FOUR SCORERS AND SEVEN YEARS AGO:
The Scoring Method for MUC-6

Nancy Chinchor, Ph.D.
Science Applications International Corporation
10260 Campus Point Drive, M/S A2-F
San Diego, CA 92121
chinchor@gso.saic.com
(619) 458-2614

Gary Dungca
Science Applications International Corporation
10260 Campus Point Drive, M/S A2-F
San Diego, CA 92121
dungca@gso.saic.com
(619) 458-4907

INTRODUCTION

The MUC-6 scoring method is based on a two-step process of mapping an item generated by a system under evaluation (the "response") to the corresponding item in the human-generated answer key and then scoring the mapped items. The resulting scores are used for decision-making over the entire evaluation cycle, including refinement of the task definition based on interannotator comparisons, technology development using training data, validating answer keys, and benchmarking both system and human capabilities on the test data.

To further understand the scoring method, we will look at the features and algorithms embodied in each of the scorers, showing their basic similarity and discussing the differences from task to task. We will show how critical the mapping algorithm is in scoring and the problems inherent in deciding what the best mapping should be. We will also discuss the result of translating the emacslisp scorers into C, including the increased accuracy of the mapping as well as the increased speed and improved memory management. The positive effects on consumer capability will be shown and future enhancements will be briefly outlined.

MODULAR STRUCTURE OF THE SCORERS

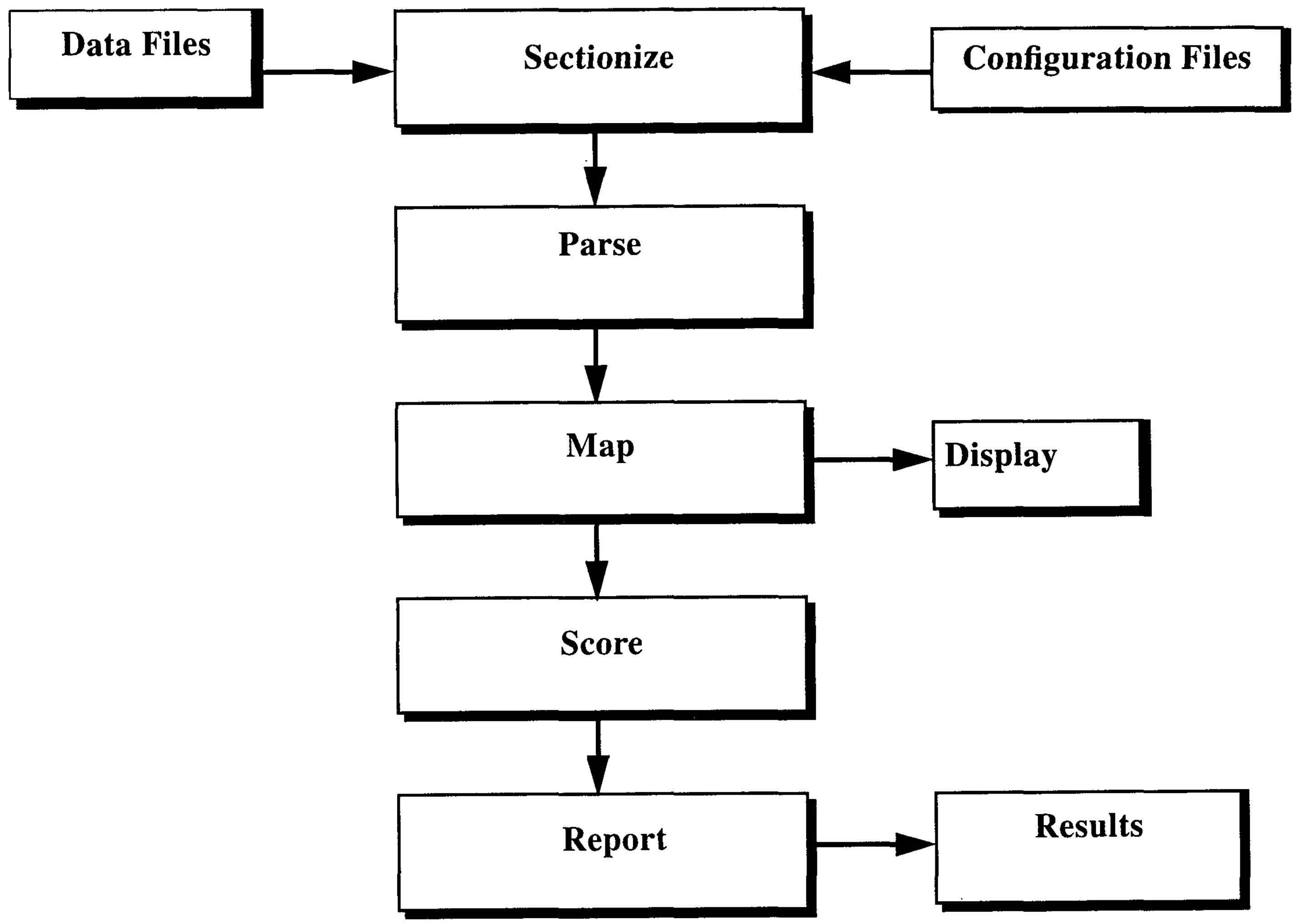
The Basic Scorer

The basic design of the scorers is shown in Figure 1. All four of the scorers follow this basic design: *sectionize*, *parse*, *map*, *score*, and *report*.

Sectionize

The input data consists of three files: one file contains the set of source text articles used in the task; one file contains the set of human-generated answer keys derived from the source articles; and one file contains the system-generated responses, also derived from the source articles. The key and response files are composed either of annotated text articles (as in the case of the Named Entity and Coreference text annotation tasks), or of relevant data extracted from those articles (as in the case of the Template Element and Scenario Template tasks). These articles are distinguishable by unique identifiers called document numbers. The scorer sectionizes the data files by creating groups based on these document numbers; each group consists of one each of the three types of input data.

Figure 1. Modular Structure of Scorer



Parse

The scorer parses the data from the input data files and creates structured internal representations of the information. The structures may represent varying levels of data abstraction, but must be consistent with the levels of abstraction in the input files. For example, singular pieces of textual data, also known as slot fills, may be combined with others to comprise a larger, cohesive body, or object, of information. Objects may also contain pointers to other objects, forming a hierarchical structure.

Map

Mapping is the process by which the scorer aligns answer key objects with a system's response objects. The broad purpose of this alignment is to pair objects that are similar in their slot content, thus optimizing the system's scores. However, because objects are generally composed of multiple slots, and because objects of the same type often have data in common, this notion of similarity between a key and a response object can be complicated. A key object may be similar to more than one response object. Likewise, a response object may be similar to more than one key object. Also, no one particular slot identifies an object; after several trials of different weightings over the last few years, it was decided that each slot should be given equal emphasis when considering an object's mapping, and that the threshold should be set so that getting one slot fill in an object correct is enough for the object to be considered as a candidate for mapping.

The object pairs that can be considered candidates for mapping are then scored and rank-ordered according to some pre-determined metric. The ordering of identically ranked object-pairs is arbitrary. The scorer then proceeds, in rank order, to align key objects with response objects to generate the final mappings.

In the event that an object has multiple candidate mappings -- a key can be possibly mapped to any of several responses, and a response can be possibly mapped to any of several keys -- that object is eventually mapped to the best remaining unmapped partner. As a consequence of this rank-ordered approach, the best ranked pairing for any particular key object may not, in fact, result in final alignment. Another key object may have already had a better rank-ordered pairing with the same response object, or the arbitrary order of like-ranked pairs may have precluded this alignment. The same may be said of a response object being compared to different answer key objects.

Another consequence of this approach is manifested in the score results: an object slot may be discredited because its content does not match the corresponding slot content of the finally aligned object, although it does match with the corresponding slot of an alternate object. This can give rise to an argument for giving credit, since matching data does exist in the overall response. However, focus must be maintained at the object level, resulting in assigning higher value to the cumulative score of all other matched slots in the object at the expense of a few mismatched slots. To either ignore the levels of abstraction or to require a switch of the alignments would likely yield a poorer score. Optimal mapping for a hierarchical template or for relational level objects is an unsolved problem in computer science. Tree-matching techniques need to be developed for this class of problems, which appear often in artificial intelligence.

Finally, after all rank-ordered pairs have been considered, the mapper forces pairings, or "connections", on the remaining unmapped objects. The purpose of connections is that without them the slot fills of an unmapped key will be scored as *missing*, and the slot fills for an unmapped response will be scored as *spurious*. However, if the two objects were connected, then the slots for which both the key and the response provided data will be scored as *incorrect*, thus reducing the scoring penalty from two (missing and spurious) to one (incorrect). After connection, any remaining objects will either be all keys or all responses, i.e., all missing or all spurious.

Score

Mapped object-pairs are scored against each other by comparing their slot fills. Slot fills are either text strings, or pointers to other objects, and every slot has a set of fills that may consist of zero, one, or multiple fills.

Response text strings are considered matched if they are identical to the key, or are identical after ignoring certain text elements that are considered non-substantive. These elements include a set of premodifiers (e.g., "the", "an", "a"), a set of postmodifiers (e.g., period, question mark, double quote), and a set of corporate-designators (e.g., "COMPANY", "CO", "INC", "LTD").

Object pointers are considered matched if the objects to which they refer are mapped to each other. Otherwise, the pointers are considered unmatched.

The scores for a particular slot fill are tallied as follows:

possible - the number of fills provided in the key.

actual - the number of fills provided in the response.

correct - the number of matches found between the key and the response fills.

incorrect - if the slot contains the same number of fills in the key and the response, this number is the number of non-matching fills in the response; if the slot contains different numbers of fills in the key and the response, a parallel process to mapping occurs which results in this number being the minimum between the number of non-correct fills in the key and the number of non-correct fills in the response. If the minimum were not taken, the systems would be unfairly over-penalized for mismatched fills. They would be given both missing and spurious scores for all mismatching fills instead of one incorrect for each mismatched fill in the response with a correlate in the key. The reasoning behind this parallels the notion of "connection" in the case of unmapped objects.

missing - the number of possible fills for which there was no response.

spurious - the number of actual fills for which no key was given.

In the event that alternative sets of fills are provided by the key, then the response set of fills is scored against each of the alternate sets in the key. The best key set is selected according to a pre-determined metric and the tallies for the remaining key sets are ignored.

If a slot fill in the key is marked as optional, then the scorer treats it in one of two ways: 1) if the response provides a slot fill, then it is scored as a normal slot fill; 2) if the response provides no slot fill, then the content of the key is ignored, and the response is not penalized for missing the fill.

During the scoring phase, the scoring tallies for all slot fills are totaled for all similar slot fills for all similar object types within the document, as well as for all similar slot fills for all similar object types across all the documents.

Report

After scoring is completed, the scorer provides a slot-by-slot score report for each document, and an overall summary score report and performance analysis for the set of all documents. For every document, the report displays the score results categorized by object type, and subcategorized within each object type by slot. A total slot score is also given for each document.

The overall summary score report displays the same object and slot-by-slot scores as the individual document score reports, but the numbers displayed are the totals across all documents. Similarly, the total slot score is the total of all the slots for all the objects across all the documents.

In addition to the scoring categories described above in the scoring section, the following metrics are also displayed in all of the score reports:

recall - the number of correct divided by the number of possible.

precision - the number of correct divided by the number of actual.

These are the basic metrics calculated for all tasks. Additional metrics are calculated for Named Entity, Template Element, and Scenario Template. See the Appendix entitled "MUC-6 Test Scores" for information on these metrics, which included Error-per-response-fill, Undergeneration, Overgeneration, Substitution, and Text Filtering.

Portability of the Scorer

The scorers are all driven by data in external files which allows the scoring software to be adapted to other similarly structured database objects without a change in code. This capability allows for flexibility from domain to domain and language to language. However, there are limits in terms of the formal structure of the database objects that we can handle and the available alphabets. The definition of the database objects is contained in the slot configuration file. The BNF of the database objects is manually translated into the form required by the software for the slot configuration file; this process could be automated in the future. Any legitimate BNF can be handled. The emacslisp version of the scoring software can handle any alphabets that the emacs text interface can handle. The C version will shortly be able to at least batch process extended alphabets and 2-byte character sets. An interface to the C versions of the scorers has not been completed for any language, but, when it is, it will be extended to cover non-English alphabets.

The configuration file contains scoring options that are allowed to be set by the user. Most of them have defaults, but certain ones require user designation, such as the filename of the messages, keys, and responses. The sizable number of options are all listed in the users' manual. The one that most concerns us here is the scoring option that allows key-to-response scoring and key-to-key scoring. During the definition of the task and when final answer keys have been made, key-to-key scoring is used to compare not only the slot-fill data itself between two versions of an answer key (usually versions produced by different annotators), but also some slot attributes that may be entered only in an answer key (alternate slot fills, optional objects and slot fills, and minimal strings). So key-to-key scoring is based on a broader range of comparisons than key-to-response scoring and is therefore a different and more difficult test. Key-to-response scores obtained in an interannotator test are probably a truer measure of human performance than key-to-key scoring provides. It is important to remember that essentially equal answer keys are being compared as key and response, and the decision as to which key is the key and which the response is arbitrary and affects the humans' scores.

ADAPTATION OF PREVIOUS MUC SCORING TECHNOLOGY TO MUC-6 TASKS

To keep costs down, SAIC reuses scoring code from past evaluations as much as possible while responding to changing consumer needs and the experience of past evaluations. In this evaluation, we wrote four scorers, one for each task. The scorer for Scenario Template was much the same as in the past except for improvements in mapping and performance (speed and memory usage) due to the translation into C. Scenario Template contains scores for relevancy judgments in the line labeled "Text Filtering" and the information retrieval metrics of recall and precision are used. The Template Element scorer was a low-cost adaptation of the Scenario Template scorer and does not include relevancy judgments because almost all articles contain these low-level objects.

The scorers for the Named Entity and Coreference tasks diverged from the usual approach because of the format of the input and the special needs for mapping and scoring. The format is SGML tagged texts. The format required a different parser than the templates or the low-level template objects. However, Named Entity input could still be internally represented as a low-level template (no pointers) object with slot fills. The score report reflects the difference in how we wanted to analyze the scores according to object type, subcategories, and placement in the document. To support this breakdown in the score report, the Named Entity scorer had to keep additional tallies. However, the mapping function was much simplified because the tags were anchored in the text and the text had to be overlapping in order for the items to be candidates for mapping.

The input for the Coreference task was close in format to the Named Entity input and had the easier mapping conditions as well, but the scoring was based on linkages rather than slot fills. The algorithm for scoring the linkages could be a simple counting process of the subsequent noun phrases in the coreference chain, but that would be computationally too expensive. The implemented algorithm for scoring the linkages was discovered by participants at MITRE Corporation [1]. It is elegant in its way of determining the minimal number of changes in the linkages required to make the response the same as the key to calculate recall and to make the key the same as the response to calculate precision. It defines recall and precision in a geometric way. Also, it is computationally inexpensive while being provably equivalent to counting subsequent noun phrases in the chain.

Translation from emacslisp into C

The semi-automated scoring program was originally developed for MUC-3 in emacslisp by General Electric using SAIC specifications. SAIC then took over the maintenance of that software for MUC-4. Emacslisp suited our needs at the time, which included rapid prototyping, a familiar text interface, and a portable lisp available to all participating and evaluating sites at no cost.

Over the years we have moved to complete automation of the scoring and have tried out several different approaches to mapping. As time passed, the limitations of emacslisp began to outweigh the its convenience. In MUC-5 the number of documents that we wanted to use in a data set caused a "memory exhausted" message during loading of the scorer. We were forced to divide the development and test articles and their answer keys into subsets making the scoring process even longer. Emacslisp does not allow applications programmers to manage memory usage, but C does. Also, even in batch mode the scoring took hours for each site. We knew that by translating to C we could get a sizable gain in speed.

After completing the translation of the scorer into C, we benchmarked 100 articles on a SPARC-20 and found that it took 10 seconds of elapsed time. We have not experienced and do not anticipate a memory problem with larger data sets because C's memory management capabilities have been taken advantage of in the coding. During the translation, we also implemented a new mapping algorithm that is more accurate than the lisp version. This change to C also allows us to implement all of the old features and add features that were put on hold because of the limitations of emacslisp. Also, we can actively pursue other more effective mapping algorithms that require more memory without very much speed impact. The speed of the C version is critical during system development, especially for those evaluation participants that use training in any part of their system. The diagnostics are produced faster as well. Overall, the scorer has become a much more useful tool for all consumers.

The status to date is that the Scenario Template and Template Element scorers are completely in C and can be run in batch mode only. The official scores were determined by the C version of each because of the more accurate mapping. The interfaces do not as yet exist, but are planned. The batch mode of the Named Entity scorer is almost completely translated into C. The Coreference scorer is still in emacslisp. The lisp versions of Named Entity and Coreference were used for the MUC-6 scoring. The scores will not change in these when converted to C because mapping candidacy is based on overlapping offsets. The official results would thus have stayed the same. The Coreference scorer will probably stay in emacslisp while the task definition is evolving in order to facilitate rapid prototyping.

FUTURE ENHANCEMENTS

We are planning to extend the scorer to handle non-English alphabets. We will begin with the Named Entity scorer, but because there is so much in common with the Scenario Template and Template Element scorers, these will be easier to modify. We are completing the Named Entity translation into C and expect to do the same with Coreference at a later date. We plan to develop a Graphical User Interface to all of the C scorers so that all of the useful emacslisp features translated and so that we can provide more features without noticeable impact on the performance.

REFERENCES

[1] Vilain, M., Burger, J., Aberdeen, J., Connolly, D., and L. Hirschman (1995) "A Model-Theoretic Coreference Scoring Scheme" Technical Report MITRE.