Modelling Events through Memory-based, Open-IE Patterns for Abstractive Summarization

Daniele Pighin
Google Inc.
biondo@google.com

Marco Cornolti[*]
University of Pisa, Italy
cornolti@di.unipi.it

Enrique Alfonseca
Google Inc.
ealfonseca@google.com

Katja Filippova
Google Inc.
katjaf@google.com

Abstract

Abstractive text summarization of news requires a way of representing events, such as a collection of pattern clusters in which every cluster represents an event (e.g., *marriage*) and every pattern in the cluster is a way of expressing the event (e.g., *X married Y, X and Y tied the knot*). We compare three ways of extracting event patterns: heuristics-based, compression-based and memory-based. While the former has been used previously in multi-document abstraction, the latter two have never been used for this task. Compared with the first two techniques, the memory-based method allows for generating significantly more grammatical and informative sentences, at the cost of searching a vast space of hundreds of millions of parse trees of known grammatical utterances. To this end, we introduce a data structure and a search method that make it possible to efficiently extrapolate from every sentence the parse sub-trees that match against any of the stored utterances.

1 Introduction

Text summarization beyond extraction requires a semantic representation that abstracts away from words and phrases and from which a summary can be generated (Mani, 2001; Spärck-Jones, 2007). Following and extending recent work in semantic parsing, information extraction (IE), paraphrase generation and summarization (Titov and Klementiev, 2011; Alfonseca et al., 2013; Zhang and Weld, 2013; Mehdad et al., 2013), the representation we consider in this paper is a large collection of clusters of event patterns. An abstractive summarization system relying on such a representation proceeds by (1) detecting the most relevant event cluster for a given sentence or sentence collection, and (2) using the most representative pattern from the cluster to generate a concise summary sentence. Figure 1 illustrates the summarization architecture we are assuming in this paper. Given input text(s) with resolved and typed entity mentions, event mentions and the most relevant event cluster are detected (first arrow). Then, a summary sentence is generated from the event and entity representations (second arrow).

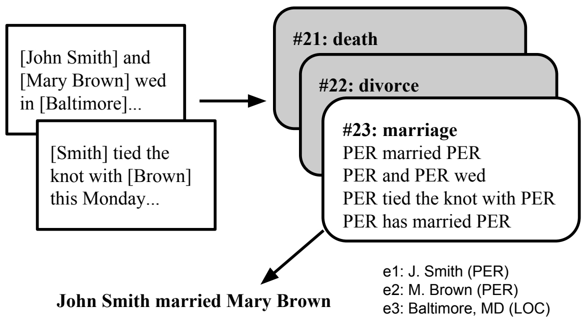
[John Smith] and [Mary Brown] wed in [Baltimore]...
[Smith] tied the knot with [Brown] this Monday...
#21: death
#22: divorce
#23: marriage
PER married PER
PER and PER wed
PER tied the knot with PER
PER has married PER
John Smith married Mary Brown
e1: J. Smith (PER)
e2: M. Brown (PER)
e3: Baltimore, MD (LOC)

Figure 1: An example of abstracting from input sentences to an event representation and generation from that representation.

However, the utility of such a representation for summarization depends on the quality of pattern clusters. In particular, event patterns must correspond to grammatically correct sentences. Introducing an incomplete or incomprehensible pattern (e.g., *PER said PER*) may negatively affect both event detection and sentence generation. Related work on paraphrase detection and relation extraction is mostly *heuristics-based* and has relied on hand-crafted rules to collect such patterns (see Sec. 2). A standard approach is to focus on binary relations between entities and extract

[*]Work done during an internship at Google Zurich.

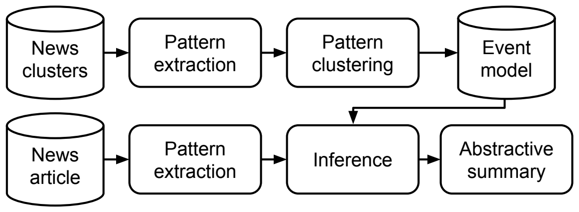

Figure 2: A generic pipeline for event-driven abstractive headline generation.

the dependency path between the two entities as an event representation. An obvious limitation of this approach is there is no guarantee that the extracted pattern corresponds to a grammatically correct sentence, e.g., that an essential prepositional phrase is retained like in *file for a divorce*.

In this paper we explore two novel, data-driven methods for event pattern extraction. The first, *compression-based* method uses a robust sentence compressor with an aggressive compression rate to get to the core of the sentence (Sec. 3). The second, *memory-based* method relies on a vast collection of human-written headlines and sentences to find a substructure which is known to be grammatically correct (Sec. 4). While the latter method comes closer to ensuring perfect grammaticality, it introduces a problem of efficiently searching the vast space of known well-formed patterns. Since standard iterative approaches comparing every pattern with every sentence are prohibitive here, we present a search strategy which scales well to huge collections (hundreds of millions) of sentences.

In order to evaluate the three methods, we consider an abstractive summarization task where the goal is to get the gist of single sentences by recognizing the underlying event and generating a short summary sentence. To the best of our knowledge, this is the first time that this task has been proposed; it can be considered as *abstractive sentence compression*, in contrast to most existing sentence compression systems which are based on selecting words from the original sentence or rewriting with simpler paraphrase tables. An extensive evaluation with human raters demonstrates the utility of the new pattern extraction techniques. Our analysis highlights advantages and disadvantages of the three methods.

To better isolate the qualities of the three extraction methodologies, all three methods use the same training data and share components of the very same summarization architecture, as shown in Figure 2: an event model is constructed by clustering the patterns extracted according to the selected extraction method. Then, the same extraction method is used to collect patterns from sentences in never-seen-before news articles. Finally, the patterns are used to query the event model and generate an abstractive summary. The three different pattern extractors are detailed in the next three sections.

Algorithm 1 HEURISTICEXTRACTOR(T, E): heuristically extract relational patterns for the dependency parse T and the set of entities E.

```
1:  /* Global constants /*
2:  global $V_p, V_c, N_p, N_c$
3:  $V_c \leftarrow \{subj, nsubj, nsubjpass, dobj, iobj, xcomp,$
4:            $acomp, expl, neg, aux, attr, prt\}$
5:  $V_p \leftarrow \{xcomp\}$
6:  $N_c \leftarrow \{det, predet, num, ps, poss, nc, conj\}$
7:  $N_p \leftarrow \{ps, poss, subj, nsubj, nsubjpass, dobj, iobj\}$
8:  /* Entry point /*
9:  $P \leftarrow \emptyset$
10: for all $C \in$ COMBINATIONS$(E)$ do
11:   $N \leftarrow$ MENTIONNODES$(T, C)$
12:   $N' \leftarrow$ APPLYHEURISTICS$(T,$ BUILDMST$(T, N))$
13:   $P \leftarrow P \cup \{$BUILDPATTERN$(T, N')\}$
14: return $P$
15: /* Procedures /*
16: procedure APPLYHEURISTICS$(T, N)$
17:   $N' \leftarrow N$
18:   while $|N'| > 0$ do
19:     $N'' \leftarrow \emptyset$
20:     for all $n \in N'$ do
21:       if $n$.ISVERB() then
22:         $N'' \leftarrow N'' \cup$ INCLUDECHILDREN$(n, V_c)$
23:         $N'' \leftarrow N'' \cup$ INCLUDEPARENT$(n, V_p)$
24:       else if $n$.ISNOUN() then
25:         $N'' \leftarrow N'' \cup$ INCLUDECHILDREN$(n, N_c)$
26:         $N'' \leftarrow N'' \cup$ INCLUDEPARENT$(n, N_p)$
27:     $N' \leftarrow N'' \setminus N'$
28: procedure INCLUDECHILDREN$(n, L)$
29:   $R \leftarrow \emptyset$
30:   for all $c \in n$.CHILDREN() do
31:     if $c$.PARENTEDGELABEL() $\in L$ then
32:       $R \leftarrow R \cup \{c\}$
33:   return $R$
34: procedure INCLUDEPARENT$(n, L)$
35:   if $n$.PARENTEDGELABEL() $\in L$ then
36:     return $\{n\}$
37:   else return $\emptyset$
```

2 Heuristics-based pattern extraction

In order to be able to work in an Open-IE manner, applicable to different domains, most existing pattern extraction systems are based on linguistically motivated heuristics. Zhang and Weld (2013) is based on REVERB (Fader et al., 2011), which uses a regular expression on part-of-speech tags to produce the extractions. An alternative system, OLLIE (Schmitz et al., 2012), uses syntactic dependency templates to guide the pattern extraction process.

The heuristics used in this paper are inspired by Alfonseca et al. (2013), who built well formed relational patterns by extending minimum spanning trees (MST) which connect entity mentions in a dependency parse. Algorithm 1 details our reimplementation of their method and the specific set of rules that we rely on to enforce pattern grammaticality. We use the standard Stanford-style set of dependency labels (de Marneffe et al., 2006). The input to the algorithm are a parse tree T and a set of target entities E. We first generate combinations of 1-3 elements of E (line 10), then for each combination C we identify all the nodes in T that mention any of the entities in C. We continue by constructing the MST of these nodes, and finally apply our heuristics to the nodes in the MST. The procedure APPLYHEURISTICS (:16) recursively grows a nodeset N' by including children and parents of noun and verb nodes in N' based on dependency labels. For example, we include all children of verbs in N' whose label is listed in V_c (:3), e.g., active or passive subjects, direct or indirect objects, particles and auxiliary verbs. Similarly, we include the parent of a noun in N' if the dependency relation between the node and its parent is listed in N_p.

3 Pattern extraction by sentence compression

Sentence compression is a summarization technique that shortens input sentences preserving the most important content (Grefenstette, 1998; McDonald, 2006; Clarke and Lapata, 2008, inter alia). While first attempts at integrating a compression module into an extractive summarization system were not particularly successful (Daumé III and Marcu, 2004, inter alia), recent work has been very promising (Berg-Kirkpatrick et al., 2011; Wang et al., 2013). It has shown that dropping constituents of secondary importance from selected sentences – e.g., temporal modifiers or relative clauses – results in readable and more informative summaries. Unlike this related work, our goal here is to compress sentences to obtain an event pattern – the minimal grammatical structure expressing an event. To our knowledge, this application of sentence compressors is novel. As in Section 2, we only consider sentences mentioning entities and require the compression (pattern) to retain at least one such mention.

Sentence compression methods are abundant but very few can be configured to produce output satisfying certain constraints. For example, most compression algorithms do not accept compression rate as an argument. In our case, sentence compressors which formulate the compression task as an optimization problem and solve it with integer linear programming (ILP) tools under a number of constraints are particularly attractive (Clarke and Lapata, 2008; Filippova and Altun, 2013). They can be extended relatively easily with both the length constraint and the constraint on retaining certain words. The method of Clarke and Lapata (2008) uses a trigram language model (LM) to score compressions. Since we are interested in very short outputs, a LM trained on standard, uncompressed text would not be suitable. Instead, we chose to modify the method of Filippova and Altun (2013) because it relies on dependency parse trees and does not use any LM scoring.

Like other syntax-based compressors, the system of Filippova and Altun (2013) prunes dependency structures to obtain compression trees and hence sentences. The objective function to maximize in an ILP problem (Eq. 1) is formulated over weighted edges in a transformed dependency tree and is subject to a number of constraints. Edge weight is defined as a linear function over a feature set: $w(e) = \mathbf{w} \cdot \mathbf{f}(e)$.

$$F(X) = \sum_{e \in E} x_e \times w(e) \qquad (1)$$

In our reimplementation we followed the algorithm as described by Filippova and Altun (2013). The compression tree is obtained in two steps. First, the input tree is transformed with deterministic rules, most of which aim at collapsing indispensable modifiers with their heads (determiners, auxiliary verbs, negation, multi-word expressions, etc.). Then a sub-tree maximizing the objective function is found under a number of constraints.

Apart from the structural constrains from the original system which ensure that the output is a valid tree, the constraints we add state that:

1. tree size in edges must be in [3, 6],
2. entity mentions must be retained,
3. subject of the clause must be retained,
4. the sub-tree must be covered by a single clause – exactly one finite verb must used.

Since we consider compressions with different lengths as candidates, from this set we select the one with the maximum averaged edge weight as the final compression. Figure 3 illustrates the use of the compressor for obtaining event patterns. Dashed edges are dropped as a result of constrained compression so that the output is *John Smith married Mary Brown* and the event pattern is *PER married PER*. Note that the root of a subclause is allowed to be the top-level node in the extracted compression.

Compared with patterns obtaines with heuristics, compression patterns should retain prepositional verb arguments whose removal would render the pattern ungrammatical. As an example consider *[C. Zeta-Jones] and [M. Douglas] filed for divorce*. The heuristics-based pattern is *PER and PER filed* which is incomplete. Unlike it, the compression-based method keeps the essential prepositional phrase *for divorce* in the pattern because the average edge weight is greater for the tree with the prepositional phrase.

4 Memory-based pattern extraction

Neither heuristics-based, nor compression-based methods provide a guarantee that the extracted pattern is grammatically correct. In this section we introduce an extraction technique which makes it considerably more likely because it only extracts patterns which have been observed as full sentences in a human-written text (Sec. 4.1). However, this memory-based method also poses a problem not encountered by the two previous methods: how to search over the vast space of observed headlines and sentences to extract a pattern from a given sentence? Our trie-based solution, which we present in the remainder of this section, makes it possible to compare a dependency graph against millions of observed grammatical utterances in a fraction of a second.

4.1 A *tree-trie* to store them all...

Our objective is to construct a compact representation of hundreds of millions of observed sentences that can fit in the memory of a standard workstation. This data structure should make it possible to efficiently identify the sub-trees of a sentence that match any complete utterance previously observed. To this end, we build a trie of dependency trees (which we call a *tree-trie*) by scanning all the dependency parses in the news training data, and index each tree in the tree-trie according to Algorithm 2. For better clarity, the process is also described graphically in Figure 4. First, each dependency tree (a) is *linearized*, resulting in a data structure that consists of two aligned sequences (b). The first sequence (*tokens*) encodes word/parent-relation pairs, while the second sequence (*structure*) encodes the offsets of parent nodes in the linearized tree. As an example, the first word "The" is a determiner ("det") for the second node (offset 1) in the sequence, which is "cat". In turn, "cat" is the subject ("nsubj") of the node in position 2, i.e., "sleeps". As described in Algorithm 2, we recursively store the token sequence in the trie, each word/relation pair being stored in a node. When the token sequence is completely consumed, we store in the current trie node the structure of the linearized tree. Combining structural information with the sequential information encoded by each path in the trie makes it possible to rebuild a complete dependency graph. Figure 4(c) shows an example trie encoding 4 different sentences. We highlighted in bold the path corresponding to the linearized form (b) of the example parse tree (a).

The figure shows that the tree contains two kinds of nodes: end-of-sentence (EOS) nodes (red) and non-terminal nodes (in blue). EOS nodes do not necessarily coincide with trie leaves, as it is possible to observe complete sentences embedded in longer ones. EOS nodes differ from non-terminal nodes in that they store one or more structural sequences corresponding to different syntactic representations of observed sentences with the same tokens.

Space-complexity and generalization. Storing all the observed sentences in a single trie requires huge amounts of memory. To make it possible to

Algorithm 2 STORE(T, I): store the dependency tree T in the tree-trie I.

```
1: /* Entry point /*
2: $L \leftarrow T$.LINEARIZE()
3: STORERECURSION($I$.ROOT(), $L$, 0)
4: return $M$
5: /* Procedures /*
6: procedure STORERECURSION($n, L, o$)
7:     if $o == L$.LENGTH() then
8:         $n$.ADDTREESTRUCTURE($L$.STRUCTURE())
9:         return
10:    if not $n$.HASCHILD($L$.TOKEN($o$)) then
11:        $n$.ADDCHILD($L$.TOKEN($o$))
12:    $n' \leftarrow n$.GETCHILD($L$.TOKEN($o$))
13:    STORERECURSION($n'$, $L$, $o + 1$)
```

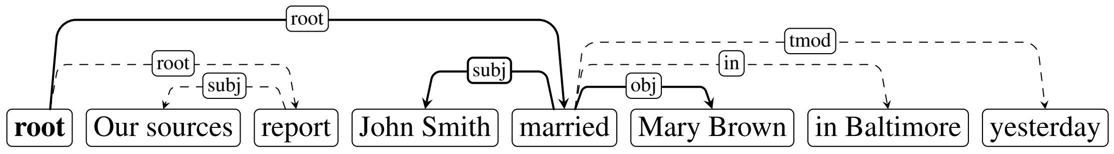

Figure 3: Transformed dependency tree with a sub-tree expressing an event pattern.

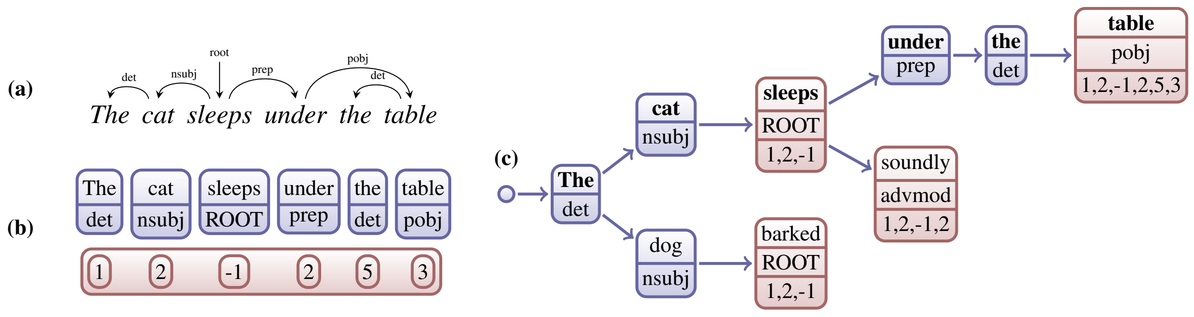

Figure 4: A dependency tree (a), its linearized form (b) and the resulting path in a trie (c), in bold.

store a complete tree-trie in memory, we adopt the following strategy. We replace the surface form of entity nodes with the coarse entity type (e.g., PER, LOC, ORG) of the entity. Similarly, we replace proper nouns with the placeholder "[P]", thus significantly reducing lexical sparsity. Then, we encode each distinct word/relation pair as a 32-bit unsigned integer. Assuming a maximum tree size of 255 nodes, we represent structure sequences as vectors of type unsigned char (8 bit per element). Finally, we store trie-node children as sorted vectors instead of hash maps to reduce memory footprint. As a result, we are able to load a trie encoding 400M input dependency parses, 170M distinct nodes and 48M distinct sentence structures in under 10GB of RAM.

4.2 ...and in the vastness match them

At lookup time, we want to use the tree-trie to identify all sub-graphs of an input dependency tree T that match at least a complete observed sentence. To do so, we need to identify all paths in the trie that match any sub-sequence s of the linearized sequence of T nodes. Whenever we encounter an EOS node e, we verify if any of the structures stored at e matches the sub-tree generated by s. If so, then we have a positive match. As a sentence might embed many shorter utterances, each input T will generally yield multiple matches. For example, querying the tree-trie in Figure 4(c) with the input tree shown in (a) would yield two results, as both *The cat sleeps* and *The cat sleeps under the table* are complete utterances stored in the trie.

Algorithm 3 LOOKUP(T, I): Lookup for matches of subset of tree T in the trie index I.

```
 1: /* Entry point /*
 2: $L \leftarrow T$.LINEARIZE()
 3: $M \leftarrow \emptyset$
 4: LOOKUPRECURSIVE($T, L, 0, I$.ROOT(), $\emptyset, M$)
 5: return $M$
 6: /* Procedures /*
 7: procedure LOOKUPRECURSIVE($T, L, o, n, P, M$)
 8:   for all $i \in [o, L$.LENGTH())$ do
 9:     if $n$.HASCHILD($L$.TOKEN($i$)) then
10:       $n' \leftarrow n$.GETCHILD($L$.TOKEN($i$))
11:       $P' \leftarrow P \cup \{i\}$
12:       for all $S \in n'$.TREESTRUCTURES() do
13:         if $L$.ISCOMPATIBLE($S, P'$) then
14:           $M \leftarrow M \cup \{T$.GETNODES($P'$)$\}$
15:       LOOKUPRECURSIVE($L, i, o+1, n', P', M$)
```

Algorithm 3 describes the lookup process in more detail. The first step consists in the linearization of the input tree T. Then, we recursively traverse the trie calling LOOKUPRECURSIVE. The inputs of this procedure are: the input tree T, its linearization L and an offset o (starting at 0), the trie node currently being traversed n (starting with the root), the set of offsets in L that constitute a partial match P (initially empty) and the set of complete matches found M. We recursively traverse all the nodes in the trie that yield a partial match with any sub-sequence of the linearized tokens of T. At each step, we scan all the tokens in L in the range $[o, L$.LENGTH()) looking for tokens matching any of the children of n. If a matching node is found, a new partial match P' is constructed by extending P with the matching token

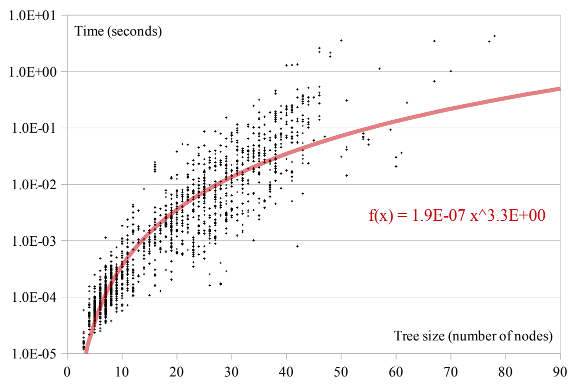

Figure 5: Time complexity of lookup operations for inputs of different sizes.

offset i (line 11), and the recursion continues from the matching trie node n' and offset i (line 15). Every time a partial match is found, we verify if the partial match is compatible with any of the tree structures stored in the matching node. If that is the case, we identify the corresponding set of matching nodes in T and add it to the result M (lines 12-14). A pattern is generated from each complete match returned by LOOKUP after applying a simple heuristic: for each verb node v in the match, we enforce that negations and auxiliaries in T depending from x are also included in the pattern.

Time complexity of lookups. Let k be the maximum fan-out of trie nodes, d be the depth of the trie and n be the size of an input tree (number of nodes). If trie node children are hashed (which has a negative effect on space complexity), then worst case complexity of LOOKUP() is $O(nk)^{d-1}$. If they are stored as sorted lists, as in our memory-efficient implementation, theoretical complexity becomes $O(nk\log(k))^{d-1}$. It should be noted that worst case complexity can only be observed under extremely unlikely circumstances, i.e., that at every step of the recursion all the nodes in the tail of the linearized tree match a child of the current node. Also, in the actual trie used in our experiments the average branching factor k is very small. We observed that a trie storing 400M sentences (170M nodes) has an average branching factor of 1.02. While the root of the trie has unsurprisingly many children (210K, all the observed first sentence words), already at depth 2 the average fan-out is 13.7, and at level 3 it is 4.9.

For an empirical analysis of lookup complexity, Figure 5 plots, in black, wall-clock lookup time as a function of tree size n for a random sample of 1,600 inputs. As shown by the polynomial regression curve (red), observed lookup complexity is approximately cubic with a very small constant factor. In general, we can see that for sentences of common length (20-50 words) a lookup operation can be completed in well under one second.

5 Evaluation

5.1 Experimental settings

All the models for the experiments that we present have been trained using the same corpus of news crawled from the web between 2008 and 2013. The news have been processed with a tokenizer, a sentence splitter (Gillick and Favre, 2009), a part-of-speech tagger and dependency parser (Nivre, 2006), a co-reference resolution module (Haghighi and Klein, 2009) and an entity linker based on Wikipedia and Freebase (Milne and Witten, 2008). We use Freebase types as fine-grained named entity types, so we are also able to label e.g. instances of *sports teams* as such instead of the coarser label ORG.

Next, the news have been grouped based on temporal closeness (Zhang and Weld, 2013) and cosine similarity (using tf·idf weights). For each of the three pattern extraction methods we used the same summarization pipeline (as shown above in Figure 2):

1. Run pattern extraction on the news.

2. For every news collection $Coll$ and entity set E, generate a set containing all the extracted patterns from news in $Coll$ mentioning all the entities in E. These are patterns that are likely to be paraphrasing each other.

3. Run a clustering algorithm to group together patterns that typically co-occur in the sets generated in the previous step. There are many choices for clustering algorithms (Alfonseca et al., 2013; Zhang and Weld, 2013). Following Alfonseca et al. (2013) we use in this work a Noisy-OR Bayesian Network because it has already been applied for abstractive summarization (albeit multi-document), it provides an easily interpretable probabilistic clustering, and training can be easily parallelized to be able to handle large training sets. The hidden events in the Bayesian network represent pattern clusters. When training is done, for each extraction pattern p_j

Original sentence	*Abstractive summary (method)*
Two-time defending overall World Cup champion Marcel Hirscher won the challenging giant slalom on the Gran Risa course with two solid runs Sunday and attributed his victory to a fixed screw in his equipment setup.	Marcel Hirscher has won the giant slalom. (**C**)
Zodiac Aerospace posted a 7.9 percent rise in first-quarter revenue, below market expectations, but reaffirmed its full-year financial targets.	Zodiac Aerospace has reported a rise in profits. (**C**)
Australian free-agent closer Grant Balfour has agreed to terms with the Baltimore Orioles on a two-year deal, the Baltimore Sun reported on Tuesday citing multiple industry sources.	Balfour will join the Baltimore Orioles. (**H**)
Paul Rudd is 'Ant-Man': 5 reasons he needs an 'Agents of SHIELD' appearance.	Paul Rudd to play Ant-Man. (**H**)
Millwall defender Karleigh Osborne has joined Bristol City on a two-and-a-half year deal after a successful loan spell.	Bristol City have signed Karleigh Osborne. (**M**)
Simon Hoggart, one of the Spectator's best-loved columnists, died yesterday after fighting pancreatic cancer for over three years.	Simon Hoggart passed away yesterday. (**M**)

Table 1: Abstraction examples from compression (**C**), heuristic (**H**) and memory-based (**M**) patterns.

Method	*Extractions*	*Abstractions*
HEURISTIC	24,630	956
COMPRESSION	15,687	657
MEMORY-BASED	11,459	967

Table 2: Patterns extracted in each method, before Noisy-OR inference.

and pattern cluster c_i, the network provides $p(p_j|c_i)$ —the probability that c_i will generate p_j— and $p(c_i|p_j)$ —the probability that, given a pattern p_j, c_i was the hidden event that generated it.

At generation time we proceed in the following way:

1. Given the title or first sentence of a news article, run the same pattern extraction method that was used in training and, if possible, obtain a pattern p involving some entities.

2. Find the model clusters that contain this pattern, $C_p = \{c_i \text{ such that } p(c_i|p) > 0\}$.

3. Return a ranked list of model patterns $output = \{(p_j, score(p_j))\}$, scored as follows:

$$score(p_j) = \prod_{c_i \in C_p} p(p_j|c_i)p(c_i|p)$$

   where p was the input pattern.

4. Replace the entity placeholders in the top-scored patterns p_j with the entities that were actually mentioned in the input news article.

In all cases the parameters of the network were predefined as 20,000 nodes in the hidden layer (model clusters) and 40 Expectation Maximization (EM) training iterations. Training was distributed across 20 machines with 10 GB of memory each.

For testing we used 37,584 news crawled during December 2013, which had not been used for training the models. Table 3 shows one pattern cluster example from each of the three trained models. The table shows only the surface form of the pattern for simplicity.

Pattern cluster (MEMORY-BASED)
organization$_1$ gets *organization*$_0$ nod for drug
organization$_1$ gets *organization*$_0$ nod for tablets
organization$_0$ approves *organization*$_1$ drug
organizations$_0$ approves *organization*$_1$'s drug
organization$_1$ gets *organization*$_0$ nod for capsules
Pattern cluster (HEURISTIC)
organization$_0$ to buy *organization*$_1$
organization$_0$ to acquire *organization*$_1$
organization$_0$ buys *organization*$_1$
organization$_0$ acquires *organization*$_1$
organization$_0$ to acquire *organizations*$_1$
organization$_0$ buys *organizations*$_1$
organization$_0$ acquires *organizations*$_1$
organization$_0$ agrees to buy *organization*$_1$
organization$_0$ snaps up *organization*$_1$
organization$_0$ to purchase *organizations*$_1$
organization$_0$ is to acquire *organization*$_1$
organization$_0$ has agreed to buy *organization*$_1$
organization$_0$ announces acquisition of *organizations*$_1$
organization$_0$ may bid for *organization*$_1$
organization$_1$ sold to *organization*$_0$
organization$_1$ acquired by *organization*$_0$
Pattern cluster (COMPRESSION)
the *sports team*$_1$ have acquired *person*$_0$ from the *sports team*$_2$
the *sports team*$_1$ acquired *person*$_0$ from the *sports team*$_2$
the *sports team*$_2$ have traded *person*$_0$ to the *sports team*$_1$
sports team$_1$ acquired the rights to *person*$_0$ from *sports team*$_2$
sports team$_2$ acquired from *sports team*$_1$ in exchange for *person*$_0$
sports team$_2$ have acquired from the *sports team*$_1$ in exchange for *person*$_0$

Table 3: Examples of pattern clusters. In each cluster c_i, patterns are sorted by $p(p_j|c_i)$.

5.2 Results

Table 2 shows the number of extracted patterns from the test set, and the number of abstractive event descriptions produced.

As expected, the number of extracted patterns using the memory-based model is smaller than with the two other models, which are based on generic rules and are less restricted in what they can generate. As mentioned, the memory-based model can only extract previously-seen structures. Compared to this model, with heuristics we can obtain patterns for more than twice more news articles. At the same time, looking at the number of summary sentences generated they are comparable, meaning that a larger proportion of the memory-based patterns actually appeared in the pattern clusters and could be used to produce summaries. This is also consistent with the fact that using heuristics the space of extracted patterns is basically unbounded and many new patterns can be generated that were previously unseen –and these cannot generate abstractions. A positive outcome is that restricting the syntactic structure of the extracted patterns to what has been observed in past news does not negatively affect end-to-end coverage when generating the abstractive summaries.

Table 1 shows some of the abstractive summaries generated with the different methods. For manually evaluating their quality, a random sample of 100 original sentences was selected for each method. The top ranked summary for each original sentence was sent to human raters for evaluation, and received three different ratings. None of the raters had any involvement in the development of the work or the writing of the paper, and a constraint was added that no rater could rate more than 50 abstractions. Raters were presented with the original sentence and the compressed abstraction, and were asked to rate it along two dimensions, in both cases using a 5-point Likert scale:

- **Readability**: whether the abstracted compression is grammatically correct.
- **Informativeness**: whether the abstracted compression conveys the most important information from the original sentence.

Inter-judge agreement was measured using the Intra-Class Correlation (ICC) (Shrout and Fleiss, 1979; Cicchetti, 1994). The ICC for readability was 0.37 (95% confidence interval [0.32, 0.41]), and for informativeness it was 0.64 (95% confidence interval [0,60, 0.67]), representing fair and substantial reliability.

Method	*Readability*	*Informativeness*
HEURISTIC	3.95	3.07
COMPRESSION	3.98	2.35
MEMORY-BASED	**4.20**	**3.70**

Table 4: Results for the three methods when rating the top-ranked abstraction.

Table 4 shows the results when rating the top ranked abstraction using either of the three different models for pattern extraction. The abstractions produced with the memory-based method are more readable than those produced with the other two methods (statistically significant with 95% confidence).

Regarding informativeness, the differences between the methods are bigger, because the first two methods have a proportionally larger number of items with a high readability but a low informativeness score. For each method, we have manually reviewed the 25 items where the difference between readability and informativeness was the largest, to understand in which cases grammatical, yet irrelevant compressions are produced. The results are shown in Table 5. *Be+adjective* includes examples where the pattern is of the form *Entity is Adjective*, which the compression-based systems extracts often represents an incomplete extraction. *Wrong inference* contains the cases where patterns that are related but not equivalent are clustered, e.g. *Person arrived in Country* and *Person arrived in Country for talks*. *Info. missing* represents cases where very relevant information has been dropped and the summary sentence is not complete. *Possibility* contains cases where the original sentence described a possibility and the compression states it as a fact, or vice versa. *Disambiguation* are entity disambiguations errors, and *Opposite* contains cases of patterns clustered together that are opposite along some dimension, e.g. *Person quits TV_Program* and *Person to return to TV_Program*.

The method with the largest drop between the readability and informativeness scores is COMPRESSION. As can be seen, many of these mistakes are due to relevant information being missing in the summary sentence. This is also the largest source of errors for the HEURISTIC system. For the MEMORY-BASED system, the drop in read-

Method	*Be+adjective*	*Wrong inference*	*Info. missing*	*Possibility*	*Disambiguation*	*Opposite*
HEURISTIC	0	7	14	3	1	0
COMPRESSION	3	10	10	0	0	2
MEMORY-BASED	0	17	4	2	0	2

Table 5: Sources of errors for the top 25 items with high readability and low informativeness.

Original sentence	*Pattern extracted (method)*	*Abstraction*
David Moyes is happy to use tough love on Adnan Januzaj to ensure the Manchester United youngster fulfils his massive potential.	David Moyes is happy. (**C**)	Fortune will start to favour David Moyes.
The Democratic People's Republic of Korea will "achieve nothing by making threats or provocation," the United States said Friday.	The United States said Friday. (**C, H**)	United States officials said Friday.
EU targets Real and Barca over illegal state aid.	EU targets Real Madrid. (**H**)	EU is going after Real Madrid.
EU warns Israel over settlement construction	EU warns Israel. (**M**)	EU says Israel needs reforms.

Table 6: Examples of compression (**C**), heuristic (**H**) and memory-based (**M**) patterns that led to abstractions with high readability but a low informativeness score. Both incomplete summary sentences and wrong inferences can be observed.

ability score is much smaller, so there were less of these examples. And most of these examples belong to the class of wrong inferences (patterns that are related but not equivalent, so we should not abstract one of them from the other, but they were clustered together in the model). Our conclusion is that the examples with missing information are not such a big problem with the MEMORY-BASED system, as using the trie is an additional safeguard that the generated titles are complete statements, but the method is not preventing the wrong inference errors so this class of errors become the dominant class by a large margin.

Some examples with high readability but low informativeness are shown in Table 6.

6 Conclusions

Most Open-IE systems are based on linguistically-motivated heuristics for learning patterns that express relations between entities or events. However, it is common for these patterns to be incomplete or ungrammatical, and therefore they are not suitable for abstractive summary generation of the relation or event mentioned in the text.

In this paper, we describe a memory-based approach in which we use a corpus of past news to learn valid syntactic sentence structures. We discuss the theoretical time complexity of looking up extraction patterns in a large corpus of syntactic structures stored as a trie and demonstrate empirically that this method is effective in practice. Finally, the evaluation shows that summary sentences produced by this method outperform heuristics and compression-based ones both in terms of readability and informativeness. The problem of generating incomplete summary sentences, which was the main source of informativeness errors for the alternative methods, becomes a minor problem with the memory-based approach. Yet, there are some cases in which also the memory based approach extracts correct but misleading utterances, e.g., a pattern like *PER passed away* from the sentence *PER passed the ball away*. To solve this class of problems, a possible research direction would be the inclusion of more complex linguistic features in the tree-trie, such as verb subcategorization frames.

As another direction for future work, more effort is needed in making sure that no incorrect inferences are made with this model. These happen when a more specific pattern is clustered together with a less specific pattern, or when two non-equivalent patterns often co-occur in news as two events are somewhat correlated in real life, but it is generally incorrect to infer one from the other. Improvements in the pattern-clustering model, outside the scope of this paper, will be required.

References

Enrique Alfonseca, Daniele Pighin, and Guillermo Garrido. 2013. HEADY: News headline abstraction through event pattern clustering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics,* Sofia, Bulgaria, 4–9 August 2013, pages 1243–1253.

Taylor Berg-Kirkpatrick, Dan Gillick, and Dan Klein. 2011. Jointly learning to extract and compress. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics,* Portland, OR, 19–24 June 2011.

Domenic V Cicchetti. 1994. Guidelines, criteria, and rules of thumb for evaluating normed and standardized assessment instruments in psychology. *Psychological Assessment*, 6(4):284.

James Clarke and Mirella Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 31:399–429.

Hal Daumé III and Daniel Marcu. 2004. A tree-position kernel for document compression. In *Proceedings of the 2004 Document Understanding Conference held at the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics,,* Boston, Mass., 6–7 May 2004.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the 5th International Conference on Language Resources and Evaluation,* Genoa, Italy, 22–28 May 2006, pages 449–454.

Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying relations for open information extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing,* Edinburgh, UK, 27–29 July 2011, pages 1535–1545.

Katja Filippova and Yasemin Altun. 2013. Overcoming the lack of parallel data in sentence compression. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing,* Seattle, WA, USA, 18–21 October 2013, pages 1481–1491.

Dan Gillick and Benoit Favre. 2009. A scalable global model for summarization. In *Proceedings of the ILP for NLP Workshop,* Boulder, CO, June 4 2009, pages 10–18.

Gregory Grefenstette. 1998. Producing intelligent telegraphic text reduction to provide an audio scanning service for the blind. In *Working Notes of the Workshop on Intelligent Text Summarization,* Palo Alto, Cal., 23 March 1998, pages 111–117.

Aria Haghighi and Dan Klein. 2009. Simple coreference resolution with rich syntactic and semantic features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing,* Singapore, 6-7 August 2009, pages 1152–1161.

Inderjeet Mani. 2001. *Automatic Summarization.* John Benjamins, Amsterdam, Philadelphia.

Ryan McDonald. 2006. Discriminative sentence compression with soft syntactic evidence. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics,* Trento, Italy, 3–7 April 2006, pages 297–304.

Yashar Mehdad, Giuseppe Carenini, and Frank W. Tompa. 2013. Abstractive meeting summarization with entailment and fusion. In *Proceedings of the 14th European Workshop on Natural Language Generation,* Sofia, Bulgaria, 8–9 August, 2013, pages 136–146.

David Milne and Ian H. Witten. 2008. An effective, low-cost measure of semantic relatedness obtained from Wikipedia links. In *Proceedings of the AAAI 2008 Workshop on Wikipedia and Artificial Intelligence,* Chicago, IL, 13-14 July, 2008.

Joakim Nivre. 2006. *Inductive Dependency Parsing.* Springer.

Michael Schmitz, Robert Bart, Stephen Soderland, Oren Etzioni, et al. 2012. Open language learning for information extraction. In *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing,* Jeju Island, Korea, 12–14 July 2012, pages 523–534.

Patrick E Shrout and Joseph L Fleiss. 1979. Intraclass correlations: uses in assessing rater reliability. *Psychological bulletin*, 86(2):420.

Karen Spärck-Jones. 2007. Automatic summarising: A review and discussion of the state of the art. Technical Report UCAM-CL-TR-679, University of Cambridge, Computer Laboratory, Cambridge, U.K.

Ivan Titov and Alexandre Klementiev. 2011. A Bayesian model for unsupervised semantic parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics,* Portland, OR, 19–24 June 2011, pages 1445–1455.

Lu Wang, Hema Raghavan, Vittorio Castelli, Radu Florian, and Claire Cardie. 2013. A sentence compression based framework to query-focused multi-document summarization. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics,* Sofia, Bulgaria, 4–9 August 2013, pages 1384–1394.

Congle Zhang and Daniel S. Weld. 2013. Harvesting parallel news streams to generate paraphrases of event relations. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing,* Seattle, WA, USA, 18–21 October 2013, pages 1776–1786.