Information Extraction via Path Merging

Robert Dale[1], Cecile Paris[2], and Marc Tilbrook[1]

[1] Centre for Language Technology, Macquarie University, Sydney
www.clt.mq.edu.au
[2] Intelligent Interactive Technology Group, CSIRO, Sydney
www.cmis.csiro.au/iit

Abstract. In this paper, we describe a new approach to information extraction that neatly integrates top-down hypothesis driven information with bottom-up data driven information. The aim of the KELP project is to combine a variety of natural language processing techniques so that we can extract useful elements of information from a collection of documents and then re-present this information in a manner that is tailored to the needs of a specific user. Our focus here is on how we can build richly structured data objects by extracting information from web pages; as an example, we describe our methods in the context of extracting information from web pages that describe laptop computers. Our approach, which we call **path-merging**, involves using relatively simple techniques for identifying what are normally referred to as named entities, then allowing more sophisticated and intelligent techniques to combine these elements of information: effectively, we view the text as providing a collection of jigsaw-piece-like elements of information which then have to be combined to produce a representation of the useful content of the document. A principle goal of this work is the separation of different components of the information extraction task so as to increase portability.
Keywords: Natural language understanding, natural language generation

1 Introduction

Information Extraction (IE [1–3]; the process of identifying a pre-specified set of key data elements from a free-text data source) is widely recognised as one of the more successful spin-off technologies to come from the field of natural language processing. The DARPA-funded Message Understanding Conferences (see, for example, [4]) resulted in a number of systems that could extract from texts, with reasonable results, specific information about complex events such as terrorist incidents or corporate takeovers. In each case, the task is manageable because (a) some other means has determined that the document being analysed falls within the target domain, and (b) the key information required is typically only a very small subset of the content of the document. A major component task is **named entity recognition** [5, 6], whereby people, places and organizations are located and tracked in texts; other processing can then take the results of

San Salvador, 19 Apr 89 (ACAN-EFE)
Salvadoran President-elect Alfredo Cristiani condemned the terrorist killing of Attorney General Roberto Garcia Alvarado and accused the Farabundo Marti National Liberation Front (FMLN) of the crime.
...
Garcia Alvarado, 56, was killed when a bomb placed by urban guerrillas on his vehicle exploded as it came to a halt at an intersection in downtown San Salvador.
...
Vice President-elect Francisco Merino said that when the attorney general's car stopped at a light on a street in downtown San Salvador, an individual placed a bomb on the roof of the armored vehicle.

Fig. 1. A typical text used in information extraction

this process to build higher order data structures, establishing, for example, who did what to who and when.

The domain-dependency that is typical of IE systems means that the cost of porting a system to a new domain can be high. As a result, in recent years there has been a move towards IE based on statistical techniques and machine learning, and these have been quite successful for the lower level tasks of named entity identification (see, for example, [7]). However, the larger scale knowledge integration tasks involved in IE are often better served by more traditional, knowledge-based approaches. In this paper we present a framework for information extraction that focuses on these higher-level processes. Particularly in cases where the knowledge to be extracted is complex in structure, statistical approaches can only solve part of the problem; informational elements, once extracted, have somehow to be built into a larger whole. We describe a new mechanism that relies on a hand-constructed knowledge template, along with a general inference mechanism we call **path merging**, to reconcile and combine the informational elements found in a text. The template provides top-down hypotheses as to the information we might find in a text; the named entities identified in the text provide bottom-up data that is merged with these hypotheses.

2 Background

A key aspect of any information extraction task is that only a small portion of the information available in a text is important. Figure 1 shows fragments of an input text in the terrorist incident domain; Table 1 shows the data structure that might be constructed as a result of analysing this document: The general approach taken in these systems is that we first identify the named entities and then work out the relations between them. Even for relatively flat output structures such as the one shown here, domain-specific hand-crafted rules of considerable complexity are required to build the templates.

Incident: Date	19 Apr 89
Incident: Location	El Salvador: San Salvador (CITY)
Incident: Type	Bombing
Perpetrator: Individual ID	urban guerrillas
Perpetrator: Organization ID	FMLN
Perpetrator: Confidence	Suspected or Accused by Authorities: FMLN
Physical Target: Description	vehicle
Physical Target: Effect	Some Damage: vehicle
Human Target: Name	Roberto Garcia Alvarado
Human Target: Description	attorney general: Roberto Garcia Alvarado
Human Target: Effect	Death: Roberto Garcia Alvarado

Table 1. The extracted results from the text in Figure 1

The goal of the KELP[3] project is to develop technology that can extract information from a collection of web pages that describe similar things, and then collate and re-present this information in such a way as for a user to make it easy to compare those things. Suppose you are interested in purchasing a new cell phone: you could check out a consumer magazine, or visit a web site that presents comparisons of the different available models, but those sources are typically not up-to-date. You might visit the manufacturers' web pages to obtain information from the original sources, but this is a painful, slow process, and comparison is hindered by the different terminology each vendor uses; further, all these sources provide information for an 'average' visitor, rather than tailoring what they present to your particular needs and interests.

In KELP, we aim to build technology that can mine the information from these source web pages, and then, using techniques we have discussed elsewhere (see, for example, [8, 9]), re-present it in a form that is tailored to needs and interests captured in a specific user profile.

Our initial experiments have been in the context of web pages that describe laptop computers. Based on an analysis of around 120 web pages describing laptop computers, we developed a template that captures the range of information we might want to extract regarding a laptop computer, and then set about developing techniques to extract the relevant data from these web pages. Part of a typical filled template, or, as we call these structures within KELP, a **knowledge object** or KO, is shown in Figure 2. We have elided this structure to make it fit the space available; it should be obvious that the quantity and complexity of the data to be extracted is significant.

In our initial experiments, we focussed our attention on the information encoded in tables on web pages: for complex products such as laptop computers, this is a typical means of information provision, although it should not be forgotten that there is also a considerable amount of useful content presented in

[3] KELP stands for *Knowledge Extraction and Linguistic Presentation*, emphasising that the project is concerned both with natural language analysis and natural language generation techniques.

```
<?xml version="1.0" ?>
  <laptop_info>
    <laptop_id>
      <manufacturer>NEC</manufacturer>
      <series>Versa</series>
      <model>Premium PIII 1GHz</model>
    </laptop_id>
  <components>
    <cpu>
      <cpu_type>Pentium III</cpu_type>
      <cpu_speed>
        <number>1</number>
        <unit>GHz</unit>
      </cpu_speed>
    </cpu>
    <memory>
      <installed_memory>
        <number>128</number>
        <unit>MB</unit>
      </installed_memory>
      <maximum_memory>
        <number>512</number>
        <unit>MB</unit>
      </maximum_memory>
    </memory>
    <storage>
      <hard_drive>
        <number>20</number>
        <unit>GB</unit>
      </hard_drive>
      <removable>
        <cd_drive>24x</cd_drive>
      </removable>
    </storage>
  ...
</laptop_info>
```

Fig. 2. A portion of a filled knowledge object

free-form text. We started out by giving our information extraction module clues as to the locations of information-bearing tables, so that, typically, the information extractor had to first work out whether the cells of the table contained object names (for example, *Versa Premium*), attributes (*installed memory*), or values (*256Mb*), and then combine these components to produce the resulting KO. This approach worked reasonably well, largely because of the predictable layout of information within tables. However, it turns out that tables are used for many general-purpose formatting tasks on the web, and so identifying the information-bearing table in the first place is far from trivial.

3 Our Approach

In part to overcome some of the problems in locating the important tabular information, and also to enable the processing of free-form text, we decided to explore techniques that were capable of using information regardless of where in a document it is found.

Our starting point is a definition of a KO, as shown in the previous section; defined in our current system via an XML DTD, this is a hierarchical structure that specifies the nature of the data to be extracted from the source documents.

The simple key to our approach is to recognize that fragments of the text can be correlated with different parts of the KO structure. For example, we know that the manufacturer will be a company name; and we know that the hard disk capacity will be measured in Mb or Gb. Thus, we can assign type information to the leaf nodes in this structure. At the same time, words in the text can serve as indicators of particular attributes: so, for example, if a sentence contains the phrase *removable storage*, we can use this to hypothesise the presence of a particular attribute in the text. We think of each such text fragment as a piece of evidence; faced with evidence from the text of a collection of attributes and values, the goal is then to combine this information to populate a KO.

The architectural model we use thus consists of the following components.

- An **annotation resource file** provides a correlation between between arbitrarily complex textual patterns and the knowledge object constituents for which these patterns provide evidence.
- A **text scannner** processes each input document, searching for the patterns specified in the annotation resource file. Each time a match is found, this generates a hypothesis, in the form of a **path fragment** associated with a piece of text. The consequence of processing a document is thus a collection of path fragments that capture the evidence found in the document.
- The **path combiner** then takes this set of path fragments and attempts to put these together to build a complete knowledge object. Path fragments may contribute to multiple hypotheses about the object being constructed, so the combiner uses the target knowledge object template as a source of constraints on what is possible.
- In most situations, this will not produce a single KO as a result; there will still be ambiguities resulting from textual fragments providing evidence for

more than one KO constituent. At this point, we resort to a collection of **inference strategies** to resolve the ambiguities.

Of course, if we had full broad-coverage natural language processing available, then this would achieve the same result: we could just parse the entire text, carry out semantic analysis, and build a representation of the text. However, such an approach is not feasible given the current state of NLP technology, so our aim here is to build on the simpler pattern-matching approaches found in the IE literature, but to augment this with the scope for more sophisticated higher-level processing. The architectural breakdown we adopt stratifies the knowledge used in a way that supports easy maintenance and portability: the annotation resource file is a relatively simple declarative knowledge source developed anew for each domain; the text scanner and path combiner are generic components that do not embody any domain-specific knowledge; and higher-level knowledge of the domain is factored out into the KO template and the inference strategies.

Section 3.1 below explains in more detail the nature and role of paths and path equations; and Section 3.2 shows how path-merging is carried out. Section 3.3 explains how arbitrary inference can be incorporated into this process.

3.1 Paths

We can view a KO as a graph structure (and for present purposes a tree) into which extracted information can be placed. The arcs between nodes in this tree are labelled with the same names as the element tags in the XML DTD; a path is a sequence of arcs in the tree. Each attribute corresponds to path from the root of tree, and each value is a data element that resides at the end of that path. Paths may share common initial subsequences: this means that we use hierarchy in the tree to cluster related pieces of information into subtrees.

We use the notation A:B:C to notate **path fragments**; if a path is from the root of the tree, the path contains an initial ':', as in :A:B:C. If the path has a value at its end, then we use a terminal ':' in the path to indicate this, as in A:B:C:. A **path equation** indicates the value at the end of a path: for example

:laptop:iodevices:keyboard:numkeys: = 131

Each piece of evidence we find in a text can be annotated with a path fragment: this fragment can be of various kinds depending upon how strong the evidence is.

Complete Paths A path fragment can be **complete**, in which case there is no doubt that a particular value is the value of a particular attribute. So, for example, if we find the string *US keyboard* in the text, we might take this to be conclusive evidence for the following path equation:

:laptop:iodevices:keyboard:type: = US

Initial Path Fragments A path fragment can be **initial**: this means that we don't have a complete path but we do have some initial sequence of arcs in a path. So, for example, if we find the string *on-board memory*, this might correspond to the following annotation:

:laptop:memory:on-board

We don't know at this point what aspect of the on-board memory is being described; it might be size or speed, for example.

Medial Path Fragments A path fragment can be **medial**: this means that we don't have a complete path but we do have the some sequence of arcs in the middle of a path. So, for example, if we find a string like *memory capacity (maximum)*, we do not know if this corresponds to main memory, graphics memory, or perhaps some other kind of memory. This corresponds to the following path fragment:

memory:maximum:bytesize

Final Path Fragments Finally, a path fragment can be **final**. This means we have some sequence of arcs at the end of a path. So, a string like *2Gb* corresponds to the following pair of path fragments:

bytesize:unit: = Gb
bytesize:number: = 2

To operationalise these notions, the annotation resource file correlates arbitrarily complex textual patterns with path fragments. These patterns are then used by the text scanner to generate hypotheses about the information in the text, expressed in terms of the path fragments; the complete analysis of a text thus results in a set of textual clues and their corresponding hypothesised path fragments.

3.2 Path Merging

A KO consists of set of paths, and a fully instantiated KO has a value for each path that makes up the KO. We can characterise an instantiated KO by a set of path equations:

:laptop:model:manufacturer: = Dell
:laptop:model:series: = Inspiron
...
:laptop:iodevices:mouse:type: = optical
:laptop:iodevices:mouse:numbuttons: = 3
...

From a text, we derive a set of path fragments. From the examples in Section 3.1 above, we would have the following set of path fragments:

:laptop:iodevices:keyboard:type: = US
:laptop:memory:on-board
memory:maximum:bytesize
bytesize:unit: = Gb
bytesize:number: = 2

Our goal is therefore to take this collection of path fragments and to derive from them a set of path equations that define an instantiated KO.

Formally there are many combinations of path fragments that we could entertain.[4] A number of cases need to be considered.

First, we do not have to do anything to path fragments which are complete; note, however, that we do not want to just forget about these, since their presence may rule out some other possible combinations (in the example above, if we are tempted to merge two path fragments that would give us a different keyboard type, then we have a good reason for not doing this).

Second, two path fragments may share some arcs: so, for example, in the above, a possible combination results in

memory:maximum:bytesize:unit: = Gb

This is a possible combination but not a necessary one: since arc labels are not unique, it's possible that this particular bytesize:unit fragment does not belong with the memory:maximum:bytesize fragment.

Third, from a formal point of view, any pair of paths can be combined, with the exception that an initial path can only appear at the front of a combined path, and a terminal path can only appear at the end of a combined path. In such cases, where there is no overlap, there is essentially missing material in the middle. We indicate missing material using '...'. So, we might have a combined path that looks something like the following':

:laptop:memory:...:bytesize:unit: = Gb

This is a case where we have some possible evidence for a memory size but we don't know if it is the standard on-board memory, the expanded-to-maximum memory size, or some other aspect of memory size. [5] Of course, not all formally possible combined paths are actually possible. The KO definition provides a way of ruling out impossible path combinations. Our technique for filtering the paths is as follows.

First, we take the set of paths that constitute a KO, called the KO **path set**; this will look something like the following:

[4] The ideas discussed here have been strongly influenced by work in graph-based unification [10], although we do not currently make use of a unification engine in our approach.

[5] Note that the presence of medial paths makes the situation more difficult than it would otherwise be; rather than just pairs of fragments being combined, in principle any number of medial path fragments can be placed between pairs of initial and final paths.

:laptop:model:manufacturer:
:laptop:model:series:
...
:laptop:iodevices:mouse:type:
:laptop:iodevices:mouse:numbuttons:
...

Then, we take the set of path fragments derived from the document. We first separate out the medial paths and the complete paths, leaving the initial paths and final paths. We then produce all possible initial × final combinations, resulting in what we call the IF **path set**. Each element of the IF path set has the form

:A:B:C:...:X:Y:Z:

We then compare each element of the IF path set against the KO path set. Any match of an IF path against the KO path set constitutes an **instantiation**: it provides a possible substitution for the '...' part. Note that any IF path may result in multiple instantiations. If an IF path results in no instantiations, then it is not completable given the KO definition, and can be discarded. The remaining IF paths make up the **filtered** IF **path set**; and each element of this set may correspond to multiple instantiations. We notate instantiations as follows:

:A:B:C:[P:Q:R]:X:Y:Z:

This indicates that P:Q:R is a possible completion derived from the KO path set. In effect, material between square brackets is hypothesized on the basis of top-down knowledge; we have not extracted direct evidence for it from the text.

Next we take the medial paths and see if they support any of these instantiations by matching them against the instantiations. Again, a medial path may support multiple instantiations. Note that a medial path may actually overlap with the initial or final path in an instantiation.

In the abstracted example used here, suppose we have the following medial fragments available: Q, P:Q, P:Q:R, C:P, and R:X. When a medial path matches one of our instantiations, we notate this by moving the square brackets, echoing the idea that the square brackets indicate material for which we have no direct evidence, and each medial path adds some evidence. The effect of this process is shown by means of the **instantiation equations** in Figure 3. The results here correspond to a **filtered** IMF **path set**: a set of paths built from the initial, medial and final path fragments in the text, and filtered using the KO path set. Note that any given initial, final or medial path fragment may figure in more than one element of the filtered IMF path set, which is to say that it can contribute evidence to more than one instantiation.

For our present purposes, we make the assumption that each path fragment derived from the text should only actually contribute to one instantiated path.[6]

[6] This is an important assumption to make processing more straightforward, but note that it may not actually hold: in an expression like *the main processor and the co-processor both have 512Mb of dedicated RAM*, the *512Mb of dedicated RAM* actually contributes to two instantiated paths.

:A:B:C:[P:Q:R]:X:Y:Z: + Q = :A:B:C:[P]:Q:[R]:X:Y:Z:
:A:B:C:[P:Q:R]:X:Y:Z: + P:Q = :A:B:C:P:Q:[R]:X:Y:Z:
:A:B:C:[P:Q:R]:X:Y:Z: + P:Q:R = :A:B:C:P:Q:R:X:Y:Z:
:A:B:C:[P:Q:R]:X:Y:Z: + C:P = :A:B:C:P:[Q:R]:X:Y:Z:
:A:B:C:[P:Q:R]:X:Y:Z: + R:X = :A:B:C:[P:Q]:R:X:Y:Z:

Fig. 3. Instantiation Equations

We want to now reduce the set of instantiations in the filtered IMF path set so that we end up with a set where each I, M or F element only plays a role once. Each path in the filtered IMF path set can be thought of as being a combination of its contributing I, M and F fragments; some combinations are more likely than others. We therefore need to assess each combination, and where there is a competing demand for a piece of evidence, determine which of the alternative uses of that evidence is most likely.

3.3 Adding Reasoning

So far we have constructed a set of hypotheses regarding the information contained in a text using a combination of bottom-up knowledge from the textual source itself, and top-down knowledge from the KO. As we have seen above, this does not necessarily result in a completely instantiated KO, and so we may need to add to this process heuristics that allow us to choose between competing hypotheses.

In our present work, we have only begun to explore how this might be approached. Note that the architectural separation we have adopted allows the incorporation at this stage of arbitrary intelligence to the process, thus focussing the knowledge-based processing in one place; here, we describe the incorporation of a simple heuristic based on a distance metric.

Ultimately, where we have competing instantiations, we can assign probabilities to each. One simple probability measure is based on the distance between the initial path fragment (which generally corresponds to the attribute being considered) and the final path fragment (which corresponds to the value being assigned to that attribute): we can assign scores so that those instantiations that have closer I and F evidence will score higher. Then, we select from this set a smaller set of paths that (a) uses each piece of evidence only once and (b) takes account of the scores. The result is a set of hypotheses that populate the KO using all the data located in the source document, with each piece of evidence being used once.

This is, of course, a very simple technique, but one that seems to work well on the basis of our initial experiments, at least for information extracted from free-form text. Far more elaborate heuristics could be incorporated in the same way; a key future target for us here is to incorporate heuristics that take account

of table layout information in order to determine the appropriate correlation of table row and column headers.

4 Conclusions

We have presented an approach to information extraction that separates out the different knowledge sources and types of knowledge required. The approach makes use of both top-down and bottom-up knowledge sources, and neatly partitions domain-specific and domain-independent processing: although we have not yet attempted this, we believe the mechanisms described here should be easily portable to a new domain by constructing a knowledge object template for that domain, and an appropriate annotation resource file. The text scanner and path combining modules are domain independent, as are the current inference strategies; as the work develops, we would expect the inference strategies to break down into those which are domain-dependent and those which are domain-independent.

5 Acknowledgements

We acknowledge the generous support of CSIRO in funding the work presented here, and the helpful comments of numerous colleagues in the Centre for Language Technology.

References

1. Cowie, J., Lehnert, W.: Information extraction. Communications of the ACM **39** (1996) 80–91
2. Appelt, D., Hobbs, J., Bear, J., Israel, D., Kameyana, M., Tyson, M.: Fastus: a finite-state processor for information extraction from real-world text. (1993)
3. Jackson, P., Moulinier, I.: Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorization. John Benjamins, Amsterdam (2002)
4. Defense Advanced Research Projects Agency: Proceedings of the Sixth Message Understanding Conference (MUC-6). Morgan Kaufmann (1995)
5. A Borthwick et al: Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In: Proceedings of the Sixth Workshop on Very Large Corpora. (1998) 152–160
6. Mikheev, A., Grover, C., Moens, M.: XML tools and architecture for named entity recognition. Markup Languages **1** (1999) 89–113
7. Bikel, D.M., Miller, S., Schwartz, R., Weischedel, R.: Nymble: a high-performance learning name-finder. In: Proceedings of the Fifth Conference on Applied Natural Language Processing, Morgan Kaufmann Publishers (1997) 194–201
8. Dale, R., Green, S.J., Milosavljevic, M., Paris, C., Verspoor, C., Williams, S.: Using natural language generation techniques to produce virtual documents. In: Proceedings of the Third Australian Document Computing Symposium (ADCS'98), Sydney, Australia (1998)

9. Reiter, E., Dale, R.: Building Natural Language Generation Systems. Cambridge University Press (2000)
10. Shieber, S.: An Introduction to Unification-Based Approaches to Grammar. CSLI Lecture Notes. Chicago University Press, Chicago (1986)