Merging Stories with Shallow Semantics

Fiona McNeill
School of Informatics
Univ. of Edinburgh
f.j.mcneill@ed.ac.uk

Harry Halpin
School of Informatics
Univ. of Edinburgh
h.halpin@ed.ac.uk

Ewan Klein
School of Informatics
Univ. of Edinburgh
ewan@inf.ed.ac.uk

Alan Bundy
School of Informatics
Univ. of Edinburgh
bundy@inf.ed.ac.uk

Abstract

We demonstrate a proof-of-concept system that uses a shallow chunking-based technique for knowledge extraction from natural language text, in particular looking at the task of story understanding. This technique is extended with a reasoning engine that borrows techniques from dynamic ontology refinement to discover the semantic similarity of stories and to merge them together.

1 Introduction

Many NLP applications would benefit from the availability of broad-coverage knowledge extraction from natural language text. Despite some recent advances in this direction (Bos et al., 2004), it is still the case that it is hard to obtain deep semantic analyses which are accurate enough to support logical inference (Lev et al., 2004).

Our problem can be stated in abstract terms. Given a formalism F for the semantic representation of natural language, such as first-order clauses in predicate calculus, and a set of sentences S, give a translation function $T(S) \rightarrow F$. The goal of such a translation would be to solve a problem P (such as paraphrasing or question-answering) where F allows P to be solved by some reasoning process, or else the domain exhibits a type of structure easily represented in the formalism F.

If we accept that current parsing technology cannot reliably combine accurate semantic analysis with robustness, then the question arises whether 'noisy' semantics can be ameliorated using some other techniques. In this paper, we adopt the hypothesis that methods drawn from dynamic ontology refinement (McNeill et al., 2004) can indeed help with this task. In the limit, we would like to be able to show that semantic content drawn from a wide variety of sources can be compared and merged to reveal the shared common ground. However, in this paper we have limited ourselves to a much more modest goal, namely merging and comparing semantic content from a set of variants of a single story.[1]

We obtained the variants by asking adults to retell the story, based on hearing the original, or reading it themselves. We have developed a system that can take in any number of such stories and produce a merged version of the stories. Our re-tellers were instructed not to elaborate upon or intentionally change the original and consequently the stories are fairly similar, not just in meaning but to an extent also in wording.

In the next two sections, we will first describe how semantic clauses are extracted from text, and second, how clauses obtained from different texts are merged.

2 Extracting Clauses from Text

The method we have adopted for extracting first-order clauses from text can be called 'semantic chunking.' This seems an appropriate term for two reasons. First, we use a syntactic chunker to identify noun groups and verb groups (i.e. non-recursive clusters of related words with a noun or verb head respectively). Second, we use a cascade of finite state rules to map from this shallow syntactic structure into first-order clauses; this cascade is conceptually very similar to the chunking method pioneered by Abney's Cass chunker (Abney, 1996).

The text processing framework we have used draws heavily on a suite of XML tools developed

[1] We used a simplified version of Oscar Wilde's fairy story *The Selfish Giant*.

for generic XML manipulation (LTXML (Thompson et al., 1997)) as well as NLP-specific XML tools (LT-TTT (Grover et al., 2000), LT-CHUNK (Finch and Mikheev, 1997)). More recently, significantly improved upgrades of these tools have been developed, most notably the program *lxtransduce*, which performs rule-based transductions of XML structures. We have used *lxtransduce* both for the syntactic chunking (based on rule developed by Grover) and for construction of semantic clauses.

The main steps in the processing pipeline are as follows:

1. Words and sentences are tokenized.
2. The words are tagged for their part of speech using the CandC tagger (Clark and Curran, 2004) and the Penn Treebank tagset.
3. Pronoun resolution is carried out using the Glencova Pronoun Resolution algorithm (Halpin et al., 2004), based on a series of rules similar to the CogNIAC engine (Baldwin, 1997), but without gender information-based rules since this is not provided by the Penn Treebank tagset.
4. The words are then reduced to their morphological stem (lemma) using Morpha (Minnen et al., 2001).
5. The *lxtransduce* program is used to chunk the sentence into verb groups and noun groups.
6. In an optional step, words are tagged as Named Entities, using the CandC tagger trained on MUC data.
7. The partially chunked sentences are selectively mapped into semantic clauses in a series of steps, described in more detail below.
8. The XML representation of the clauses is converted using an XSLT stylesheet into a more conventional syntactic format for use by Prolog or other logic-based systems.

The output of the syntactic processing is an XML file containing word elements which are heavily annotated with attributes. Following CoNLL BIO notation (Tjong et al., 2000), chunk information is recorded at the word level. Heads of noun groups and verb groups are assigned semantic tags such as `arg` and `rel` respectively. In addition, other semantically relevant forms such as conjunction, negation, and prepositions are also tagged. Most other input and syntactic information is discarded at this stage. However, we maintain a record through shared indices of which terms belong to the same chunks. This is used, for instance, to build coordinated arguments.

Regular expressions over the semantically tagged elements are used to compose clauses, using the heuristic that an `arg` immediately preceding a `pred` is the subject of the clause, while `arg`s following the `pred` are complements. Since the heads of verb groups are annotated for voice, we can treat passive clauses appropriately, yielding a representation that is equivalent to the active congener. We also implement simple heuristics that allow us to capture simple cases of control and verb phrase ellipsis in many cases.

3 Knowledge Refinement and Merging

Once the clauses have been extracted from the text, each story becomes a list of predicates, representing verbs, each with a number of arguments (possibly zero), representing nouns. Two stories can thus be compared by considering how the clauses from one story relate to the clauses from another story. This is done both by considering how the predicates (verbs) from one story relate to those from another story and also by considering the arguments (nouns) that these related predicates connect. This allows us to consider not just the similarity of the words used in the story but also, to some extent, the structure of the sentences.

The aim of merging is to build up, initially, a *working merged story* that includes all aspects of each story so far; then, when all stories have been merged, to refine the working merged story by removing aspects of it that are considered to be peripheral to the main core. The output is a single story in the same format as the inputs, and which reflects common elements from across the set of variants.

If text is represented in such clause form, then the number of ways in which these clausal representations of the story can differ is strictly limited. Clauses have only two attributes: predicates and arguments. The predicates may find an exact match, an inexact match or no match. If the predicates find some kind of match, their arguments can then be examined. Each of these will find an exact, inexact, or no match with the corresponding ar-

gument in the related predicate; additionally, their ordering and number may be different. Thus it is possible to create an exhaustive list of the possible differences between clauses. We currently consider only WordNet information concerning synonyms, hypernyms and hyponyms when determining matches: we do not perform inference using antonyms, for example, nor do we consider implication cases.

The techniques that are used in the merging process were inspired by work on dynamic ontology refinement (McNeill et al., 2004), which deals with the problem of reasoning failure caused by small inconsistencies between largely similar ontologies. This is achieved by analysing ontological objects that were expected to, but do not, match, and diagnosing and patching these mismatches through consideration of how they differ through reasoning. Examples of mismatches that are common between similar ontologies are changed names of predicates (often to sub- or super-types), changed arity, and changed types of arguments. These types of ontological mismatches are currently handled by our system, since in the domain of stories people often use different names or different levels of description for things. The application of these techniques for determining differences and similarities between the story representations therefore forms the basis of the merging process.

In order to merge a new story with the current working merged story (WMS), the facts in the WMS are examined one by one in an attempt to match them to a fact in the story to be merged. Such a match may be exact (the predicates and all their arguments are identical), inexact (the predicates have the same name but their arguments differ), similar (the predicates are synonyms) or related (the predicates are hyponyms or hypernyms). Information about synonyms, hyponyms and hypernyms is extracted from WordNet and used as the type hierarchy for our refinement (see Section 5; for an explanation of our usage of WordNet, see the WordNet project (Miller, 1995) for general details). Another potential kind of match is where the arguments match but no link can be found between the predicates; however, this is not considered in the current system. If a match of any kind is found for a predicate from the WMS, the predicate from the new story with which it matches is appended to its entry in the WMS. Each entry in the WMS is annotated with a score to indicate in how many stories a match of some kind has been found for it. For example, an entry in the WMS, `([1] play(child))` may find an inexact match with `cavort(child)` in a new story, to create an entry of `([2] play(child), cavort(child))` in the new WMS.

Once all the facts in the WMS have been examined and matched where possible, there will usually be some facts in the story to be merged that have not been found as a match for anything in the WMS. These should not be ignored; they have no match with anything thus far, but it may be that stories to be merged in future will find a match with them. Thus, they are added to the merged story with a score of 1. The initial merged story is found by simply annotating the first story to be merged so that each entry has a score of 1. It is possible, but not necessary, to provide a range value to the merging process, so that matches are only sought in the given vicinity of the fact in the WMS. If no range value is given, this is set to be arbitrarily large so that all of the story to be merged is examined.

Once all of the stories to be merged have been examined, we have a complete WMS, which needs to be processed to produce the merged output. A threshold value is used to determine which of these should be immediately discarded: anything with a score less than the threshold. Those with a score of more than or equal to the threshold must be processed so that each is represented by a single fact, rather than a list of different versions of the fact. If all versions of the fact are identical, this single version is the output. Otherwise, both a 'canonical' name and 'canonical' arguments for a fact must be calculated. In the current system, a simple approach is taken to this. For the predicate, if there is more than one version, then the most commonly occurring one is chosen as the canonical representative. If there are two or more that are jointly the most commonly occurring, then the most specific of the names is chosen (i.e., the one that is lowest in the class hierarchy). When choosing the arguments for the merged predicate, any argument that has a match in at least one other version of the predicate is included. If the match is inexact — i.e., the arguments are not of the same class, but are of related classes — then the most specific of the classes is chosen.

4 Worked Example

We now work through a simplified example to illustrate the merging process. Consider the following three lists of facts, which are drawn from different retellings of our example story:

Story 1: `come(child), play(garden), visit(friend), forget(friend), come(giant), yell(child)`

Story 2: `go(giant), visit(friend), be(giant), come(child), play(garden)`

Story 3: `be(giant), come(giant), play(garden), bellow(child,anger,giant), happy(giant)`

The first WMS (working merged story) is produced by marking up the first story:

```
([1] come(child)),
([1] play(garden)),
([1] visit(friend)),
([1] forget(friend)),
([1] come(giant)),
([1] yell(child))
```

This is then merged with the second story. The first fact of Story 1, `come(child)`, matches exactly with the fourth fact of the Story 2; the fifth fact matches inexactly with the fourth fact of the Story 2, and so on. The resulting WMS is:

```
([2] come(child), come(child)),
([2] play(garden), play(garden)),
([2] visit(friend) visit(friend)),
([1] forget(friend)),
([1] come(giant)),
([1] yell(child)),
([1] come(giant)),
([1] go(giant)),
([1] be(giant))
```

This is then merged with Story 3 to produce:

```
([3] come(child), come(child),
     come(giant)),
([3] play(garden), play(garden),
     play(garden)),
([2] visit(friend) visit(friend)),
([1] forget(friend)),
([1] come(giant)),
([2] yell(child),
     bellow(child,anger,giant)),
([1] come(giant)),
([1] go(giant)),
([2] be(giant), be(giant)),
([1] happy(giant))
```

We then proceed to merge all the automatically extracted knowledge representations of the three stories. To create the output merged story, those predicates with a score of 1 are ignored. The others are each merged to produce a single predicate. For example, `([3] come(child), come(child), come(giant))` becomes `come(child)`: `giant` does not match with any other argument and is dropped. `([2] yell(child), bellow(child,anger,giant))` becomes `yell(child)` because `yell` is a subclass of `bellow`, and thus preferred, and `child` is the only argument that has a match in both facts. Thus the resulting merged story is:

```
come(child),
play(garden),
visit(friend),
yell(child),
be(giant)
```

It is clear from this example that our current approach is, at times, naive. For example, the decisions about which arguments to include in output facts, and how to order facts that are unmatched in working merged stories could be made significantly more effective. We view this current system as a proof of concept that such an approach can be useful; we certainly do not consider the system to be complete approach to the problem. Further work on the system would result in improved performance.

5 Extracting Ontological Information from WordNet

In order to perform some of the tasks involved in merging and matching, it is necessary to have information about how words are related. We extract this information from WordNet by getting the ontology refinement engine to call WordNet with a word and retrieve both its synset (i.e., synomym set) and its hyponyms and hypernyms. We collect these for every sense of the word, since our natural language pipeline currently does not include word-sense disambiguation. When it is necessary during the merging process to obtain information about whether two words are related (i.e., when two words do not match exactly), we extract synonym and hyper/hyponym information from WordNet for these words and examine it to discover whether an exact match exists or not. We treat synonyms as equivalent, and we treat hypernym and hyponym synsets as the super- and subtype of a word respectively, and then traverse the type hierarchy for exact matches. To avoid spurious equivalence we use a bound to restrict the search, and from our experience a bound of two type-levels in either direction and a stop-list of 'upper ontology' types yields good results.

6 Related Work

This work breaks some new ground by being the first to use dynamic refinement to compare and merge information from natural language texts. In general, recent work in natural language processing has currently relied heavily on 'purely statistical' methods for tasks such as text similarity and

summarization. However, there is also a rich logical tradition in linguistic semantics, and work in this vein can bring the two closer together.

Current work in story understanding is focusing on the use of logical forms, yet these are not extracted from text automatically (Mueller, 2003). The natural language processing and story conversion pipeline are improvements over a pipeline that was shown to successfully compare stories in a manner similar to a teacher (Halpin et al., 2004).

The merging task is a more logic-based approach than similar techniques like information fusion used in multi-document summarization (Barzilay et al., 1999). Our approach has some features in common with (Wan and Dale, 2001), however, we have chosen to focus exclusively on merger at the semantic level, rather than trying to also incorporate syntactic structure.

There has also been a revival in using weighted logical forms in structured relational learning, such as Markov Logic Networks (Domingos and Kok, 2005), and this is related to the scoring of facts used by the current system in merging texts. As mentioned at the beginning of this paper, the conversion of unrestricted text to some logical form has experienced a recent revival recently (Bos et al., 2004). Although our approach deliberately ignores much semantic detail, this may be compensated for by increased robustness due to the reliance on finite-state methods for semantic translation and chunking.

7 Further Work

The work we have done thus far suggests many avenues for further work. One obvious improvement would be to enable the system to deal with more complex input, so that stories could be represented with nested predicates and with a more complex notion of time. Time can be conceived of ontologically in a number of differing manners with highly differing properties, and relating these notions of time to the extraction of tense (which the *lxtransduce*-based chunker currently does automatically) would be a fruitful task (Hayes, 1996). Making decisions about what to include in a merged story is currently done in a fairly naive manner. Further research into what constituted a good merged story and under what circumstances it is advantageous to be sceptical or generous as to what should be included in the merged story, would allow this to become much more effective. Once the system has been suitably improved, it should be tested on a more complex domain, such as news stories. Finally, the primary benefit of the use of knowledge representation is the possibility of using inference. The current system could easily take advantage of an external knowledge-base of domain-specific facts and rules to aid refinement and merging.

We have not implemented a baseline for the system using purely word-based statistical features, such as reducing the words to the their morphological stem and then using WordNet synsets to compare the stories without any predicate structure. This is because at this stage in development the extraction of the correct clauses is itself the goal of the task. If connected to larger system, comparison with a purely statistical model would be useful. However, we would hazard a guess that in the domain of computational story understanding, it is unlikely that purely statistical methods would work well, since stories by their nature consist of events involving the actions of characters in a particular temporal order, and this type of structural complexity would seem to be best accounted for by some structure-preserving features that attempt to model and extract these events explicitly.

8 Conclusion

Although many questions remain unanswered, the development of the current system demonstrates that this kind of refinement-based approach to matching and merging texts can be produce promising results. Much could be done to improve the effectiveness of both the clause extraction and the merging components of the system, and the breadth of task that these techniques have been tested on remains very narrow. Nevertheless, this work represents a reasonably successful first investigation of the problem, and we intend to use it as the basis for further work.

References

Steven Abney. 1996. Partial parsing via finite-state cascades. *Natural Language Engineering*, 2(4):337–344.

Breck Baldwin. 1997. CogNIAC: A High Precision Pronoun Resolution Engine. In *Operational Factors in Practical, Robust Anaphora Resolution for Unrestricted Texts (ACL-97 workshop)*, pages 38–45.

R. Barzilay, K. McKeown, and M. Elhadad. 1999. Information fusion in the context of multi-document summarization. In *In Proceedings of Association for*

Computational Linguistics, pages 550–557, Maryland.

Johan Bos, Stephen Clark, Mark Steedman, James Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *In Proceedings of the 20th International Conference on Computational Linguistics (COLING '04)*, pages 1240–1246, Geneva, Switzerland.

Stephen Clark and James Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL 2004)*, pages 104–111, Barcelona, Spain.

Pedro Domingos and Stanley Kok. 2005. Learning the structure of Markov Logic Networks. In *Proceedings of the International Conference on Machine Learning*, pages 441–448, Bonn.

Steve Finch and Andrei Mikheev. 1997. A workbench for finding structure in texts. In Walter Daelemans and Miles Osborne, editors, *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*. Washington D.C.

Claire Grover, Colin Matheson, Andrei Mikheev, and Marc Moens. 2000. LT TTT—a flexible tokenisation tool. In *LREC 2000—Proceedings of the 2nd International Conference on Language Resources and Evaluation*, pages 1147–1154.

Harry Halpin, Johanna Moore, and Judy Robertson. 2004. Automatic analysis of plot for story rewriting. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 127–133, Barcelona, Spain.

Pat Hayes. 1996. A catalog of temporal theories. Technical Report UIUC-BI-AI-96-01, University of Illinois.

Iddo Lev, Bill MacCartney, Christopher D. Manning, and Roger Levy. 2004. Solving logic puzzles: From robust processing to precise semantics. In *2nd Workshop on Text Meaning and Interpretation at ACL 2004*, pages 9–16.

Fiona McNeill, Alan Bundy, and Chris Walton. 2004. Facilitating agent communication through detecting, diagnosing and refining ontological mismatch. In *Proceedings of the KR2004 Doctoral Consortium*. AAAI Technical Report.

George Miller. 1995. Wordnet: A lexical database for english. *Communications of the ACM*, 11(38):39–41.

Guido Minnen, John Carroll, and David Pearce. 2001. Applied morphological processing of English. *Natural Language Engineering*, 7(3):207–203.

Erik T. Mueller. 2003. Story understanding through multi-representation model construction. In Graeme Hirst and Sergei Nirenburg, editors, *Text Meaning: Proceedings of the HLT-NAACL 2003 Workshop*, pages 46–53, East Stroudsburg, PA. Association for Computational Linguistics.

Henry Thompson, Richard Tobin, David McKelvie, and Chris Brew. 1997. LT XML: Software API and toolkit for XML processing.

Erik F. Tjong, Kim Sang, and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of the Conference on Natural Language Learning (CoNLL-2000)*. Lisbon, Portugal.

Stephen Wan and Robert Dale. 2001. Merging sentences using shallow semantic analysis: A first experiment. In *Proceedings of the 2001 Australasian Natural Language Processing Workshop*, Sydney, April. Macquarie University.