The MetaMap Mapping Algorithm

Alan R. Aronson

April 26, 2000

1. Overview

The algorithm used by MetaMap to construct its final mapping from text to the UMLS Metathesaurus is described in this document. The scope of the mapping effort is limited by segmenting incoming text into phrases using the SPECIALIST parser. MetaMap processes each phrase by generating variants of each phrase word, finding Metathesaurus strings (called candidates) with one or more of these variants, evaluating each candidate and finally choosing non-overlapping candidates to form the final mapping. This last step is the most complicated part of the algorithm, although variant generation is almost as complex. Both are made complicated by their recursive nature.

Input to the mapping algorithm consists of an ordered list of evaluations (evaluated candidates). The algorithm recursively produces partial mappings by searching the evaluation list for one which does not overlap the current partial mapping. If it finds such an evaluation, it adds it to the partial mapping and continues looking for more evaluations at the beginning of the list. If it cannot extend the current mapping, it records the current result and backtracks by throwing away the most recently added evaluation and continuing the search at that point. Details of the algorithm along with examples are given in the next section.

2. The Algorithm

The phrase *obstructive sleep apnea* will be used as a source of examples throughout the description of the MetaMap mapping algorithm. The Metathesaurus candidates for this phrase are

Meta Candidates (8):
1000 Obstructive sleep apnoea (Sleep Apnea, Obstructive) [Disease or Syndrome]
901 Apnea, Sleep (Sleep Apnea Syndromes) [Disease or Syndrome]
827 Apnea [Finding]
827 Obstructive (Obstructed) [Functional Concept]
827 Sleep [Functional Concept]
827 Sleep <3> (Sleep brand of diphenhydramine hydrochloride) [Organic Chemical, Pharmacologic Substance]

755 Sleeplessness (Sleep Initiation and Maintenance Disorders) [Mental or Behavioral Dysfunction, Sign or Symptom]
755 Sleepy [Finding]

and the final mapping is

Meta Mapping (1000):
1000 Obstructive sleep apnoea (Sleep Apnea, Obstructive) [Disease or Syndrome]

Note that the candidates are ordered by their score. If the candidate is not the preferred name of its Metathesaurus concept, the preferred name is included in parentheses. Semantic types for the concept are listed in square brackets. In this case the final mapping consists of the single concept "Sleep Apnea, Obstructive". If that concept were not present in the Metathesaurus, the final mapping would consist of the concepts "Sleep Apnea Syndromes" and "Obstructed" represented by the strings "Apnea, Sleep" and "Obstructive". (Actually, this is true only for normal MetaMap processing. MetaMap has an option, `--prefer_multiple_concepts`, which would result in two top-scoring mappings consisting of "Obstructive", "Apnea", and either "Sleep" or "Sleep <3>". Use of the option causes a simple change in the evaluation metric; subsequent mapping construction does not change.)

2.1 Input

Each candidate in the previous section is represented internally by a Prolog term of the form

```
ev(NegValue, MetaString, MetaConcept, MetaWords, SemTypes, MatchMap, InvolvesHead, IsOvermatch).
```

`NegValue` is the negative of the evaluation score for the candidate, `MetaString` and `MetaConcept` are the matching string and its concept, `MetaWords` are the lowercased words found in `MetaString`, `SemTypes` is the list of semantic type abbreviations for the concept, `MatchMap` is the correspondence between phrase words and `MetaString` words, and `InvolvesHead` and `IsOvermatch` are self-explanatory flags. For mapping purposes, the most important parts of an evaluation are the `NegValue` and the `MatchMap`. We only care about how well a candidate scored and what text words it matched.

The ev/8 term for the best scoring candidate is

```
ev(-1000, 'Obstructive sleep apnoea', 'Sleep Apnea, Obstructive',
[obstructive,sleep,apnoea], [dsyn], [[[1,1],[1,1],0],
[[2,3],[2,3],0]], yes, no).
```

One thing to note is that concept "Sleep Apnea, Obstructive" is equivalent to its *uninverted* form "Obstructive Sleep Apnea"; MetaMap only *sees* the latter. Everything else except MatchMap is fairly simple. MatchMap consists of a list of terms, *mapping components*, of the form

```
[PhraseComponent, ConceptComponent, VariationLevel].
```

The words in the text and in the (uninverted) candidate are numbered left to right starting with 1. Each `PhraseComponent` and `ConceptComponent` is a list of two numbers indicating the

range of words within the text or candidate. The `VariationLevel` is the score for how varied the concepts words are from the text words, 0 meaning no variation. (In this regard spelling variants and words differing only due to case are considered to be identical and hence involve no variation.) For our example phrase and the string “Obstructive sleep apnoea”, the first mapping component is `[[1,1],[1,1],0]`. It means that the first word of the text (*obstructive*) matches the first word of the string (*Obstructive*) with no variation. The second mapping component is `[[2,3],[2,3],0]`, and it means that words 2 and 3 of the text (*sleep apnea*) match words 2 and 3 of the string (*sleep apnoea*), again with no variation.

Another example is the candidate

```
ev(-901, ‘Apnea, Sleep’, ‘Sleep Apnea Syndromes’, [sleep,apnea],
[dsyn], [[[2,3],[1,2],0]], yes, no).
```

Here the single mapping component is `[[2,3],[1,2],0]` and means that words 2 and 3 of the text (*sleep apnea*) match words 1 and 2 of the string (*Sleep Apnea*) without variation, where word 1 of the string is Sleep because of uninversion of the original string.

The list of candidates shown in the last section is not the original list formed by MetaMap. *Redundant evaluations* (essentially due to different strings from the same concept) have been removed before MetaMap displays the candidates. Formally, an evaluation is redundant if it occurs later in the list than another evaluation involving the same concept and has the same *phrase involvement*, i.e., involves the same text words. An example of a redundant evaluation is

```
ev(-1000, ‘Sleep Apnea, Obstructive’, ‘Sleep Apnea, Obstructive’,
[obstructive,sleep,apnea], _432514, [[[1,3],[1,3],0]], yes, no).
```

This evaluation is redundant since it occurred after the first one (for string “Obstructive sleep apnoea”) in the candidate list above, it involves the same concept, “Sleep Apnea, Obstructive”, and has the same phrase involvement of all three words of the text. It is not necessary to have identical phrase components (even ignoring variation). In the current case, the redundant evaluation has only one mapping component whereas the first evaluation has two. The only thing that matters is that in both cases the same text words are involved in the match. (Note that the expression `_432514` appears in this evaluation since the semantic types of the concept have not been computed yet. The expression is just a Prolog variable.)

Besides removing redundant evaluations before displaying the candidate list, MetaMap also removes *subsumed evaluations* after displaying the list and before computing final mappings. A subsumed evaluation is one whose score is strictly worse that one higher in the list ~~(or the same but for the same concept)~~ and has the same phrase involvement. In our example, the only subsumed evaluations are the last two candidates for “Sleeplessness” and “Sleepy” since they both have lower scores than either “Sleep” or “Sleep <3>” and all involve the second text word, *sleep*. A subsumed evaluation can never be part of a mapping that scores as well as one containing the evaluation that subsumes it.

2.2 Partial Mappings

Before describing the algorithm for computing partial mappings, it is worthwhile to enumerate all mappings for our *obstructive sleep apnea* example. The first candidate, "Obstructive sleep apnoea", matches all the text and is therefore a complete mapping by itself. Continuing down the list, the second candidate, "Apnea, Sleep" matches everything but *obstructive*. Starting down the list again, we find that "Obstructive" is the only candidate that can extend the partial mapping; and since the text is exhausted, we have the second mapping. Nothing else involving "Apnea, Sleep" works, so we continue down the list to "Apnea". Two more scans of the list add "Obstructive" and "Sleep" to the partial mapping resulting in the third mapping. Backtracking on "Sleep" produces "Sleep <3>" for the fourth and final mapping. One additional consideration is the quality of the complete mappings. Once the components have been determined, the complete mapping can be evaluated just as individual candidates were earlier. This is a measure of the mapping quality and allows us to determine the best mappings, i.e., those with the highest evaluation scores.

The pseudocode for constructing mappings is given below. It assumes a ranked list e_1, e_2, …, e_n, of n evaluations and a stack s_1, …, s_m of m evaluation indexes representing a partial mapping. Thus each s_j is an index between 1 and n. Initially m=1 and s_1=1, i.e., the stack contains the single evaluation e_1. Note that the empty stack is represented by m=0.

```
    ***Extend the mapping***
A:  i = s_m + 1
B:  if i > n
        go to C
    if e_i does not overlap anything in the stack
        push it onto the stack
        go to A
    else
        i = i + 1
        go to B
    ***Record the mapping and continue***
C:  if the stack is empty
        stop
    else
        save the stack; it represents a mapping
        i = s_m + 1
        pop the stack
        go to B
```

The mappings found by the above algorithm are evaluated (using the candidate evaluation metric) and sorted according to the evaluation score.