Temporal Indexing Through Lexical Chaining

Reem Al-Halimi, Rick Kazman

rnkazman@cgl.uwaterloo.ca
ralhalim@neumann.uwaterloo.ca

Department of Computer Science
University of Waterloo
Waterloo, ON, Canada N2L 3G1

1 Introduction

As computerized repositories of information move increasingly away from traditional structured record- and field-oriented relational and hierarchical databases, to databases which include large bodies of unstructured data, new techniques will be needed to aid in the storage, indexing and retrieval of this information [9]. The sources of such data are all around us: televised newscasts, movies, videotaped lectures, requirements elicitation, computer-scanned photographs, computer-captured and computer-generated music. All of these new computer media types are either inherently unstructured, or are only structured within a limited domain [14]. The domain of interest for the research being presented here is video-conferences: conferences where the participants are physically remote, and communicate through video and audio channels, as well as potentially sharing computer applications. Such conferences are already being used to reduce some of the need for business travel, and to extend the reach of university courses to remote communities. However, the data resulting from such conferences is currently stored as a set of parallel streams of unstructured data.

Occasionally, videotaped meetings are indexed so that they may be queried like a database or browsed like an encyclopedia, but typically this indexing only occurs through the input of enormous amounts of manual labor [12]. There is little research into the management and *automatic* indexing of such repositories of information. Such research is clearly needed if the potential of distributed communication is to become a reality.

This paper presents a method for indexing transcriptions of conference meetings by topic. The work described here is part of a research project that aims at designing and prototyping tools for transforming video-conferences into a repository of organizational memory and knowledge, but to do so with relatively no reliance on a semantic analysis of those video-conferences. The project has been based upon the assumption that: a) such information is a valuable information resource, and that b) without tools to index and manage the information, it will be nearly useless. The research has also assumed that a true automatic semantic analysis of video-conferences is a distant goal, since it requires either sophisticated natural language recognition or sophisticated image recognition by a computer, neither of which will be possible in unrestricted domains in the foreseeable future.

To appear in *WordNet: An Electronic Lexical Database and some of its Applications,* C. Fellbaum (ed.), MIT Press, 1997.

We have been investigating ways to index the information based upon attributes of the data which are more easily derived ([7], [8]). In particular:

1. indexing by intended content: both manually and automatically creating meeting agendas and associating them with meeting topics in real time;
2. indexing by temporal structure: viewing video-conferences as “content-less” simultaneous streams of data (the video and audio records of the participants) and indexing based upon the temporal relations of those streams;
3. indexing by topic: using speech recognition and lexical chaining to identify topics within the audio portion of video-conferences.

Furthermore, we have been investigating methods to present the analyzed information to a user:

4. in real-time during a video-conference, and
5. post-facto, in order to permit easier browsing of the stored contents based upon its semantic properties.

Topics 1, 2, 4 and 5 have been dealt with elsewhere ([7], [8]) and will not be further discussed here. This paper will primarily focus on our progress in indexing by topic, and the ways in which WordNet has been applied to this goal.

2 Previous Work

Texts discussing a certain topic use words that are related to each other and to that topic. These words and relations form structures known as *lexical chains*. Hasan [5] formulated lexical chaining as a measure for coherence of stories made up by children. Later, arguing that cohesive harmony reflects text coherence, she developed lexical chaining as a tool for measuring coherence of any text. Hasan’s chains are linear linked lists consisting of text words and relations connecting these words. She defines seven types of these word-to-word relations: synonymy, meronymy, repetition, taxonomy, atonymy, co-taxonomy (two objects are taxonyms of the same object), and co-meronymy (two objects are part of the same object). In addition, Hasan specifies six chain-to-chain relations: epithet-thing, medium-process, process-phenomenon, actor-process, process-goal, and process-location of process.

Morris and Hirst [10] designed an algorithm to automatically build lexical chains using Roget’s International Thesaurus. Unlike Hasan’s lexical chains, however, Morris and Hirst’s are groups of related words. They define six types of lexical relations: meronymy, synonymy, antonymy, taxonomy, repetition, and collocation. In addition, Morris and Hirst’s lexical chains are not interrelated. St-Onge’s LexC system[6] is an implementation of Morris and Hirst’s algorithm that uses WordNet instead of a thesaurus. LexC is the basis of LexTree, our implementation of the ideas presented in this paper.

3 Lexical Trees for Indexing

In order for lexical trees to reflect the semantic structure of the text, it is our assertion that we need to diverge from Hasan’s and Morris and Hirst’s notion of lexical *chains*—one-dimensional structures—and move to two dimensional lexical *trees*. Our modifications to Hasan’s and Morris and Hirst’s schemes fall in two categories: chain structure, and chain parameters. We will discuss each of these topics in turn.

3.1 Chain Structure

Previous work on lexical chaining has defined lexical chains either as a linear linked list of words (see for example [5], [10]). These definitions do not account for the role of the chain's lexical relations in characterizing the chain. Lexical relations vary in number within the text, thus conveying information about the text. For example, in an article discussing growing roses, we are more likely to find words related to the main topic, namely roses (words like *shrub*, *leaf*, *petal*, *root*, possibly *flower*, or even a scientific name such as *angiosperm*), than we are to find words related to a relevant but minor topic such as fertilizer. As a result, there are more instances of, and therefore, more connections among, rose related words than fertilizer related words.

Therefore, the exact amount of text information a chain retains depends on the number and type of relations it contains. Hasan [5] notes that any word in a chain can be related to multiple other words in that chain. If we preserve *all* lexical relations, we will quickly generate a graph of a possibly huge number of edges. Both building and traversing such a graph requires an $O(n^2)$ algorithm, where n is the number of words being considered. An algorithm of such high complexity precludes the possibility that lexical *graph* processing could be incorporated into a real-time system, as is required of indexing video-conferences. On the other hand, if we restrict the number of relations in a chain to any constant number N where $N \geq 2$, we will be creating an artificial chain form that does not necessarily reflect the distribution of the lexical relations in the text since the actual density of connections to a word will be lost. Such a restriction on the form of a chain could have the effect of unintentionally culling important information about main text concepts.

In order to balance these competing concerns, we have decided to disallow cycles in the chain without restricting the number of relations per chain word, thus simplifying the structure—we create *trees*, not graphs—while retaining most of the inherent text information. Cycles of lexical relations contribute little *new* information (e.g. the fact that words A is a part of B, and B is a part of C contributes no more information than knowing that A is a part of B, B is a part of C, and A is a part of C).

We do, however, consider the lack of cycles found while building a chain as an indicator of a sparse non-intuitive subtree such as that in Figure 2. The subtree in this figure is part of LexTree's chains characterizing an article discussing computer security. Some leaves in the subtree are either completely disjoint or so far apart in WordNet as to be pruned by LexTree’s algorithm (as will be discussed in section 3.3), producing a sparse subtree that contains several topics simultaneously.

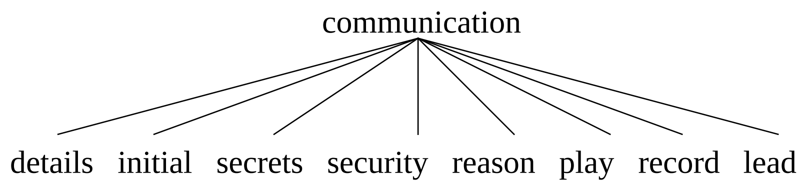

Figure 1: A Sparse Lexical Tree

Figure 2 shows an example of lexical relations, found in WordNet, which are used by LexTree to characterize the following text:

> Inference is the process of creating explicit representations of knowledge from implicit ones. It can be viewed as the creation of knowledge itself. Deductive inference proceeds from a set of assumptions called axioms to new statements that are logically implied by the axioms. Inductive inference typically starts with a set of facts, features or observations, and it produces generalizations, descriptions and laws which account for the given information and which may have the power to predict new facts, features or observations. [13]

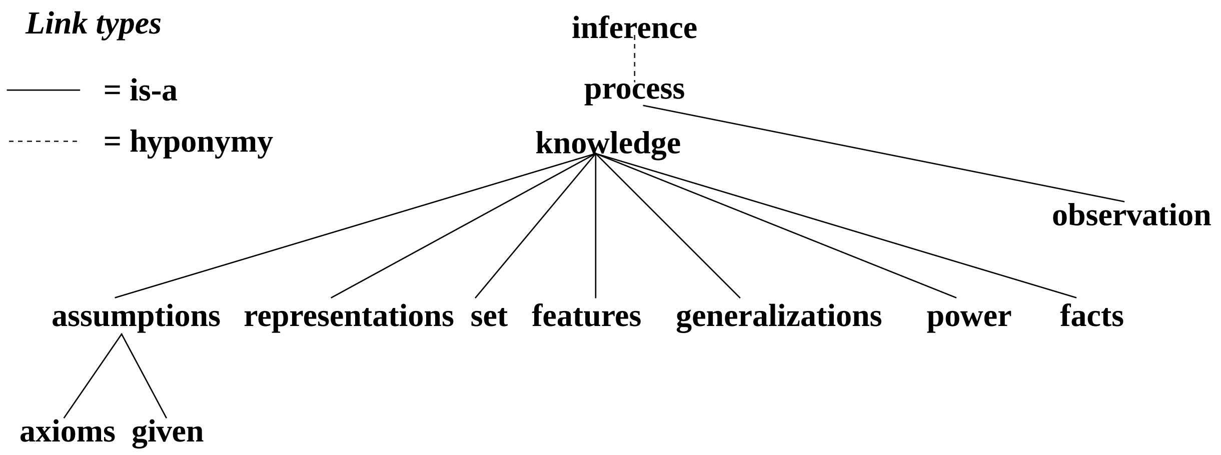

Figure 2: The Lexical Tree for the “Inference” Text

Compare the tree of Figure 2 to the graph in Figure 3, where *all* lexical relations found in WordNet between the words are retained. The main concept in the above text, namely *inference*, is reflected in Figure 2 by the inclusion of the *knowledge* sub-tree which is, in turn, linked to a set of related features of knowledge and forms of knowledge, which are treated as synonyms of knowledge. Note that the word “set” has been wrongly disambiguated here, but this does not damage the overall effect of the tree. Note also that a cycle can exist between any set of related (but not necessarily reiterated) words.

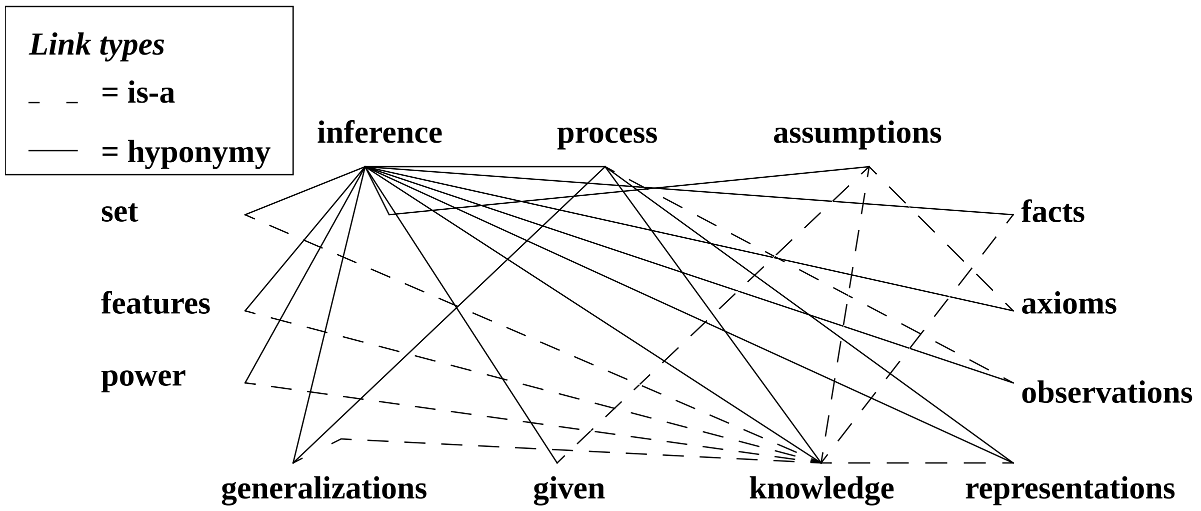

Figure 3: A Lexical Graph for the “Inference” Text

3.2 Tree Parameterization

By restricting the form of the tree, we are forced into selecting a subset of relations from the set of all possible relations in the tree. Our selection is based on three criteria: relation informativeness; relation type compatibility; and word distance. Each criteria is described in more detail next.

3.2.1 Relation Informativeness

Relation informativeness deals with the amount of new information a relation type adds to the tree. This degree of informativeness is context independent. It is the same for a relation type regardless of other relation types in the tree. LexTree uses all noun relations defined in WordNet in addition to *repetition* which is the relation between a noun and a different instance of the same noun. In LexTree, we define five constants each pertaining to the informativeness of a relation type:

1. **Identity-value**: which reflects the informativeness of repetition.
2. **Similarity-value**: which reflects the informativeness of synonymy.
3. **Antonymy-value**: which reflects the informativeness of the antonymy relation.
4. **Holonymy-value**: which reflects the informativeness of holonymy.
5. **Hyponymy-value**: which reflects the informativeness of the hyponymy relation.

The exact values of these constants can be set by the user. Currently, identity-value has the highest value, similarity has the second highest value, antonymy has the third highest, meronymy next, and hyponymy has the lowest level of informativeness because it generalizes the concept to be connected to the tree thus introducing a less specific topic into the tree. Note that we connect the concept to the tree node not the other way around, so the relation goes from the concept to the tree node.

3.2.2 Relation Type Compatibility

Relation type compatibility, on the other hand, is context dependent. It reflects the fact that different relation types interact differently with each other, and these interactions affect the consistency of the information in the tree. For example, a tree that consists mainly of meronymy and synonymy relations, such as that for the “inference” text, as shown in Figure 2, is more focused toward a certain aspect of the main topic than one that contains a mixture of meronymy and antonymy relations.

There are two types of relation type compatibilities: *within connection compatibility*, and *within tree compatibility*. The compatibilities within a connection are what Hirst and St-Onge call “allowable synset paths” [6]. These are paths that LexC allows between synsets of the two words it tries to connect. To form their set of allowable paths, Hirst and St-Onge define three relation directions: *right* for synonymy and antonymy,[1] *upward* for meronymy and hyponymy, and *downward* for hypernymy and holonymy. In LexTree, we redefine the *right* direction for similarity, synonymy, and repetition only, and introduce a *left* direction for antonymy. Using these directions, LexTree then applies the following synset path rules: [2]

1. Right directions are compatible with all other directions since synonymy, similarity, and repetition do not change the current meaning of the word.

2. Left directions are not allowed on the same path as any other direction except for the right direction, otherwise we would be allowing a more general or more specific form of the antonym that is not semantically related to the original word. For instance, an *artifact* is the antonym of *natural object* in WordNet. If we allow left relations on the same path as an upward relation, we could relate *artifact* with *nest* since *nest* is a kind of *natural object*. Antonym coverage is slight in WordNet, and so this situation seldom obtains currently, but the rule makes sense and provides protection against future augmentations to WordNet’s antonym coverage.

3. Opposite directions are incompatible (e.g. holonymy and meronymy) and cannot coexist on the same path. This condition guarantees that the path found is consistent and accurately specifies the relation between the two connected words. For instance, *waste* is a kind of *material* and so is *rock*. If we allowed opposite directions on the path, we could connect *rock* to *material*, an upward direction, and then connect *material* to *waste*, a downward direction. This is an undesirable result, hence opposite directions are ruled out.

---

[1]. Actually Hirst and St. Onge call this direction “horizontal”, but show it as pointing rightwards.

[2]. In a previous version of LexTree, left direction relations were allowed once only in the synset path. Such a restriction was meant to avoid meaning shifts throughout the path. Consider, for example, that an antonym of *cool* is *excited*, but antonyms of *excited* are (in addition to *cool*): *calm*, *nonchalant*, *tame* and *unreactive*. However, we found that, in practice, such a restriction is unnecessary because the antonymy relation for nouns in WordNet is very restrictive, and thus LexTree was unlikely to produce wrong paths due to multiple antonymy relations in the path.

Once a path is found, LexTree gives this path a relation type according to the following rules in order:

1. If the path contains an antonymy relation, then the overall relation type is antonymy.
2. If the path contains any holonymy relations, then the overall relation type is holonymy.
3. If the path contains any meronymy relations, then the overall relation type is meronymy.
4. If the path contains none of the above relations, then the overall relation type is synonymy.

While the “within connection" compatibilities refer to compatibilities within a synset path, the “within tree" compatibilities are concerned with the relations in the tree as a whole. This compatibility type takes into account the text context. The importance of this compatibility stems from the fact that some relations are more informative than others, and the degree of informativeness depends on the context. For example, if the discussion is about different means of communication, we learn more about the topic if we think of *letter* and *list* as different types of *communication*. On the other hand, if the discussion is about letters, for example, it is more informative to relate *list* and *letter* through a part-whole relation.

When LexTree finds a relation between a word and a tree node, it checks this relation’s compatibility with the other relations in the tree. Since each of the tree’s existing relations is assumed to be compatible with all other existing tree relations, LexTree chooses the first relation in the path from the root to the current leaf that is neither a similarity nor synonymy relation if one exists (since these relations carry the least information), else it chooses the synonymy relation. LexTree then checks the compatibility of this relation with the new relation. In LexTree, we define five types of tree relation compatibilities, in order of their priority:

1. **Same Direction Compatibility**: This is when the new relation is of the same type as the tree relation. It is the highest level of compatibility.
2. **Same Right Direction Compatibility**. When one relation is a synonymy relation, and the other is a repetition relation.
3. **Direction Change Compatibility**: When one relation is synonymy or repetition and the second relation is anything other than synonymy or repetition.
4. **Mixed Direction Compatibility**: Compatibility of relations, other than those in 1, 2, 3, or 5, with each other (e.g. meronymy with antonymy).
5. **Opposite Direction Compatibility**: When the two relations are of opposite types (e.g. synonymy and antonymy, holonymy and meronymy). This is the lowest level of compatibility.

LexTree modifies the weight (or value) of the new relation according to its direction compatibility (relation evaluation will be discussed further in section 3.3). These compatibilities are between different relations in the *tree* itself, not in the synset path between *tree* nodes. Although the two types—within connection compatibility and within tree compatibility—have several aspects in common, they differ in several important aspects. In particular, mixed directions are not allowed within a synset path but are allowed within a tree. In addition, a synset path cannot contain more than one antonymy relation while there is no limit on the number of antonymy relations in a tree. Also, we cannot have both holonymy and meronymy in the same synset path, while this is not a problem in a tree.

These differences derive from the different goals of both compatibility types. Within connection, or synset path, compatibilities are meant to define as accurately as possible the relation type between the words to be connected. Within tree compatibilities, however, are meant to produce a consistent and meaningful view of the text context through promoting the coexistence of certain relation types in the same subtree, and penalizing the coexistence of other types such as opposite direction relations.

3.2.3 Word Distance

The final relation parameter to be considered is the *distance* between the words to be connected by the relation. The further the two words are from each other within the source text, the less likely it is that they belong to the same topic, and the weaker their relation. We use the number of intermediate syntactic connectives, words such as *which* and *who* that begin or join phrases, as a distance measure. Such a measure is independent of the text medium, and depends only on the text content. Yet this measure captures the number of different contextual concepts that separate the two words. The set of syntactic connectives is predefined in LexTree and is simple to modify. No use of WordNet is being made for this criterion.

It is interesting to note that topics change more frequently at the prologue and epilogue of a text (spoken or written), than they do in the text's main body. As a result, the effect of distance on a relation is non-uniform (as well as non-linear).

3.3 LexTree’s Algorithm

Given a text file, LexTree builds a set of lexical trees that characterize the text's semantic structure. As shown in Figure 4, building lexical trees involves several steps: the system first selects text words to be included in the tree: stop-words as well as words that have no noun sense are not selected; then it determines the best relation between the chosen word and the tree node, for all trees if any exist; next, it chooses the trees to connect the chosen word to; if no trees are chosen, LexTree creates a new tree whose root is the chosen word. In what follows we will describe LexTree's algorithm in more detail.

LexTree follows the same word selection strategy as LexC (step 2.2 in Figure 4). It discards all words that do not have a noun sense from the tree building process. It also ignores words that appear in the stop word list since they are general words that do not make any significant contribution to the text's semantic structure.

If LexTree decides that a word is connectable, it then selects the appropriate trees to connect the word to (step 2.2.1). Unlike LexC, LexTree connects a word to *all* trees where the connection value between the word and the tree is higher than a preset threshold (step 2.2.1.2). Such policy accounts for the possibility of a keyword referring to more than one topic simultaneously. The *connection value* between a tree and a word is the best connection LexTree finds from among all possible relations between the word and the tree node (steps 2.2.1.1.1 and 2.2.1.1.2).

```
REPEAT
1.  READ next word from input file
2.  IF (word is a possible compound word component) AND
        (word has a noun sense in WordNet) AND
        (compound word buffer is empty) THEN
2.1       PUSH word in compound word buffer
    ELSE
2.2     IF (compound word buffer is not empty) THEN
2.2.1           attempt to join word and item in compound word buffer
        END IF
    ELSE
2.3     IF (word is not a general word) AND
           (word has a noun sense in WordNet) THEN
2.3.1      FOR all trees within a suitable span
2.3.1.1        FOR all words in the tree
2.3.1.1.1          CHECK WordNet for relations between word and tree word
2.3.1.1.2          CHOOSE best relation
               END FOR
2.3.1.2        IF (best relation > connection threshold) THEN
2.3.1.2.1          ADD word and best relation to tree.
               END IF
           END FOR
2.3.2      IF no relations are found THEN
2.3.2.1        MAKE word a node in a new tree
           END IF
        END IF
    END IF
UNTIL end of input file
```

Figure 4: LexTree's Tree Building Algorithm

LexTree evaluates a relation using the parameters described in section 3.2. The weight of a relation is:

$$Weight = R - M + \sum_{1}^{n} S_i + C + dist + T$$

where

- *R* = relation informativeness value.
- *M* = Maximum possible weight.
- *S* = within synset direction compatibility for direction i.
- *C* = within tree connection compatibility.
- *dist* = number of synsets between the word and the tree node.

- T = number of syntactic connectives between the two words in the text.

The best relation is the relation with the highest weight. LexTree searches for paths between the text word and the tree node as long as the relation weight is higher than a preset minimum weight threshold. This means that we can have synset paths that are longer than LexC's maximum of 5 as long as the path's weight is higher than the minimum weight threshold.

One important distinction between LexC and LexTree's algorithm is that LexTree does not have LexC's three levels of relation strength: *extra-strong*, *strong*, and *regular*. Instead, LexTree searches for the relation with the highest weight. Synonymy and repetition are no longer given absolute preference. Rather they are weighted parameters in the relation weight equation. Their value is affected by all other relation parameters such as their text distance.

The trees produced by LexTree represent thetopic semantic structure of the input. A topic, therefore, is represented by a lexical tree eventhough it might not be expressed explicitly in the tree.

3.4 Results

We conducted a preliminary study in order to verify the usability of our lexical trees for automatic indexing of arbitrary text. This study was aimed at demonstrating the existence of lexical trees in texts and comparing tree structures developed by humans to those developed by LexTree. In the study, the subjects were given a journal article and asked to choose a set of text words which, in their opinion, characterized the article. The words which the subjects chose from the text were constrained to be either single words or compounds. The subjects were then asked to group the words which they had chosen from the text.

In order to allow the subjects to create trees that are most intuitive to them, no restriction was made on the grouping criteria they could use. We also removed the article's title as it would have provided extra information that is not necessarily available in spoken language and some forms of written text. The text we chose was an article from an electronic journal that the subjects were unlikely to have previously read (and which they had not in fact read), thus ensuring that their trees were produced entirely from their understanding of the text rather than from their previous experiences. In this way, their tree formation would be as constrained as possible to be a text comprehension task, in much the same way that LexTree has only the words of the text, and no context, to use in building its trees.

The trees built by subjects were similar to each other: there was a high degree of overlap in the keywords which they chose to characterize the text, as well as the grouping criteria which they used. And, despite the fact that they were restricted to using one word keywords only, their trees were similar to the text's outline, as determined by the authors. This indicates that it is possible to build sound topic trees using one word (or compound word) text keywords only.

We then compared the subjects' trees to the trees produced by LexTree for the same text. Half of the text words the subjects used in characterizing the text's topics were found in a single tree of 160 words which LexTree produced (the text was about 1800 words). We will call this tree the *characteristic tree* of the text. The rest of the words chosen by the subjects either connected to sparse trees, or failed to connect to any tree.

Surprisingly, however, more than 75% of the subjects' grouping criteria were in the characteristic tree. This is surprising because the subjects were not required to use words from the original text in expressing their grouping criteria. Furthermore, even when they did use a word that is not used in the text, this word was found closely related to some other word in LexTree's tree. Since their grouping criteria were always important subtopics in the text, these preliminary results give a strong indication of the usefulness of lexical trees in text indexing and retrieval.

4 Applications

Performing lexical chaining on transcripts of meeting, and in the future on recognized speech, allows us to search and query the text based upon *topics*—as identified by the lexical trees which are formed—rather than upon keywords, which is the most common form of search mechanism on text. For example, in a stored meeting which was discussing customer service, one might be interested in knowing places where the participants discussed handling customer complaints, particularly responsiveness. Within the meeting, however, these words may not have been used, or may have been used along with other, similar words. Our approach is to create a lexical forest from the user's query, and match this to lexical trees which have been stored as indexes into the video-conference.

The failings of keyword searches have been long known in the information retrieval community. For example, Furnas *et al* found that even within a constrained area of discourse people rarely agree on what to name things [3]. Expert cooks, in one experiment, were asked to extract keywords from recipes. The probability that two people applied the same term to an object was 18%. In another test, a group of 48 typists were shown typed pages with editorial markups and asked to devise names for each markup operation. Here the terminology overlap was only 7%. The operation of crossing out a word was variously called *delete*, *remove*, *change*, *spell*, *make into*, and so on. The authors found that heavy aliasing (using about twenty synonyms to describe the same concept) was required for moderate retrieval rates.

In a related study, Freeman observed students using a commercial multimedia encyclopedia explicitly made for use in a classroom setting. Overall the students liked the product. Unfortunately, with no overview of the system, it was difficult for the students to get a sense of the encyclopedia's contents. Thus they were unsure how to use the word searching abilities of the program, the primary means provided for retrieving information. Lacking a model of how the information was partitioned and organized, it took trial and error to discover those “sweet search words” that the encyclopedia makers happened to choose. Freeman recounts:

> The fact that all the items, including pictures were classified by keywords restricted the users to the vocabulary and perceptions of the editors and contributors. It has already been noted that a picture can convey so many messages and meanings according to the observer's own inclinations and preferences, which makes it well nigh impossible to classify a picture that will satisfy everyone's requirement. Certain words or phrases which achieve a desired result are passed on through the schools like precious nuggets. [2]

By structuring our search software so that it looks for related trees, we give the user the ability to search a stored multimedia database by topic, rather than by keyword. We not only give the user the ability to search for concepts which are synonyms of the concepts specified in the query, but with the same lexical relations as those specified in the query.

Our current query matching strategy measures the distance between the topics in the query and those in the text trees. The closer the concepts in the query are to those in the tree, the stronger the evidence of the relatedness between the query and the tree. Through such a measure, LexTree finds the tree that is closest to the query. The text pertaining to this tree should be the system's response to the query. For example, assume that we have a business-related query with the following set of topics: *organization, administrative unit, function*. We would like to match these topics to all stored lexical trees, in order to find the texts which most closely match those trees. However, some of these words might match other trees. For example, *unit* and *function* could easily match terms from mathematics texts. Now, consider the following simple trees stored in our database, and represented in Figure 5:

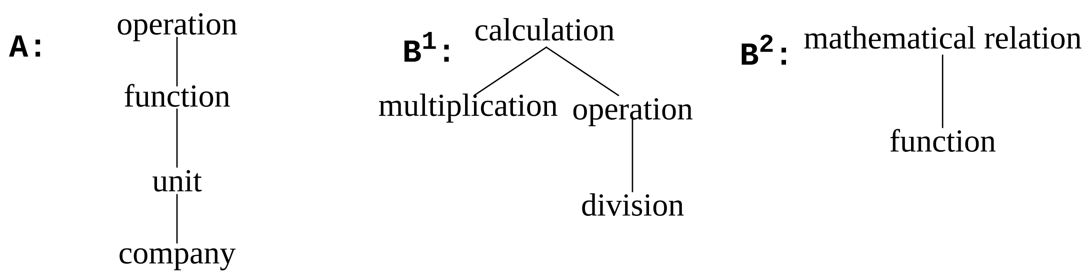

Figure 5: Sample Trees A and B

- tree A consists of the following words indexed by LexTree from a text on business: *company, unit, function, operation*
- trees B^1 and B^2 consist of the following words indexed by LexTree from a text on mathematics: *function, mathematical relation, division, operation, multiplication, calculation*

LexTree takes each topic in the query *organization, administrative unit, function*, and determines the word with the minimum distance to that topic from each tree in its database. The tree with the closest matches (highest number of lexical relationships and minimum lexical distance) is returned as "matching" the query, provided that the number of lexical relationships is higher than a predetermined percentage of the topics in the query (currently set at 75%).

The semantic distances between the nodes of the two trees, as determined by LexTree, are given below:

Query: `administrative unit organization function`

Comparison with trees in A:

```
distance between function(0) and unit(0) = 3
distance between function(0) and function(0) = 0
distance between function(0) and operation(0) = 0
distance between organization(0) and company(0) = 4
distance between organization(0) and unit(0) = 1
distance between organization(0) and function(0) = 3
distance between organization(0) and operation(0) = 3
distance between administrative_unit(0) and company(0) = 4
distance between administrative_unit(0) and unit(0) = 1
distance between administrative_unit(0) and function(0) = 2
distance between administrative_unit(0) and operation(0) = 2
```

Comparison with trees in B:

```
No connections
```

Since LexTree found two exact matches (zero distance) and several "near" matches (distance of 2--4) with tree A, it will prefer this tree over tree B where no matches were found. This indicates that the text represented by tree A is more relevant to the query than that represented by Tree B. Or, to be more precise, lexical relations inherent in the query matches tree A better than it matches either of trees B^1 or B^2. The query matched many words in tree A, it matched neither of the "mathematics trees". A indication of the strength of this method is that there was no match found between the query word *function* and the indexed word *function* in tree B^2, nor was there a match found between *function* and *operation* in tree B^1 even though *function* matched exactly with these same words in tree A. This is because the lexical relations in the query indicated a different sense of *function* than that used in trees B^1 and B^2.

5 Conclusions and Future Work

There are a number of challenges which we have encountered in creating LexTree, and the system surrounding it. For example, one problem which we have noted is WordNet's inconsistent lexical coverage. Any corpus of lexical information has difficulty keeping up with the rate of change of language, and this problem is particularly acute when the domain of interest is *spoken* language. This inherent problem is compounded by two additional factors: the purpose of a meeting, the domain being tested here, is often to describe or define new concepts; and the meetings which we have been testing have involved participants in high-technology areas, which is a particularly rich source of neologisms. Thus, no lexical corpus will contain all of the relevant words, or all of the relevant word associations. Concepts such as "smart card", "hyper link", "wearable computer", "virtual reality", or even "video conference" may not be included or related in a lexical corpus. However, treating these as separate, unrelated words is a serious failing of a system which is attempting to discern word clusters and themes in a stored meeting.

Our approach to dealing with such neologisms is simple, and takes advantage of the fact that such terms are typically used as rigid compounds. While one might hear "smart card" in a conversation, one does not hear "smart green card", "smart little card" or "smart money card" [1]. Compounds of this sort most commonly have the form Noun Noun, or Adjective Noun. Given this characterization, we can pre-process the text, looking for compounds of this form which are repeated throughout the text, and we can dynamically add these to our lexicon (although we can not make any inferences about the lexical relations that such compounds enter into, other than knowing that a smart card is a type of card, virtual reality is a type of reality, a wearable computer is a type of computer, and a video conference is a type of conference).

We also suffer from a lack of part of speech information in creating lexical trees. Many English words have, for example, verb, noun and adjective senses. Currently we just treat all such ambiguous words as nouns. In the future, we intend to employ a naive "parsing" technique, such as a statistical part-of-speech tagger, to aid in narrowing the space of possibilities which LexTree considers for a given word. For example, if an ambiguous word follows a determiner, preposition, or adjective, we will feel relatively secure in treating that word as a noun. If a word is interposed between a determiner and a noun, we will treat it as an adjective.

Another area of investigation is in the use of lexical relations in determining tree similarity. As was shown in section 4, we can already make some surprisingly fine distinctions between senses of words, just by constraining the sense of the words found in its immediate vicinity. However, our way of determining the similarity of two trees is to treat each of them as a “bag” of words, and comparing all words in the query with all words in a stored tree, irrespective of how they are connected together. We feel that we can use tree relations to hone our tree matching strategy even further (or, to be more precise, to move from “bag” matching to true tree matching). The major research issue here involves determining how to assign relative weights to similarity of relations and similarity of words in deriving an overall similarity metric.

Finally, we are still experimenting with usability testing and the proper parameterization of trees so that they correspond most closely to a human’s subjective notion of lexical relationships. Although we can, through user testing, empirically determine the optimal parameters for building lexical trees given the parameter space that we have chosen, there is a much larger space of parameters which we have not been able to experiment with. For example, is our notion of tree distance the correct one?

We feel that, with all of the problems inherent in this activity, it is an exciting and challenging area, and one with potentially enormous benefits. It promises the ability to treat arbitrary multimedia data as an anlyzable and queryable database of information. In addition, by transforming a user’s query into a lexical tree and matching this to stored trees, we give users the ability to query this database by topic, rather than being restricted to the vicissitudes of querying by keyword.

6 References

[1] A. Di Sciullo, E. Williams, *On the Definition of Word,* MIT Press: 1987.

[2] D. Freeman, “Multimedia Learning: The Classroom Experience”, *Computers in Education*, Vol. 15, No. 1-3, 1990, pp. 189-194.

[3] G. Furnas, T. Landauer, L. Gomez, S. Dumais, “The Vocabulary Problem in Human-Systems Communication”, *Communications of the ACM*, Vol. 30, No. 11, 1987, pp. 964-971.

[4] M. Halliday, R. Hasan, *Cohesion in English*, Longman: 1976.

[5] R. Hasan, “Coherence and Cohesive Harmony”, in J. Flood (ed), *Understanding Reading Comprehension*, IRA: Newark, Delaware, 1984, 181-219.

[6] G. Hirst, D. St-Onge, “Lexical chains as representations of context for the detection and correction of malapropisms“, this volume.

[7] R. Kazman, R. Al-Halimi, W. Hunt, M. Mantei, “Four Paradigms for Indexing Video Conferences”, *IEEE Multimedia*, to appear Spring 1996, 17 pages.

[8] R. Kazman, W. Hunt, M. Mantei, “Dynamic Meeting Annotation and Indexing", *Proceedings of the 1995 Pacific Workshop on Distributed Multimedia Systems*, (Honolulu, HI), March 1995, pp. 11-18.

[9] R. Kazman, J. Kominek, “Information Organization in Multimedia Resources”, *Proceedings of SIGDOC ‘93*, 1993, 149-162.

[10] J. Morris, G. Hirst, “Lexical Cohesion, the thesaurus, and the structure of text”, *Computational Linguistics*, 17(1), March 1991, 211-232.

[11] W. Myers, “MCC: Planning the revolution in software”, *IEEE Software*, November 1985.

[12] G. Salomon, T. Oren, K. Kreitman, ``Using Guides to Explore Multimedia Databases'', *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences*, vol. 4, 1989, pp. 3-12.

[13] S. Tanimoto, *The Elements of Artificial Intelligence: An Introduction using Lisp*. Computer Science Press: Rockville, Maryland, p.6.

[14] H. Zhang, S. Tan, S. Smoliar, G. Yihong, “Automatic Parsing and Indexing of News Video”, *Multimedia Systems Journal*, 2(6), 1995, 256-266.