KEYNET: Fast indexing for semantically rich information retrieval

Kenneth Baclawski*and J. Elliott Smith
Northeastern University
College of Computer Science
Boston, Massachusetts 02115
(617) 373-4631
FAX: (617) 373-5121
{kenb,esmith}@ccs.neu.edu

April 3, 2006

Abstract

We propose an architecture and index structure for semantically rich information retrieval from a subject-specific collection of documents or other information objects. The technique assumes that information objects have been labeled using small semantic networks called keynets. The index structure is compatible with parallel machine architectures and scales up well, allowing very large collections to be searched very quickly using semantically complex queries.

1 Introduction.

With the expansion of the Internet and the development of new "Information Highways," computer-based communication is becoming the defining technology of this decade. A number of proposals have been made to build a coherent structure over these new high-bandwidth networks to convert them into a National Information Infrastructure (NII). For example, the proposed I-95 Information Market [T+93] proposes an infrastructure that facilitates the free-market purchase, sale and exchange of services. The amount of information that will be available on the I-95 or any other NII is immense: on the order of billions of objects and hundreds of terabytes of data. Finding information in such an environment is a monumental task but essential to the success of the infrastructure. Any solution to this problem must

*This material is based upon work supported by the National Science Foundation under Grant No. IRI-9117030.

satisfy two important requirements: it must be scalable to handle the large number of information objects and it must be semantically rich enough to support effective information retrieval (IR).

We propose an architecture and indexing strategy for a search engine that would satisfy these requirements. The KEYNET system would support information retrieval for a collection of information objects in a single subject area, such as a collection of biological research articles, a set of court cases, files containing remote geophysical sensor data, or even collections of software programs and modules. KEYNET assumes that the information objects in its collection have been annotated using small semantic networks of key concepts (keynets) that are consistent with a subject-specific concept ontology.[1] Although there are certainly important distinctions between semantic networks, knowledge frames and objects in an object-oriented database, none of these distinctions are important for the purposes of KEYNET, so they will be used interchangeably within the context of this article.

A keynet is similar to an abstract or review of a document both in size and in being separately accessible from its corresponding document. If good tools are available, it could be generated by the author of the document with no more effort than is now taken to write the abstract or to select the keywords. It may eventually be possible to use natural language processing techniques to generate keynets, but this is only possible for textual information objects. A third possibility is to have professional annotators construct the keynets. This is less costly than one might expect. The biological research literature consists of some 600,000 articles per year. It would cost less than $30 million per year to annotate this literature with keynets, a tiny amount compared to the cost of generating this literature in the first place[BF93].

To give a simple example of a keynet consider an imaginary paper entitled, "POOQ: A parallel, object-oriented query system." Let's say that the paper uses some known dynamic programming algorithms to optimize queries for use on parallel machine architectures. A keyword-based approach would classify this paper by using phrases such as "parallel algorithms," "object-oriented databases," "dynamic programming," and "query optimization."

Keywords have the advantage that they may include topically relevant words and phrases that do not appear in the documents themselves. Furthermore, some information objects (like images, scientific data and software) have no text that can reasonably be used by traditional IR technology. Keynets enrich the possible semantics as compared with keywords based on simple subject classification schemes by adding relationships between keywords. In addition to being more expressive than sets of keywords, keynets would also generally be larger, though still much smaller than the entire information object.

To describe a keynet for the imaginary POOQ paper above, we must first describe a hypothetical ontology for Computer Science. Suppose that one of the classes in this ontology is concerned with the concept of translation or transformation. The keynet for the POOQ paper could begin with an instance of the transformation class which is linked via an attribute

[1] We are misusing the word "annotate" slightly here. Although a keynet may refer to portions of its document, its role is to classify or abstract rather than to explain the portions of the document to which it refers.

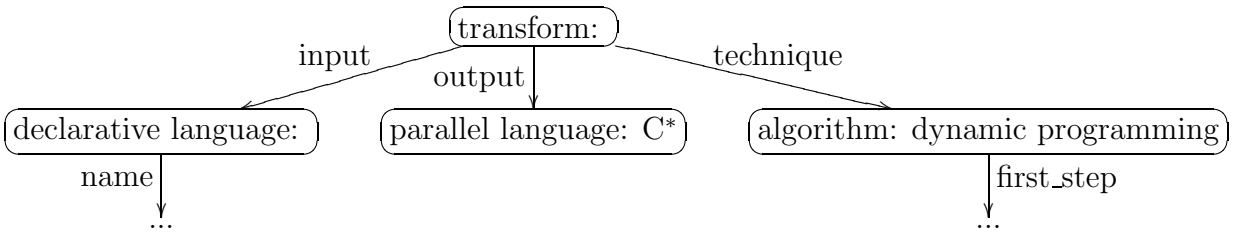


Figure 1: Example of part of a keynet

edge labeled "input" to an object having the subtype "declarative language," and so on. Using semantic networks the keynet would look something like that shown in Figure 1. In this figure, the label of each node consists of its type followed by its value (if any) separated by a colon. Since the output language is well-known, no further elaboration is needed. However, the input language, POOQ, is not well-known, so it must be specified further with attributes like its name and other attributes not shown.

The I-95 infrastructure proposal specifically mentions the need for "content labels [on information objects] to permit users to learn about available resources." These content labels are designed to deal with the problem of scaling up to a full-scale NII. Keynets would provide a solution to the design of such content labels. In spite of supporting the rich semantic content possible with semantic networks, the `KEYNET` system uses efficient indexing techniques based on hashing to make it a fast and scalable search engine.

Since the purpose of the `KEYNET` system is information retrieval, the paper begins with some background on IR in section 2. Despite the perceived inefficiency of semantic network techniques in IR, there are systems that use such methods, and we discuss such related work in section 3. We then describe the overall system architecture of `KEYNET`.

The `KEYNET` system is an information retrieval system for a subject area as specified in a database called the *ontology*. The ontology is discussed in section 5. The objects in the collection will be called *documents* even though, as noted earlier, the `KEYNET` system works equally well with information objects that are not documents in the usual sense of this term. The most common use of `KEYNET` begins with a user query. The query may be expressed using natural language text which is parsed to obtain a semantic network, or else a graphic interface may be used to express the query directly in terms of semantic networks. The same ontology is used for defining the structure of semantic networks of queries as that used for keynets. For example, the query "What systems use dynamic programming to generate C* code?" would be translated into a semantic network like that in Figure 2.

Having converted the original query into a semantic network, the next step is to break the network into small semantic networks having a bounded size. We call these fragments *probes*. Fragmentation is discussed in section 6. These fragments are indexed using a hash index structure defined in section 7. The keynets are also fragmented and the fragments inserted into the index as described in subsection 7.3. The search step consists of hashing the probes and looking for matches with fragments of keynets of documents. In the example,

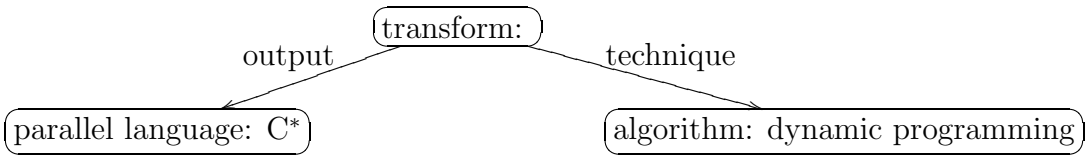


Figure 2: Semantic network of a query

the POOQ document would have fragments that match every probe of the query. The search algorithm is described in more detail in subsection 7.2.

We are in the process of developing a prototype `KEYNET` system for information retrieval on two document collections. The first is a small document collection and ontology prepared by the Biological Knowledge Laboratory [BFH+93] at Northeastern University. The second will be a full-size, randomly-generated document collection. The first collection will be used for testing recall and precision, while the second collection will be used for testing database performance. For a discussion of techniques for rapid generation of random databases see [GEBW93].

2 Information Retrieval.

Information retrieval systems are primarily concerned with the problem of providing mechanisms for a user to select a small set of relevant documents (or parts of documents like chapters, figures, tables, etc.) from a large document collection. This objective differs from database management in a number of ways. Generally speaking, IR systems are not concerned with answering detailed queries about the content of the documents in the collection. On the other hand, whereas a database system can answer detailed queries very precisely, most are not capable of answering the vaguely worded queries about relevance that an IR system can handle.

A database system takes for granted that a query is precisely stated, and the issue becomes how efficiently the query can be evaluated. By contrast, since IR queries are not required to be precise, one measures performance in a different manner. The two most common general measures of retrieval quality in IR research are called *recall* and *precision*. The former is the ratio of documents retrieved versus the number of available documents relevant to the query, i.e., the fraction returned out of all desirable documents (a measure of alpha risk or type I error). The latter is the ratio of the number of relevant documents retrieved versus the total number of documents retrieved, or the useful fraction of what was actually retrieved (beta risk or type II error).

Recall and precision presume that categories assigned by human experts are correct, complete, and well-specified; yet emergent concepts are seldom clearly formulated. An ideal information retrieval mechanism must not only capture poorly expressed concepts, but also somehow adapt as ideas change; its conceptual ontology must evolve over time.

Most commercial IR systems are based on a boolean model of relevance. The query terms

are matched to keywords or to words in the document content, and relevance is determined by the satisfaction of a boolean expression specified by the user. A variety of techniques such as word stemming, truncation, thesauri and lexicons have been used to extend this model[Sal89].

In contrast to boolean methods, the so-called "vector" methods use a notion of relevance that is less sharply defined: a document has a degree of relevance (salience) rather than simply being relevant or irrelevant. Documents are represented as points in a multidimensional vector space. One can then compare documents to each other as well as to queries. One commonly used measure is the cosine of the angle between the points regarded as vectors [Sal89]. Other possible measures of salience include path distance between nodes in a graph, or the number of levels that must be traversed to connect two categories in a hierarchy of abstractions.

While vector methods use probability and statistical methods to improve retrieval effectiveness, they are the same as boolean methods in their reliance on simple linguistic units, such as combinations of words or phrases, as the basis for retrieval. Since fragments of natural language do not always communicate a concept unambiguously for every combination of speaker, listener, writer or reader, retrieval errors inevitably arise from irrelevant discourse. Consequently, to improve retrieval effectiveness IR systems often label documents using keywords or phrases that may never appear in the document itself.

Moreover, relevance requires relative judgement; material irrelevant for categorization may still be relevant for other user purposes. Retrieval that merely matches against text in a document body presumes concepts can be completely characterized by statistical correlations. Yet as Jacobs [Jac93] observes, "statistical methods must be an aid rather than a replacement for knowledge acquisition". Text statistics are best used to identify patterns that depend on specialized words and phrases not obvious to casual readers. Mechanisms that recognize complex ideas are better constructed by human experts, whose understanding of a concept may transcend language.

The power available in a contemporary pattern-matching IR system comes mostly from its lexicon. Efficiency motivates use of simple combinations of lexical categories, such as can be represented in regular expressions. Yet more complicated patterns and mechanisms provide a major advantage in category definition and retrieval despite the time and effort required to create them, rendering knowledge-based approaches preferable to statistical methods.

Unfortunately, knowledge-based approaches that utilize semantic networks are currently considered so inefficient that they are explicitly omitted from some IR textbooks. For example, [FBY92] dismisses them on the basis of "the amount of manual effort that would be needed to represent a large document collection."

3 Related Work.

Despite their reputation in IR circles as cumbersome, inefficient and suitable only for small databases, at least one IR researcher has used knowledge-based indices successfully [FHL+91]. Fuhr *et al*'s AIR/X performs automatic indexing of documents using terms (descriptors)

from a restricted vocabulary. Probabilistic classification determines indexing weights for each descriptor using rule-based inference.

After building a system similar to AIR/X, Jacobs [Jac93] determined that "the combination of statistical analysis and natural language based categorization is considerably better than either alone." His paper describes an automated set of statistical methods for pattern acquisition that operate inside a knowledge-based approach for news categorization (an area closely related to document classification and other information retrieval tasks).

Both of these systems attempt to automate the process of generating index terms. `KEYNET`, by contrast, is not directly concerned with how the index terms are obtained but rather in the data structures and indexing techniques that would make use of the index terms.

Several distinct families of databases for semantic networks have been developed. Such databases are often called knowledge-base systems. Some of the best known of these are: Conceptual Dependency, ECO, KL-ONE, NETL, Preference Semantics, PSN and SNePs (see [Leh92]). All of these support link types, frame systems and so on, but few if any explicitly concern themselves with performance measures familiar in traditional database work, such as minimizing the number of *disk accesses* required to retrieve complex structures (i.e., graphs assembled from multiple frames and their typed relations). Contemporary knowledge-base systems traverse semantic networks one frame/relation at a time. Unless the knowledge-base fits entirely in the main memory of a single processor, these traversals result in large amounts of virtual memory paging.

In [Lev91], Levinson describes a technique for pattern-oriented (graph) retrieval. Levinson's approach discovers common structures among graphs in a database, and then uses pattern associativity to reduce the required number of tests for subgraph isomorphism. A separate paper, [Lev92], compares techniques for subgraph isomorphism testing suitable for pattern-oriented retrieval. The baseline retrieval method described in [Lev92] considers a flat set of N networks with no pattern associativity information, in which each query requires N isomorphism tests. The first improvement indexes commonly occurring substructures in the graphs, and tests only a subset of the entire database. A second improvement creates a multilevel index of subgraph relationships between successive levels of substructures and the domain of graphs in the database (establishes a partial order). A final elaboration applies multilevel indexing to connectivity and label information required during subgraph isomorphism testing (via the refinement method), rather than to the graph substructures themselves. This produces a tree of "node descriptors" in order of increasing specificity, with actual index pointers at its leaves.

In `KEYNET` we adapt subgraph isomorphism techniques to determine the relevance of documents that were previously annotated with keynets. Our approach compiles (fragments of) search paths through a tree of node descriptors into a hash table, reducing lookup computation (although increasing database construction overhead).

The `KNOWIT` system of Sølvberg, Nordbø and Aamodt [SNA92] embeds a semantic network in a front-end query refinement system. Queries to the ESA-QUEST bibliographic database are expanded according to a semantic model that describes the meaning of a concept entirely

in terms of its relations to other concepts in the model. The KNOWIT system is a front-end to a traditional IR system, whereas KEYNET uses a different kind of search strategy.

Chu and Chen [CC92] explain that cooperative query answering (CQA) substitutes retrieval terms to and from some abstract object representation (such as a type abstraction hierarchy). Separate frameworks can be used for grouping "data" (collections of objects with the same type and many common properties) and "knowledge" (relationships between objects of different types in specific problem domains), for example data by (type) class and knowledge by subject. (The "pattern instances" that link data and knowledge are mildly reminiscent of semantic network nodes.)

One example of CQA is "Neighborhood query answering" (NQA) which first generalizes query terms and then re-specializes them into a collection of "nearby concepts" in a compound query. Neighborhood specifiers such as "morning", "afternoon" or "cross-country flight" substitute for specific times or airline names. Chu and Chen present a formal system of rewriting rules and nearness measures for query relaxation in NQA.

Another example of CQA is "Associative query answering" (AQA) which traces behaviour dependencies among "cooperating objects" under a given subject. Virtual "pattern instances" with restricted scope and behaviour provide many-to-many linkages between data objects and knowledge subjects. Knowledge hierarchies based on generalization, composition or abstraction are used to support deductive reasoning that obtains information relevant to query construction but not explicitly contained in a user's request, using a logic-based rule language and inference engine, flexible goals, object contexts, and so on.

In general, CQA is a front-end to a traditional database management system. Moreover, the user is required to specify the degree of relaxation explicitly; it is not done automatically.

The EDS TemplateFiller system [SMHC93] applies Message Understanding (MUC) text-filtering techniques to the generation of knowledge frames for one or a few specific subject areas from entire texts (computer product announcements). TemplateFiller fills in slots for frames that exist in a predefined schema of templates, ignoring subjects that are not in the schema.

There are many other MUC-style systems; in fact, there is an annual competition among them. Such a system could automate the construction of the keynets for a collection of specialized textual documents. In fact, we are building a system of this kind for biological research papers [BFH+93].

4 System Architecture.

The KEYNET system is an information retrieval system for a subject-specific collection of documents. The subject area is defined by a database called the *ontology*. The ontology includes a number of components such as its schema, sublanguage grammar and thesaurus. The schema specifies the structure of knowledge frames and handles the regular features of knowledge in the subdiscipline. Less regular features are specified in the thesaurus which can specify relationships between concepts in the schema as well as between individual terms in the sublanguage. The sublanguage grammar is used for parsing natural language queries

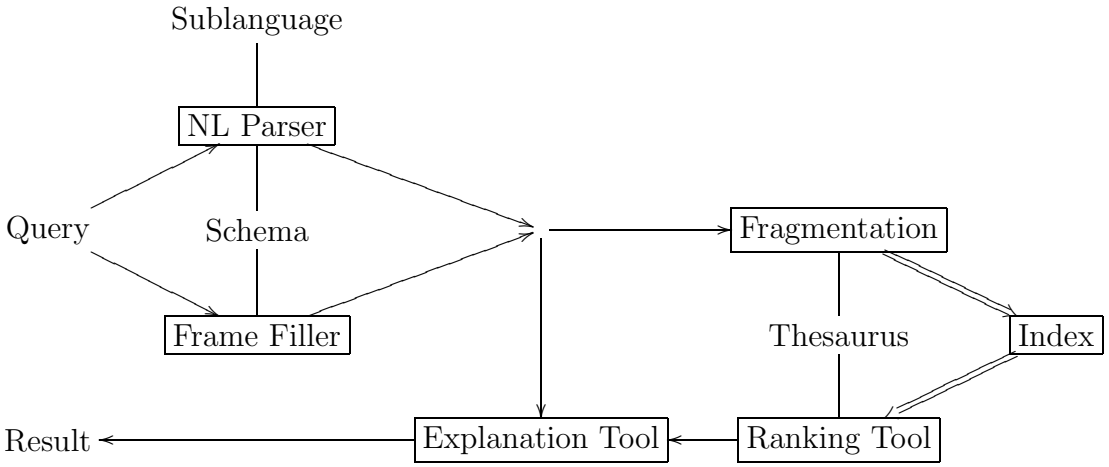


Figure 3: KEYNET System Architecture

as well as for generating natural language from frames. The ontology is discussed in more detail in section 5.

The KEYNET system processes queries using a series of modules as in Figure 3. A user query may be formulated using natural language text which is parsed into frames using the Natural Language (NL) parser which, in turn, uses information in the Sublanguage database and the knowledge frame schema. The topic of NL parsing is beyond the scope of this article and is not described further. Alternatively, a user query may be directly expressed as knowledge frames using a graphical frame fill-in tool. Still another possibility (not shown in the diagram) is for the user to formulate the query using natural language and then to edit the resulting frames if they were not properly parsed by the NL parser.

However the knowledge frames are obtained, the next step is to break the frames into small semantic networks, called probes, each having a bounded size. This fragmentation is also done for the keynets of documents, and fragments thereby obtained will be called *index terms*.

The fragmentation of a query may also involve semantic "broadening" in which additional probes may be constructed by replacing concepts and terms by closely related concepts and terms. The thesaurus database specifies both the weight of such similarity links and the behavior of substitution during broadening. Broadening the query increases recall at the expense of precision. The user can specify how broadly or narrowly the query is to be interpreted by adjusting a threshhold value for semantic broadening.

The result of the fragmentation step is a large number of probes that can be indexed in parallel. This is shown in the diagram by using double arrows. The indexing step is analogous to the technique used by biologists to study the genome. A chromosome (a very long strand of DNA) is probed using small pieces of DNA which can attach to the chromosome only where they match in a precise fashion.

The KEYNET system separates the semantic broadening and indexing steps. As a re-

sult efficient hashing techniques may be employed rather than the somewhat less efficient tree-structured indices. Tree-structured indices can be used to solve both the indexing and broadening steps using a single structure. For example, in a B-tree, entries that are alphabetically close are also close in the index. Techniques for indexing more than one dimension are known (for example, the hB-tree of Lomet-Salzberg [LS90]), but they are complex and are not yet commonly available in commercial systems. Semantic networks not only have a large number of "dimensions" but these dimensions are also not typically linear. Semantic proximity is very difficult to express or even to approximate using multidimensional vector spaces as in the vector models of IR. Accordingly, in the `KEYNET` system we use hash indices rather than tree indices. The cost of using hashing is that a much larger number of probes must be processed, but they can be processed in parallel.

The result of the indexing operation is a set of document identifiers each of which has a rough measure of relevance based on the number of probes that "hit" the document. This measure can be used to rank the documents. Alternatively, one can use more sophisticated graph isomorphism techniques. These more sophisticated techniques would use the thesaurus as well as the original query. As these techniques are beyond the scope of this article, they will be covered elsewhere.

The last step before presenting the result to the user is a tool for explaining how the documents were retrieved. This can be as simple as highlighting the passages that caused the retrieval. This technique works well only when no semantic broadening has occurred. More complex forms of matching will require not only highlighting but also natural language explanations.

Not shown in the diagram is the possibility of an additional form of user interaction known as *relevance feedback*. The user can indicate which documents in a set of retrieved documents are actually relevant to the original query. Such feedback can be used to modify the weights in the thesaurus, resulting in a customized thesaurus for each user.

Also not shown are the modules involved in constructing the ontology and the keynets of documents. These issues are the subject of the next section.

5 Ontology.

The word ontology literally means "a branch of metaphysics relating to the nature and relations of being." Our use of the word is much more restrictive, dealing only with the nature of, and relationships among, concepts within a narrow subject area. Attempts to specify ontologies for scientific disciplines are very common, with most disciplines having some kind of subject classification scheme by this time. However, as Lakoff points out,[Lak87], "human categorization is based on principles that extend far beyond those envisioned in the classical theory." As a result, simple classification methods leading to taxonomies of concepts are inadequate for expressing the rich variety of human categorization techniques.

The `KEYNET` system depends on having a background ontology that defines the structure and behavior of keynets. The Ontology Builder[BF93] is a system related to `KEYNET` that provides support for constructing and maintaining subject-specific ontologies that have much

richer semantics than simple taxonomies.

One of the important features of the Ontology Builder is that it will automatically generate tools for entering keynets. We currently have a prototype called the M&M-Query system that has such a tool specialized for entering knowledge frames for the Materials & Methods sections of biological research papers[BF93]. This prototype furnished an important "proof of concept" for the feasibility of using such tools for annotating documents.

6 Fragmentation.

No matter how the original query is formulated, it is eventually converted into a graph (semantic network) which is then given to the Fragmentation Module. The output of this module is a set of small fragments called *probes*, a term borrowed from biology. This module is responsible for substitutions (broadening) and the computation of the probes.

Broadening is controlled by a weight associated with each possible substitution. Successive substitutions multiply the weights until a user-specified limit is reached. This prevents a combinatorial explosion.

After computing the substitutions, the resulting graphs are broken into probes. The simplest kind of probe is a single node. Each node is labeled with a type and possibly a value. Types are defined in the schema of the ontology, while values are defined in the lexicon. Roughly speaking, probes consisting of a single node correspond to the keywords of a traditional keyword-based IR system.

The next more complex probe is a pair of nodes connected by an attribute edge. Attribute edges are directed and have a label. The attribute labels are defined in the schema of the ontology. The most complex probe that is used is one having two attribute edges and two or three nodes. Note that fragments can consist of more than one node together with the edges (relationships) between them, so that any given node as well as any given relationship edge will generally occur many times in the set of all fragments.

The algorithm for fragmentation can be expressed as follows:

for every node n:
    output the node n
    for every variant n' of n:
        output the variant n'
    for every outgoing edge e:
        output the edge e with nodes n and t
        for every variant n' of the source node n:
            output the edge e with nodes n' and t
        for every variant t' of the target node t:
            output the edge e with nodes n and t'
    for every pair of (undirected) edges e and f incident on nodes t and u, respectively:
        output the triple (n, t, u)
        for every variant n' of the node n:

output the triple (n', t, u)
for every variant t' of the node t:
output the triple (n, t', u)
for every variant u' of the node u:
output the triple (n, t, u')

Each probe has an associated weight. The weight is computed using weights taken from the ontology as well as substitution weights and possibly even weights specified in some way in the original query. The number of probes can be limited by specifying a lower limit below which probes will be discarded. Larger probes have a weight larger than the sum of the weights of its constituent parts since a match with such a probe would be much more meaningful than just matches with the individual nodes.

The fact that a match with a larger probe is more meaningful than one with a smaller probe suggests that indexing with larger probes is more useful than with smaller ones, in general. However, using larger probes expands the size of the index greatly with index terms that are much less likely to be matched. One must trade-off the inclusion of index terms that have high weight but little likelihood of being matched against index terms that have lower weight but greater likelihood of being matched.

Returning to the query example in Figure 2, the fragmentation step would produce six probes: one for each node, one for each edge, and one for the whole network. This assumes no broadening of the query. There are several ways one can broaden this query. The specific language, C*, could be broadened to any parallel language. The algorithm category could be replaced with a more general category such as "algorithm: search." Just using these two variants, the number of probes increases from six to twelve.

Even for a relatively small query, the fragmentation step can generate a large number of probes. However, for a fixed broadening limit, the number of probes is a constant times the size of the query. For a more precise statement of this see Theorem 1 and Corollary 2 in the Appendix.

7 Index Structure.

After fragmentation, the probes are hashed and matched with entries in the index. If a probe matches an entry in the index, then it is said to have *hit* the entry. Each entry consists of a document identifier and additional information to be discussed below. A document can be hit by many probes, and the sum of the weights of all probes that hit the document (suitably normalized) can be used to give an approximate measure of the relevance of the document to the query. Alternative measures of relevance are discussed in the next section.

In the rest of this section, we discuss how the index is structured and how operations are performed on the index. The details of the structure depend to some extent on the architecture of the machine to be used. The main distinction is whether the memory is globally shared (as in tightly coupled machines like the KSR-1) or local as in parallel computers like the MasPar or Connection Machine.

7.1 Data Structure.

The overall structure of the index is a hash table, with each component being called a *bucket*. The buckets, in turn, have the structure of a cache, a structure not often used in an index. For this reason, we call our index structure a *hash-cache index*.

Each probe is converted to a hash index using a standard hashing algorithm as in Knuth[Knu73, section 6.4]. For a probe p, let $h(p)$ be its hash value, a string of exactly n bits. We write $h_k(p)$ for the first k bits of $h(p)$, where $k \leq n$. Similarly, write $h^l(p)$ for the last l bits of $h(p)$. Write $w(p)$ for the weight of the probe p. Depending on the machine architecture, there will either be a fixed number of buckets in the table or the number of buckets can increase as needed. When the number of buckets can vary, we employ the dynamic hashing scheme due to Litwin[Lit80] which he has recently generalized to the case of distributed systems[LNS93]. When the number of buckets is fixed, the number is a power of two, say 2^m. The bucket for a probe p is then determined by $h_m(p)$. The rest of the hash value, $h^{n-m}(p)$, is then used for searching within the bucket. The same technique holds for the case of a variable number of buckets except that m is not necessarily the same for every bucket.

Within each bucket data is arranged as a tree, where $h_{n-m}(p)$ is the index. The tree must be balanced for a parallel machine architecture. Note that probes are not represented in the tree structure. One only has (parts of) hash values and document identifiers. For this to work, it is necessary for the number of bits n in the hash value to be large enough so that collisions will be very unlikely. Unlike traditional hashing algorithms, `KEYNET` can tolerate occasional collisions because of the redundancy inherent in using large numbers of probes. This optimization results in a significant reduction in the overall size of the index since (partial) hash values are much smaller than probes in general. It also improves the speed of indexing by replacing relatively costly string comparisons with fixed-size integer comparisons. For more detail about the probability model underlying this optimization, see the Appendix.

7.2 Searching.

To search for a keynet κ, one first performs substitutions on the nodes of κ, obtaining a list of variants for each node. One then fragments κ into a set of probes having at most three vertices, at most one of which is a variant node. See Corollary 2 in the Appendix for a bound on how many probes can be generated. Let $\mathcal{P}(\kappa)$ be the set of probes.

The next step is to compute $h(p)$ for every $p \in \mathcal{P}(\kappa)$. This can be done in parallel. The hash values are then sent to their buckets. More precisely, at a bucket with address a, one collects the weights $w(p)$ and partial hash values $h^{n-m}(p)$ of the probes that satisfy $h_m(p) = a$. Each partial hash value is then used for searching within its bucket. This step uses well-known algorithms.

The tree searching can be done in parallel. However, in this case, one must face the problem that not all buckets will have the same number of probes. To process all the probes in parallel will require a time proportional to the maximum number of probes occurring in a

bucket. This can be large, and it wastes resources because only a few (or even just one) of the parallel processors will be doing useful work when the last few probes are being processed. However, because of the statistical nature of the algorithm, it is acceptable to discard a small number of probes. When this is done, the number of parallel processing cycles needed can be reduced to reasonable values. For example, if there are 1,000 processors and 2,000 probes, and if one is willing to discard 2 probes on average, then it suffices to process at most 7 probes at each bucket. See the discussion and table in the Appendix for details.

The result of each search within a bucket is a set (possibly empty) of document identifiers. Each document identifier is associated with the weight $w(p)$ of the probe. Sets of document identifiers at different buckets are then merged, with the weights being accumulated for each document. The document identifiers having the highest accumulated weight are then passed to the next module for post-processing.

7.3 Insertion.

Insertion of keynets into the index begins with a fragmentation step similar to the one for searching except that no substitutions are done on the nodes. The number of index terms is then limited by the bound in Theorem 1. Assuming that the knowledge frames are not too large, one can expect that the number of index terms will be no more than about 4 or 5 times the number of nodes in the keynet. The index terms are then hashed as before, but now the partial hash values and document identifiers are inserted into the buckets.

There can, of course, be more than one document identifier associated with the same hash value. However, if the number of these exceeds a specified limit, then the document identifiers are dropped from the index, although the partial hash value is not. Instead, the partial hash value is associated with a marker to indicate that document identifiers were deleted.

A hash value with a large number of associated identifiers does not discriminate enough for it to be useful for retrieval. Traditional IR systems refer to nondiscriminating words as "stop words," and standard lists of stop words are available. Index terms that have too many associated identifiers will be called *stop terms*.

If documents are labeled by keynets having about 50 nodes, then there would be around 200 or so index terms per document. A collection of one million documents will then have over 200 million probes. However, when the nondiscriminating index terms are dropped, the number of terms will be substantially smaller, perhaps only 50-100 million. The resulting index should take up less than a gigabyte of memory. If each document requires 30-100K bytes, then the the index will be about two orders of magnitude smaller than the document collection itself.

When a bucket overflows, there are several strategies that can be used. If expandable hashing is being used, one simply allocates a new bucket (or buckets) and redistributes the trees into the new buckets. If there is a fixed number of buckets, then occurrences of identifiers will be selectively deleted from the bucket in a manner similar to a cache. The details of the caching algorithm are beyond the scope of this article. The effect of selective deletion of identifier occurrences is to "forget" index terms that are less useful. This does not

mean that the documents indexed by the forgotten terms are no longer accessible, since each document will have many terms associated with it. Eventually, of course, all the references to a given document could be forgotten at which point the document would cease to be accessible. However, this would only happen after a relatively long period of time. It would require more than just a loss of interest in this particular document. It would mean that there was no longer any interest in any feature of the document.

7.4 Deletion.

Explicit deletion of documents from the index can be done using a technique similar to the one used for insertion. The only peculiar feature of deletion is that stop terms can change back to being ordinary index terms if enough documents are deleted. However, it may not be necessary to delete documents from the index. As noted in the subsection above, one can allow a document collection to increase in size even with a fixed size index. In this case, documents do not get abruptly removed, but rather gradually get less and less accessible as interest in its index terms declines.

8 Summary.

KEYNET is a graph-oriented method for document indexing and retrieval. Documents must be annotated with small semantic networks that represent their key concepts. A larger semantic network (part of a subject-specific ontology) determines which node and link types (basic concepts and relations) are considered pertinent to a subject during document retrieval.

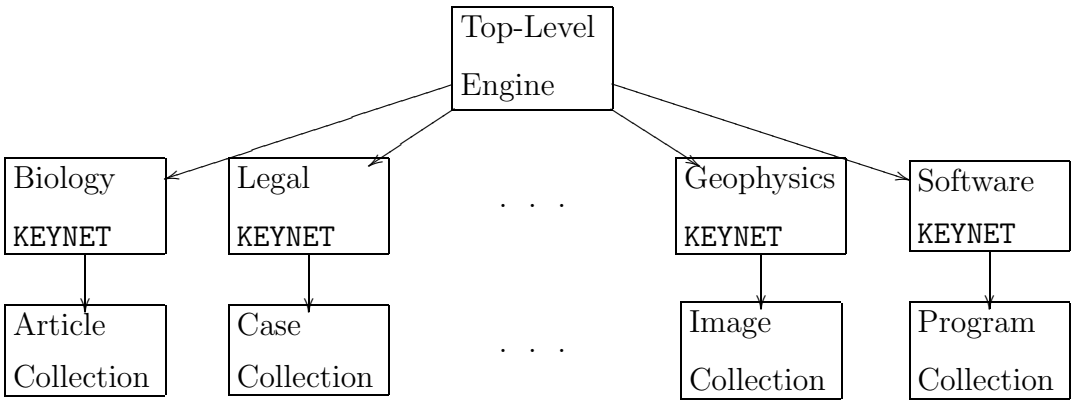
The query graph actually used for retrieval may substitute more general or specific concepts for those specified by the user. Retrieval does *not* match large components of the query graph against whole keynets of documents. Instead, the query graph is *fragmented* into small probes of bounded size. These fragments are matched against document keynets, and resulting retrieval sets of potentially relevant documents are combined using fragment-oriented weights.

The graph representations, their fragmentation, and post-retrieval merging of document sets associated with distinct fragments naturally introduce a "fuzziness" appropriate to information retrieval notions of relevance, and facilitate use of parallel processing resources (at appropriate stages). Although it was not the inspiration for KEYNET, it is interesting to compare it with human memory. Like human memory it is associative and semantically rich. It is also fault-tolerant: loss of a few probes during retrieval or loss of some of the buckets would make it somewhat harder to find a document but would not prevent it. It accomplishes this fault-tolerance by randomly distributing many index terms for each document throughout the entire index. Human memory is believed to have a similar feature. Human memory is highly parallel, and in the parallel version of the KEYNET system, the index has a fixed size with memories fading rather than abruptly disappearing.

The KEYNET system employs a number of optimizations to ensure that it scales up to large document collections and so that it has high performance. Fragmentation combined with

pattern associativity of graph structures and linear hashing techniques produces tractable complexity of communications and computation, despite necessary isomorphism testing and index manipulation. The system is therefore compatible with the requirements for search engines needed in proposals for an NII.

The proposed retrieval mechanism can be applied to instances of itself, producing a quasi-encyclopedic classification of documents. The following diagram suggests how one could organize search engines in the NII:



The top-level search engine appears to be a "hotspot" in this scheme, but it would not have to be consulted for every query. Most individuals would only use a few search engines and would not need to consult the top-level engine once the locations of these engines were known.

Using currently available technology, each search engine could support collections having one million or more documents. The top-level search engine could, in turn, support one million search engines. The entire structure would therefore index 10^{12} documents having an aggregate storage size of 10^{16} bytes, i.e., 10,000 Terabytes.

References

[BF93] K. Baclawski and N. Fridman. M&M-Query: Database support for the annotation and retrieval of biological research articles. In *Proc. ACM SIGMOD Conference*, 1993. Submitted.

[BFH+93] K. Baclawski, R. Futrelle, C. Hafner, M. Pescitelli, N. Fridman, B. Li, and C. Zou. Data/knowledge bases for biological papers and techniques. In *Proc. Sympos. Adv. Data Management for the Scientist and Engineer*, pages 23–28, 1993.

[CC92] Wesley W. Chu and Qiming Chen. Neighborhood and associative query answering. *Journal of Intelligent Information Systems*, 1(3/4):355–386, December 1992.

[FBY92] William B. Frakes and Ricardo Baeza-Yates. *Information Retrieval / data structures and algorithms.* Prentice-Hall, Englewood Cliffs, NJ, 1992.

[FHL+91] Norbert Fuhr, Stephan Hartmann, Gerhard Lustig, Michael Schwantner, Konstadinos Tzeras, and Gerhard Knorz. AIR/X: A Rule-Based Multistage Indexing System for Large Subject Fields. In *Proc. User-Oriented Content-Based Text and Image Handling Conference (RIAO-91)*, pages 606–623, Centre de Hautes Etudes Internationales d'Informatique Documentaire, Paris, France, 1991.

[GEBW93] J. Gray, S. Englert, K. Baclawski, and P. Weinberger. Quickly generating billion record synthetic databases. In *Proc. ACM SIGMOD Conference*, 1993. Submitted.

[Jac93] Paul S. Jacobs. Using statistical methods to improve knowledge-based news categorization. *IEEE Expert*, pages 13–23, April 1993.

[Knu73] Donald Knuth. *Art of Computer Programming*, volume 3: Sorting and Searching. Addison-Wesley, Reading, MA, 1973.

[Lak87] George Lakoff. *Women, Fire and Dangerous Things.* The University of Chicago Press, Chicago, IL, 1987.

[Leh92] Fritz Lehmann. Semantic networks in artificial intelligence, parts i and ii. *Computers and Mathematics with Applications*, 23(2-9), 1992.

[Lev91] Robert Levinson. A self-organizing pattern retrieval system and its applications. *International Journal of Intelligent Systems*, 6:717–738, 1991.

[Lev92] Robert Levinson. Pattern associativity and the retrieval of semantic networks. *Computers and Mathematics with Applications*, 23(6-9):573–600, 1992.

[Lit80] Witold Litwin. Linear hashing: A new tool for file and table addressing. In *Proc. VLDB*, pages 212–223, 1980.

[LNS93] Witold Litwin, Marie-Anne Neimat, and Donovan A. Schneider. LH* – Linear Hashing for Distributed Files. In *SIGMOD Record*, volume 22, pages 327–336, 1993.

[LS90] David B. Lomet and Betty Salzberg. The hB-Tree: A Multiattribute Indexing Method with Good Guaranteed Performance. *ACM Trans. Database Systems*, 15(4):625–658, December 1990.

[Sal89] Gerard Salton. *Automatic Text Processing.* Addison-Wesley, Reading, MA, 1989.

[SMHC93] H. Kelly Shuldberg, Melissa Macpherson, Pete Humphrey, and Jamil Corley. Distilling Information from Text: The EDS TemplateFiller System. *Journal of the American Society for Information Science*, 44(9):493–507, 1993.

[SNA92] Ingeborg Sølvberg, Inge Nordbø, and Agnar Aamodt. Knowledge-based information retrieval. *Future Generation Computer Systems*, 7:379–390, 1991/1992.

[T+93] D. Tennenhouse et al. I-95 The Information Market. Technical Report MIT/LCS/TR-577, Massachusetts Institute of Technology, August 1993.

Appendix: Graph-Theoretic and Probabilistic Details

In this appendix, we sketch the derivation of some results from graph theory and probability that are useful for `KEYNET`.

Graph Theoretic Results

Theorem 1 *Let G be an acyclic, undirected graph with v vertices and e edges. If there are at most n edges incident on an vertex, then there are at most $1 + 2e + n(e-1)/2$ connected subgraphs having at most 3 vertices.*

To prove this, one first reduces to the connected case, in which case $e = v - 1$. There are clearly $2e+1$ connected subgraphs having at most 2 vertices, so the problem is to bound the number of connected subgraphs having exactly 3 vertices. One can show that this number is maximized when all the vertices of G have either valence 1 or valence n (or as close to this as possible). There will be at most $(e-1)/(n-1)$ vertices with valence n, and each of these defines $\binom{n}{2}$ connected subgraphs having 3 vertices.

Allowing substitutions means that some of the vertices have *variants* that are distinguishable from the original vertex. Subgraphs are allowed to have at most one variant vertex. The following is an easy calculation given the one in the theorem above:

Corollary 2 *Let G be an acyclic, undirected graph with v vertices and e edges. If there are at most n edges incident on an vertex, and if each vertex has at most m variants, then there are at most $1 + m + (3m+2)e + (3m+1)n(e-1)/2$ connected subgraphs having at most 3 vertices, where at most one of the vertices is a variant.*

Probability Models

If the hash function $h(p)$ is properly chosen, then the hash values will be approximately uniformly distributed over their range. If the hash value is n bits long, then this range is $[0, 2^n - 1]$. If there are N items in the hash table, then the probability of randomly generating a hash value that corresponds to an item in the table is $N2^{-n}$. In particular, if a probe p does not match any item in the table, then it will accidentally match ("collide") with the hash value of an item in the table with probability $N2^{-n}$. This can be made arbitrarily small by choosing n large enough. For example, if N is 67 million (i.e., 2^{26}) and if n is 40, then the probability of a collision is $2^{-14} = 6 \times 10^{-5}$.

The probability model being used here is called the *finite occupancy process*. The remaining results are concerned with the distribution of a set of independent hash values distributed uniformly among a set of buckets. Let P be the number of hash values (i.e., the number of probes), and let B be the number of buckets. Suppose that one can only process up to N probes per bucket. Let D_{PBN} be the number of probes that are not processed (and hence discarded), and let $FD_{PBN} = D_{PBN}/P$ be the fraction of all probes that are discarded. Both of these are random variables. We would like to compute the expectation (average) of these random variables:

Proposition 3 *Let P, B, N be positive integers with $B > 10$, and write α for the ratio P/B. Then the expectation of FD_{PBN} is approximately equal to*

$$\frac{1}{\alpha} \sum_{k=N+1}^{\infty} (k-N) \frac{\alpha^k}{N!} e^{-\alpha}.$$

The expectation is computed as follows. First approximate the finite occupancy process with the Poisson process. In Poisson process, the probability that there are k probes in a given bucket is $\frac{\alpha^k}{k!}e^{-\alpha}$. If $k \leq N$, then no probes are discarded; if $k = N+1$, then one probe is discarded; and so on. Thus the expected number of probes discarded in a single bucket is

$$\sum_{k=N+1}^{\infty} (k-N) \frac{\alpha^k}{k!} e^{-\alpha}.$$

The total number of probes discarded, D_{PBN}, is the sum of the number of probes discarded at each bucket. There are B buckets, and they are stochastically independent (unlike the case of the finite occupancy process). Hence

$$E(D_{PBN}) = B \sum_{k=N+1}^{\infty} (k-N) \frac{\alpha^k}{k!} e^{-\alpha}.$$

Finally, since $FD_{PBN} = D_{PBN}/P$, the result follows.

The following table gives values of N such that the expectation of FD_{PBN} is less than 0.01, 0.001 and 0.0001, for various values of the ratio P/B:

P/B	$E(FD) < 0.01$	$E(FD) < 0.001$	$E(FD) < 0.0001$
0.5	3	4	5
1	4	5	6
2	6	7	9
3	7	9	11
5	10	12	14
10	16	19	22
20	27	31	35