Text Generation for Abstractive Summarization

Pierre-Etienne Genest, Guy Lapalme
RALI-DIRO
Université de Montréal
P.O. Box 6128, Succ. Centre-Ville
Montréal, Québec
Canada, H3C 3J7
{genestpe,lapalme}@iro.umontreal.ca

Abstract

We have begun work on a framework for abstractive summarization and decided to focus on a module for text generation. For TAC 2010, we thus move away from sentence extraction. Each sentence in the summary we generate is based on a document sentence but it usually contains a smaller amount of information and uses fewer words. The system uses the output of a syntactic parser for a sentence and then regenerates part of the sentence using a Natural Language Generation engine. The sentences of the summary are selected among regenerated sentences based on the document frequency of contained words, while avoiding redundancy. Date and location were handled and generated especially for cluster categories 1 and 2. Even though our initial scores were not outstanding, we intend to continue work on this approach in the coming years.

1 Introduction

This year's (new) summarization task was Guided Summarization. Each multi-document cluster was tagged as being one of five categories. Each category had aspects that were meant to guide the summarization towards answering specific questions. The main reason invoked for designing this new task definition was to prompt the development of richer, information-focused approaches to summarization, and, hopefully, approaches that are different from pure extraction.

Also keeping in mind our conclusions of the HexTac experiment (Genest et al., 2009b), that showed intrinsic limitations to pure extraction techniques, we decided to take on this year's challenge by going "all in" in a new direction. We developed an exclusively abstractive system based on text generation.

An abstractive approach requires a representation of the text that serves as an intermediate step before the generation of new sentences. Ours is based on the concept of *Information Items* (InIt). An InIt is the smallest element of coherent information that we can extract from a text or sentence. It can be something as simple as some entity's property or as complex as a whole description of an event or action. We believe that such a representation could eventually allow for directly answering guided topic aspects, by generating sentences that address specific information needs.

This approach has the advantage of generating typically short, information-focused sentences to produce a coherent, information rich, and less redundant summary. However, the difficulties are great: it is difficult for a machine to properly extract information from sentences at an abstract level, and text generated from noisy data will often be flawed. Generating sentences that do not all sound similar and generic will be an added challenge that we have for now circumvented by re-using the original sentence structure to a large extent. We do believe that efforts in abstractive summarization need to be made and constitute the future of summarization research, even though we are aware that the unavoidable diffi-

Original Sentence The Cypriot airliner that crashed in Greece may have suffered a sudden loss of cabin pressure at high altitude, causing temperatures and oxygen levels to plummet and leaving everyone aboard suffocating and freezing to death, experts said Monday.

Information Items

1. airliner – crash – *null* (Greece, August 15, 2005)
2. airliner – suffer – loss (Greece, August 15, 2005)
3. loss – cause – *null* (Greece, August 15, 2005)
4. loss – leave – *null* (Greece, August 15, 2005)

Generated Sentences

1. A Cypriot airliner crashed.
2. A Cypriot airliner may have suffered a sudden loss of cabin pressure at high altitude.
3. A sudden loss of cabin pressure at high altitude caused temperatures and oxygen levels to plummet.
4. A sudden loss of cabin pressure at high altitude left everyone aboard suffocating and freezing to death.

Generated Sentence in the Summary On August 15, 2005, a Cypriot airliner may have suffered a sudden loss of cabin pressure at high altitude in Greece.

Original Sentence At least 25 bears died in the greater Yellowstone area last year, including eight breeding-age females killed by people.

Information Items

1. bear – die – *null* (greater Yellowstone area, last year)
2. person – kill – female (greater Yellowstone area, last year)

Generated Sentences

1. 25 bears died.
2. Some people killed eight breeding-age females.

Generated Sentence in the Summary Some people killed eight breeding-age females.

Figure 1: Two examples of Information Item (InIt) text generation from a single sentence, and the generated sentence selected for the summary. Examples taken from clusters D1012C-B and D1025E-A.

culties will lead to relatively poor early results.

For now, we are still in the early stages of developing our abstractive summarization framework. As time was running out before the TAC competition, we have restricted our implementation of information items to dated and located subject–verb–object triples, but we intend to extend InIts in the future to include other (richer) types of information content. We use a parser to analyse each sentence and extract subject-verb-object triples, to which we attribute a date and a location when possible.

Our system generates the summary sentences, rather than extracting sentences directly from the original text. From each InIt and the sentence from which it originates, we have developed rules to generate new sentences, that include only one item of

information. To determine which of the generated sentences should be used in the summary, we would have liked to choose from among the InIts directly. For example, selecting the most frequent InIt, or InIts containing the most frequent subject-verb pair seems reasonable at first. However, during development, no such naive implementation of selecting InIts provided satisfactory results, because of the low frequency of those constructs. Instead, we compute a score based on the frequencies of the terms in the generated sentence. Finally, generating the summary requires some care, because each InIt potentially has a date and a location associated with it. The generated sentences, as they appear in the summary, also include automatically generated date and location modifiers.

Figure 1 shows two examples of sentences that can be generated from a source document sentence and the way they were realised in the summary in which they appear. As currently implemented, our approach of generated sentences can be seen as a kind of refined sentence compression for summarization. However, it differs in three important ways from the definition of the task of compression usually used (Knight and Marcu, 2000):

- Our generated sentences intend to cover only one item of information and not all the important information of the original sentence.
- An input sentence may have several generated sentences associated to it, one for each of its InIts, where it normally has only one compressed sentence.
- Generated sentences sometimes include words that do not appear in the original sentence (like 'some' in the second example), whereas sentence compression is almost always reduced to word deletion.

Because we were trying something completely new, we did not have time to attempt dealing differently with the update part of the task this year.

2 Our Summarization System

Our approach consists of 6 steps described in the following sections.

2.1 Preprocessing

The preprocessing stage formats the input for easier use. First, it patches the SGML-like files into well-formed XML. From the XML, we extract the text for each document. A few minor modifications are made in the text to make the parsing easier, like removing quotation mark characters, parentheses and their content, replacing some contractions with their full form, and pre-segmenting the text into sentences. At the end of preprocessing, the result is a text file with one sentence per line.

2.2 Annotation and Parsing

We run an information extraction engine on the preprocessed document cluster. This produces annotations on the cluster of the words' part-of-speech tags, and words or groups of words that are locations and dates.

We also parse each sentence of the cluster of documents, resulting in a full syntactical dependence tree.

2.3 Information Item Retrieval

We then extract the information items from each sentence, an information item being defined as a subject–verb–object triple. Sample InIts are given in figure 1.

Every verb encountered forms the basis of a candidate InIt. We also identify the verb's subject and object, if they exist, from the parse tree. Many candidates are rejected for various reasons: the difficulty of generating a grammatical and meaningful sentence from them, the observed unreliability of parses that include them, or because it would lead to incorrect InIts. We implemented the rejections using the following rules:

- Verb is a present participle
- Verb is in infinitive form
- Verb has a conjunction with another verb
- Verb is 'to be'
- Verb is not identified as a verb by the ANNIE POS Tagger
- Subject or object is identified as a verb by the ANNIE POS Tagger
- Subject or object is a relative pronoun
- Triple is part of a conditional clause

- Triple has no subject and no object

Experimentally, those criteria were selected because the InIts containing them often lead to generated sentences with incorrect grammar or content. Roughly half the candidates are rejected. More work is required to handle more cases and thus reject fewer candidate InIts in this initial step.

2.4 Sentence Generation

From each InIt retrieved, we generate a new sentence. Our main tools to do this are the original parse tree of the sentence from which the InIt is taken, and an NLG realiser to generate the new sentence. Sample generated sentences are illustrated in figure 1.

Our process can be described as translating the parts that we want to keep from the dependency tree provided by our parser, into a format that the realiser understands. This way we keep track of what words play what role in the generated sentence and we select directly which parts of a sentence appear in a generated sentence for the summary.

Sentence generation follows the following steps:

- Generate a Noun Phrase (NP) to represent the subject if present
- Generate a NP to represent the object if present
- Generate a NP to represent the indirect object if present
- Generate a complement for the verb if one is present and there was no object
- Generate the Verb Phrase (VP) and link all the components together, ignoring anything else found in the original sentence

NP Generation

Noun phrase generation is based on the subtree of its head word in the dependency parse tree, and it is always done in the same way, whether the noun is a subject, object, indirect object, or noun complement (calls to build NPs are recursive). The head in the subtree becomes the head of the NP. Additional parts of the NP are added according to the children of the head in its parse subtree. The following children of the head in the parse tree are 'translated' for the NLG realiser:

- A determiner is kept as is, or changed from "the" to "a" if the NP's modifiers have been removed
- A preposition leads to building a Prepositional Phrase (PP); see below
- A number modifier becomes a 'pre-modifier'
- A noun modifier leads to building an NP and placing it as pre- or post-modifier according to its original position in the original sentence
- A noun that is in a relation of conjunction with the head noun leads to building an NP for that noun and a conjunction between them
- A genitive noun leads to building a genitive NP and setting it to pre-modifier
- An adjective modifier is set as pre- or post-modifier according to its position in the original sentence
- All other children in the subtree are ignored and effectively removed during generation

PP Generation

Prepositional phrases are generated when they are the complement of a noun phrase or when they replace the object as complement of a verb. The head of the PP is the preposition. If the preposition has a noun complement, then we generate an NP for it, otherwise we do not generate the PP.

Verb Complement Generation

When an InIt has no object, then we attempt to find another complement instead, should the verb have no interesting meaning without a complement. The first modifier of the verb that follows the verb in the sentence will be used, except if the complement is the subject of the verb or if it is a punctuation. Prepositional phrase modifiers are generated as described above. All other modifiers are included as verb complements in full, with all of their subtree from the parse. This step includes complements in the form of a verb in the infinitive form, such as in the sentence "George decided to leave", which would be generated in its entirety.

VP Generation

Finally, the verb phrases are also generated from the verb and some of its children. Guests, auxiliaries, negation and perfect form are all kept. Then the NPs generated for the subject, object and indirect object are added with their appropriate function. The verb complement is added if it was generated. If there is an object but no subject, the VP is set to pas-

sive. The tense (past or present) of the VP is set to the tense of the verb in the original sentence.

2.5 Dates and Locations

TAC 2010 categories 1 and 2 both include aspects related to time and space, whereas the other categories do not. For these two categories, we use our date and location annotations.

We do not include any words identified as a date or a location in the sentence generation process. In particular, words considered a date or location are ignored while building a NP, PP or complement. Instead, when exactly one date and/or exactly one location appear in the subtree of the verb of an InIt triple, that InIt is assigned a date and/or location, as illustrated in the examples of figure 1. Only dates that can be resolved are used, other dates are ignored. These dates and locations are generated at the time of summary generation instead of sentence generation, as will be described in section 2.7.

Dates are resolved when it is easy to do so, and ignored otherwise. If the date is parsable (such as "January 23" or "3 March, 2006"), then that date is used directly. The words "yesterday" and "today" are interpreted according to the date of publication. The days of the week ("Monday", "Tuesday", etc.) are interpreted as referring to the latest such day before the date of publication. This is not always right but works properly most of the time.

Date and location are used in the summary of the first example of Figure 1. They are not shown in the second example because it is in category 4, different from 1 or 2.

2.6 Sentence Ranking in each Run

Our sentence scoring criteria resemble those of our 2009 sentence selection module (Genest et al., 2009a). We use the document frequency (DF) – the number of documents that include an entity in its original text – of the lemmas included in the generated sentence as the main scoring criterion. The generated sentences are ranked based on their average DF (the sum of the DF of all the unique lemmas in the sentence, divided by the total number of words in the sentence). Lemmas in our stop list and lemmas that are included in a sentence already selected in the summary have their DF reduced to 0, to avoid favoring frequent empty words, and to avoid redundancy in the summary.

Only the generated sentences are considered for the sentence ranking process in run 1, and the DF score of each lemma was computed on the unedited documents.

For our second run, instead of computing the average DF and redundant lemmas on the the generated sentences themselves, it is computed on the original sentences, ignoring information about the generated sentences – the one actually appearing in the summary. We rank the generated sentences by the rank of the original sentences in which their InIt appeared. This is similar to the process used in sentence compression, where we may choose to select sentences, and then compress them.

2.7 Summary Generation

The total number of words from all the sentences that have been selected for the summary are set to exceed 100 at this point in the processing.

Abstractive summarization requires text and sentence planning. Text generation patterns can be used, based on some knowledge about the topic, and in the case of guided summarization, based on answering specific aspects of the category.

For now, we restrict text planning to a temporal ordering of the sentences, and adding dates and locations to the generated sentences when appropriate, that is for clusters 1 and 2.

We select sentences from the ranking performed before, until we go over the word limit of 100 words. Those sentences are ordered by the date of their InIt when it can be determined. Otherwise, the day before the date of publication of the article that included an InIt is used instead. We plan to improve our temporal analysis in the future. All generated sentences with the same date are grouped in a single coordinated sentence. The date is included directly as a pre-modifier "On *date*," in the first InIt of the coordination, and the other InIt's of that date are added to form a coordinate sentence.

Each InIt with a known location has its generated sentence appended with a post-modifier "in *location*", except if that location has already been mentioned in a previous InIt of the summary.

At the end of this process, the size of the summary is always above the word limit of 100. We remove

the least relevant InIt (see section 2.6) and restart the summary generation process. We keep taking away the least relevant InIt in a greedy way, until the length of the summary is under the 100-word limit. This naive solution to never exceed the limit was selected because InIt's were initially thought to always lead to short generated sentences. However, it turns out that some of the generated summaries are too short because some InIts that were removed can be quite long. It will be worth it to find a better algorithm to optimize the information content within the size limit.

2.8 Resources Used

This year's system is programmed almost entirely in Java, and makes use of the Minipar parser, the GATE framework, and the SimpleNLG natural language realiser. The preprocessing step also makes use of bash, sed and XSLT.

2.8.1 Minipar

Syntactical parsing is an essential part of our approach. For now, we use the MINIPAR parser (Lin, 1998) from the command-line to build a syntax dependency tree for each sentence of the source documents. MINIPAR provides syntactical dependency relations between each word and its parent in the parse tree, which may be a virtual node that in turn may have a syntactical role and/or an antecedent in the parse tree. A lemma and a part-of-speech is also provided for each word. MINIPAR always produces a complete parse tree, but it is sometimes incorrect. Incorrect parse trees may lead to false or misleading InIts and ungrammatical generated sentences.

2.8.2 GATE and GeoNames lists

We use the GATE framework (Cunningham et al., 2002) to generate annotations on the documents of each cluster. We specifically use certain IE features of GATE, the so-called ANNIE plug-ins, especially the Tokenizer, POS_Tagger, and Gazetteer. We use grammatical rules and data already included in the GATE standard build, with the notable addition of two geographical location lists found on GeoNames (GeoNames, 2010) – those named 'US' and 'cities1000', that include a comprehensive list of US locations and city names.

2.8.3 SimpleNLG

SimpleNLG (Gatt and Reiter, 2009) is a natural language generation framework implemented as a Java library. It realises sentences based on the words we choose and the syntactical relations we identify between them. The syntactical roles known to SimpleNLG are not the same as those of Minipar, and some roles have different names in one and the other. For example, Minipar has *determiners* ('the' or 'a' for example) whereas SimpleNLG calls them *specifiers* to noun phrases.

3 Results and Discussion

As shown in tables 1 and 2, linguistic quality of our summaries was very low, with both runs arriving always in the bottom 5. This low linguistic score is understandable, because this was our first try at text generation and abstractive summarization, whereas we think that most of the other runs used sentence extraction, with at most minor modifications made to the extracted sentences.

Our method for text generation still needs to be refined in several ways, starting from the way we identify InIts, as we have already discussed in the introduction. Even in the context of the methodology outlined in section 2, and specifically 2.4, many improvements can still be made. Errors specific to the current state of our approach came from two major sources: incorrect sentence analysis from the parser, and insufficiently detailed and sometimes inappropriate rules for "translating" a part of a parse into generated text.

Although the linguistic quality was very low, our pyramid and overall responsiveness scores were near the average. This indicates that, even in a rough first try where little effort has been given to content selection, our method is capable of producing summaries with reasonable content and of reasonable overall quality. There is a correlation between linguistic quality and the other two manual scores for most runs, but, as we can see in figure 2, our two runs stand out. We believe that this shows that we are doing something quite different from the others. By extension, we hope that increasing the linguistic quality of our approach to the level of the top systems could yield content and overall scores above their current ones.

Part A	Pyr.	Ling. Q.	Overall R.
Rali1	0.315	2.174	2.304
Rali2	0.282	2.239	2.326
Avg	0.309	2.820	2.576
Best	0.425	3.457	3.174
Models	0.785	4.910	4.760
Part B			
Rali1	0.199	2.000	1.826
Rali2	0.183	2.043	1.891
Avg	0.202	2.743	2.094
Best	0.321	3.326	2.717
Models	0.673	4.820	4.710

Table 1: Scores of pyramid, linguistic quality and overall responsiveness for our two runs, the average of automatic systems, the best score of any automatic system, and the average of the human models.

Part A	Pyr.	Ling. Q.	Overall R.
Rali1	29	39	29
Rali2	25	40	30
Part B			
Rali1	29	40	29
Rali2	27	41	33

Table 2: Ranks for each manual evaluation metric of our two runs in the TAC 2010 competition, out of the 43 submitted automatic runs.

There is no significant difference in performance between our two runs, except perhaps a small advantage in the pyramid score of our first run, even though individual summaries were quite different. This indicates that on average, the content selection strategies were probably equivalent.

4 Conclusion and Future Work

We are satisfied with the results considering that our participation was rushed and that we only offered a first draft of what is to come. Having a low linguistic quality was expected, but we are pleased at achieving average performance in content and even near-average performance in overall responsiveness. It means that our text generation was good enough to produce understandable summaries.

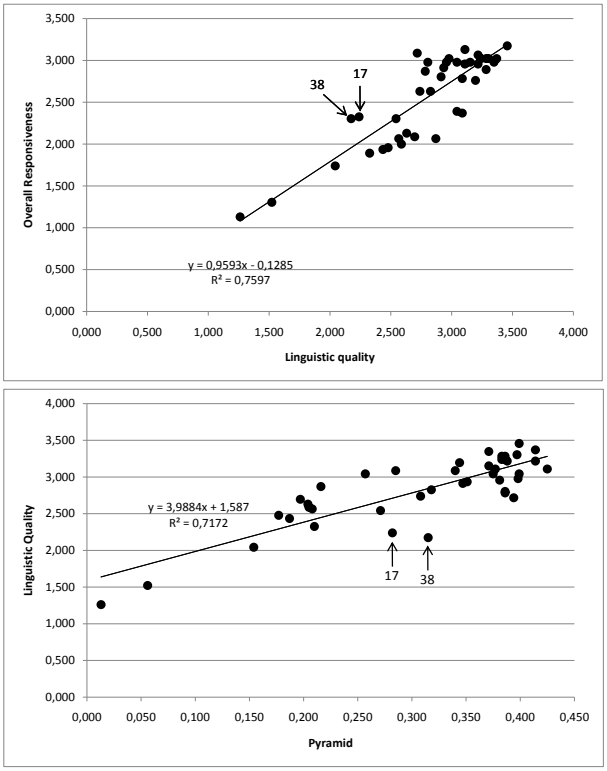

Figure 2: Scatter plots of overall responsiveness with respect to linguistic quality (top) and pyramid score with respect to linguistic quality (bottom) in part A, for all the systems competing in TAC 2010. Run 38 is Rali1 and run 17 is Rali2.

There are still many improvements to be made. One great disappointment is that we ended up not addressing the aspects directly for each category, other than having a special treatment of date and location for categories 1 and 2. To address this and to get closer to our high-level goals, we need to implement actual abstrative summarization, building upon the current system. This means specifically that we want all the content selection to be conducted within the abstraction of Information Items and that these units will be extracted directly. This year was closer to doing a sort of compressed sentence extraction, but this will soon change.

Generating sentences should rely less on the original sentence structure and more on the information meant to be transmitted. We want to move away from the current way we do things, which is too similar to rule-based sentence compression. At the core of moving toward actual abstraction, we need to re-

define InIts and find better ways in which they could be manipulated (compared, joined, etc.). Specifically, we intend to use tools and techniques that will enable us to find words and phrases of similar meanings, and to allow the generation of a sentence that is an aggregate of information found in several source sentences.

As we have just discussed, a complete overhaul of our approach will probably be necessary, but we have learned a lot with this year's participation and we have great plans in store for the years to come.

References

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. 2002. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, PA, USA*.

Albert Gatt and Ehud Reiter. 2009. Simplenlg: a realisation engine for practical applications. In *ENLG '09: Proceedings of the 12th European Workshop on Natural Language Generation*, pages 90–93, Morristown, NJ, USA. Association for Computational Linguistics.

Pierre-Etienne Genest, Guy Lapalme, Luka Nerima, and Eric Wehrli. 2009a. A symbolic summarizer with 2 steps of sentence selection for tac 2009. In *Proceedings of the Second Text Analysis Conference*, Gaithersburg, Maryland, USA. National Institute of Standards and Technology.

Pierre-Etienne Genest, Guy Lapalme, and Mehdi Yousfi-Monod. 2009b. Hextac: the creation of a manual extractive run. In *Proceedings of the Second Text Analysis Conference*, Gaithersburg, Maryland, USA. National Institute of Standards and Technology.

GeoNames. 2010. http://geonames.org/.

Kevin Knight and Daniel Marcu. 2000. Statistics-based summarization - step one: Sentence compression. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 703–710. AAAI Press.

Dekang Lin. 1998. Dependency-based evaluation of minipar. In *Proc. Workshop on the Evaluation of Parsing Systems*, Granada.