Merging Sentences using Shallow Semantic Analysis: A First Experiment

Stephen Wan and Robert Dale
Language Technology Group, Macquarie University
{swan|rdale}@ics.mq.edu.au

Abstract

Multiple document summarisation involves the selection of important information in a collection of related documents, and the compilation of this information into one summary. This paper addresses a key problem in multiple document summarisation, that of merging two semantically similar sentences. Unlike the case of single document summarisation, when important sentences are extracted from multiple related documents, one is likely to find repetition of content within these extracted sentences. To avoid the resulting summary being repetitive and hence incoherent, there is a need for a merging process to handle such repetition. We present work in progress on an algorithm that takes annotated lexical dependency trees corresponding to an abstraction of the key semantics of the input sentences and produces a representation of their combined content that can be used to generate a new sentence.

1 Introduction

Multiple document summarisation (henceforth MDS; see, for example, [McKeown et al 1999]) is a technology that enables the combination of information from a set of similar documents. The kinds of simple sentence extraction techniques used in single-document text summarisation systems (see, for example, [Paice 1990]) provide us with no way of merging content from multiple sources. In a MDS system operating on a set of similar documents, a process will first identify pairs of sentences that are semantically similar in content[1] (for an example of a heuristically-based process, see [Barzilay et al 1999]). A merging process then fuses together the information in these sentences. If the process of identifying similar sentences has worked well, the contents of the two sentences will overlap; however, they will rarely be identical. The task faced by the merging process is therefore twofold: it must recognise when the two sentences contain the same content so that this information appears in the resulting summary only once, and it must recognise when the sentences contain complementary information so that the contributions of both are incorporated.

Given that we cannot rely on the availability of a deep semantic analysis, the key challenge we are faced with is how to develop a merging process that operates reliably. In this paper, we present a first version of an algorithm that achieves this by operating upon an annotated lexical dependency tree of each sentence to be merged, and we discuss the nature of the representations used. The nature of the information encoded within the annotated lexical dependency tree is part of the research undertaken in this project.

Our goal is to develop a general mechanism that is domain-independent in nature, but which, for any given application domain, is able to use declaratively-specified rules that

[1] There are a number of ways in which this might be achieved; see [McKeown et al 1999] for an approach that uses a machine learning classification tool.

capture information about syntactic conventions and word usage in that domain in order to optimise performance. In our initial experiments, we have been focussing on the application of the mechanism in the news domain.

The remainder of this paper is structured as follows. In Section 2, we discuss the form of representation we use and how it is produced. In Section 3, we describe the merging algorithm that operates over these representations. In Section 4, we present the results of our experiments so far, and in Section 5 we draw some conclusions and discuss the next steps in the work.

2 Representations

A traditional view of NLP would suggest that the best way to merge the contents of two documents is to build sophisticated representations of their individual contents, and then to integrate these two representations. However, MDS systems are required to operate over real texts, and so approaches that are based on a deep semantic analysis are beyond the current state of the art. Our response to this is to try to identify some level of semantic representation that is shallow enough to be produced reliably from unrestricted text, but which provides enough abstraction to allow similarities and differences in semantic content to be recognised. The representation developed in this work is called the annotated lexical dependency tree. Ultimately, this representation only needs to be as rich as the merging process requires. This trade-off, which specifies the degree to which semantic information is encoded in the representation, is a topic of investigation.

A key observation here is that a fairly standard representation of syntactic structure is not enough for our needs because multiple syntactic constructions may correspond to the same semantic content. As a result, since the merging process operates on the level of semantics, the use of syntactic representations would result in an increased number of declarative rules used in the merging process in order to accommodate these differing syntactic representations that may be similar semantically. Hence, it is useful to use a representation that abstracts across the superficial differences in syntax, so as to represent the key semantic content necessary for the merging process. Two common phenomena that exemplify the problem here are the superficial differences between active and passive forms, and between variations of reported speech; we have to be able to abstract across such differences.

We settle on a dependency-based representation, where the head–modifier relationships in the results of an analysis mirror the semantic dependencies in the text. We use Collins' statistically based parser [Collins 1996] to produce an analysis of input sentences; as well as providing a standard phrase structure representation, this analysis also indicates the lexical head of each phrase. From these structures, we then construct a *lexical dependency tree* for each sentence. In such a tree, each node consists of a lexeme and its corresponding part of speech tag; the immediate daughters of a node are the head lexical elements of the constituents are dependent on that node. Typically, a lexical dependency tree for a sentence has a verb at the root of the tree. The syntactic subject and object, as well as any prepositional phrases that are complements of the verb, then form the children of this root node. The lexical dependency tree becomes the basis of an abstract representation that encodes key elements of the semantic content of the sentence; we call this representation an *annotated lexical dependency tree.*

The process that builds this representation does so by applying a predefined series of tree transformations. The purpose of these tree operations is to abstract across superficial syntactic differences present in the unrefined lexical dependency tree; these differences are irrelevant to the merging process, since it operates on non-closed class content and on certain key semantic relationships in the sentence. For example, in the case of passive sentences, auxiliary verbs are removed, and grammatical subject and objects are replaced by logical subjects and objects; in effect, the shallow representation has something of the flavour of a lexically-based case frame representation. In constructing this slightly richer representation, any information regarding the original syntax or lexical items used that would be lost by virtue of the rewriting is maintained by means of featural annotations in the output structure; this information can be used when generating a new output sentence from the results of the merging process. An example showing the analysis for a passive sentence is presented in Figure 1.[2] The analysis for the corresponding active sentence is shown later in the paper; the semantic representation is very similar, modulo the annotations on the nodes as discussed above.

Input Sentence:

The White House was visited by Kofi Annan who had just visited the Middle East.

Part of Speech Tags:

The/DT White/NNP House/NNP was/VBD visited/VBN by/IN Kofi/NNP Annan/NNP who/WP had/VBD just/RB visited/VBN the/DT Middle/NNP East/NNP ./.

Lexical Dependency Tree:

```
|_ was, POS: VBD,
   |_ House, POS: NNP,
      |_ The, POS: DT,
      |_ White, POS: NNP,
   |_ visited, POS: VBN,
      |_ by, POS: IN,
         |_ Annan, POS: NNP,
            |_ Kofi, POS: NNP,
            |_ who, POS: WP,
               |_ visited, POS: VBN,
                  |_ just, POS: RB,
                  |_ East, POS: NNP,
                     |_ the, POS: DT,
                     |_ Middle,POS: NNP,
                     |_ ., POS: PUNC.,
```

Annotated lexical dependency tree:

```
|_ visited, POS: VBN, features: passive
   |_ Annan, POS: NNP, features: subject
      |_ Kofi, POS: NNP,
      |_ who, POS: WP,
         |_ visited, POS: VBN,
            |_ just, POS: RB,
            |_ East, POS: NNP,
               |_ the, POS: DT,
               |_ Middle, POS: NNP,
   |_ House, POS: NNP, features: object
      |_ The, POS: DT,
      |_ White, POS: NNP,
```

Figure 1. The representations derived from an input sentence

Currently, besides abstracting out the common content in the passive and active voice, the tree writing operations which build the annotated lexical dependency tree also remove auxiliary verbs and annotate relationships between conjuncts. As in the case of the passive and active voice, sentences describing the act of quotation can be realised syntactically in a variety of surface forms. These are also normalised and the relationships between the speaker, audience and reported speech is indicated by means of annotations. Correspondences between verbs and their nominalisations are not

[2] The part of speech tags used here are those used in the Penn Treebank.

currently handled, since the current implementation of the merging process does not merge across content that is expressed using different parts of speech. For similar reasons, the correspondences between relative clauses and other forms of noun modification are not abstracted out. This limitation will be overcome in subsequent refinement of the algorithm presented here.

3 The Merging Algorithm

3.1 Recursive Top-Down Merging

Given two sentences in the annotated lexical dependency tree described above, the goal of the *Merge* process is to produce a composite structure that contains the relevant information drawn from each of the two input representations. We do this by recursively traversing the trees in parallel in a top-down breadth-first fashion, as shown in the algorithm in Figure 2.

```
Merge(t1, t2) :-
        if  isLeafCondition (t1,t2) then
            return findAppropriateLeaf(t1,t2);
        else
            host ¬ chooseHost(t1,t2);
            pairings ¬ pairDaughters(t1, t2));
            if isEmpty(pairings) then
                return null;
            else
                for ⟨p1, p2⟩ ∈ pairings
                    newDaughter ¬ merge(p1, p2);
                    if newDaughter is not null then
                        addDaughter(host, newDaughter);
                    else
                        if not sufficientlyMerged (⟨p1, p2⟩) then
                            return null;
                return host;
```

Figure 2. The Merge Algorithm

This algorithm works as follows. One of the two input trees is arbitrarily chosen as the host; this serves as the base for a new tree containing the merger of the two input trees. Beginning with the root nodes of each tree, the merging engine then attempts to apply one of a number of declaratively specified *pairing rules.* These rules identify daughter nodes, which potentially represent the same content. The result of these pairing rules, applied using a subprocess described below, is a vector of paired daughters. If the results vector is non-empty, the merging engine is then invoked recursively for each pair of daughter nodes indicated in the vector. If these recursive merges are successful, their results form the daughters of the output structure root node. This recursive process continues until the leaves of the tree are reached. If two leaf nodes are semantically compatible, the most appropriate node is chosen and returned as the result of the merger. Appropriateness is determined by considering the relationships of the nodes in question to the choice of the host. Where possible, by choosing consistently from the same host tree, the result of the *Merging* process is likely to maintain any collocational relationships between lexemes. This is necessary because the annotated lexical

dependency tree is not able to fully abstract out the semantic content of the input sentence, and thus its meaning is dependent on information at the lexico-grammatical level. Semantic compatibility might be determined by a variety of means; in the current version of the system, we use a combination of lexical identity, part of speech equivalence, verb base form equivalence and shared membership of Wordnet synsets [Miller et al 1990].

If a recursive merge fails, the merging engine checks to see if the pairing of daughters was essential for the merging of the two original input nodes. This is determined by examining the relationship between the daughters and the host node. If the daughters do not contain essential content, the failure to merge two particular branches is simply ignored. For example, if the merging of two annotated lexical dependency trees has been partially successful and the intermediate result contains the verb and both the logical subject and object, failure to merge additional modifiers of one of the participants should not prevent the merger from ultimately succeeding, since the inclusion of the modifier may not be crucial. Such a failure may simply be due to a lack of linguistic knowledge in the system, for example when the system fails to detect that two noun phrase modifiers are similar because they are expressed using different parts of speech. In future work, failure can be used to detect when information from two different sources conflicts, which may be worth reporting to the user. However, this is beyond the scope of the current project.

3.2 Pairing Daughters

Within the merging process, there is a process called the *Pair Daughters* that applies the pairing rules. The algorithm is presented in Figure 3. This process iterates through each pairing rule with the purpose of finding one that is appropriate to apply to the two particular input nodes in question. This is determined by evaluating the constraints attached to each pairing rule. If a rule is applicable, the *Pair Daughters* process then has two primary tasks. The first, described above, is to determine which information should be combined. The second task is to add any extra content that might be found in one semantic representation and not in the other. If a branch is found to contain information not present in the other tree, it is added to the vector of paired nodes as a pairing with itself. In this way, complementary information from the two semantic representations is combined.

```
pairDaughters (t1, t2) :-
   for R ∈ ruleLibrary
      if evaluateConstraints(R, t1, t2) then
         for ⟨d1, d2⟩ ∈ potentialPairings(R, t1, t2)
            if isCompatible(d1, d2) then
               push(pairedDaughters, ⟨d1, d2⟩);
         for d ∈ (daughters(t1) ∪ daughters(t1))
            if d not present in pairedDaughters then
               push(pairedDaughters, ⟨d, d⟩);
         return pairedDaughters;
   return empty list;
```

Figure 3. The Pair Daughters Algorithm

The first task requires information regarding legitimate pairings of daughters, which are based on the relationships they have with the parent nodes being merged. These legitimate pairings are specified in the pairing rules. It is up to the *Pair Daughters* process to determine whether or not each instantiation of the specified pairing of daughters is valid by checking for semantic compatibility between the daughters.

Rule 1: Constraints: D1 POS: Verb Feature: Communication D2 POS: Verb Feature: Communication Pairings to try: (D1.speaker, D2.speaker) (D1.audience, D2.audience) (D1.speech, D2.speech)	Rule 2: Constraints: D1 POS: Verb D2 POS: Verb Pairings to try: (D1.subject, D2.subject) (D1.object, D2.object) (D1.time, D2.time)

Figure 4. Two examples of declarative pairing rules

Examples of pairing rules are presented in Figure 4. By organising the *Merge* process in terms of a simple recursive engine and a set of prioritised pairing rules, the substantive content of the merging process can be represented by means of declarative rules. When applying this process to a new domain, only the rules need to be modified.

As mentioned above, each pairing rule has a set of constraints that determine whether the rule can be applied. Elements to be paired are specified by paths of the form *Parent.RelationshipOfChild*, where the relationships are just those annotations added in the process of constructing the annotated lexical dependency tree. The rules shown here require that the elements being paired are of the same part of speech, and are at the same level of embedding; other pairing rules which relax these constraints are currently under investigation.

The set of pairing rules is ordered. When attempting to merge two nodes, the *Pair Daughters* process iterates through the rules until one can be successfully applied. The ordering is prioritised in such a way that rules with more specific preconditions are attempted first, allowing us to capture special case situations where the daughters of the two nodes should be paired in a particular way. In this way, the prioritisation mechanism provides a way of representing the relaxation of constraints on merging, with subsequent rules being less strict in their application.

There are two failure cases for the *Pair Daughters* process, and it is important to distinguish between these. The first is the case where a rule was unsuccessful but the merging engine should go on to try other pairing rules. The second is the case where the content is detected to be conflicting; in this situation, the two trees do not contain the same content after all and should not be merged. Though not currently used, this second failure case might be useful in flagging conflicting information. However, as mentioned above, conflicting information is beyond the scope of the current project.

4 Results

A full evaluation of the merging algorithm is currently underway, using a corpus of mergeable documents used in the Multigen project [McKeown et al 1999]. A preliminary

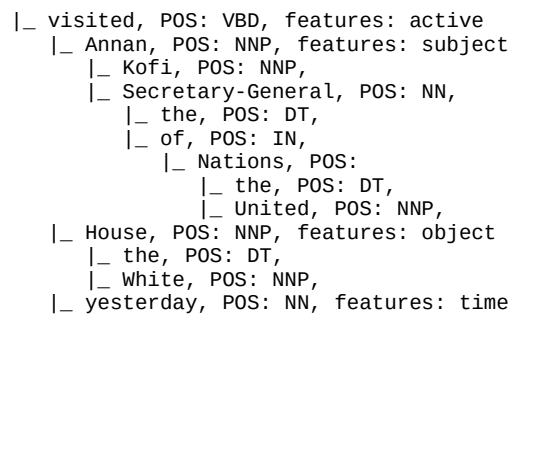
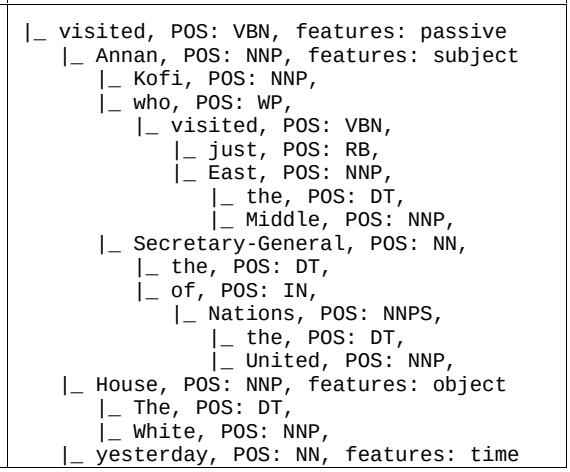
evaluation shows that the technique described here is able to successfully merge a wide range of sentences, with some exceptions which we comment on later.

Figure 5 shows the result of a merger between the passive voice sentence shown in Figure 1, and a similar sentence in the active voice. The result has merged the shared semantic content without repetition. It has also included a relative clause describing Kofi Annan found in the sentence described in Figure 1. The final sentence has been constructed by traversing the tree structure resulting from the merging process. As mentioned earlier, a description of the sentence generation process is beyond the scope of this paper.

At present, the pairing rules we have developed may not work reliably in certain situations. In particular, complex constructions that arise from certain stylistic conventions of news domain text may confuse the process which constructs the semantic representations, and hence may lead to erroneous results in the merging process. By examining the frequency of use of these stylistic conventions, we aim to identify extra domain specific linguistic knowledge that can be added to the set of pairing rules, to further optimise the summarisation system for the news domain.

Second Sentence:

Kofi Annan , the Secretary-General of the United Nations , visited the White House yesterday .

Annotated lexical dependency tree of Second Sentence	Annotated lexical dependency tree of Merged Content
	

Resulting Sentence:

The White House was visited yesterday by Kofi Annan, the Secretary-General of the United Nations, who had just visited the Middle East.

Figure 5. Merging sentences in active and passive voice

5 Conclusions

In this paper, we have presented a basic mechanism that allows the merging of two sentences with similar semantic content, without relying on the availability of deep semantic analysis. One of our main aims is to identify a level of annotated lexical dependency tree that can be constructed reliably, but which at the same time allows a merging process to produce useful results. We have demonstrated how the use of an

annotated lexical dependency tree developed from a lexical dependency tree provides the kind of input structure we need, and we have described a rule-driven merging process that operates over these representations. In addition, the algorithm described shows how it is possible to tune the system easily, given the separation of the (possibly domain-specific) pairing rules from the domain-independent merging engine.

Our current work focuses on a corpus analysis to identify specific linguistic phenomena that cause problems for the mechanism; initial results show that we need to add pairing rules to compare noun phrase modifiers expressed via different lexical and syntactic categories.

Ultimately, there is a balance to be struck between the complexity of the pairing rules and the richness of the input semantic representation used. So, for example, our rewriting of the lexical dependency tree into an annotated lexical dependency tree abstracts across the difference between active and passive forms; however, an alternative solution would have been to construct an increased number of pairing rules that could explore structures closer to the syntactic structure. Consequently, an important end-goal of this research is the determination of rules of thumb for deciding what should be represented in the annotated lexical dependency trees in order to achieve a smaller and simpler set of pairing rules.

References

Regina Barzilay, Kathleen R McKeown, and Michael Elhadad [1999] Information Fusion in the Context of Multi-Document Summarization. In *Proceedings of the 37th Annual Meeting of the ACL*, College Park, Maryland, June 1999..

Michael Collins [1996] A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, California, June 1996.

K McKeown, J Klavans, V Hatzivassiloglou, R Barzilay and E Eskin [1999] Towards multidocument summarization by reformulation: Progress and prospects. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence* (AAAI--99).

George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller [1990] Introduction to WordNet: An On-Line Lexical Database. *International Journal of Lexicography*, 3(4):235--312.

Christopher Paice [1990] Constructing Literature Abstracts by Computers: Techniques and Prospects. *Information Processing and Management*, Vol. 26, No. 1, pages 171–186.

Adwait Ratnaparkhi [1996] A Maximum Entropy Part-Of-Speech Tagger. In *Proceedings of the Empirical Methods in Natural Language Processing Conference*, May 17-18, University of Pennsylvania

James Allan, Jaime Carbonell, George Doddington, Jon Yamron, and Y. Yang [1998] Topic Detection and Tracking Pilot Study: Final Report. In *Proceedings of the Broadcast News Understanding and Transcription Workshop*, pages 194--218, 1998.