SRI INTERNATIONAL FASTUS SYSTEM MUC-4 TEST RESULTS AND ANALYSIS

Douglas E. Appelt, John Bear, Jerry R. Hobbs, David Israel, and Mabry Tyson

SRI International
Menlo Park, California 94025
appelt@ai.sri.com
(415) 859-6150

INTRODUCTION

The system that SRI used for the MUC-4 evaluation represents a significant departure from system architectures that have been employed in the past. In MUC-2 and MUC-3, SRI used the TACITUS text processing system [1], which was based on the DIALOGIC parser and grammar, and an abudctive reasoner for horn-clause logic. In MUC-4, SRI designed a new system called FASTUS (a permutation of the initial letters in *Finite State Automata-based Text Understanding System*) which we feel represents a significant advance in the state of the art of text processing. The system shares certain modules with the earlier TACITUS system, namely modules for text preprocessing and standardization, spelling correction, Hispanic name recognition, and the core lexicon. However, the DIALOGIC system and abductive reasoner, which were the heart and soul of the previous system, were replaced by a system whose architecture is based on cascaded finite-state automata. Using this system we were capable of achieving a significant level of performance on the MUC-4 task with less than one month devoted to domain-specific development. In addition, the system is extremely fast, and is capable of processing texts at the rate of approximately 3,200 words per minute, measured in CPU time on a Sun SPARC-2 processor. (Measured according to elapsed real time, the system about 50% slower, but the observed time depends on the particular hardware configuration involved.)

OVERVIEW OF THE FASTUS ARCHITECTURE

The architecture of the FASTUS system is described in detail in the associated system summary. It can be summarized as a three-phase process. The first phase consists of scanning the text to identify proper names, correcting spelling, and similar preprocessing tasks to ensure that the text is in a standardized format for the remainder of the processing.

The second phase consists of a finite-state machine that accepts the sequence of words from the text, and produces as output a sequence of linguistic consituents — noun groups consisting of determiners, prenominals and head noun, verb groups consisting of auxilliaries plus the main verb together with any intervening adverbs, and particles, which is a catch-all category including prepositions, conjunctions, and genitive markers. The output of the second pass is filtered to include only the longest consitutents spanning any given portion of the sentence.

The linguistic consituents from the second phase are given as input to another finite-state machine. The transitions of this third-phase machine are based on the head of each constituent, and each transition builds some piece of an "incident" structure, which can be thought of as a "proto-template." When a final state of the machine is reached, the incident structure that has been produced through that point is saved, and merged with all other incident structures produced by the same sentence. (There may be several, because the machines are non-deterministic). These incident structures are then merged with incident structures from the rest of the text according to a set of merging heuristics. The incident structures are converted to the format of MUC-4 templates in a post-processing phase.

CONTROLLING THE FASTUS SYSTEM

In the course of designing the system, we paramaterized a number of characteristics of the system's operation because we believed that the parameterized behavior would reflect tradeoffs in recall versus precision. Subsequent testing revealed that many of these parameters result in both higher recall *and* higher precision when in one state or the other, and therefore we left them permanently in their most advantageous state. Those parameters that seemed to affect recall the the expense of precision were set to produce a test run in which we attempted to maximize the system's recall. The effect of these parameters could be described in general as distrusting the system's filters' ability to eliminate templates corresponding to stale dates, uninteresting countries, and military incidents. We observed a small but measurable increase in recall at the expense of precision by distrusting our filters.

The following parameters were implemented and tested on 300 texts before arriving at the decisions for the settings on the final run.

- *Conservative Merging.* When this option is selected, the system would not merge incidents that had non-overlapping targets with proper names. When not selected, any merges consistent with the incident types were permitted. Testing revealed that merging should *always* be conservative.
- *Civilian Target Requirement.* This filter would reject any template that did not have at least one non-military target, including templates that identified a perpetrator, but no physical or human target at all. This option appears to produce a recall-precision tradeoff of about one or two points.
- *Subjectless Verb Groups.* This parameter would allow the system to generate an incident structure from a verb together with its object, even if its subject could not be determined. Although early tests showed a recall-precision tradeoff, subsequent and more thorough testing indicated that this should always be done.
- *Filter Many-Target Templates.* This filter would disallow any template that had more than 100 targets, on the supposition that such templates often result from vague or general, and hence irrelevant, descriptions. This turns out to be a correct heuristic, but only if the number of targets is evenly divisible by 100. (An airline bombing with 307 victims is certainly interesting, while "70,000 peasants have been killed" is probably vague).
- *Military Filtering.* This heuristic causes the system to eliminate all military targets from templates, on the belief that we may have incorrectly merged a military incident with a civilian incident and incorrectly reported the union of the two. Tests show that this filtering improves precision slightly.
- *Liberal Perpetrator Org.* Setting this parameter would cause the system to pick any likely perpetrator organization out of the text, ignoring whatever the text actually says. Testing showed that this parameter had no effect, which was such a surprising result that we distrust it, and regard our testing as inconclusive.
- *Spelling Correction* This parameter controls how much spelling correction the system does. Our experiments indicated that spelling correction hurts, primarily because novel proper names get corrected to other words, and hence lost. We tried a weaker version of spelling correction which would correct only misspelled words that did not occur on a large list of proper names that we had assembled. This showed an improvement, but spelling correction still had a small negative effect. This was also a surprising result, and we were not willing to abandon spelling correction, and ran all tests with weak spelling correction enabled, although to some extent a complete lack of spelling correction is compensated for by the presence of common misspellings of important domain words like "guerrilla" and "assassinate" in the lexicon.
- *Stale Date Filtering.* This parameter causes filtering of any template that has a date that is earlier than two months before the date of the article. Eliminating this filtering produces an increase in recall

at the expense of precision, the magnitude of which depends on how well our date detection currently works. We would expect about a one-point tradeoff.

- *Weak Location Filtering.* If the system's location dection finds that the location of an incident is impossible according to the system's location database, it eliminates the template. If this flag is set, the template will be produced using only the country as the location. Testing shows that this is always desireable.

THE RESULTS ON TST3 AND TST4

On TST3, we achieved a recall of 44% with precision of 55% in the all-templates row, for an F-score ($\beta = 1$) of 48.9. On TST4, the test on incidents from a different time span, we observed, surprisingly, an identical recall score of 44%, however our precision fell to 52%, for an F-score of 47.7. It was reassuring to see that there was very little degradation in performance moving to a time period over which the system had not been trained. We also submitted a run in which we attempted to maximize the system's recall by not filtering military targets, and allowing incidents with stale dates. On TST3, this led to a two-point increase in recall at the expense of one point in precision. On TST4, our recall did not increase, however our precision fell by a point, giving us a lower F-score on this run. These results were consistent with our observations during testing, although our failure to produce even a small increase in recall on TST4 was somewhat disappointing.

The runtime for the entire TST3 message set on a SPARC-2 processor was 11.8 minutes (about 16 minutes of elapsed real time with our configuration of memory and disk). These times are quite consistent with our runs over the development sets. During the course of development, the overall run time for 100 messages increased approximately 50%, but we attribute this increase to the decision to treat more sentences as relevant. It appears possible to increase the coverage of the system without an unacceptable increase in processing time.

DEVELOPMENT HISTORY

During December of 1991 we decided to implement a preprocessor for the TACITUS system, at which point the FASTUS architecture was born. The system was originally conceived as a preprocessor for TACITUS that could be run in a stand-alone mode. Considerably later in our development we decided that the performance of FASTUS on the MUC-4 task was so high that we could make FASTUS our complete system.

Most of the design work for the FASTUS system took place during January. The ideas were tested out on finding incident locations in February, and with some initial favorable results in hand, we proceded with the implementation of the system in March. The implementation of the second phase of processing was completed in March, and the general outline of phase three was completed by the end of April. On May 6, we did the first test of the FASTUS system on TST2, which had been withheld as a fair test, and we obtained a score of 8% recall and 42% precision. At that point we began a fairly intensive effort to hill-climb on all 1300 development texts, doing periodic runs on the fair test to monitor our progress, culminating in a score of 44% recall, 57% precision in the wee hours of June 1, when we decided to run the official test. As the chart in Figure 1 points out, the rate of progress was rapid enough that even a few hours of work could be shown to have a noticeable impact on the score. Our scarcest resource was time, and our supply of it was eventually exhausted well before the point of diminishing returns.

CONCLUSIONS

FASTUS was more successful than we ever dreamed when the idea was originally conceived. In retrospect,

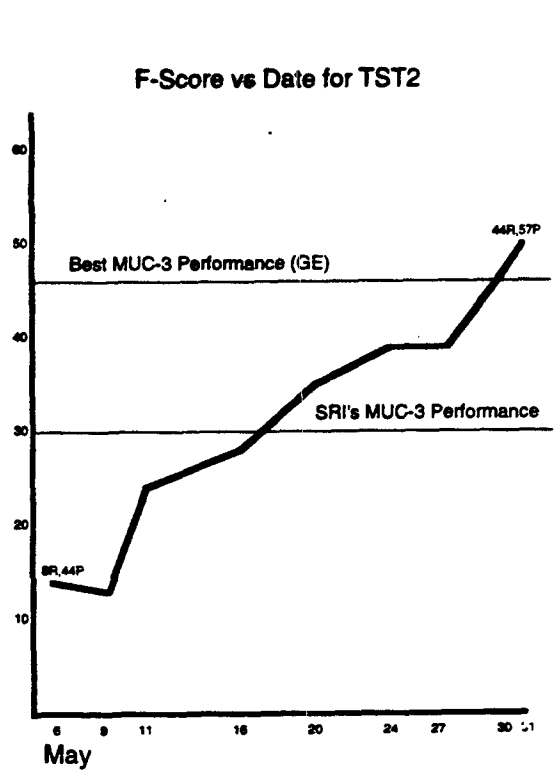

Figure 1: Plot of F-Score versus Date for FASTUS Development

we attribute its success to the fact that its processing is extremely well suited to the demands of the task. The system's phase-3 works successfully because the input from phase-2 is already reliably processed. Phase two does only the linguistic processing that can be done reliably and fast, ignoring all the problems of making attachment decisions, and the ambiguity introduced by coordination and appositives. This input is adequate for phase-3 because the domain pragmatics are sufficiently constrained that given this initial chunking, the relevant information can be reliably detected and extracted.

One source of frustration with the development of this system is that we never had the opportunity to produce a decent developer's interface. We believe that phase-2 is almost completely domain independent, with all the domain specific knowledge embedded in the phase-3 automata. We feel that with some careful thought devoted to such an interface, we could produce a general text processing system that could be brought up to our current level of performance on a MUC-like or TIPSTER-like task in even less than the three and a half weeks of effort that we required.

Another discovery of this experience is that a MUC-like task is much easier than anyone ever thought. Although the full linguistic complexity of the MUC texts is very high, with long sentences and interesting discourse structure problems, the relative simplicity of the information-extraction task allows much of this linguistic complexity to by bypassed – indeed much more than we had originally believed was possible. The key to the whole problem, as we see it from our FASTUS experience, is to do exactly the right amount of syntax, so that pragmatics can take over its share of the load. For the MUC task, we think FASTUS displays exactly the right mixture.

Finally, we point out that while FASTUS is an elegant engineering achievement, the whole host of linguistic problems that were bypassed are still out there, and will have to be addressed eventually for more complex tasks, and to achieve ever higher performance on simpler tasks. The nature of competitive evaluations is that they force everyone to deal with the easiest problems first. However, the hard problems cannot be ignored forever, and scientific progress requires that they be addressed.

ACKNOWLEDGEMENTS

This research was funded by the Defense Advanced Research Projects Agency under Office of Naval Research contracts N00014-90-C-0220, and by an internal research and development grant from SRI International.

References

[1] Hobbs, J., et al., "Description of the TACITUS System as Used for MUC-3," Proceedings of the MUC-3 Workshop, 1991, pp. 200–206.