Conciseness through Aggregation in Text Generation

James Shaw
Dept. of Computer Science
Columbia University
New York, NY 10027, USA
`shaw@cs.columbia.edu`

Abstract

Aggregating different pieces of similar information is necessary to generate concise and easy to understand reports in technical domains. This paper presents a general algorithm that combines similar messages in order to generate one or more coherent sentences for them. The process is not as trivial as might be expected. Problems encountered are briefly described.

1 Motivation

Aggregation is any syntactic process that allows the expression of concise and tightly constructed text such as coordination or subordination. By using the parallelism of syntactic structure to express similar information, writers can convey the same amount of information in a shorter space. Coordination has been the object of considerable research (for an overview, see (van Oirsouw87)). In contrast to linguistic approaches, which are generally analytic, the treatment of coordination in this paper is from a synthetic point of view — text generation. It raises issues such as deciding when and how to coordinate. An algorithm for generating coordinated sentences is implemented in PLANDoc (Kukich et al.93; McKeown et al.94), an automated documentation system.

PLANDoc generates natural language reports based on the interaction between telephone planning engineers and LEIS-PLAN[1], a knowledge based system. Input to PLANDoc is a series of messages, or semantic functional descriptions (FD, Fig. 1). Each FD is an atomic decision about telephone equipment installation chosen by a planning engineer. The domain of discourse is currently limited to 31 message types, but user interactions include many variations and combinations of these messages. Instead

---

[1]LEIS is a registered trademark of Bell Communications Research, Piscataway, NJ.

of generating four separate messages as in Fig. 2, PLANDoc combines them and generates the following two sentences: "*This refinement activated DLC for CSAs 3122 and 3130 in the first quarter of 1994 and ALL-DLC for CSA 3134 in 1994 Q3. It also activated DSS-DLC for CSA 3208 in 1994 Q3.*"

2 System Architecture

Fig. 3 is an overview of PLANDoc's architecture. Input to the message generator comes from LEIS-PLAN tracking files which record user's actions during a planning session. The ontologizer adds hierarchical structure to messages to facilitate further processing. The content planner organizes the overall narrative and determines the linear order of the messages. This includes combining atomic messages into aggregated messages, choosing cue words, and determining paraphrases that maintain focus and ensure coherence. Finally the FUF/SURGE package (Elhadad91; Robin94) lexicalizes the messages and maps case roles into syntactic roles, builds the constituent structure of the sentence, ensures agreement, and generates the surface sentences.
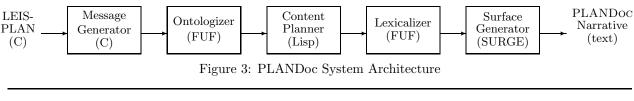
3 Combining Strategy

Because PLANDoc can produce many paraphrases for a single message, aggregation during the syntactic phase of generation would be difficult; semantically similar messages would already have different surface forms. As a result, aggregation in PLANDoc is carried out at the content planning level using semantic FDs. Three main criteria were used to design

```
((cat message)
 (admin ((PLANDoc-message-name RDA)
         (runid r-reg1)))
 (class refinement)
 (action activation)
 (equipment-type  all-dlc)
 (csa-site 3134)
 (date ((year 1994) (quarter 3))))
```

Figure 1: Output of the Message Generator

```
This refinement activated ALL-DLC for CSA 3134 in 1994 Q3.   (E1 S3 D2)
This refinement activated DLC for CSA 3130 in 1994 Q1.       (E2 S2 D1)
This refinement activated DSS-DLC for CSA 3208 in 1994 Q3.   (E3 S4 D2)
This refinement activated DLC for CSA 3122 in 1994 Q1.       (E2 S1 D1)
Equipment: E1= ALL-DLC, E2= DLC, E3= DSS-DLC
       Site: S1= CSA 3122, S2= CSA 3130, S3= CSA 3134, S4= CSA 3208
       Date: D1= 1994 Q1, D2= 1994 Q3
```

Figure 2: Unaggregated Text Output

LEIS-PLAN (C) → Message Generator (C) → Ontologizer (FUF) → Content Planner (Lisp) → Lexicalizer (FUF) → Surface Generator (SURGE) → PLANDoc Narrative (text)

Figure 3: PLANDoc System Architecture

the combining strategy:

1. **domain independence**: the algorithm should be applicable in other domains.

2. **generating the most concise text**: it should avoid repetition of phrases to generate shortest text.

3. **avoidance of overly-complex sentences**: it should not generate sentences that are too complex or ambiguous for readers.

The first aggregation step is to identify semantically related messages. This is done by grouping messages with the same action attribute. Then the system attempts to generate concise and unambiguous text for each action group separately. This reduces the problem size from tens of messages into much smaller sizes. Though this heuristic disallows the combination of messages with different actions, the messages in each action group already contain enough information to produce quite complex sentences.

The system combines the maximum number of related messages to meet the second design criterion–generating the most concise text. But such combination is blocked when a sentence becomes too complex. A bottom-up 4-step algorithm was developed:

1. **Sorting**: putting similar messages right next to each other.

2. **Merging Same Attribute**: combining adjacent messages that only have one distinct attribute.

3. **Identity Deletion**: deletion of identical components across messages.

4. **Sentence Breaking**: determining sentence breaks.

```
(E2 S1 D1)       (E1 S3 D2)        (E2 S1 D1)
(E2 S2 D1)       (E2 S1 D1)        (E2 S2 D1)
(E1 S3 D2) --> (E2 S2 D1) --> (E1 S3 D2)
(E3 S4 D2)       (E3 S4 D2)        (E3 S4 D2)
  by Site       by Equipment        by Date
```

Figure 4: Step 1. Sorting

3.1 Step 1: Sorting

The system first ranks the attributes to determine which are most similar across messages with the same action. For each potential distinct attribute, the system calculates its rank using the formula $m - d$, where m is the number of messages and d is the number of distinct attributes for that particular attribute. The rank is an indicator of how similar an attribute is across the messages. Combining messages according to the highest ranking attribute ensures that minimum text will be generated for these messages. Based on the ranking, the system reorders the messages by sorting, which puts the messages that have the same attribute right next to each other. In Fig. 2, *equipment* has rank 1 because it has 3 distinct equipment values – ALL-DLC, DLC, and DSS-DLC; *date* has rank 2 because it has two distinct date values – 1994 Q1 and 1994 Q3; *site* has rank 0. Attribute *class* and *action* (Fig. 1) are ignored because they are always the same at this stage. When two attributes have the same rank, the system breaks the tie based on a priority hierarchy determined by the domain experts. Because the final sorting operation dominates the order of the resulting messages, PLANDoc sorts the message list from the lowest rank attribute to the highest. In this case, the ordering for sorting is *site*, *equipment*, and then *date*. The resulting message list after sorting each attribute is shown in Fig. 4.

3.2 Step 2: Merging Same Attribute

The list of sorted messages is traversed. Whenever there is only one distinct attribute between two adjacent messages, they are merged into one message with a conjoined attribute, which is a list of the distinct attributes from both messages. What about messages with two or more distinct attributes? Merging two messages with two or more distinct attributes will result in a syntactically valid sentence but with an undesirable meaning: "*This refinement activated ALL-DLC and DSS-DLC for CSAs 3122 and 3130 in the third quarter of 1993.*"

By tracking which attribute is compound, a third message can be merged into the aggregate message if it also has the same distinct attribute. Continue from Step 1, (E2 S1 D1) and (E2 S2 D1) are merged because they have only one distinct attribute, *site*. A new FD, (E2 (S1 S2) D1), is assembled to replace those two messages. Note that although (E1 S3 D2) and (E3 S4 D2) have the date in common, they are not combined because they have more than one distinct attribute, *site* and *equipment*.

Step 2 is applied to the message list recursively to generate possible crossing conjunction, as in the following output which merges *four* messages: "*This refinement activated ALL-DLC and DSS-DLC for CSAs 3122 and 3130 in the third quarter of 1993.*" Though on the outset this phenomenon seems unlikely, it does happen in our domain.

3.3 Step 3: Identity Deletion

After merging at step 2, the message list left in an action group either has only one message, or it has more than one message with at least two distinct attributes between them. Instead of generating two separate sentences for (E2 (S1 S2) D1) and (E1 S3 D2), the system realizes that both the subject and verb are the same, thus it uses deletion on identity to generate "*This refinement activated DLC for CSAs 3122 and 3130 in 1994 Q1 and [this refinement activated] ALL-DLC for CSA 3134 in 1994 Q3.*" For identical attributes across two messages (as shown in the bracketed phrase), a "deletion" feature is inserted into the semantic FD, so that SURGE will suppress the output.

3.4 Step 4: Sentence Break

Applying deletion on identity blindly to the whole message list might make the generated text incomprehensible because readers might have to recover too much implicit information from the sentence. As a result, the combining algorithm must have a way to determine when to break the messages into separate sentences that are easy to understand and unambiguous.

How much information to pack into a sentence does not depend on grammaticality, but on coherence, comprehensibility, and aesthetics which are hard to formalize. PLANDoc uses a heuristic that always joins the first and second messages, and continues to do so for third and more if the distinct attributes between the messages are the same. This heuristics results in parallel syntactic structure and the underlying semantics can be easily recovered. Once the distinct attributes are different from the combined messages, the system starts a new sentence. Using the same example, (E2 (S1 S2) D1) and (E1 S3 D2) have three distinct attributes. They are combined because they are the first two messages. Comparing the third message (E3 S4 D2) to (E1 S3 D2), they have different *equipment* and *site*, but not *date*, so a sentence break will take place between them. Aggregating all three messages together will results in questionable output. Because of the parallel structure created between the first 2 messages, readers are expecting a different *date* when reading the third clause. The second occurrence of "1994 Q3" in the same sentence does not agree with readers' expectation thus potentially confusing.

4 Future Directions

In this paper, I have described a general algorithm which not only reduces the amount of the text produced, but also increases the fluency of the text. While other systems do generate conjunctions, they deal with restricted cases such as conjunction of subjects and predicates(Dalianis&Hovy93). There are other interesting problems in aggregations. Generating marker words to indicate relationships in conjoined structures, such as "respectively", is another short term goal. Extending the current aggregation algorithm to be more general is currently being investigated, such as combining related messages with different actions.

5 Acknowledgements

References

Dalianis, Hercules, and Hovy, Edward. 1993. Aggregation in Natural Language Generation. In *Proceedings of the Fourth European Workshop on Natural Language Generation*, Pisa, Italy.

Elhadad, Michael. 1991. FUF: The universal unifier - user manual, version 5.0. *Tech Report CUCS-038-91*, Columbia Univ.

Robin, Jacques. 1994. *Revision-Based Generation of Natural Language Summaries Providing Historical Background: Corpus-based analysis, design, implementation and evaluation.* Ph.D. thesis, Computer Science Department, Columbia Univ.

Kukich, K., McKeown, K., Morgan, N., Phillips, J., Robin, J., Shaw, J., and Lim, J. 1993. User-Needs Analysis and Design Methodology for an Automated Documentation Generator. In *Proceedings of the Fourth Bellcore/BCC Symposium on User-Centered Design*, Piscataway, NJ.

McKeown, Kathleen, Kukich, Karen, and Shaw, James. 1994. Practical Issues in Automatic Documentation Generation. In *Proceedings of the 4th Conference on Applied Natural Language Processing*, Stuttgart, p.7-14.

van Oirsouw, Robert. 1987. *The Syntax of Coordination* Beckenham: Croom Helm.