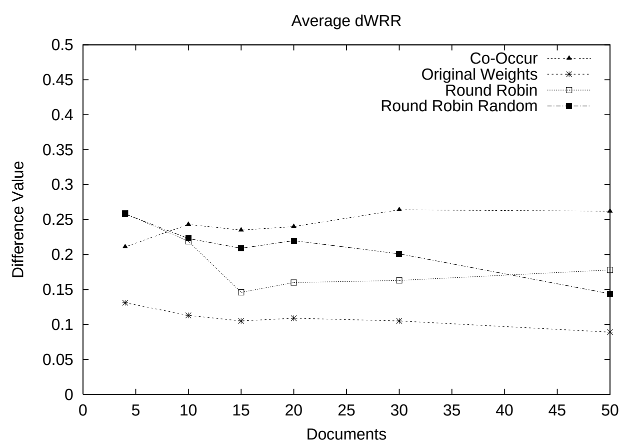
(a) Average Rank Difference (dR) at Document Cut-offs (smaller is better)



(b) Average Rank Difference - Rel (dRR) at Document Cut-offs (larger is better)



(c) Average Weighted Rank Difference (dWR) at Document Cut-offs (smaller is better)



(d) Average Weighted Rank Difference - Rel ($dWRR$) at Document Cut-offs (larger is better)

Figure 37: Rank Difference Measures - TREC

performs noticeably better when the set contains mainly homogeneous collections. Taking into consideration the previously mentioned drawbacks of the Original Weights (Raw Score) approach and that Co-occurrence fusion is more effective in the first data source but less effective in the TREC experiment, it can be concluded that Co-occurrence fusion is a better alternative than the other three fusion strategies. Co-occurrence fusion obtains better results without noticeably increasing the execution time of a query on a PDA. This is due to the simplicity of the fusion algorithm and the ability to combine parts of the algorithm with the search engine itself yielding a negligible (e.g., less than 1 sec.) increase in query time.

The last goal was to propose new techniques to more closely analyse how a collection fusion technique merges the documents to create a result list by comparing it to a reference collection. The measures (dR, dRR, dWR, $dWRR$) show by how much on average documents tend to be out of position relative to a reference collection. Fusion strategies that have a better value in these measures, considering the average and the median, tend to produce better results. The four proposed measures used along with the Precision and Recall measures help to provide insight into the effectiveness, strengths and weaknesses of a given collection fusion technique.

PalmIRA has been implemented with the capability to search PalmOS textual databases including: MemoPad, Mail, ToDo, Address, and DateBook. The results from each database can then be fused into one global result list using Co-occurrence fusion. PalmIRA is currently freely available on the internet[22].

4.2 Future Work

The opportunities for future work exist in: decreasing the storage cost of the inverted index, decreasing the synchronization data transfer cost, decreasing the retrieval time, increasing the effectiveness of Co-occurrence fusion and evaluating Co-occurrence fusion outside the context of PalmIRA.

Using a compressed inverted index may help decrease the storage cost and transfer cost of the inverted index with a trade-off of some time. PalmOS databases store the Unique ID of the last record added. When a new record is added, the previously stored unique ID is incremented and assigned to the new record. This method of assigning unique ids to the Palm records offers opportunities to compress the term posting lists composed of unique record ID and weight items. The posting lists for each term are sorted by record id. There exists the possibility to encode the difference of the unique record ids portion of the posting list. A number of index compression methods are introduced in [53]. The PDA domain offers a number of trade-offs that differ from the traditional PC based compressed inverted index.

One of the major bottlenecks of the synchronization process is the time required to transmit data via the PC - PDA link (e.g., USB). During each synchronization, the inverted index is rebuilt from scratch on the PC by downloading all the documents from the PDA and then transmitting the inverted index to the PDA. Research in the area of incremental updates of inverted files ([3]) may offer the opportunity to update the index on the PDA without completely rebuilding it.

An attempt to increase the efficiency of the retrieval algorithm may be to alter the number of term posting lists that reside in each `irWeight` record. The current inverted index implementation (Section 2.6) packs as may posting lists as possible into each PalmOS record with a maximum size of 64KB. The number of records increases as the number of posting lists stored in each PalmOS record decreases. A binary search is used to find the correct record and a sequential search is used to find the term posting list in that record. By decreasing the number of posting lists in each record, a savings in the sequential search may occur.

The effectiveness of Co-occurrence fusion may be able to be improved. In the work by Craswell et al. [10] [9], the authors describe a collection fusion technique that gives more weight to documents that contain interesting query terms closer to the beginning of the document. In Section 2.5.3, by following the linked list of nodes, it is possible to determine the first n non-stop-words in a document. It is possible to give the first n non-stop-words a higher weight than query terms that appear closer to the end of the document. Would this help

[22]http://database.cs.ualberta.ca/PalmIRA/

to increase effectiveness of the single and/or multiple collection retrieval?

Co-occurrence fusion, the collection fusion approach we propose (described in Section 3.3.4) could be used in a meta-search environment if at least one of the following holds true:

1. if the co-occurrence information is transmitted by each search engine
2. if the entire document is parsed to extract co-occurrence information
3. if the context of the query terms is transmitted by the search engines, then the co-occurrence information can be extracted from it

Finally, the Co-occurrence collection fusion idea may be able to be altered to work in a multiple collection content-based image retrieval environment. Using the idea that importance is determined by the number of features that co-occur, Co-occurrence fusion might be able to efficiently but still effectively merge the results from the multiple collections.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[2] C. Baumgarten. A probabilistic solution to the selection and fusion problem in distributed information retrieval. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 246–253, 1999.

[3] E. W. Brown, J. P. Callan, and W. B. Croft. Fast incremental indexing for full-text information retrieval. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB)*, pages 192–202, Santiago, Chille, September 1994.

[4] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *Proceedings of DEXA-92, 3rd International Conference on Database and Expert Systems Applications*, pages 78–83, 1992.

[5] J. P. Callan, Z. Lu, and W. B. Croft. Searching Distributed Collections with Inference Networks . In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–28, Seattle, Washington, 1995. ACM Press.

[6] A. Le Calve and J. Savoy. Database merging strategy based on logistic regression. *Information Processing and Management*, 36(3):341–359, 2000.

[7] Data Sets at ftp.cs.cornell.edu/pub/smart/.

[8] Stop-word list available at ftp.cs.cornell.edu/pub/smart/english.stop.

[9] N. Craswell. *Methods for distributed information retrieval*. PhD thesis, Australian National University, 2000.

[10] N. Craswell, D. Hawking, and P. B. Thistlewaite. Merging results from isolated search engines. In *10th Australasian Database Conference ACD1999*, pages 189–200, 1999.

[11] D. Dreilinger and A. E. Howe. Experiences with selecting search engines using metasearch. *ACM Transactions on Information Systems*, 15(3):195–222, 1997.

[12] M. J. Folk and B. I. Zoellick. *File Structures*. Addison-Wesley, 2^{nd} edition, 1992.

[13] L. R. Foster. *PalmOS Programming Bible*. IDG Books Worldwide Inc., 2000.

[14] E. A. Fox, M. Koushik, J. A. Shaw, R. Modin, and D. Rao. Combinating evidence from multiple searches. In *Proceedings of the First Text REtrieval Conference (TREC-1)*, pages 319–328, 1992.

[15] E. A. Fox and J. A. Shaw. Combination of multiple searches. In *Proceedings of the Second Text REtrieval Conference (TREC-2)*, pages 243–252, 1993.

[16] W. B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.

[17] J. C. French and A. L. Powell. Metrics for evaluating database selection techniques. Technical Report CS-99-19, University of Virginia, 1999.

[18] J. C. French, A. L. Powell, J. P. Callan, C. L. Viles, T. Emmitt, K. J. Prey, and Y. Mou. Comparing the performance of database selection algorithms. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 238–245, 1999.

[19] S. Gauch, G. Wang, and M. Gomez. ProFusion: Intelligent fusion from multiple, distributed search engines. *J.UCS: Journal of Universal Computer Science*, 2(9):637–649, 1996.

[20] L. Gravano, C. K. Chang, H. García-Molina, and A. Paepcke. Starts: Stanford proposal for internet meta-searching. In *Proceedings of the 1997 ACM SIGMOD Conference*, pages 207–218, 1997.

[21] L. Gravano and H. García-Molina. Generalizing GlOSS to vector-space databases and broker hierarchies. In *Proceedings of the 21st International Conference on Very Large Databases VLDB*, pages 78–89, 1995.

[22] L. Gravano and H. García-Molina. Merging ranks from heterogeneous internet sources. In *Proceedings of the 23th International Conference on Very Large Databases (VLDB)*, pages 196–205, 1997.

[23] L. Gravano, H. García-Molina, and A. Tomasic. The effectiveness of gloss for the text database discovery problem. In *Proceedings of the 1994 ACM SIGMOD Conference*, pages 126–137, 1994.

[24] J. E. Hopcraft, R. Motwani, and H. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2^{nd} edition, 2001.

[25] E. Horowitz and S Sahni. *Fundamentals of Data Structures in Pascal*. Computer Science Press, 4^{nd} edition, 1994.

[26] Palm Inc. Files and databases, 2000. http://oasis.palm.com/dev/kb/manuals/1733.cfm.

[27] Palm Inc. Palm OS memory architecture, 2000. available at http://oasis.palm.com/dev/kb/manuals/1145.cfm.

[28] L. S. Larkey, M. Connell, and J. P. Callan. Collection selection and results merging with topically organized U.S. patents and TREC data. In *Proceeding of the Ninth International Conference on Information and Knowledge Management CIKM'00*, pages 282–289, 2000.

[29] S. Lawrence and C. L. Giles. Inquirus, the NECI meta search engine. In *Seventh International World Wide Web Conference*, pages 95–105, Brisbane, Australia, 1998. Elsevier Science.

[30] K. Liu, W. Meng, C. T. Yu, and N. Rishe. Discovery of similarity computations of search engines. In *Proceeding of the Ninth International Conference on Information and Knowledge Management CIKM'00*, pages 290–297, 2000.

[31] W. Meng, C. Yu, and K. Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, (To Appear).

[32] W. Meng, C. T. Yu, and King-Lup Liu. Detection of heterogeneities in a multiple text database environment. In *Fourth International Conference on Cooperative Information System COOPIS'99*, pages 22–33, 1999.

[33] General Microsystems. Intelligentfind. www.intelligentfind.com.

[34] J. Noble and C. Weir. *Small Memory Software: Patterns for Systems with Limited Memory*. Addison Wesley, 2001.

[35] Porter Stemming Algorithm available at http://www.tartarus.org/ martin/PorterStemmer/.

[36] A. L. Powell, J. C. French, J. P. Callan, M. Connell, and C. L. Viles. The impact of database selection on distributed searching. In *Proceedings of the 23nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 232–239, 2000.

[37] Y. Rasolofo, F. Abbici, and J. Savoy. Approaches to collection selection and results merging for distributed information retrieval. In *Proceeding of the Tenth International Conference on Information and Knowledge Management CIKM'01*, pages 191–198, 2001.

[38] J. Savoy, A. Le Calve, and D. Vrajitoru. Report on the TREC-5 experiment: Data fusion and collection fusion. In *Proceedings of the Fifth Text REtrieval Conference (TREC-5)*, pages 493–502, 1996.

[39] E. Selberg and O. Etzioni. The MetaCrawler architecture for resource aggregation on the Web. *IEEE Expert*, 12(1):11–14, 1997.

[40] J. A. Shaw and E. A. Fox. Combination of multiple searches. In *Proceedings of the Third Text REtrieval Conference (TREC-3)*, pages 105–108, 1994.

[41] A. Soffer, D. Cohen, and M. Herscovice. Pirate search. http://www.haifa.il.ibm.com/projects/software/iro/PirateSearch/in

[42] A. Steidinger. Comparison of different collection fusion models in distributed information retrieval. In *DELOS Workshop on Information Seeking, Searching and Querying in Digital Libraries*, Zurich, Switzerland, December 2000.

[43] G. Towell, E. M. Voorhees, N. Kumar Gupta, and B. Johnson-Laird. Learning collection FUsion strategies for information retrieval. In *Proceedings of the Twelfth International Conference on Machine Learning ICML-1995*, pages 540–548, 1995.

[44] Y. Tzitzikas. "Democratic Data Fusion for Information Retrieval Mediators". In *ACS/IEEE International Conference on Computer Systems and Applications*, pages 530–536, June 2001.

[45] C. L. Viles and J. C. French. Dissemination of collection wide information in a distributed information retrieval system. In *Proceedings of the 18nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 12–20, 1995.

[46] E. Voorhees and D. Harman. Overview of the ninth text retrieval conference (TREC-9). In *Proceedings of the Ninth Text REtrieval Conference (TREC-9)*, pages 1–14, 2000.

[47] E. M. Voorhees. Siemens TREC-4 report: Further experiments with database merging. In *Proceedings of the Fourth Text REtrieval Conference (TREC-4)*, pages 121–130, 1995.

[48] E. M. Voorhees, N. Kumar Gupta, and B. Johnson-Laird. The collection fusion problem. In *Proceedings of the Third Text REtrieval Conference (TREC-3)*, pages 95–104, 1994.

[49] E. M. Voorhees, N. Kumar Gupta, and B. Johnson-Laird. Learning collection fusion strategies. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 172–179, 1995.

[50] E. M. Voorhees and R. M. Tong. Multiple search engines in database merging. In *Proceedings of the Second ACM International Conference on Digital Libraries*, pages 93–102, Philadelphia, Pa., 1997. ACM Press, New York.

[51] G. Wilson and J. Ostrem. *Palm OS Programmer's Companion*. PalmSource Inc., 2002. http://www.palmos.com/dev/support/docs/palmos/Companion_Front.html.

[52] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishing, 1994.

[53] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishing, 2^{nd} edition, 1999.

[54] R. R. Yager and A. Rybalov. On the fusion of documents from multiple collection information retrieval systems. *Journal of the American Society for Information Science*, 49(13):1177–1184, 1998.

[55] C. T. Yu, W. Meng, K. Liu, W. Wu, and N. Rishe. Efficient and effective metasearch for a large number of text databases. In *Proceeding of the Eighth International Conference on Information and Knowledge Management CIKM'99*, pages 217–224, 1999.

[56] B. Yuwono and D. L. Lee. Server ranking for distributed text retrieval systems on the internet. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications DASFAA*, pages 41–50, 1997.

[57] X. Zhu and S. Gauch. Incorporating quality metrics in centralize/distributed information retrieval on the world wide web. In *Proceedings of the 23nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 288–295, 2000.