Incrementally Segmenting Incoming Speech into Pragmatic Fragments

Gregory S. Aist
Computer Science Department
University of Rochester
gaist@cs.rochester.edu

1 Introduction

The state-of-the-art in spoken dialog systems is, in general, to process incoming speech one utterance at a time, and then respond. We have been working to move computer understanding closer to contemporary models of human language understanding, which operate incrementally, with information shared across multiple levels of processing. Such incremental methods have been shown to have various advantages over their nonincremental counterparts, such as better parsing (Stoness et al. 2005)

In this paper we describe work on a level of processing which operates early on in language understanding: speech segmentation. Our algorithm has a number of distinct features. First, it is incremental, operating on incoming speech one word at a time. Second, it is pragmatic, making use of knowledge of the specific domain in order to make predictions. Third, it is linguistically based, making predictions of fragments based on a lightweight unification grammar. And finally, the predictions are both syntactically and pragmatically relevant – the advice of the segmentation module is passed on to the parser and to the user interface, for further action as needed.

2 Segmentation Algorithm

The segmentation algorithm has several components. There is a lookahead word which allows a "peek" at the word ahead of the current word being processed. There is a cache which stores incoming words until they are consumed (or discarded). There is a queue which stores the identified fragments. There is a fragment finder which uses a small unification grammar to identify Xbar fragments such as the vbar *move* or the nbar *large triangle.* A prefix inspector invokes the fragment finder to calculate whether a given word sequence is a valid prefix of a fragment. Reporting code notifies the other components in the dialog system (such as the parser and user interface) of fragments that have been identified.

The segmentation algorithm operates as follows.

1. The cache and queue are set to the empty list.
2. In response to an incoming token, if the token is end-of-utterance, go to step 1. Otherwise continue to step 3.
3. The fragment finder runs over the cache plus the current token.
4. If a fragment is found (*move*), and the words in that fragment plus the lookahead do not yield a fragment (*move a*), then report the fragment, add the fragment to the rightmost side of the queue, and clear the cache – that is, close off the current fragment and move on.
5. If no fragment is found and the words are a valid prefix, then add the current token to the rightmost side of the cache. (e.g. *central* is not itself a fragment, but is a valid prefix for the region name *central park*) – that is, save a word for later.
6. If no fragment is found (*a large*) and the words are not a valid prefix (*a large* is a prefix to an np, but not an nbar, which is what we seek here), then remove the leftmost word from the cache and add the current token to the right side of the cache – that is, skip a word.
7. Go to step 2.

Figure 1 shows an example of a fragment being used to indicate the results of incremental understanding to the user, in a testbed domain (Aist et al. 2005). If the user had meant to specify one of the triangles with a star or a circle on it, the word following *large triangle* would not have been *to* but rather something like *with* or *that (has)*. It is true that a later phrase might modify the *large triangle* – such as *the one with the star on it* – but the system as a whole could treat such refinements as a repair.

A trace for *move a large triangle to central park*:

Words: MOVE a
Cache: []
Queue: []
Fragment(move)? Yes – vbar
Fragment(move a)? No
Report: vbar(move)

Words: move A large
Cache: []
Queue: [vbar(move)]
Fragment(a)? No
Valid prefix(a)? No

Words: move a LARGE triangle
Cache: [a]
Queue: [vbar(move)]
Fragment(a large)? No
Valid prefix(a large)? No

Words: move a large TRIANGLE to
Cache: [large]
Queue: [vbar(move)]
Fragment(large triangle)? Yes – nbar(large triangle)
Report: nbar(large triangle) ------- **(See Figure 1)**

Words: move a large triangle TO central
Cache: []
Queue: [vbar(move), nbar(large triangle)]
Fragment(to)? Yes – pbar(to)
Fragment(to central)? No
Report: pbar(to)

Words: move a large triangle to CENTRAL park
Cache: []
Queue: [vbar(move), nbar(large triangle), pbar(to)]
Fragment(central)? No
Valid prefix(central)? Yes – nbar(central X)

Words: move a large triangle to central PARK </s>
Cache: [central]
Queue: [vbar(move), nbar(large triangle), pbar(to)]
Fragment(central park)? Yes – nbar(central park)
Report: nbar(central park)

Words: move a large triangle to central park </s>
Cache: []
Queue: [vbar(move), nbar(large triangle), pbar(to), nbar(central park)]

Currently the segmentation algorithm handles ambiguity issues such as prepositional phrase attachment by deferring to the general parser. That is, a sequence such as "a large triangle with a star on the corner" would be segmented into the stream

[a, nbar(large triangle),
pbar(with), a, nbar(star),
pbar(on), the, nbar(corner)]

and the decision about whether the appropriate referent(s) were

(a) a large triangle decorated with a star, or
(b) (1) a large triangle
and (2) a star up in the corner of the screen

would be deferred to later processing, where a complete deep parse is calculated.

3 Related Work and Future Directions

This algorithm is related to work in deterministic parsing (e.g. Marcus 1980) in that it uses lookahead to make its decisions. It is not itself deterministic since the fragments need not make it into the final parse. (That depends on the decisions of the parser itself).

The present algorithm is similar to left-corner parsing in that it integrates bottom-up and top-down information in order to make its decisions. The classic formulation of left-corner parsing uses alternating top-down and bottom-up steps. The current algorithm uses a cache of as-yet-unprocessed words that represent, in a sense, its bottom-up data; as further evidence comes in, the entire cache is reanalyzed to check for the top-down match of a fragment. (This is fast since in practice the fragments are small.)

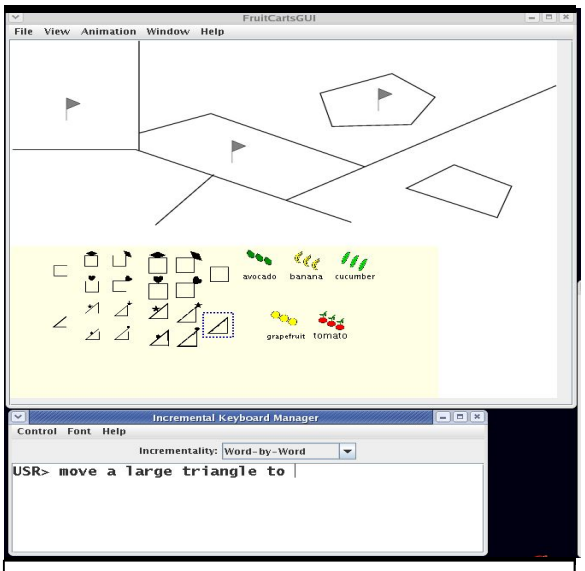

Figure 1. In response to the nbar *large triangle*, the (plain) large triangle has been highlighted. The middle flag is in Central Park.

Finally, the relationship of the present algorithm to current parsing methods such as probabilistic chart parsing and head-driven parsing (e.g. Collins 2003, Charniak 2001, van Noord 1997) is at the present time one of producer and consumer. It is possible that the queue of Xbars might be amenable to reanalysis to yield a complete parse, but that remains ground for future exploration.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0328810. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

G.S. Aist, E. Campana, J. Allen, M. Rotondo, M. Swift, and M. Tanenhaus. 2005. Variations along the contextual continuum in task-oriented speech. Proceedings of the 27th Annual Conference of the Cognitive Science Society, Stresa, Italy, July. Paper number 769.

E. Charniak. 2001. Immediate-head parsing for language models. ACL 2001.

M. Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics.*

M. P. Marcus. 1980. *Theory of Syntactic Recognition for Natural Languages*. MIT Press.

G. van Noord. 1997. An efficient implementation of the head-corner parser. *Computational Linguistics* **23**(3): 425-456.

S.C. Stoness, J. Allen, G. Aist, and M. Swift. 2005. Using real-world reference to improve spoken language understanding. AAAI Workshop on Spoken Language Understanding, Pittsburgh, Pennsylvania, July. pp. 38-45.