TIPSTER PHASE I FINAL REPORT

Bill Caid, Stephen Gallant, Joel Carleton, David Sudbeck

HNC, Inc.
5501 Oberlin Drive
San Diego, CA 92121
(619) 546-8877

1. DESCRIPTION OF SYSTEM USED IN 24-MONTH EVALUATION

1.0 Technical Approach

Overview: During Phase I of the TIPSTER program, HNC developed a unique approach to machine learning of similarity of meaning. This approach, embodied in a system called "MatchPlus", exploits this learned similarity of meaning for concept-based text retrieval, routing and visualization of textual information. MatchPlus uses an information representation scheme called "context vectors" to encode similarity of usage. Key attributes of the context vector approach are as follows:

- Words, documents, and queries are represented by context vectors. A context vector encodes a representation of the meaning of a word, query, or document as a high-dimension, fixed length, real vector.

- Elements of the context vector, called "features", define a "meaning space" used for classification and retrieval of documents. The selection of features is automatically determined by the system and defines the "frame of reference" for the formation of the context vectors. The *direction* of the context vector provides an encoding of the meaning of the associated text.

- A context vector is assigned to each unique word and phrase in the system lexicon during system initialization. Word and phrase context vectors are then *learned* from free text based on the context of their occurrences in the training text using a constrained self organization technique. This learning is *fully automatic*: no external knowledge bases, dictionaries or thesauri are needed for learning the vectors.

- Once trained, vectors for words with similar usage in the training text (for example *NASA* and *Space Shuttle*) will point in approximately the same direction. Vectors for words with unrelated usage (like *NASA* and *peanut*) will be approximately orthogonal. The degree of similarity of usage is analogous to similarity of direction.

- Word and phrase context vectors are used to form context vectors for documents and queries. Document and query context vectors are computed as the weighted sum of the context vectors associated with stems and phrases in the document or query. Long documents may be split and represented by several vectors. Document context vectors are normalized to prevent long documents from being favored over short documents. When complete, each document's context vector contains an estimate of the meaning of the document relative to the current feature set.

The context vector representation scheme offers a number of advantages. Documents can be classified (clustered), retrieved, and routed by computing conventional geometric distances. This is because documents that have similar context vectors (i.e., point in roughly the same direction) will have similar information content. Simple Euclidean distance measurements between sets of context vectors are a measure of similarity of meaning. Additionally, since all tokens in the system (words, queries, and documents) have the same vector representation, these comparisons of similarity can be carried out at any level of detail: word, sentence, paragraph, document, or set of documents. This provides a mechanism for automatically finding high relevance "hot spots" within a document. The hot spot detection feature is referred to as "highlighting". Highlighting can be performed at both the paragraph-within-a-document and word-within-a-paragraph levels.

Document retrieval is performed by simply finding documents having context vectors that are close to the query context vector. A document's relevance to the query is determined by comparing the dot product of the query context vector with that of the document's context vector. A large dot product implies strong relevance to the query. The documents in the retrieval list are ranked according to the magnitude of the dot products.

Routing is performed in a similar fashion. The context vector for each incoming document is compared to the context vector of each routing query, and if the dot product exceeds the similarity threshold, the document is routed to the user(s) associated with that query.

Context vectors can be used alone or in combination with a Boolean filter, thus the name "*MatchPlus*". Using a compound query approach, *MatchPlus* can augment and enhance existing Boolean search systems because the most relevant documents are at the top of the retrieval list. In this mode, *MatchPlus* will provide relevance-ranked documents within the set of documents that meet the Boolean filter criterion, thus helping to reduce information overload common to Boolean-based systems.

Features: The key features of the *MatchPlus* approach from a user's perspective are as follows:

- Documents are retrieved based on meaning rather than word match and are ranked based on relevance to the query. *MatchPlus* will find related documents even if the key word is not present in those documents.
- The unified representation approach allows the entire text of documents to be used as a search query.
- Adaptive learning by *steering* (refining) the query context vector using relevance feedback can quickly improve the quality of a query.
- The system learns similarity of meaning from usage and can track the evolving meaning of terms.
- Easy-to-use graphical query interface is based on X Windows/MOTIF.

Architecture: Key attributes of the current *MatchPlus* system architecture are:

- Allows operation on very large corpora (in excess of 10^7 documents) in networked and/or distributed processing environments. Allows heterogeneous corpus contents and formats.
- Supports Boolean, context, and relevance feedback query modes, and can be integrated with existing "key word" match systems to provide performance improvements and relevance rankings.
- Extendible and hardware-independent architecture that provides easy-to-use graphical user interface.

Summary of Conclusions: The research conducted during Phase I of the TIPSTER program has resulted in the following conclusions:

- The context vector approach to text representation is viable.
- *Matchplus*, which exploits the concept of "similarity of use", is fully automatic. Demonstration of *MatchPlus's* ability to learn "similarity of use" has been demonstrated in the legal, medical and scientific domain as well as in foreign languages such as Spanish and Japanese.
- *MatchPlus's* vector representation provides a detailed explanation capability that can be utilized to answer basic informational questions such as:
  1. Why was this document retrieved?
  2. What section of the document is most relevant?
  3. What is the commonalty with this group of documents?
  4. What is the relationship of this concept with the rest of the text and what, if any, are the senses in which it is used?
  5. Has this concept changed or has it been used in a different context over time?

Next Steps: To fully exploit the context vector representation further research is required in these general areas:

- Use of context vectors in non-text domains such as images.
- Refinement of the clustering technique to improve retrieval and routing performance and speed as well as automated subject indexing, visualization, and word sense disambiguation.
- Continued participation in TREC evaluations and continued in-house retrieval and routing testing in an

effort to understand how *MatchPlus's* concept retrieval system can be improved for information retrieval and routing.

Related Research: The *MatchPlus* context vector approach appears to be directly extensible to text in all languages and domains. Concepts for further extending the approach to provide content-addressable access for image, video, sound, and sensor data have also been developed. Rome Labs (USAF) funded further development of the HNC context vector-based approach as part of the Automated Librarian SBIR contract. Phase II of this SBIR, called Image Content Addressable Retrieval System (*ICARS*) will extend the context vector approach to the image domain. The context vector representation can also provide a vehicle for visualization of the information content of free text. Any word or set of words has a vector representation. As such, graphical representations of information content can be achieved based on the context vector representation. HNC is currently developing a text information visualization system as part of non-TIPSTER, ORD-sponsored activities. When complete, this system, called *DOCUVERSE,* will have the capability to perform icon-based browsing of text as well as graphically displayed directed queries.

Current Deployment Status: Preliminary prototype test versions of *MatchPlus* have been in use at Wright-Patterson AFB (FASTC) for over one year. *MatchPlus* is also being evaluated at USAIA (FSTC) and other agencies for analysis of classified data. HNC has received a contract to develop a prototype *MatchPlus* system for a large legal publishing firm. Current plans call for the evaluation of *MatchPlus* as part of the US Patent and Trademark Office Automated Patent System upgrade effort. The current implementation of the *MatchPlus*/TIPSTER system is based on the C language and uses X Windows/MOTIF as the graphical user interface.

1.1 Processing Flow and Key Modules

1.1.1 *MatchPlus* Functional Overview

HNC's approach to the TIPSTER text retrieval and routing problem is called "*MatchPlus*". *MatchPlus* is a neural network-based approach to the problem of free text retrieval, classification and routing. The key technical feature of *MatchPlus* is the representation of words, documents, and queries by "context vectors". A context vector encodes a representation of the meaning of a word, query or document as a high-dimension, fixed length, real vector. Context vectors for new words are "learned" from the text corpus using a process called "bootstrapping". Using the bootstrapping technique, *MatchPlus* learns the usage of words and meanings of documents that contain those words by using only the text corpus.

Once bootstrapping is complete, the resulting word context vectors are then used to form a "document context vector" for each document in the corpus. Document context vectors are computed as the weighted sum of the context vectors associated with stems and word groups in the document.

Documents can be classified, retrieved and routed by computing conventional geometric distances. Simple Euclidean distance between sets of context vectors are a measure of similarity of meaning. Thus, document retrieval is performed by simply finding documents that are "close" to the query context vector. Document relevance to the query is assessed via a dot product of the query context vector with each document context vector. Large dot products imply strong relevance to the query. The retrieval list is ranked according to magnitude of the dot product.

Routing is performed in a similar fashion. Context vectors are computed for incoming documents and compared to each routing query. If the dot product exceeds the similarity threshold, the document is routed to the user(s) associated with that query.

Context vector similarity assessment techniques can be used alone or in combination with a "Boolean filter", thus the name "*MatchPlus*". Using a compound query approach, *MatchPlus* can augment and enhance existing Boolean search systems since the most relevant documents are at the top of the retrieval list. In this mode, *MatchPlus* will provide relevance-ranked documents within the set of documents that meet the Boolean filter criterion, thus helping to reduce "information overload" common to Boolean-based systems.

The key attributes of the *MatchPlus* system architecture are:

- Design accommodates very large corpus (more than 1 million documents).
- Architecture supports operation in a distributed CPU environment.

- Provisions for heterogeneous format corpus built into system.
- X Windows/MOTIF GUI under Sun/OS.

The sections below provide a more in-depth examination of the *MatchPlus* system. This discussion will "walk-through" the operation of the system in each high level mode and will trace the processing that occurs to a query when a retrieval is performed.

1.1.2 *MatchPlus* Operating Modes

The *MatchPlus* system operates in three main modes:

- System Generation
- Retrieval
- Routing

The system generation mode provides initialization and maintenance capabilities and is used to "teach" the system the meanings of words given a training corpus of text. Once the meanings of the stem words have been derived, these are used to compute document context vectors.

Retrieval mode allows the user to enter queries to the *MatchPlus* system and find documents that meet the query specification(s).

Routing mode, as the name implies, provided routing services for incoming documents. In a sense, routing is the inverse of retrieval. In retrieval, there is one query and many documents that might apply to the query. In routing, there is one document and many routing queries that may apply to the document.

In order to provide a better basis for understanding the operation of the *MatchPlus* system, the key data structures used by the *MatchPlus* system will be described prior to the detailed explanation of the operating modes.

1.1.3 Key Data Structures and Control Files

An object oriented design was utilized during the design of the *MatchPlus* architecture. Therefore, to gain a better understanding of the operation of the system, a description of these data objects is required. *MatchPlus* makes use of several "key" data structures. These structures are key in the sense that many software components use the information contained within these structures. These data structures are described below. In an effort to standardize and reuse software components, "access routines" and "standard packages" have been developed. These packages provide a common format interface to widely used facilities such as hash tables, linked lists, etc. Since these data structures are conventional data structures and are not TIPSTER specific, they will not be discussed.

1.1.3.1 Corpus Description Data Structure

The corpus description (CD) data structure, as it's name implies, carries information about where documents that comprise the corpus are located, how they are formatted, etc. The CD is used to compress the amount of information that is required to completely describe the corpus of text to the system. In general, there are no restrictions about the number of documents per file, the number of files per directory or the number of directories that comprise the corpus. The only restriction is that one document cannot span a file. A schematic of the CD is shown in Figures 1A and 1B. As can be seen from the figure, the CD contains a number of sub-objects. These sub-objects are:

- The root object is of type "tCorpus" and contains a series of base addresses and lengths of each of the other object arrays. Additionally, it also contains the number of elements (dimension) of the context vectors used by the system.
- The array of objects of type "tDocDescr" contain information about each specific document in the corpus. This information includes the (internal) document ID, start FSEEK address, length in bytes, a code for the file that contains the document, a status and a pointer to the document context vector. There is one object of type tDocDescr for each document in the corpus.
- The array of objects of type "tFileDescr" contains information about the files that comprise the corpus. There is one object for each file. This object contains information such as which host in a network contains the file, a code for the fully qualified Unix path for the file, a pointer to the file name, a tag to indicate how the file will be deformatted and pointers to the deformatting functions and/or deformatting script file name. Using this scheme, all documents in a file must have the same format. However, each file in the corpus can have a different format if desired.

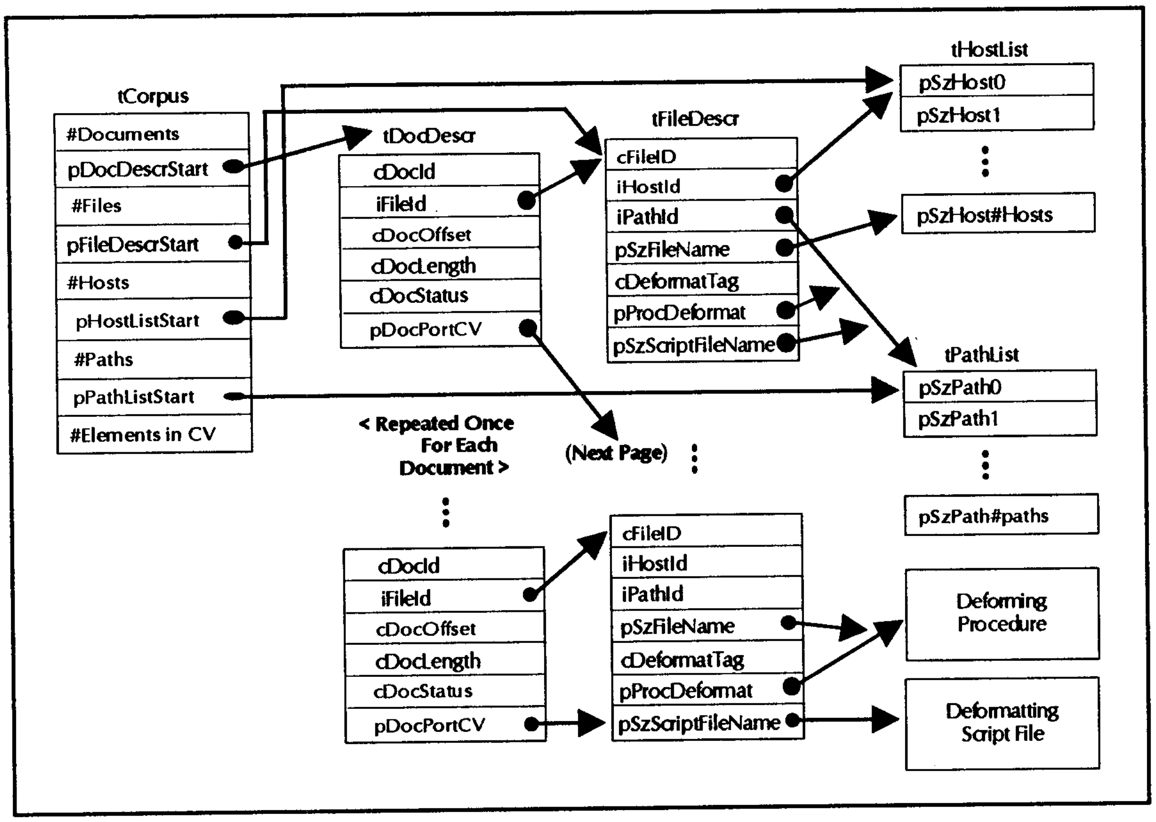

Figure 1A. Corpus Description Data Structure

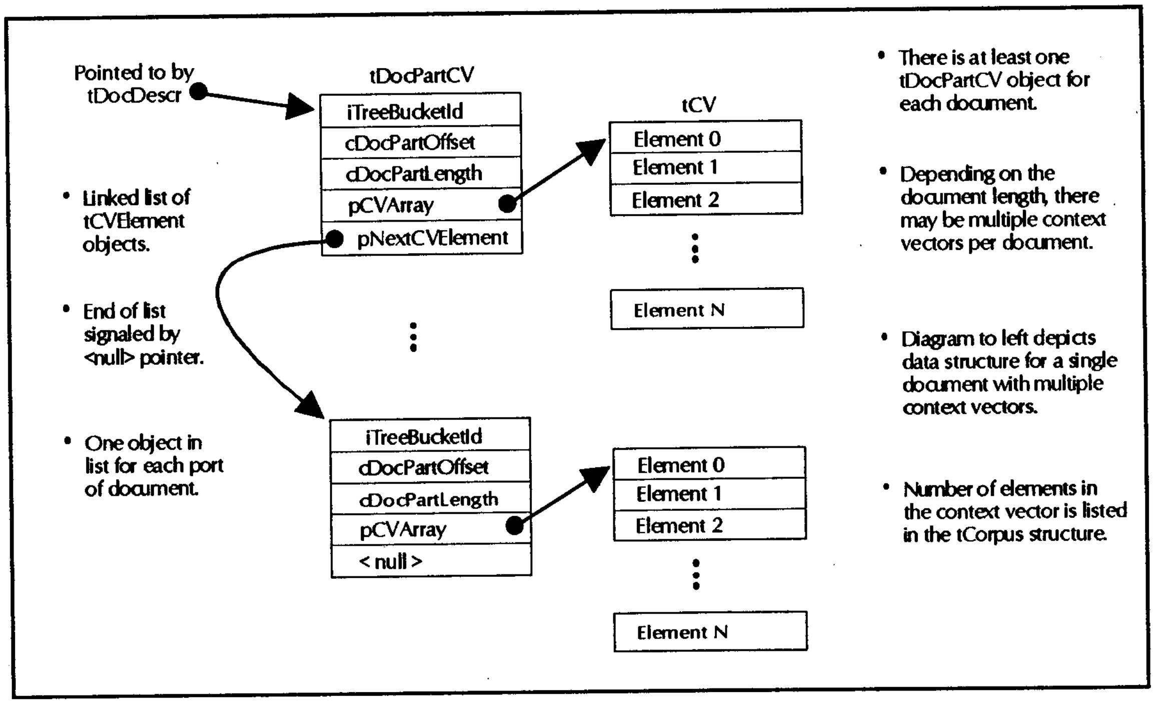
Pointed to by tDocDescr
tDocPartCV
iTreeBucketId
cDocPartOffset
cDocPartLength
pCVArray
pNextCVElement
tCV
Element 0
Element 1
Element 2
Element N
• Linked list of tCVElement objects.
• End of list signaled by <null> pointer.
• One object in list for each part of document.
< null >
• There is at least one tDocPartCV object for each document.
• Depending on the document length, there may be multiple context vectors per document.
• Diagram to left depicts data structure for a single document with multiple context vectors.
• Number of elements in the context vector is listed in the tCorpus structure.

Figure 1B. Corpus Description Data Structure (cont.).

- The array of objects of type "tHostList" contains a list of character strings with the host ID. This feature would only be required in a distributed implementation.

- The "tPathList" object is an array of character strings that contain the paths to each data file in the corpus. Using this approach all files could reside in the same directory. Alternatively, they could each reside in a separate directory or some intermediate combination.

1.1.3.2 Stem Information Data Structure

The StemInfo (SI) data structure contains information about the stem words that comprise the corpus and is shown in Figure 2. Access through the SI data structure is via the HNC-common "hashing package". In addition to a standard hashing function, this package provides a key capability: caching. The hashing package has a provision to allow the hashed objects to be either memory resident or "cached". If the objects are to be cached, a compile time parameter allows the user to determine how many objects will be kept in memory. This provision allows the application to control the amount of virtual memory used during execution. This is a key capability, since Sun/OS has a fixed upper bound of 500 Mb for virtual memory usage for a single task. Without the caching capability, *MatchPlus* would consume well over a gigabyte in virtual memory during the generation of the system!

Like the CD data structure, SI is composed of a series of sub-objects. These sub-objects are:

- The root object is of type "tStemInfoDescr" and contains a pointer to the hash description object and the total number of unique stems found in the corpus.

- The "tHashDescr" object contains information about the hashing function that provides access to the individual stems. This information includes a pointer to the hash table, page sizes and status flags, a pointer to the hashing key comparison function and paging control function and a pointer to the hash table itself.

- The hash table is of type "tStemHashTable" and consists of an array of pointers to the stem description objects.

- The information about each stem is contained in the "tStemDescr" object and contains a pointer to the stem string, the stem status, a count of the number of documents that contain the stem (used for normalization), a pointer to the context vector for this stem and a pointer to the inverted index entries for this stem.

- The stem context vector, like all context vectors within *MatchPlus*, is of type "tCV". This is an array of floating point vector elements that encode the "meaning" of the stem as learned from the training corpus.

- The inverted index entries for each stem are contained in a linked list. The objects in the linked list are of type "tDocList" and are managed by the HNC-standard linked list package. The objects in the linked list contain the document ID and number of occurrences of the stem in that document. The document ID is used as an index in the CD data structure.

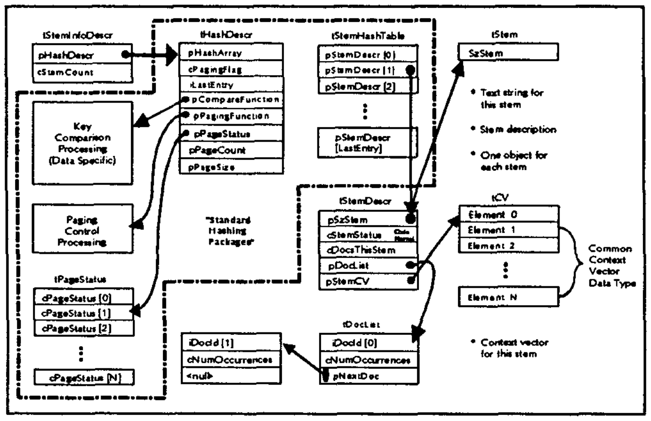

Figure 2. StemInfo Data Structure

```
#               /* @(#)system_op_info.file    1.4  6/11/92 */

# Note: the following is my current thought, if you have any Qs
# please let me know, we can discuses about it
# 6/1/92:PQ
# This file will be updated by calling update_system_op_info_file()
# previous_last_doc_num and total_doc_num will be updated by calling
# update_system_op_info_file in the end od Func. InitializeCorpusDescr

# Retrieval:    Start always FIRST_DOC_INX_HIDDEN
#               End   always last doc num
retrieval                       # process key
1                               # Previous last doc number,during retrieval,it always 1
1000                            # Last doc number in current corpus

#Generate DocList operation range definition
# Start doc number and end doc number will be updated
# by calling InitializeCorpusDescr when system_op is (1) or (2)
# (1) regenerate_system:        Start = 1, End = last_doc_num
# (2) add_more_docs   :         Start = previous_last_doc_num, End = last_doc_num
# (3) within_this_range:        Start = user_define, End = user_define
inverted_index                  # process key
regenerate_system               # regenerate_system,add_more_docs or within_this_range
1                               # Start of doc number
1000                            # End of doc number
```

Figure 3. System.Op.Info.File

1.1.3.3 System Operational Info File

The system operational information file (SOIF), shown in Figure 3, is not a data structure, but a simple flat ASCII file that controls system generation and maintenance operations. This file allows control of the range of operations performed by the system expressed using the internal document ID. Specifically, this file allows control of the range of documents used for:

- Retrieval operations
- Formation of the corpus description data structure
- Generation of stem hash table entries and the associated stem information file
- Bootstrapping
- Generation of document context vectors

In general, the only difference between a "new" system generation and an incremental update (maintenance) is the range of documents involved in the operation.

1.1.4 System Generation Overview

Generation of the *MatchPlus* system consists of four steps. If a "ground-zero" system build is being performed, all steps must be performed. For incremental builds and maintenance, step 2 may be omitted or performed on a subset of the whole corpus. These four steps are detailed below.

- Initialize Primary Tables: This step initializes tables that are used as part of other operations. Specifically, this step performs the following actions:

    - Allocates the Corpus Description data structure. Using control information provided by the SOIF and other files, this step writes the document start and length information into the Corpus Description structures.

- Reads the stop word list file and generates and saves the stop word hash table.
- Reads the stemming exception list file and generates and saves the exception word hash table.
- Reads the stem word group file and generates and saves the stem word group hash table (not yet implemented).
- Reads the core stem list and loads this data into the StemInfo data structure.

• Form Secondary Tables: This step derives information from the training corpus as specified by the System.Op.Info.File. Actions performed in this step are:

  - Load data into the StemInfo data structure. This operation consists of building the stem list and associated hash table, sorting the stem occurrence information and forming the inverted index. The resulting information in the StemInfo structure is saved to disk.
  - Pre-bootstrap (if double bootstrapping is specified) or loading core stem context vectors into the StemInfo structure.
  - Bootstrapping. This operation will make two complete passes through the corpus to determine the stem context vector for non-core words based on their usage in the training corpus. The resulting learned stem context vectors are stored in the StemInfo structure and then the completed StemInfo data structure is saved to disk.

• Compute Document Context Vectors: This operation forms context vectors for all documents in the corpus. Unlike the bootstrap operation where a subset of the corpus can be used for training and stem context vector generation, all documents that are to be retrieved must have a context vector calculated. Context vectors for documents are calculated from the context vectors of the stems that comprise the document. The resulting vector is normalized such that the system does not "favor" long documents over short ones.

• Generate Cluster Tree: This operation forms the document context vector cluster tree. This capability will result in a centroid-consistent cluster tree that is used to reduce document retrieval times.

1.1.5 Document Retrieval

Document retrieval is implemented in two sets of processing steps and is shown in Figure 4. The first set of steps is initialization and consists of the following:

• StemInfo data structure is restored from disk (if not already memory resident).

• Stop word, exception word and word group hash tables are restored from disk (if not already memory resident).

• Cluster tree is restored from disk (if not already memory resident).

The second set of processing steps is event driven operation and consists of the following steps:

• Query processing is performed. The user may specify a topic to be automatically processed or may invoke interactive mode such that queries are entered via the X Windows GUI.

• Retrieval query processing module processes and parses the query into Boolean and context components.

• Query components are saved to disk for possible later use.

• A query context vector is formed from the query components. Boolean and context terms are treated equally in this step.

• For the specified Boolean query terms, the inverted index is used to determine which documents contain these terms. A list of documents that meet the Boolean filter is formed.

• The query context vector is used in conjunction with the cluster tree to find context-relevant documents via the dot product operation. A ranked list is formed.

• The Boolean list and the context-relevant lists are merged and an aggregate list is formed. This list is sorted.

• When the user selects a document for display on the GUI, the document is deformatted (if needed) and displayed on the screen.

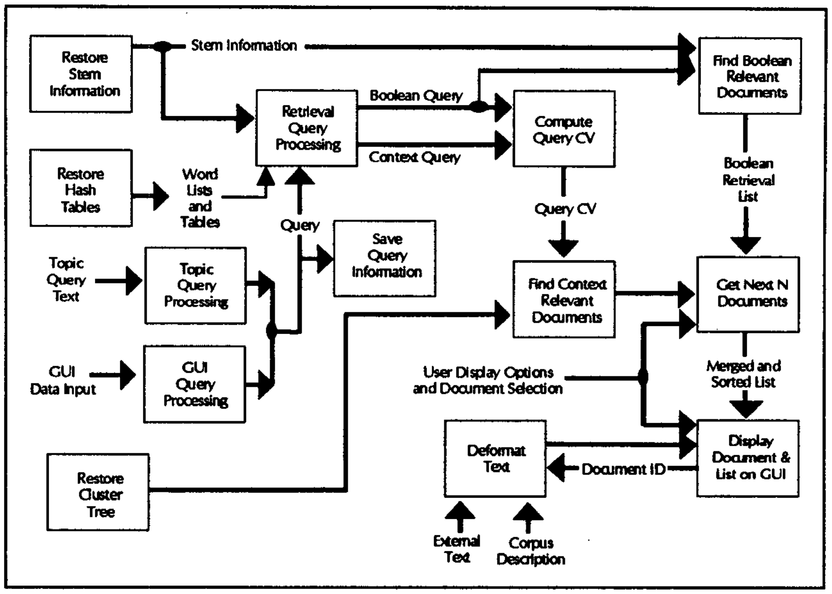

Figure 4. Document Retrieval

1.1.6 Document Routing

Document routing is, in a sense, the inverse of retrieval. In retrieval, there is one query and may documents. In routing, there are many queries and one document. The process flow for document routing is shown in Figure 5. This flow shares many processing components with the retrieval flow. Routing is broken down into three main operations:

- Initialization: The initialization operation restores all data structures and tables as needed. This includes Corpus Description, StemInfo, and the stop word, exception word and word group hash tables.

- Query Processing: This operation is divided into GUI and topic query processing. If interactive mode is selected, the GUI is used to assemble a routing query. If topic mode is selected, the specified topic text is read. Queries used for document retrieval can also be restored and used for routing. For the selected mode, the query is parsed into Boolean and context terms and a similarity threshold, to be used for route/no-route decisions. A routing context vector is then computed using both the Boolean and context terms. This query processing step is performed for each routing query to be used in the system. The routing information (context vector, Boolean terms and match threshold) is stored in a table along with the user ID of the route.

- Document Routing: For each document to be routed, the following steps are performed:

  - If required, the document is deformatted into the *MatchPlus*-internal format.

  - Document preprocessing is performed: stop words are removed, stemming exceptions are identified and stems are produced.

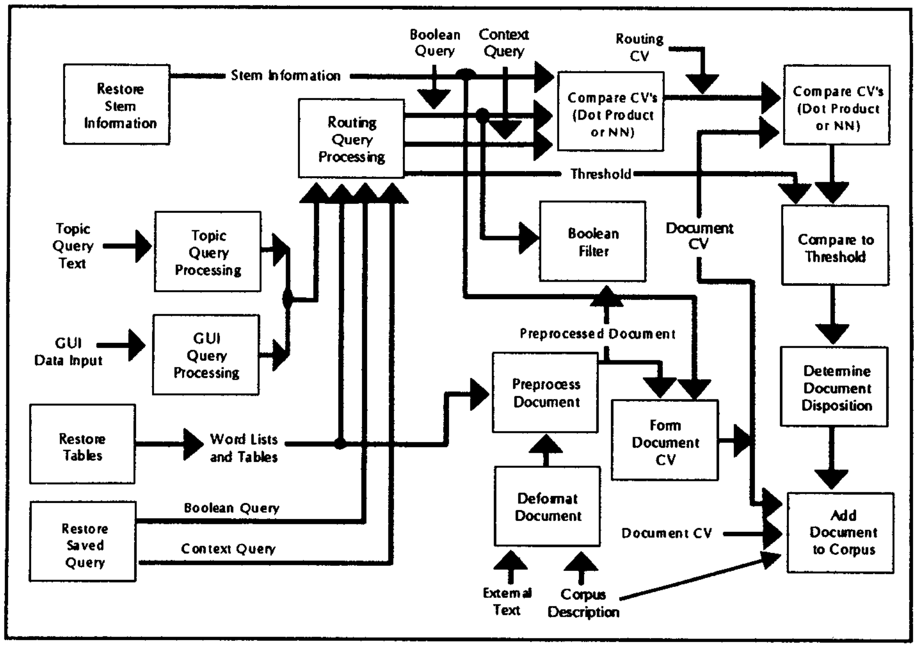

Figure 5. Document Routing

- Additionally, a context vector for the preprocessed document is computed.

- For each routing query in the system:

  * The preprocessed document is passed through the Boolean filter operation to see if the filter criteria is met for this routing query.

  * The dot product of the routing query context vector and the document context vector is computed.

  * The Boolean filter is passed or the dot product threshold is exceeded, then the document is routed to the user associated with this routing query.

- If specified, the document is added to the corpus, Corpus Description and StemInfo data structures.

1.2 System Throughput

Building the *MatchPlus* system can be broken down into three main areas: inverted index generation, learning of stem context vectors, and generation of document context vectors. The approximate times to build the system as well as the retrieval time are given in Table 1. The hardware is a Sun Sparc 10 with 512 megabytes of RAM.

BUILD STAGE	TIME
Inverted Index Generation	~10,000 documets an hour
Stem Context Vector Learning	~2000 documents an hour
Generation of Document Vectors	~25,000 documents an hour
Retrieval	~3000 documents per CPU second

Table 1. System Build Times

It should be noted that the stem context vector learning does not need to be applied to the entire data set. For the TIPSTER 24 month evaluation the system was only trained (i.e. learning of stem context vectors) on 80,000 documents (approximately 40 hours).

For the TIPSTER tests the use of a boolean filter greatly increased the retrieval speed. Instead of ranking the entire database using the document context vectors a broad boolean filter would return a small subset (50,000-100,000 documents) which would then be ranked by context vector. Another approach the *MatchPlus* system utilizes to speed up retrievals is the use of clustering (refer to section 4.0 for a discussion of clustering). HNC has been successful in speeding up retrievals by a factor of 20 with no degradation in performance by utilizing document clusters.

1.3 Key Innovations

During Phase I of the TIPSTER project the HNC team, through its research and experimentation, has formed some highly significant conclusions. These conclusions not only encompass the main' objective of TIPSTER phase I, text retrieval and routing, but address much larger issues involving the processing of information. The major conclusions and innovations are as follows:

- **Context vector approach to text information representation is viable.** The assertion that a vector space model can be utilized to retrieve from large (gigabyte) heterogeneous databases has been proven. In addition the use of initial hand entered context vectors is unnecessary and in fact does NOT perform as well as a fully automatic approach. Unforeseen uses for context vectors have also been discovered. These include document clustering, word sense disambiguation, visualization, image retrieval, and automated hyperlinks in a hypertext environment.

- **Fully automated learning of similarity of usage has been demonstrated.** Even very low frequency words (e.g. last names) are trained such that they are associated with logical "concepts". This is done automatically with no thesauri or knowledge base, only the relationships as they occur in the text are used. Beyond the obvious benefit for retrieving documents and investigating word relationships, other possible applications of "similarity of usage" is in the analysis of how particular concepts evolve and change over time. For example a person's name may always be mentioned with a certain company, but if new text were added in which the name appeared with a different company this change in relationship would trigger an alarm for an analyst. Refer to Figures 6 and 7 for examples of automated learning for specific words (referred to as "stem trees"). The text to the right of the bar graph is an example of the context in which a specific word is found in the text.
- **Context Vector representation is language independent.** *MatchPlus* uses no language dependent knowledge basis (e.g. WordNet, thesauri). Small (10-20 meg) systems have been built both in Spanish and Kanji and the "learning" of similarity of use witnessed in English carried over to the other languages.

- **Conventional neural networks can be effectively applied to context vector operations.** In a routing environment basic networks can be trained and have shown to give a 20 % improvement over adhoc methods. Additionally, clustering algorithms have been utilized to perform automated word sense disambiguation and document clustering. Automated word sense disambiguation with context vectors has benefits beyond information retrieval such as machine translation. Document clustering also has further applications in visualization and automated subject indexing.

- **Constrained learning law (bootstrapping) promotes learning stability.** Constraining the geometry of the vector space can produce a stable solution independent of the number of training passes through the corpus.

2. SYSTEM GOALS

The primary goal of the HNC's *MatchPlus*/TIPSTER effort was to apply advanced adaptive and neural network techniques to improve the state of the art in text retrieval and routing technology. More specifically HNC sought to design a vector space model for text that encodes a representation of similarity of meaning. To reach this goal HNC developed an adaptive technique to learn similarity of meaning for words using only free text as examples. Once the similarity of meaning was encoded in a vector representation the *MatchPlus* system needed to exploit the learned relationships to provide improved precision and recall over current techniques. In addition it was HNC's intent to apply neural network learning algorithms to automatically improve queries based on user feedback.

The *MatchPlus*/TIPSTER effort had a number of secondary goals. These included the development of a generic representation for word similarity that could be exploited for other uses (e.g. visualization, key word extortion), use of a minimum of human knowledge for system generation and retrieval (e.g. dictionaries, thesauri), and the creation of a basis for multi-media and multi-language capabilities.

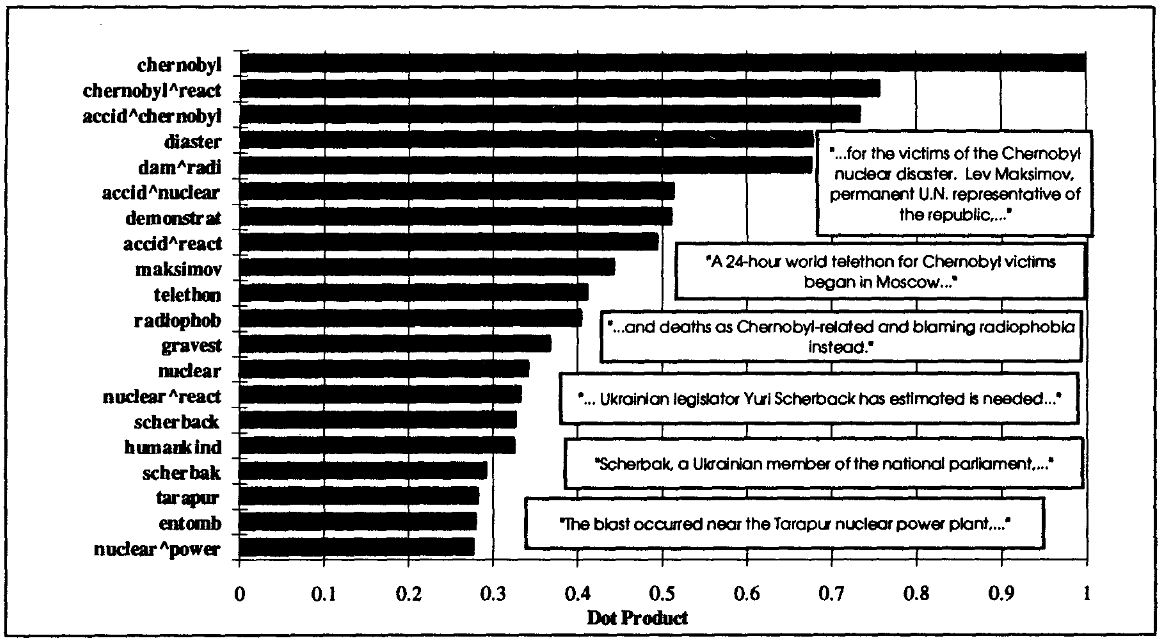

Figure 6. "Stem Tree" for Chernobyl

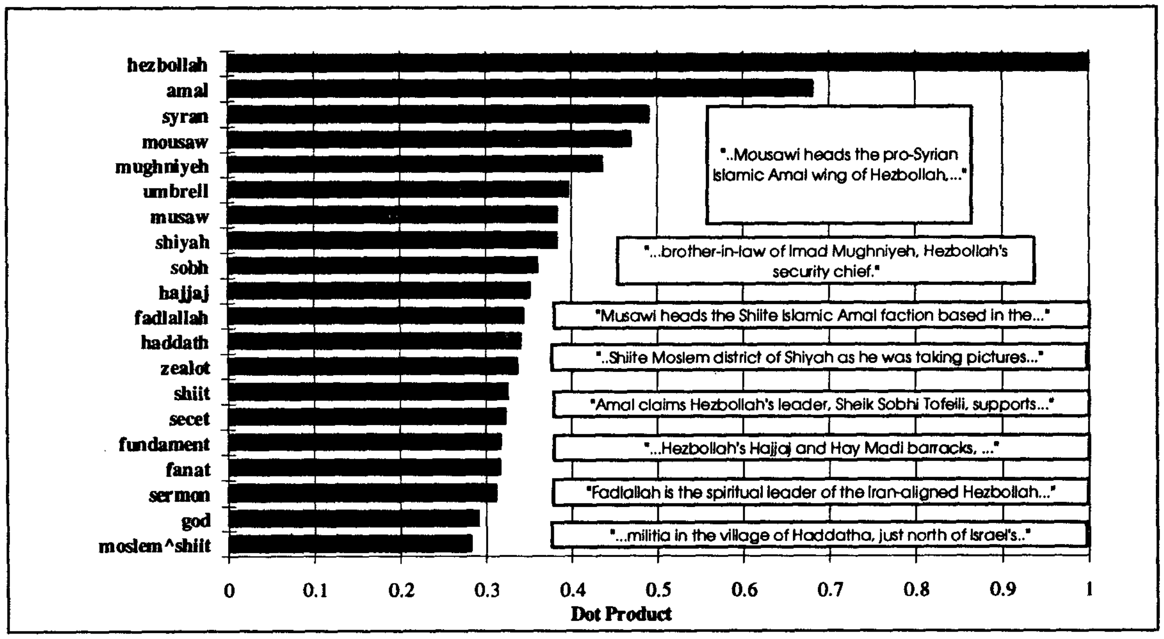

Figure 7. "Stem Tree" for *Hezbollah*

3. EVOLUTION OF THE SYSTEM

At the start of the 2 year TIPSTER project HNC had nothing more than a preliminary prototype that embodied the conceptual idea for the use of context vectors for information retrieval. The original prototype was built on less than 500 documents. Throughout the TIPSTER program the original *MatchPlus* system underwent a significant amount of changes and enhancements. The goals of these changes and enhancements are explained in section 2. The evolution of the system was driven by both the performance on the TIPSTER data (i.e. recall and precision numbers) as well as a software design that enabled the processing of gigabytes of text such as the TIPSTER collection. Some of the major software design strategies implemented for processing large databases are listed as follows:

- Caching algorithms to prevent the overflow of virtual memory.
- File splitting for files that span more than a single disk
- Heap data structures to quickly get the top "N" documents from a collection of "M" documents given that "N" is much smaller than "M".
- Internal memory management to prevent excessive memory allocation and memory freeing.
- Use of clustering to prevent searches on the entire database.
- Re-ordering of document vectors by cluster to reduce the number of disk seeks thus, speeding up retrieval.

Changes and enhancements not related to the size of the data, but rather the goal of learning similarity of use in a vector representation and exploiting that representation were made in various areas. Those areas and the enhancements or changes made are as follows (for detailed results of the enhancements refer to section 4):

1) Preprocessing

- Addition of word phrases.
- Use of a stemming exception list prevents words such as "army" from being stemmed to "arm")

2) Bootstrapping

- Use of a fully automated learning algorithm (i.e. the use of the manually entered core stem context vectors from the original system was replaced by a fully automated learning technique.
- Use of a batch update for the training of stem context vectors.
- Ability to transplant the learned relationships from one corpora to another and suffer no degradation in performance.
- Ability to automatically disambiguate words given only free text examples.
- Improvement over original stem context vector learning law that provides adjustable constraints and comes to stability.

3) Generation of Document Context Vectors

- Ability to cluster documents.

4) Query Processing

- Use of broad boolean filters
- Automated parsing of the TIPSTER topics which provides optimum performance.
- Relevance feedback to improve an original adhoc query.

5) Routing

- Use of perceptron learning to create a routing query given a set of relevance judgments

6) Graphical User Interface

- Word, paragraph and document highlighter to quickly get to the most relevant information.
- Simplistic method for user relevance feedback
- Automated subject index
- Examination of learned relationships (i.e. stem trees)
- Document headline display to quickly scan a set of retrievals

- Ability to process natural language queries.

4. EXPERIMENTS & PERFORMANCE

A number of research areas were identified as possible candidates for improvements in *MatchPlus* performance. These areas covered a wide range of processes including preprocessing, learning and query formation. HNC utilized the official retrieval results from the document detection runs sent in for the 12 month, 18 month and 24 month evaluations. These evaluations consisted of a set of 150 topics and 3 CD-ROMs with about 1 gigabyte of document data each. The document data consisted of the Wall Street Journal, San Jose Mercury News, AP Newswire, Information from Computer Selected disks, Federal Register, U.S. Patents and short abstracts from the Department of Energy. These sources had a varied length, a varied writing style, a varied level of editing and a varied vocabulary. In addition to utilizing the official test collection, the HNC team created subsets of the TIPSTER corpus. These subsets contained exhaustive relevance judgments for each of the topics (either judged internally or judged by the TIPSTER relevance assessors) and varied in size. A 1,000 document corpus (~10 megabytes), 10,000 document corpus (~60 megabytes), 30,000 document corpus (~120 megabytes), and a 80,000 document corpus (~320 megabytes) all containing selections from the AP Newswire, Wall Street Journal, Federal Registry and the Department of Energy were utilized for testing. In addition to the recall (number of relevant items retrieved / total number of relevant items in collection) and precision (number of relevant items retrieved / total number of items retrieved) numbers obtained from the evaluation code and the test collections mentioned above, HNC analyzed the quality of the learned relationships via the "stem trees" (refer to figures 6 and 7). By evaluating the effectiveness of *MatchPlus* to learn similarity of use as well retrieval performance HNC was able to meet two goals: superior retrieval and routing performance and the ability to learn similarity of use.

4.1 Preprocessing Tests

Preprocessing consists of scanning each document for individual tokens. Each token (delineated by a space, tab, comma, etc.) is then stemmed (i.e. endings such as "ed" and "ing" are removed). Stemming serves the function of treating words such as "run" "runs" and "running" as the same stem (word) as well as greatly decreasing the size of the data structures. A stemming exception list is consulted (to remove stemming errors such as stemming "army" to "arm") as well as a stop word list. The stop word list removes frequently used word such as "and" and "the" that contribute little to the overall meaning of the text.

The use of word pairs (e.g. "white house") was investigated. The list was manually generated by presenting all two word combinations that were found in the text to an analyst who determined if it was a valid word pair. Our investigations found it was best to split the word pairs into their individual words. For example if "white house" was found in the corpus or the query there would be three different context vectors used for the calculations, one for "white" one for "house" and one for the word pair "white house". The use of word pairs in both the bootstrapping and the query processing gave an improvement of 10%-15%.

Use of classical IR tools such as stop word lists and stemming improved performance. Eliminating stop words, stemming and stemming exception words caused a decrease in performance of 12%.

Late in the project HNC was able to experiment with a head list from Compton's New Media. A sample of this list is in Table 2.

take	took	taken	taking
tall	taller	tallest	
tangle	tangleweed	tangles	tangled

Table 2. Encyclopedia Britanica Head List

Replacing the *MatchPlus* baseline stemmer (the Lovins stemmer) with the head list gave no improvement in retrieval performance. On the TIPSTER tests the head list did get equivalent performance to the baseline stemmer. It is interesting that the list did not help more. As seen in Table 2, the two words "took" and "take" are treated as a single word while the baseline stemmer (which simply removes suffixes) would treat these as different words. It seems that with the context vector approach if terms such as "take" and "took" are used in similar ways in the text the context vector learning approach will encode that, likewise, if they are used in dissimilar ways the context vectors for the two words will not point in the same direction. If in preprocessing, the two words are treated as a single word (e.g. take is the same as took) the context vector learning will never be given the chance to determine if in fact the two words are used in a similar fashion.

4.2 Bootstraping Tests

Major effort was spent experimenting and researching the vector learning law. The main objectives were superior retrieval performance and the ability to automatically learn similarity of use for any domain and corpus size. The following are the major experimental areas and the conclusions reached.

4.2.1 Automatic Representation Tests

The key discovery of Phase I was the ability to automatically learn a vector representation for words using only free text examples. *MatchPlus's* original approach used a set of 940 core context vectors. Each core word was compared to a set of 80 features (words). On a scale from -5.0 to +5.0 the amount of "similarity" or "relation" was entered. For example if the core word were "protein" and the features were "agriculture", "DNA", etc. its feature vector might look like Table 3.

Core Word	**Feature 1**	**Feature 2**	**Feature 3**	**Feature 4**	**etc.**
	agriculture	DNA	electronics	human	....
protein	2.0	4.0	0.0	1.0	.....
man	0.0	2.0	0.0	4.0	.....

Table 3. Manually enter context vector for word *"protein"*

An additional 200 random elements (floating point numbers) were augmented to the 80 manually entered features. All 280 feature elements were then modified according to the bootstrap algorithm. To investigate the "benefit" of the manually entered core context vectors the features were replaced by random numbers , making the entire context vector (280 features) completely random. This approach gave a 3% to 6% improvement over the hand entered set on a subset of the TIPSTER corpus. The creation of hand entered vectors are time consuming, corpus dependent and language dependent. The full benefits of hand entry are certainly contingent on the set of core words and the set of features that are chosen. A more domain specific set of manual context vectors may be of some benefit but with the broad range of features and the heterogeneity of the tipster corpus, hand entry gives no apparent retrieval improvement. *MatchPlus's* ability to learn word relationships from random initial vectors without hand entered context vectors is a key discovery that enables the *MatchPlus* system to work in virtually any subject domain and any language.

4.2.2 Freezing Core Context Vectors Tests

The motive for "freezing" (i.e. not allowing the hand entered context vectors to be modified during bootstrapping) was to prevent the collapsing of the vector space. The original bootstrapping algorithm modified word vectors such that each vector pointed in more or less the same direction. This collapsing phenomenon resulted in words and ultimately documents residing in a much restricted vector space. Individual words and documents were indistinguishable. The hypothesis was if the hand entered context vectors were sufficiently spread out and kept constant this would prevent the vectors from collapsing. Due to the limited number of hand entered vectors (940) and the observation that similarity of use encoded manually was drastically different from the similarity of use learned automatically during bootstrapping "freezing" hand entered context vectors did not prove successful.

4.2.3 Staged Bootstrapping Tests

It was thought a staged approach to vector learning may improve performance. The conjecture was that words within a certain frequency range should be trained to stability, then use these vectors to train words within another frequency. The original bootstrapping algorithm made two passes through the entire corpus. On the first pass only words that appeared in corpus 3 times or more were modified. All remaining words were modified on the second iteration. Further experiments involved up to 4 passes with varying frequency thresholds for stem modification on any single iteration. These experiments proved to give no significant change in performance. In fact, the current implementation performs two passes through the corpus modifying everything on each pass. Performance after one iteration is only slightly worse (2%-3%) than two iterations while a third iteration gives a slight improvement (2%-3%) and levels off with more than three passes through the corpus.

4.2.4 Vector Size (Dimensionality) Tests

The choice of the number of features (vector elements) for each stem was 280. Investigations as to the number of features actually utilized was performed in various ways. Using subsets of the TIPSTER corpus (1,000, 10,000 and 30,000 documents) in which relevance judgments were available the vector size was both increased and decreased. Since document context vectors are represented in the same space as the word context vectors the dimensionality was the same. The results of the experiments are given in Table 4. Each entry is in comparison to the baseline of 280 features.

Vector Dimension	Match Filtered Query	Context Vector only Query
50	-12 %	-47 %
140	-8 %	-24 %
280	0 %	0%
512	+5 %	+4 %

Table 4. Vector Dimension vs. Retrieval Performance

To further give evidence as to whether or not all dimensions of the vectors were being utilized an eigan analysis was performed on the 1,000 document corpus. All the stem context vectors (approximately 16,000) were multiplied creating a 280 by 280 symmetric matrix. A singular value decomposition was then performed to determine if the vectors spanned the entire space. Figure 8 plots the eiganvalues for a 280 dimensional system and figure 9 plots the eigenvalues for a 512 dimensional system. It is clear that the 280 features are spanning the space, any fewer would cause the stem vectors to overlap each other and, as evidenced in Table 4, cause a degradation in performance. Figure 9 indicates that 512 dimensions is perhaps too large for this size corpus.

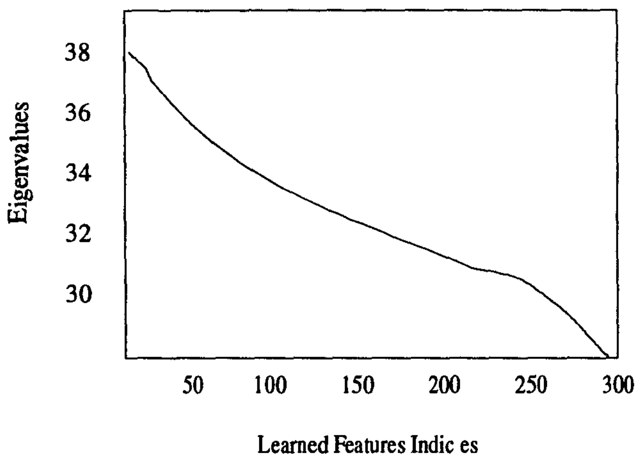

Figure 8. Eigen Analysis of 280 Dimension Stem Context Vectors

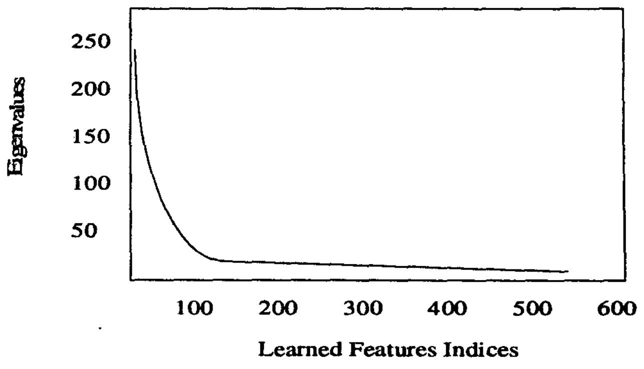

Figure 9. Eigen Analysis of 512 Dimension Stem Context Vectors

Empirical tests showed that increasing the vector size improves recall. For the TIPSTER 24 month evaluation the precision (number of relevant documents retrieved/total number of documents retrieved) for a 280 feature vector versus a 512 feature vector showed little difference while the recall (number of relevant documents retrieved/total number of relevant documents) increased approximately 5%.

4.2.5 Thesaurus Training Tests

HNC obtained an electronic thesaurus in an attempt to encode the relationships found in the thesaurus into the vector space model. The reasoning behind this experiment is as follows. The bootstrapping algorithm is designed to encode word relationships in a vector space model using examples from free text. If a pre-existing knowledge base (e.g. thesaurus) were used as an initial training example this may help the *MatchPlus* system in learning word relationships. The thesaurus training vectors would act as an initial starting point for training on the TIPSTER corpus much like the hand entered context vectors did. Past information retrieval programs have used thesauri for query expansion. *MatchPlus* trained on the thesaurus, which did exhibit very good stem trees, then using the trained vectors trained on the TIPSTER corpus. In one experiment thesaurus words were allowed to be bootstrapped (trained) by the corpus. This gave a degradation in performance (10%-15%) over our baseline system. An alternate approach to thesaurus training involved "freezing" the stems that were trained by the thesaurus during bootstrapping. There was a significant degradation in performance (40%-50%). Through the use of stem tree analysis after thesaurus training and after the TIPSTER corpus training it became apparent that "similarity of use" is NOT the same as synonymy. Referring to the stem trees in figure 6 and figure 7, it is clear that the related "concepts" to both "Chernobyl" and "Hezbollah" are much more than synonyms. The stem trees reveal the "true" meaning of concepts as they are used in the training data.

4.2.6 Training Window Size Tests

The original bootstrapping algorithm used as its window vector three words on either side of the "target" update stem (refer to figure 10). These stems were summed up using a Gaussian distributed weight and applied to the "target" stem. There was no consideration paid to sentence and paragraph boundaries (i.e. the window vectors could be part of another sentence or paragraph). Only the document boundaries were used (i.e. the window for a "target" update stem did NOT carry over into the next document). The conjecture was that like a single document, each sentence contained a complete idea or concept. Likewise a paragraph contained a unique concept and if these "concepts" were treated as individual documents the system would perform better. Tests were run in which bootstrapping occurred on sentence and paragraph markers (i.e. the window did not use words to the right or left of the target update stem if they began or ended a sentence or paragraph). Using only paragraph boundaries as well as sentence and paragraph boundaries made little difference in performance. Using an early version of the bootstrapping algorithm there was a slight improvement in performance (2%-4%). Although with the latest bootstrapping implementation there is a small degradation in performance (less than 1%). Our conjecture that each sentence and each paragraph contains unique "concepts" is incorrect for the TIPSTER corpus. While some of the document sets do not contain paragraph markers (Department of Energy, Federal Registry) it is apparent that even when there exist paragraph markers they do not express complete thoughts but are inserted for aesthetic reasons.

4.2.7 Batched Learning Tests

A learning law experiment that proved to improve retrieval performance significantly for all size builds (1,000 documents, 10,000 documents, etc.) was the batching of the adjustment that was to be made to each stem. The original algorithm would make immediate updates as it encountered each stem in the corpus. For example if the word "stock" was the first word in the first document its initial random context vector would get modified by the neighboring words. When "stock" was encountered again in the document or in one of the next documents either as a target for updating or as a neighboring word contributing to a target stem's update, the "altered" context vector was used. This created a "smearing" condition in which the context vector for any word was continually being changed and other stems used these "intermediate" changes when calculating their neighborhoods. Batching was implemented to remove this condition. The algorithm was modified such that each update that was to be applied to any stem was stored but never actually applied to the target word until the end of the iteration (i.e. when all the documents were processed). This "batching" technique provided a more stable algorithm in that each word context vector remained the same as it became a target for updates and a neighbor, contributing to another word's update. Table 5 shows performance improvement with batching for various corpus sizes.

CORPUS SIZE	PERFORMANCE CHANGE
1k	+25 %
10k	+22 %
30k	+19 %

Table 5. Batching vs. Non-Batching Performance

HNC did experiment with batching the updates on the document boundaries as opposed to the entire corpus but that did not provide as big an improvement as the full batching.

As mentioned above, the learning algorithm uses a window size when calculating a neighborhood for a target word. The original approach used 3 words to the left of the target and 3 words to the right of the target. Various other window widths were tried including using the entire sentence as the window (again with the conjecture that complete concepts are contained in a single sentence). The results indicate a window width of 3 words works well but further investigations with alternate weightings need to be conducted (i.e. Gaussian weight is applied to neighboring stems which causes the closest words in proximity to the target word to have the largest weight). Table 6 has some comparisons for various window width sizes against the baseline of 3 words on either side that were run on the 1k corpus.

WINDOW SIZE	PERFORMANCE CHANGE
4 words	-2 %
5 words	-5 %
entire sentence	-1 %

Table 6. Window Size Experiments

Many experiments were conducted that involved transplanting a set of stem context vectors from one corpus to another. For example a set of vectors trained from a 1,000 document subset could be transplanted into a 10,000 document system. There are approximately 16,000 unique stems in the 1,000 document corpus and 60,000 stems in the 10,000 document corpus. During a transplant the trained context vectors for each stem are copied into the corresponding stem for the 10,000 document build. The stems not found in the 1,000 document build are set to zero since they are untrained (experiments were conducted in which the untrained stem context vectors were initialized to random numbers and the retrieval performance indicated it was better to set them to zero so they have no affect when generating document context vectors). Once the transplant is complete the document context vectors are calculated. The results of various transplant experiments are presented in Table 7.

Type of Corpus Transplant	Transplant Performance
1k transplanted into 10k	-7.0 % prec. , -0.6 % recall
10k transplanted into 30k	-3.2 % prec. , -2.1 % recall

Table 7. Transplant Results

Given these encouraging results and the amount of time required to bootstrap the entire TIPSTER collection, various subsets of the TIPSTER corpus were trained and transplanted into the entire collection. It has become apparent that when working with gigabyte corpus sizes it is important to train on a sufficiently large portion of the text. A 10,000 document subset (45 meg) and an 80,0000 document subset (320 meg) were used for training and transplanted into CD's 1 and 2 of the TIPSTER collection. The official results (avg. prec and total relevant) are given in Table 8.

Training Set Size	Precision	Relevant Documents
10k documents	.2648	7240
80k documents	.2837 (+4 %)	7541 (+7 %)

Table 8. Transplant Results for CD's 1 and 2 from TIPSTER Collection

It seems contradictory that the results in Table 6 indicate it is possible to get "near" baseline performance by transplanting from a smaller corpus while Table 7 indicates performance can suffer from a smaller corpus training and transplant. If one looks at the percent of untrained stems in the above tests it becomes apparent that 10,000 documents (45 meg) is not large enough to characterize the first two CD's (1,200 meg, 740,000 documents). The percentages are presented in Table 9.

Donor and Recipient of Transplant	Percent of Trained Vectors
1k to 10k	32 %
10k to 30k	21 %
10k to CD's 1 and 2 (1,200k)	7 %
80k to CD's 1 and 2 (1,200k)	23 %

Table 9. Percent of Trained Vectors for Various Transplants

Most of the transplanting experiments involved transplanting a smaller training set into a larger set. Some tests were conducted where the training set was the same size and larger than the "target" set and the results indicated that only a small (less than 3%) degradation in performance is detected. This result indicates that a sufficiently large and sufficiently domain compatible (i.e. wall street journal transplanted into associated press, NOT wall street journal transplanted into New England Medical Journal) can be transplanted continuously, eliminating the need to bootstrap new documents, thus greatly increasing the speed of building new systems or updating existing systems. For existing systems, when new data needs to be added it is sufficient to simply create a document context vector for each new document.

4.2.8 Word Sense Disambiguation

Late in the project the concept of "word sense" disambiguation was investigated. "Word sense" disambiguation refers to finding all the "senses" in which a word is used. For example the word "star" may be used in the context of "the moon and the stars", or in the context of "star wars and the Strategic Defense Initiative", or "movie star". These senses can be disambiguated from the text as well as the query resulting in improved retrieval performance. This process is completely automatic and uses no thesauri or knowledge bases. The word sense disambiguation uses a k-means clustering algorithm to cluster a given set of context vectors. If, for example the word "virus" is to be disambiguated each neighborhood window (i.e. the 3 words on either side of "virus") is calculated. Then each of these neighborhood vectors (the number of vectors is simply the frequency of the word "virus" in the corpus) are given to the k-means clustering algorithm with a pre-specified number of resulting clusters. Once these clusters have been calculated they can be used in various ways. During document context vector generation the appropriate sense of the word can be calculated by taking the neighborhood context vector found in the document and calculating the dot products with the centroids of each of the clusters for the word. The closest cluster would then be used when calculating the document context vector. In an identical fashion the "best" sense could be automatically determined in a query. Given the following query: "movie stars who are in cowboy films" it would be expected that the cluster involving "movie stars" would be used and NOT the cluster involving "the moon and the stars". Evidence supporting the above claim can be found in Table 10. The Wall Street Journal from 1990 through 1992 was used. The example shows the list of 4 clusters that have been calculated for the word "virus". The clusters are listed in order of dot product (highest dot product is first) with the word "internet". It is clear the correct sense would have been determined if the query were "internet, virus".

Sense	Related Word 1	Related Word 2	Related Word 3	Related Word 4
1	computer virus	computer	michel-angelo	portable computer
2	mixup	transmit	tobacco	genetic
3	methy-lpredin	secrete	recombine	necros
4	hiv	immun-deficiency	infect	retrovirus

Table 10. Word Senses for *"virus"* in the Context of *"internet"*

There has been limited testing with "word sense" disambiguation. Using the TIPSTER collection and the topic queries the performance is slightly better (less than 5 %) than our baseline system which does not utilize the word senses. Future experiments include using more than one word sense for queries and/or documents, over specifying the number of clusters wanted then running a "combiner" to determine the appropriate number of senses for each word on an individual basis (this has been done for document clustering with some success, see sections below), re-bootstrapping after the word senses have been calculated, only modifying the sense that matches closest with the window context vector, and using alternate clustering algorithms that automatically determine the number of clusters.

4.2.9 Learning Law Tests

One of the most important experiments involved the way in which the word context vectors were updated. More specifically the point at which the update was made and the weight applied to that update were investigated. The original learning technique used a "moving average" type of approach with a positional weighting. The "window" (i.e. the 3 words on either side of the target stem) was computed using the Gaussian weighting function. This window vector was then added to the unnormalized target stem and the resulting vector was normalized (refer to Figure 10).

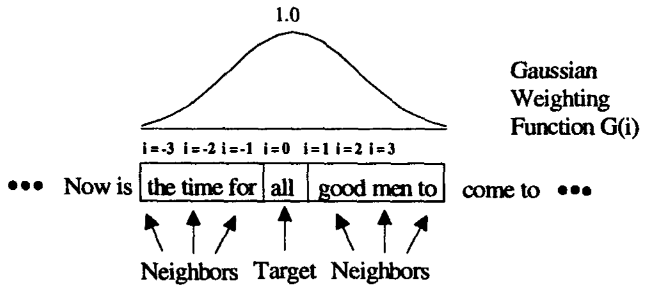

Figure 10. Original Bootstrap Window Calculation

The "moving average" approach produced modest results as evaluated by the TIPSTER tests. Further investigations indicated the algorithm was not moving the stem context vectors a sufficient amount in the direction of their "window". This could be perceived when looking at the learned relationships for a specific word. This "stem tree" calculation uses a single stem and calculates the dot product with every other stem in the corpus and produces a list in order of stems that are closest to the specified stem. The list produced after training was very similar to the list produced before training, using the initial random context vectors. The new approach applies a constraint to the geometry of the vector space. The constraint will determine how close any target can get to its neighbors. The influence of each neighbor on the target is determined individually by Euclidean distance. The original implementation only used the weighted sum of neighborhood stem context vectors. The amount of the effective error and the corresponding change in position is determined by the co-importance of the neighbor and the target. "Overtraining" resulting in poor performance, a characteristic of the original approach, is eliminated when constraints are applied to the learning law.

4.3 Document Context Vector Generation

The approach to document vector generation is to simply add up each of the stems, weighted by the inverse document frequency (IDF) weight (see Figure 11).

$$DocCV_k = \sum_{i=1}^{t} w_i(StemCV_i)$$

$$w_i = \log(\frac{N}{n_i})$$

Figure 11. Document vector equation

4.3.1 Weighting Tests

Various weighting techniques were experimented with. One approach was to eliminate the IDF weight completely and use a flat weighting scheme. Like the approach found in Figure 11 if a word is repeated a great deal throughout the document its vector is continually added to the document causing the document vector to point in the same direction as the repeated stem (i.e. overly biased toward the high frequency stems), regardless of the other words present in the document. A variation to the IDF weighting was experimented with. Instead of adding the stem each time it is encountered in the document, the stem vector is added once and given a weight equal to the log of the frequency of the stem in the document (Chris Buckley reported an improvement over IDF weighting using this approach - See TREC2 Conference, Aug 30, 1993). Once again, this did not prove to be beneficial with regards to the TIPSTER tests but in terms of solving the problem of a single stem "overpowering" the other stems in the document vector representation it has proved successful. A final attempt at alternative document generation weighting used the importance factor formula that was used in the bootstrapping algorithm. This too did not provide a significant change in performance. The original approach (the IDF weighting) does seem to give the best performance on the TIPSTER tests over the variations tried by HNC and others. Table 11 provides a summary of the performance results for the 10k corpus.

WEIGHTING	PERFORMANCE CHANGE
IDF	0 %
Flat	-1.8 %
Modified IDF	-2.7 %
Importance	-3.8 %

Table 11. Document Weighting Experiments

4.3.2 Document Clustering Tests

Document clustering uses the same algorithm as word sense disambiguation (K-means). Each document has a context vector. When document clustering is performed the number of clusters is pre-specified. Each document vector is put into one of these clusters.

The initial motivation behind document clustering was to improve retrieval speed. Although the results are preliminary, it is apparent that retrieval time can be greatly reduced. As the number of clusters searched for a retrieval goes down the retrieval speed also goes down. Obviously as the number of clusters searched is decreased the system's recall goes down. Early tests on the 10,000 document subset of the TIPSTER corpus indicate that by only looking at 500 documents from the top "n" clusters (variable number of clusters are searched for each topic) there is less than a 5% degradation in recall and precision. For the non-cluster system 10,000 dot products are required for each query while the clustered system only requires 500 dot products for each query.

An extremely noteworthy discovery involving document clustering is an automated cluster explanation capability that is inherent in the context vector approach. By taking the centroid vector of each of the resulting document clusters and "dotting" them (i.e. calculating the dot product) with every word and word phrase in the corpus the system can automatically elucidate the meaning of the cluster (i.e. what the documents in the cluster are about). would be used for all topics regardless of the number of words contained in the "concept" section (the average number of words in the concepts section is ~25). The relative threshold would require "x" percent of the words from the "concepts" section to appear in a document

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
launch	at&t	snow	commando	interface
maiden	sprint	gust	guerilla	specific
titan	cable	shower	shiite	portable
payload	transmit	appalachians	lebanon	architecture
pad	fcc	dakato	gunman	processor
navigation	tariff	wisconsin	insurgent	logic
shuttle	mci	wyoming	moslem	programming
rocket	gte	scatter	afghan	database
navstar	fiber	thunderstorm	manilla	graph
gooch	transmitted	commuter	alih	diagnostic
nasa	coaxial	lake	clash	software
unmanned	marketplace	buffalo	hezbollah	prolog
orbit	distance	minnesota	islamabad	maintenance
booster	optic	eastem	fled	hardware
delt	phone	upper	mujahedeen	interact
atl	copper	idaho	palestin	concur
satellite	breakup	picket	pakistan	function
space	fts	nebraska	vorontsov	computer
challenger	deregulation	wind	wound	query
discovery	competition	valley	beirut	language

Table 12. Document Cluster Closest Stems

This can be used to aid in visualization and in creating automated subject indexes. An example of the cluster explanation is given in Table 12. This sample was taken from the 1,000 document test system which was clustered into 20 groups (5 of the clusters are presented in the table).

4.4 Query Processing

The TIPSTER evaluations consisted of 150 topic descriptions (see appendix C for an example). Numerous experiments were conducted to come up with the optimal automated parsing strategy for all the topics over all of the TIPSTER corpus. Additional experiments involved alternative weighting schemes and relevance feedback.

4.4.1 Boolean Filters

The *MatchPlus* system works best in conjunction with a gross boolean filter (i.e. specifying many terms with a small match requirement). Given the structure of the topic queries, it was apparent the "concepts" portion of the topic would be a likely candidate for the boolean filter. This in fact did turn out to be the best approach. The two match threshold approaches were an absolute threshold and a relative threshold. The absolute threshold would use the same threshold for all the topics. For example if the threshold were 4 the system would first retrieve all the documents that had at least 4 of the words contained in the "concepts" section of the topic description. Once those documents were retrieved they were ranked by dot product between their document vectors and the query context vector. The threshold of 4 before it was retrieved. This threshold, like the absolute threshold, was swept and it was determined that a constant threshold of 3 (4 was slightly better for very large corpora) performed the best.

4.4.2 Query Weighting

As is the case when generating document context vectors, query context vectors are a weighted sum of each of the individual query stem vectors. Again, there were various experiments to determine the best weighting scheme. The original approach used the traditional tf*idf weighting (tf being the term frequency in the query). The importance factor equation was experimented with as well as the variation of the tf*idf weight, log(tf)*idf, which was described as giving better performance for other IR systems (see Buckley, TREC2, Aug 30, 1993). As was found for generating document context vectors, the original approach proved to be the best.

4.4.3 Query Expansion

The idea of query expansion was experimented with during TIPSTER Phase I. Traditional adhoc query expansions involved adding words from a thesaurus or a preset knowledge base such as WordNet. For example if the query were "dog" the terms "canine" and "pooch" might be automatically added to the query. *MatchPlus*'s approach was to augment query terms with terms that the system had automatically determined were related. Most likely these terms would NOT be simply synonyms but "concepts" that are related or found in similar documents. For example if the query were "NASA" the top 8 related terms added to the original query would be: "space shuttle, unpiloted, challenger, unmanned, payload, booster, launch, rocket". This automated query expansion proved to be unsuccessful. It is apparent that the original query term(s) already have, encoded in them, the related terms that were used to augment the query. The vector for "NASA" is already "close" to terms such as "space shuttle" and "rocket" so documents with only these two terms and NOT NASA will be retrieved. This "built in" query expansion is one of the benefits of the vector space model and such things as a thesaurus and WordNet (which have not been successful for query expansion in the TIPSTER tests) are not necessary. Future research needs to be done with regard to augmenting query terms with their related words. For example the augmented terms could be used to broaden and enrich the boolean filter, or the terms could be selected and de-selected by a user in a feedback situation.

4.4.4 Relevance Feedback

The use of relevance feedback for adhoc queries proved to be a success. The experiment consisted of taking the top 20 documents retrieved for each query and making a relevance judgment. The context vectors for the relevant documents were then added back into the original query vector and the new query was used to retrieve the rest of the documents. This approach gave a 3 % to 5 % increase in performance over our baseline scores.

4.4.5 Vector Only Query

An extremely encouraging experiment involved using a context vector query with NO boolean filter. The entire topic (excluding the domain and definitions sections) was used in creating this automatic adhoc query. As reported in the tipster 24 month proceedings the context vector query with no boolean matching only performed 3 % worse for relevant documents and 8 % on precision. The comparison is with a similarly formed query but with the additional requirement that documents with at least 4 of the terms in the "concept" section of the topic be present in the document for it to be retrieved. This result indicates that good retrieval performance can be obtained using only the context vector approach. No inverted index (used in calculating boolean "hits") is necessary, greatly reducing the system build time and the system storage requirements.

4.5 Routing Experiments

The vector space model lends itself very nicely to conventional neural network training algorithms. One of those algorithms, the single cell perceptron learning, proved to be very useful in the routing environment. There were two approaches experimented with using the perceptron algorithm. The first, the "stem weighting" approach calculated weights for each of the query terms for each topic description. The input for the network was the dot product between each query term and a previously judged document (either relevant or not relevant for that particular topic query), and the relevance judgment. An example input is given in Table 13.

	surrogate	mother	court	case	relevance
Doc 1	.3	.4	.7	.9	0
Doc 2	.8	.7	.5	.2	1

Table 13. Stem Weighting Input for Perceptron Learning

The output consists of weights for each of the query terms which are applied when adding up each of the query terms to form the final query vector. For the example given in Table 12 one would expect the weight for the term "case" to be low because it did not play a significant part in retrieving the relevant document but did contribute significantly in retrieving the non-relevant document. The second approach that utilizes the perceptron network is the "full context vector" method. The input consists of a judged document context vector (280 floating point values) and the relevance judgment. The network returns with 280 weights that are directly inputted into the query context vector. The stem weighting approach performed better than the "full context vector" approach. This is most likely due to the fact the "full context vector" approach required many more training examples (i.e. relevance judgments) because it was calculating 280 weights while the "stem weighting" was calculating on average 25 weights. Although there were an abundance of relevance judgments for each topic there were many more examples of non-relevant documents than relevant documents.

The idea of combining different types of query runs (from the same system as well as from different systems) proved to be very successful. This idea of "data fusion" was inspired by the observation that any two TIPSTER retrieval runs came up with for the most part non-intersecting sets of documents. Using information about each retrieval approach's performance (e.g. "stem weighting" works better than an adhoc query) retrieval lists were combined in two different ways. The first way determined the best approach for each topic and used that approach to retrieve all the documents for that query. For example, if the context vector only query (no boolean matching) worked best for topic 51 then that query was used to retrieve all the documents. The second approach, which proved to work slightly better, combined document lists for each topic by taking documents from each retrieval approach (again, based on the accuracy weights) and combining them to create a single list. For example, the top ranked retrieved document from the best approach ("stem weighting") would be chosen first, the top ranked document for the second best approach would be next, etc.. An experiment using the latter combining method showed that by combining the best single retrieval run from the University of Massachusetts INQUERY system with *MatchPlus*'s best run the single best performance by any single participant in TREC and TIPSTER could be improved 5 %.

5. EVALUATION SUMMARY

5.1 Results

For TIPSTER Phase I 24-month evaluation HNC submitted 5 adhoc and 5 routing runs. The results for the adhoc runs which were run on Disks 1 & 2 and used topics 101-150 are given in Table 14.

Run	Relevan t Docs	Relative Perf.	Prec. @ 100	Relative Perf.
1	7205	0.0 %	.4520	0.0 %
2	7173	-0.4 %	.4748	+5.0 %
3	7504	+4.1 %	.4616	+2.1 %
4	7202	0.0 %	.4464	-1.2 %
5	6926	-3.9 %	.4128	-8.7 %

Table 14. Adhoc Retrieval Results

The total number of relevant documents for the adhoc was 11,657. For each of the topics 101-150, 1000 documents were retrieved. The 5 different run types are described as follows:

1. Totally automated, use entire topic with a match threshold of 4 terms on the concepts section (baseline system). The training size was 320 megabytes of text.
2. Use run 1 for first 20 retrievals, read these documents and mark relevant ones. Add context vectors for relevant documents to original query context vector and retrieve remaining 980 documents. The training size was 320 megabytes of text.
3. Same query type as run 1 but the system uses a larger context vector size (512 dimensions versus 280 dimensions). The training size was 320 megabytes of text.
4. Same as run 3 but a smaller corpus was used for the learning of the context vectors. The training size was 45 megabytes of text.
5. Context Vector only query (i.e. No boolean match filter). The training size was 320 megabytes of text.

The routing results which were run on Disk 3 and topics 51-100 are given in Table 15.

Run	Relevant Docs	Relative Perf.	Prec. @ 100	Relative Perf.
1	5752	0.0 %	.4128	0.0 %
2	6531	+11.9 %	.4748	+13.1 %
3	5966	+3.6 %	.4616	+4.9 %
4	6436	+10.6 %	.4464	+7.5 %
5	5950	+3.3 %	.4520	+8.7 %

Table 15. Routing Retrieval Results

The total number of relevant documents for the routing was 10,489. For each of the topics 51-100, 1000 documents were retrieved. Runs 1,2 and 4 used a technique called "data fusion". Multiple query types were used then the resulting retrieval lists were combined to produce a single list of documents. The following were the query types used for combining (refer to section 4, routing for details of routing techniques):

- Stem weighting (neural network training)
- Full context vector weighting (neural network training)
- Adhoc automated query, boolean filter (Runs 1 and 2 only)
- Adhoc automated query, context vector only (Run 4 only)

The 5 different run types are described as follows:

1 Stem weighting for entire run.

2. Data fusion 1: combines 4 different types of retrievals inside each topic.

3. Adhoc query, fully automatic with a match threshold of 4 terms on the concepts section.

4. Use same approach as Run 1 but include a context vector only query as one of the query types.

5. Data fusion 2: combines 4 different types of adhoc and routing approaches by topic (i.e. the same query approach is used for retrieving all the documents for a particular topic).

5.2 Interpretation of Results

Increasing the size of the vector training set improves performance. Adhoc run 3 was trained on 320 megabytes of data while run 4 was trained on 45 megabytes. There is nearly a 5% improvement with the increased amount of training data.

Increasing the vector size from 280 dimensions to 512 dimensions helps performance. For the larger vector size *MatchPlus* performed 5% better on the number of relevant documents and 8% better for precision at 100 documents. The increased dimensionality provided more distinguishability between document context vectors thus fewer non-relevant documents were retrieved.

A probabilistic combination of multiple runs gives superior performance to any single query formation technique. The best combination run (Route run 2) gave over a 10% improvement for both precision and recall over the best single routing method (Route run 1).

For further details regarding "unofficial" results from numerous experiments refer to the section on "Experiments and Performance".