

Word Sense Disambiguation Using Semantic Graph

Narayanan Unny E

Pushpak Bhattacharyya

Abstract

This work describes a method of word sense disambiguation by finding similar words in a text. We have used some characteristic properties of the text and its constituent words for the disambiguation task. Using the WordNet, the algorithm constructs a semantic structure on the text illustrating the relations among the words of the text. This structure is then used for disambiguating the constituent words.

1 Introduction

The problem of word sense disambiguation has been a classical problem in the field of natural language processing. The solution to this problem can in turn solve problems in machine translation, information retrieval, automatic hyperlinking and other such applications involving natural language processing.

The popular approaches to word sense disambiguation can be categorized as (a) supervised (Gale et al., 1992), (Ng and Lee, 1996) (b) unsupervised (Yarowsky, 1995), (Resnik, 1997) and (c) machine readable dictionary based (Cowie et al., 1992), (Miller et al., 1994), (Agirre and Rigau, 1995). In the first case, a correspondence between the sense of a word and its context of usage is established using a training corpus consisting of labeled senses of a word along with its usage context. Though the supervised approach achieves a high rate of accuracy, it is limited to only a small subset of the vocabulary. This subset is decided by the words present in the documents of the training corpus. On the other hand the unsupervised approach has a wider coverage but at a lower accuracy.

Our work falls under the third category where the WordNet (George A. Miller, 1990) is used. Due to the well-structured organization of WordNet, the disambiguation algorithms using it have been able to attain high degrees of accuracy.

The novelty in our work is the use of lexical chain (Morris J. and Hirst G., 1991) like structure. Though many algorithms that perform hyperlinking (Green S, 1999) using lexical chains claim implicit word sense disambiguation, there is perhaps no algorithm that uses properties of lexical chains in explicit sense disambiguation of words. Our algorithm involves the disambiguation of words by building a directed acyclic graph (DAG) structure over the words in the text and using the properties of this structure to disambiguate the words in the text.

2 WordNet as a hypergraph

WordNet can be defined in simple terms as a network of concepts. Sets of synonyms form the building blocks. This set of synonyms is termed the *synset*. These synsets are linked to each other by a group of semantic relations called *hypernymy*, *hyponymy*, *meronymy*, *holonymy* and *antonymy*. Another view of the WordNet could be that of a hypergraph with synsets forming the nodes and the relations forming the arcs. In this structure, we find that the nodes, which lie close to each other, are more related than the words that lie far apart. The shortest distance between two synsets in terms of the link distance separating them is known as the *conceptual distance* (Rada et al., 1989). Lesser the conceptual distance between two synsets, more similar they are. As a result, we can conclude that *the similarity of two synsets in the*

WordNet graph is inversely proportional to the link distance separating them. This observation forms the basis of our algorithm.

3 *Text Structure*

In this section, we look at some unique properties of a text that can be exploited to aid the word sense disambiguation process.

A typical text consists of a flow of ideas that are expressed in the form of structured sentences. These ideas are expressed by a series of words, the words being placed in such a way that the flow of the idea is clear. Depending on the ideas expressed in a text, one can classify the text as belonging to a certain topic. A human reader performs this task of classifying a text by judging the priorities given by the author to each of the ideas expressed in the text.

The words conveying the main ideas in a text are commonly known as the *keywords* in the text. Although nouns and verbs play important roles in expressing the ideas, we feel that the nouns carry the main burden of expressing. We hypothesize that, given the topic of a text, there is a high probability that most of the words in the text are very similar to the topic. For instance, in a text about *Car* we expect to find a lot of words related to car, like *steering*, *wheels* etc. This condition of similarity implies that most of the synsets corresponding to the words in a text would lie close to each other when mapped on to the WordNet graph. We can then impose this condition as a *constraint of proximity in the WordNet* that a particular word from the text must satisfy. This type of constraint imposed on the senses of a group of contiguous words to disambiguate them, has been used in many algorithms. The first algorithm that used such a condition was the algorithm by Michael Sussna (1993). In this, he considers a group of contiguous words in a window. The sense combination for the words in the window is chosen such that the sum of distances between the synsets corresponding to the word forms and word sense combinations is minimized.

Another algorithm that considers this proximity of synsets is the one by Eneko Agirre and German Rigau (1995). In this, they define a measure of *conceptual density* as the number of words in the context window that lie in the sub hierarchy of synsets of the candidate word. The synset of a word is chosen such that the conceptual density is maximized.

The main drawback of these algorithms is that they assume that words lying close to each other in the text have similar senses. This might not be necessarily true. It has also been observed that the reference of a word can extend up to a large distance in the text. Use of anaphors is a good example of such a case. Hence, though we can be sure that the text is populated with similar words pertaining to the main topic of the text, we cannot make any such assumption about the contiguous words in the text. The proof of such a conclusion lies in the result obtained by Eneko Agirre and German Rigau. In the evaluation of their algorithm it was seen that the precision of disambiguation increased with the increase in the window size used for disambiguation. This indicates that the constraint of proximity in the WordNet graph works better as the context size increases.

Another flaw in the above description of a text structure is that a text can talk about more than one topic. We must take this factor into consideration when designing WSD algorithms. The idea of capturing the flow of ideas in the text has been previously used in lexical chains. The aim there was automatic hyperlinking (Green S., 1999). In our algorithm we use the idea of lexical chains to obtain more effective disambiguation.

The discussion so far can be summarized by an illustration of how the words in a text can represent the ideas in a text when they are mapped onto the synsets of the WordNet graph.

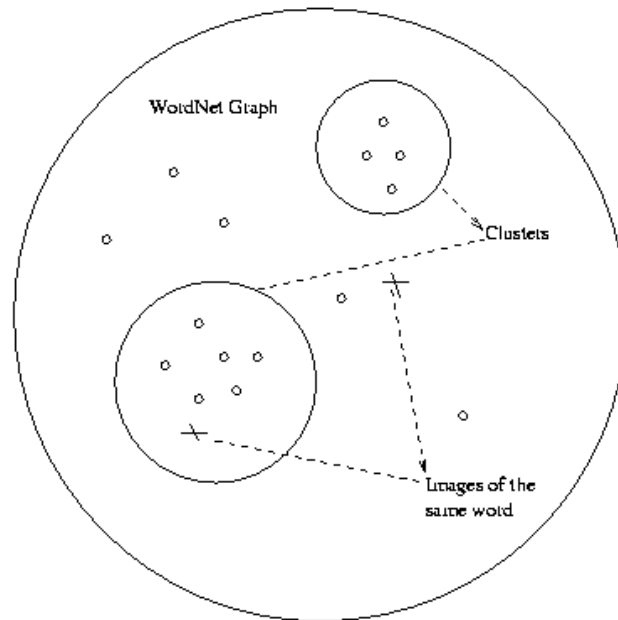


Figure 1 Illustration of clustering in WordNet graph

In figure 1, the large circle represents the WordNet graph. The smaller circles with a number of points inside them correspond to the clusters formed by putting together similar words in the text. The points that we find in them are the synsets corresponding to the words in the text. Each cluster stands for an idea that has been expressed in the text.

When the words are not disambiguated there is more than one image of it in the WordNet graph. If our previous assumption of clustering is true then we can choose the image of the word in such a way that we get a closely packed cluster. The question now is - where do we start to cluster a text. This is the place where the property of monosems helps us. Monosemous words have only a single sense and therefore it serves as the right starting point for clustering.

Our algorithm uses the principles discussed above to produce a cluster of words in the text, and in the process disambiguate them.

4 *Semantic Graph*

The algorithm works by building a Directed Acyclic Graph (DAG) like semantic structure from the words in the text. We call this structure as the *Semantic Graph* of the text. In this semantic graph, the nodes correspond to the different senses of the words in a text. The arcs between them represent the dependency between the nodes. Here the arcs are directed. The direction denotes the dependency – child is dependent on the parent. Furthermore each node has a score associated with it. The magnitude of this score determines the sense of the word. The score at a node corresponds to the probability of a word taking that particular sense.

5 *The Algorithm*

The algorithms that use lexical chains to capture the flow of idea usually start with the first word of the text. With this word as the starting point, other words in the text are added to the lexical chain that is most similar to the candidate word. The similarity is measured in terms of its link distance in the WordNet graph. The problem with such an approach is that we do not know a-priori the sense of the first word. At this point monosems are used. Monosems have only a single sense and hence do not need any disambiguation. Hence we choose the monosems in the text as the starting point for the chain building process.

The individual steps in the algorithm that integrates all these concepts discussed above is given below.

1. Collect the monosems: In this initial step we collect all the monosems in the given text. These form the roots of the semantic DAG that gets built as the algorithm proceeds. For a single word in the text we have all the senses of the word as nodes in the DAG. The number of roots of the DAG becomes equal to the number of monosems in the text.

2. Initialize the scores of monosems: For each node in the DAG, it has a score corresponding to the probability of the word taking that sense. To start with, the roots of the DAG are monosems. These are initialized with a score of 1.

3. Find the link distance to other words: By the definition of a node in the DAG we can recognize that each node is the synset corresponding to a word in the text. We take a word from the text that has not been added to the DAG yet and for all synsets corresponding to the word, we find the link distance between the synsets corresponding to the current node of the DAG and the selected word from the text. This involves searching the WordNet graph starting from one of the synsets and proceeding in a breadth first fashion till we reach the other synset. When we talk about the links here, it must be noted that the reference is to all kinds of relation links in the WordNet. To decrease the time and space complexity caused due to a breadth first search we restrict the search to a cut-off radius say, **search_depth**. The value of this variable decides the depth to which the search proceeds. Any link distance that is more than the **search_depth** is taken as infinite. Placing such a restriction on the depth of the search has an advantage that the precision of disambiguation also increases. **Search_Depth** can control the value of the similarity assigned to pairs of synsets.

4. Form the semantic DAG: After finding the distance to the synsets of a word we compare the distances that we have obtained. The word along with its sense, to which a path has been found, is added as the child of the DAG node. Two situations arise in this case:

- *The new DAG node is not present:* In this case the node is created and added as a child of the current DAG node.
- *The new DAG node is already present:* This condition can happen when the combination of the word and the sense corresponding to the new DAG node had already been found by a sibling of the current DAG node. In such a case a pointer is added to the existing DAG node making it the child of the current DAG node.

From the way the DAG is constructed we find that it is done in a breadth first fashion. That is, the frontier of the DAG is expanded one level at a time. After a particular level has been added to the dag, it goes onto expanding the next level. When a DAG node is chosen as the current node, the word corresponding to the current selection is permanently removed from the list of words in the text. Therefore *a node corresponding to a particular word at a particular location in the text can occur only once in the DAG*. This is to avoid cycles. Note that, this does not prevent the same word occurring at different points in the text, from occurring more than once in the DAG at different levels.

5. Pass the score of the parent to the child: Having initialized the scores of the roots to 1, we need to assign scores to the subsequent nodes at each level. We make a small deviation from our previous notation and let us denote a node at level i and sense j as W_j^i . The score of W_j^i will depend on the following three parameters,

- *Score of parent*: Since a DAG node is added based on its proximity to the synset of the corresponding parent DAG node, the probability of the node depends on the probability associated with its parent word taking the corresponding sense. Hence, the probability of W_j^i taking the sense j depends upon the probability of its parent W_k^{i-1} taking the sense k . Therefore,

$$Score(W_j^i) \propto Score(W_k^{i-1})$$

- *Distance between the child and the parent*: The score of the DAG node also depends on the distance between the parent word and itself in the text. The intuition behind this is that similar words are distributed quite close to each other. This phenomenon is evident in texts having multiple paragraphs. Mostly, the topic of discussion changes during the transition from one paragraph to another. Similar words can be found inside a paragraph rather than between paragraphs. Therefore,

$$Score(W_j^i) \propto \frac{1}{Dist(W^i, W^{i-1})}$$

- *Link distance in the WordNet*: The score of a word also depends on the link distance of the parent word with the current word in the DAG. The lesser the distance between them, the more similar the child node is to the parent node. We are then more certain about the sense of the child node. The score then is

$$Score(W_j^i) \propto \frac{1}{Link_dist(W^i, W^{i-1})}$$

where,

$Link_dist$ is the numerical value of the distance between the parent and child nodes in terms of the number of links separating them in the WordNet graph. If this distance is larger than the parameter $search_depth$ then $link_dist$ is taken as infinite.

Hence, when we combine all these measures together into a score we would get,

$$Score(W_j^i) = \frac{Score(W_k^{i-1})}{Dist(W_j^i, W_k^{i-1}) * Link_dist(W_j^i, W_k^{i-1})}$$

Since the structure is a DAG, a node can have more than one parent. The aggregate score of the node then becomes the sum total of the scores contributed by each of its parents.

6. Judge the sense for a particular word: Having assigned the scores to all the nodes of the DAG, we compare the scores of all the senses of a word. We choose that sense of the word, which has got the highest score. It must be noted that, since we remove the word from the text after having added it to the DAG, we will find all the senses of a particular word¹ at the same level of the DAG. The scores that we have assigned to the different senses of the word are not normalized, so it can happen that some senses have zero score. This serves a purpose, as we will discuss later.

The DAG structure that we had produced during the algorithm is nothing but a subgraph of the WordNet. This subgraph contains most of the words in the text. In the created unique DAG structure, one property is that as we descend down the hierarchy of nodes at each level, the accuracy of disambiguation decreases. This is because of the cascading effect of errors in the disambiguation process. To start with, we were sure about the sense of the monosems that occupied the root, but as the score was passed from one level to another, the certainty about the proportionality between the score and the sense decreased. This means that if there was any error in a level and a wrong sense of a word got a high score then this error can pass on to the descendants of the node and cause errors in them. Hence one technique to improve precision is to maintain a cut-off at some level of the DAG. All the levels below the cut-off

¹ Here, word refers to the word at a particular location of the text and not to a particular word form.

level can be excluded from the disambiguation procedure. In the experiments given below we have not used any level cut-offs.

6 Algorithm with an Example

We will demonstrate each step of the algorithm using an example of a text document. We use a text from the Semcor corpus. The extract is given below.

Nevertheless, "we feel that in the future Fulton County should receive some portion of these available funds", the jurors said. "Failure to do this will continue to place a disproportionate burden" on Fulton taxpayers. The jury also commented on the Fulton ordinary's court which has been under fire for its practices in the appointment of appraisers, guardians and administrators and the awarding of fees and compensation.

In the above extract of the text the underlined words are the nouns in the text and most of them have multiple senses (for example, *portion* has 6 WordNet senses under the noun category).

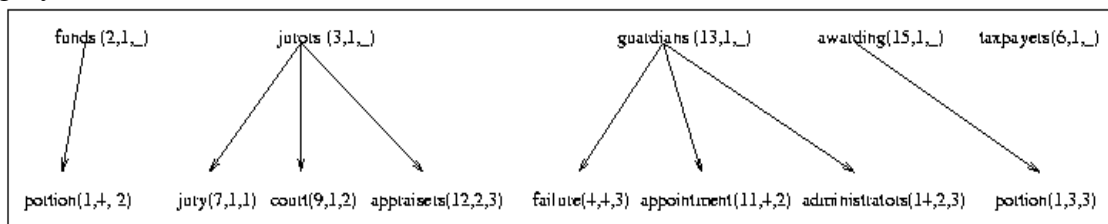


Figure 2 Partial DAG structure

(Step 1) Collect the monosems: Some of the monosems in the text under consideration would be *funds*, *jurors*, *guardians*, *awarding*, and *taxpayers*.

(Step 2) Initialize the scores of monosems: The words *funds*, *jurors*, *guardians*, *awarding* and *taxpayers* will form the roots of the DAG and will have a score of 1.

(Step 3) Find the link distance to other words: From each leaf of the DAG, link distance is found to the words in the text. For example, a link distance of 2 is found from the sense no. 1 of *funds* to the sense no. 4 of *portion*. Hence, sense no. 4 of *portion* is added as a child of *funds*.

(Step 4) Form the semantic DAG: The above step is repeatedly performed to grow the DAG structure. A part of the DAG structure is shown in figure 2.

In figure 2, monosems occupy the roots of the DAG. The three numbers given in brackets along with the words represent the position of the word in the text, sense number and the link distance to that sense from the parent node respectively.

(Step 5) Pass the score of the parent to the child: The nodes of the DAG are then assigned scores based on the relation,

$$Score(W_j^i) = \sum \frac{Score(W_k^{i-1})}{Dist(W_j^i, W_k^{i-1}) * Link_dist(W_j^i, W_k^{i-1})}$$

where the summation is over all the parents W_k^{i-1} of W_j^i . The DAG structure with the scores assigned to the nodes is shown in figure 3.

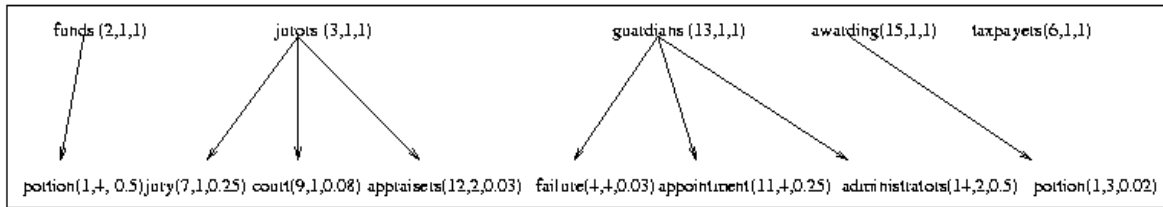


Figure 3 DAG structure with scores

Figure 3 illustrates the final structure of the DAG that we had built. The numbers assigned in brackets to each of the node in the DAG represents the index of the word in the text, the sense number and the score of that particular node. For example, for the node *portion* the index is 1, sense 4 and the score is 0.5.

(Step 6) Judge the sense for a particular word: The sense for a word is chosen such that the node corresponding to that sense in the DAG has the highest score.

For example, in figure 3 we find that the word *portion* at 1st position of the text has different scores for its different senses. We choose the sense of *portion* which has the highest score. In this case it corresponds to the 4th sense of *portion*. Though the actual sense of *portion* is 3, but 4 is found to be close to the 3rd sense. The algorithm hence is found to narrow down the choice from 6 senses to two in this case. Also we find a good example of disambiguation in the case of *court* which has 9 senses.

7 Evaluation

The evaluation of this algorithm was done using three different parameters namely precision², recall³, and coverage⁴. For our experiments, we chose documents from the Semcor corpus (Miller et al., 1993) in which the words have been tagged with sense, part of speech and other lexical information.

7.1 Disambiguation with a single sense (fine level)

In our first experiment, we chose the semcor corpus **br-a01**. The reason for this is that previous works on disambiguation of nouns like Eneko Agirre's were also evaluated using the same document. From this text we extracted out the nouns and these nouns without any sense tagging were passed to our algorithm as input. The output obtained is *the same set of nouns with the senses*. The senses are then compared with the actual senses of these nouns. The performance is then computed from the counts we obtain. We had earlier discussed that the scores assigned to the senses are not normalized and hence it can happen that, for a word all its senses have zero score. In such a case the word is not disambiguated. We performed the experiment runs for three different values of the **search_depth**. The results are summarized below.

Search Depth	Precision	Recall	Coverage
3	62.87	32.30	51.38
4	57.50	42.46	73.84
5	51.91	45.84	88.30

Table 1 Results of the experiment

² Precision is defined as the ratio of number of words disambiguated correctly, to the number of words disambiguated in total.

³ Recall is defined as the ratio of number of words *correctly* disambiguated to the total number of words that were input to the algorithm.

⁴ Coverage is defined as the ratio of number of disambiguated words to the total number of words that were input to the algorithm.

From the results we find a neat variation in the precision and recall values with the values of the **search_depth** parameter. This variation of precision can be explained by the fact **search_depth** restricts the distance upto which two synsets can be considered to be similar. If we increase the value of this parameter, even synsets that are far apart are considered similar, resulting in a dilution of the similarity measure. This in turn affects the precision of the disambiguation.

7.2 Disambiguation at a coarse level

It is a known fact that in the WordNet the sense distinction between the words is very fine. Due to this, an ordinary human user may not be able to disambiguate a word as correctly as had been done by experts in the Semcor documents. To test this, we modified our algorithm in such a way as to produce two senses of the word which gave the highest and the second highest scores. We mark a word as correctly disambiguated if any of the two senses produced matches the actual sense given in the Semcor tagging.

The results of this experiment appear in table below.

Search Depth	Precision	Recall	Coverage
3	67.66	34.76	51.38
4	66.66	49.23	73.84
5	66.20	58.46	88.30

Table 2 Results of the second experiment

The results here too follow the pattern of the first experiment, but the values of precision and recall are better.

7.3 Comparison of results

The results of comparison of our experiments with the algorithm using *conceptual density* (Eneko Agirre and German Rigau, 1996) have been summarized in the table below.

Algorithm	Precision	Recall	Coverage
Conceptual density	47.30	39.40	83.20
Coarse evaluation	67.66	34.76	51.38
Fine evaluation	62.87	32.31	51.38

Table 3 Comparison of different algorithms

The comparison is based on the evaluation performed on the **br-a01** text of Semcor. We find from the table that the performance of our algorithm is very good. The reason for this is that we use the whole text in building the DAG and hence the context in which a word is judged, is not limited to a set of neighboring words. As far as the efficiency is concerned, we have a trade-off between the efficiency and the recall. As we increase the value of **search_depth** parameter, the efficiency in terms of time and space decreases but the performance in terms of the recall increases and precision decreases.

8 Conclusion

We would like to conclude by noting that the sense disambiguation relies heavily on the context of word in the text and must include as much of the whole text as possible.

The algorithm that has been introduced in the paper attempts to find the senses of words in the text by relating them with the words in the entire text rather than some specific set of words forming the window of words.

Another observation that we can make is the variation of the precision and recall values by varying the **search_depth** parameter. This variable gives the user of the algorithm control over its performance.

9 Further Work

Due to the large number of nouns present in a typical text and due to the breadth first search processes for finding the links between synsets, the efficiency of the algorithm is quite poor. We will reduce these overheads by splitting the text into segments and then trying to disambiguate the segments one at a time.

Another possible improvement is to modify the scoring to reflect the transition of the topic across paragraphs. Normally in a text, there is a shade of transition in the topic across the paragraphs. This information can be integrated into the disambiguation algorithm so that the score contributed by a DAG node built from a word in one paragraph is not propagated to a word in the succeeding paragraphs.

References

- Cowie J., Guthrie L., and Guthrie J. 1992. *Lexical Disambiguation Using Simulated Annealing*. Proceedings of the Fifth International Conference on Computational Linguistics COLING-92. 157-161.
- Eneko Agirre and German Rigau. 1995. *A Proposal for Word Sense Disambiguation Using Conceptual Distance*. Proceedings of the First International Conference on Recent Advances in NLP. Bulgaria
- Gale W., Church K., and Yarowsky D. 1992. *One Sense per Discourse* Proceedings of the DARPA Speech and Natural Language Workshop. Harriman. New York.
- Green, S. 1999. *Building Hypertext Links by Computing Semantic Similarity*, IEEE Transactions on Knowledge and Data Engineering, Vol. 11-5, 713-730.
- Rada R., Milli H., Bicknell E. and Blettner M. 1989, *Development and Application of a Metric on Semantic Nets*, in IEEE transactions on Systems, Man and Cybernetics, Vol.19, no.1, 17-30.
- Michael Sussna. 1993. *Word Sense Disambiguation for Free-text Indexing Using a Massive Semantic Network*. Proceedings of the Second International Conference on Information and Knowledge Management. Arlington, Virginia USA.
- Miller G. A. 1990. *Five Papers on WordNet*. Special Issue of International Journal of Lexicography 3(4).
- Miller G. A., Leacock C., Randee T., and Bunker R. 1993. *A Semantic Concordance*. Proceedings of the 3rd DARPA Workshop on Human Language Technology. 303-308. Plainsboro. New Jersey.
- Miller G. A., Chodorow M., Landes S., Leacock C. and Thomas R. G. 1994. *Using a Semantic Concordance for Sense Identification*. Proceedings of the ARPA Human Language Technology Workshop. 240-243.
- Morris J. and Hirst G. 1991. *Lexical Cohesion, the Thesaurus, and the Structure of Text*. Computational Linguistics. Vol. 17. No. 1. 211-232
- Ng H. T. and Lee H.B. 1996. *Integrating Multiple Knowledge Sources to Disambiguate Word Sense: An Exemplar-Based Approach*. Proceedings of the 34th Annual Meeting of the Association of Computational Linguistics (ACL-96). Santa Cruz
- Resnik P. 1997. *Selectional Preference and Sense Disambiguation*. Proceedings of ACL Siglex Workshop on Tagging Text with Lexical Semantics, *Why, What and How?* Washington.
- Yarowsky D. 1995. *Unsupervised Word Sense Disambiguation rivaling Supervised Methods*. Proceedings of the 33rd Association of Computational Linguistics.

Affiliations :

Narayanan Unny E,
Computer Science and Engineering Department,
Indian Institute of Technology Bombay,
Mumbai
nue@cse.iitb.ac.in

Pushpak Bhattacharyya,
Computer Science and Engineering Department,
Indian Institute of Technology Bombay,
Mumbai
pb@cse.iitb.ac.in