Comparison of LVG and MetaMap Functionality

Alan R. Aronson

November 14, 1994

The Lexical Variant Generator (LVG) and MetaMap are two computer modules which generate lexical variants for words occurring in free text. These notes characterize the similarities and differences between the two. Section 1 compares LVG and MetaMap at a general level; Section 2 describes variant generation, itself; and Section 3 summarizes the findings.

1. Introduction

LVG and MetaMap both compute lexical variants but were developed for quite different purposes: LVG's *raison d'être* is lexical variant generation whereas MetaMap's main purpose is to map text to corresponding concepts in the UMLS® Metathesaurus (Meta), one of the UMLS knowledge sources. Besides generating lexical variants, LVG has the subsumed ability to normalize words and the supplementary ability to produce tokens suitable for use as a word index. Generating lexical variants is one of four major components of MetaMap. The other three components parse text, search for and evaluate Meta terms, and combine the best Meta terms into a coherent mapping from text to Meta concepts.

Both LVG and MetaMap were developed in a Unix environment on Sun Workstations. LVG is entirely written in C, and MetaMap is mostly written in Prolog using C modules to implement some of its basic functions. LVG is implemented as a main command, `lvg`, and two secondary commands, `norm` (for normalization) and `wordind` (for word index). The LVG commands are filters and can be used in combination with other Unix filters. MetaMap is implemented as a single command, `metamap`. It has both interactive and batch modes of operation but is not a filter.

2. Variant Generation

The most striking difference in the way that LVG and MetaMap generate variants is that LVG consists of a collection of independent submodules which can be combined in any way in order to generate the variants desired by the user. MetaMap's variant generation algorithm is completely fixed. LVG provides complete flexibility with no guidance on creating a complete variant

generation strategy. MetaMap provides no flexibility but engenders a variant generation strategy developed iteratively and with attention to the linguistic and domain-specific consequences of alternative substrategies. The rest of this section describes the LVG submodules (2.1); the MetaMap variant generation algorithm (2.2), and an LVG approximation to the MetaMap algorithm (2.3).

2.1 LVG Variant Generation

LVG's functionality is determined by that of its submodules which are listed below[1] along with examples of their usage. Each submodule acts as a filter by performing its function on its input producing its output. Designing a compound variant generation strategy consists of constructing one or more sequences of the filters called *flows*.

- `l`—Lowercase

```
lan@nls3> echo 'NLS' | lvg -fl
NLS|nls|2047|255|l|1
```

- `u`—Uninvert

```
lan@nls3> echo 'red blood cells' | lvg -fu
red blood cells|red blood cells|2047|255|u|1
lan@nls3> echo 'cells, blood, red' | lvg -fu
cells, blood, red|red blood cells|2047|255|u|1
```

- `w`—Sort the words in ascending order

```
lan@nls3> echo 'red blood cells' | lvg -fw
red blood cells|blood cells red|2047|255|w|1
```

- `g`—Remove Genitive markers

```
lan@nls3> echo "Paget's Disease of Bone" | lvg -fg
Paget's Disease of Bone|Paget Disease of Bone|2047|255|g|1
```

1. Two submodules, Generate Spelling Variants and Remove Stop Words have not been implemented yet.

- p—Remove punctuation (non-alphanumeric) characters

```
lan@nls3> echo '(normal 4000-10,800/cu mm)' | lvg -fp
(normal 4000-10,800/cu mm)|normal 400010800cu mm|2047|255|p|1
lan@nls3> echo '1.5 to 8.0' | lvg -fp
1.5 to 8.0|15 to 80|2047|255|p|1
```

- b—Generate the base form (using rules of inflection)

```
lan@nls3> echo 'veiled' | lvg -fb
veiled|veil|1024|1|b|1
lan@nls3> echo 'cells' | lvg -fb
cells|cell|128|1|b|1
cells|cell|1024|1|b|1
lan@nls3> echo 'veiled cells' | lvg -fb
veiled cells|veiled cells|2047|255|b|1
```

- B—Generate the base form for each word in the term

```
lan@nls3> echo 'veiled cells' | lvg -fB
veiled cells|veil cell|2047|1|B|1
```

- `i`—Generate inflectional variants

```
lan@nls3> echo 'veiled cells' | lvg -fi
veiled cells|veiled cell|128|1|i|1
veiled cells|veiled cellses|1024|128|i|1
veiled cells|veiled cellses|128|8|i|1
veiled cells|veiled cell|1024|1|i|1
veiled cells|veiled cellser|1|2|i|1
veiled cells|veiled cellsest|1|4|i|1
veiled cells|veiled cellsed|1024|32|i|1
veiled cells|veiled cellsing|1024|16|i|1
veiled cells|veiled cellser|2|2|i|1
veiled cells|veiled cellsest|2|4|i|1
veiled cells|veiled cellss|128|8|i|1
veiled cells|veiled cellss|1024|128|i|1
```

- `d`—Generate derivational variants

```
lan@nls3> echo 'veiled cell' | lvg -fd
lan@nls3> echo 'cell' | lvg -fd
cell|cellular|1|255|d|1
lan@nls3> echo 'cellular' | lvg -fd
cellular|cell|128|255|d|1
cellular|cellula|128|255|d|1
cellular|cellula|128|255|d|1
cellular|cellularity|128|255|d|1
lan@nls3> echo 'cell' | lvg -fdd
cell|cell|128|255|dd|1
cell|cellula|128|255|dd|1
cell|cellula|128|255|dd|1
cell|cellularity|128|255|dd|1
```

2.2 MetaMap Variant Generation[1]

MetaMap variant generation begins by computing a set of variant generators for a given phrase. A variant generator is any *meaningful* subsequence of words in the phrase where a subsequence is meaningful if it either occurs in the SPECIALIST lexicon or is a single word. For example, the variant generators for the noun phrase “of liquid crystal thermography” are *liquid crystal thermography, liquid crystal*, *liquid*, *crystal* and *thermography* (prepositions, determiners, conjunctions and punctuation are ignored). Note the multi-word generators. A simpler example is the phrase “ocular complications” which has only two generators, *ocular* and *complications*.

Variants are computed for each of the variant generators according to the scheme pictured in Figure 1.

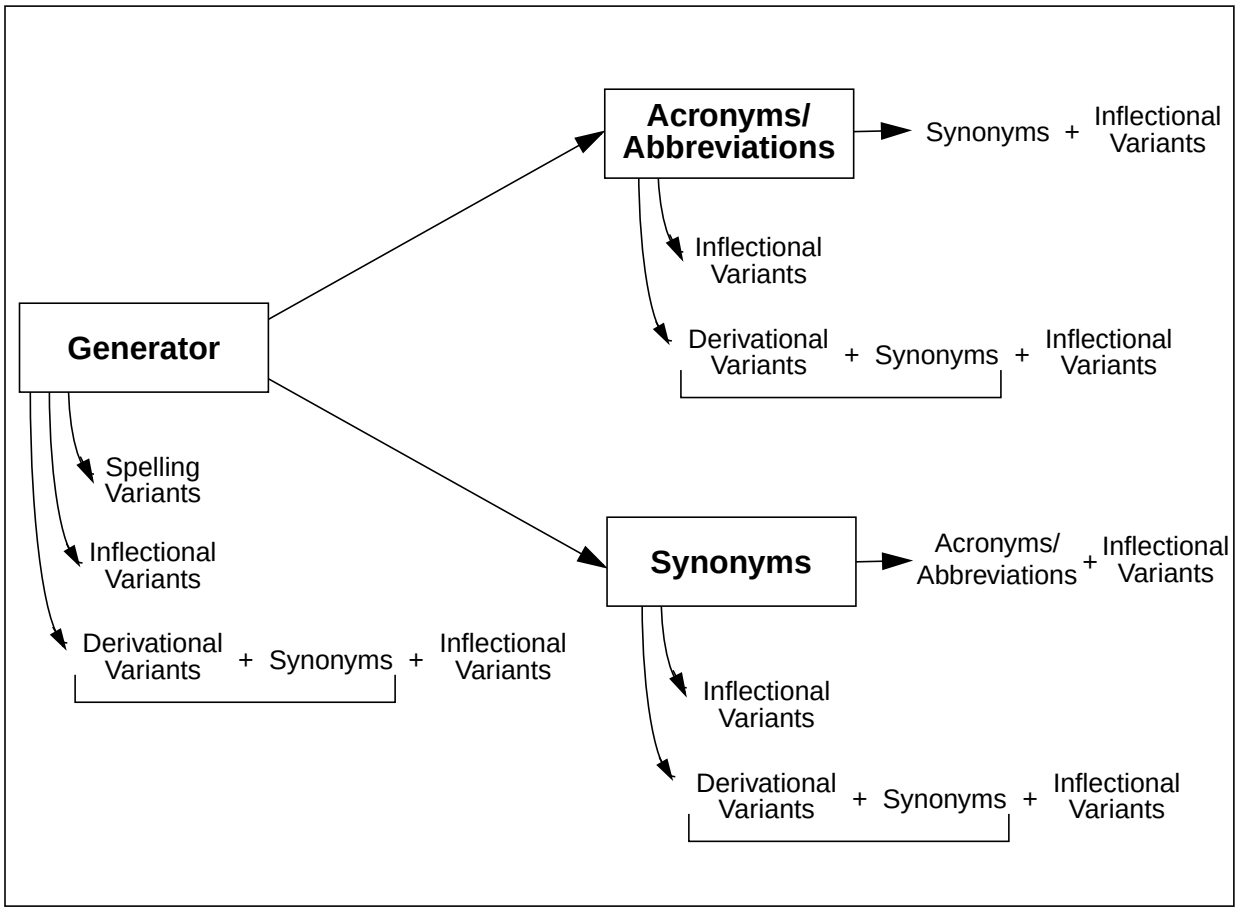

Figure 1. MetaMap Variant Generation

1. This section is a minor revision of Section 0.1.3 Noun Phrase Variants of the report *Mapping Noun Phrases in Free Text to Meta Terms* (June 8, 1993).

The computation for each generator proceeds as follows:

1. Compute all acronyms, abbreviations and synonyms of the generator. This results in the three sets Generator, Acronyms/Abbreviations and Synonyms which are highlighted with boxes in the figure;
2. The elements of the three sets are augmented by computing the inflectional and derivational variants (and spelling variants for the generator, itself). In addition, the derivational variants are augmented with their synonyms, and then both are inflected;
3. For each member of the Acronyms/Abbreviations set, compute synonyms and their inflections; and
4. For each member of the Synonyms set, compute acronyms/abbreviations and their inflections.

The issue of whether to recursively generate variants of a given type is handled as follows:

- Inflectional variants are not recursively generated since any inflectional variant of an inflectional variant is already an inflectional variant of the original variant;
- Acronyms and abbreviations are not recursively generated since doing so almost always produces incorrect results. For example, the abbreviation na of sodium has expansions “nurse’s aide” and “nuclear antigen” both of which are unrelated to sodium; and
- Derivational variants and synonyms are recursively generated since this often produces useful variants.

The variants computed for the generator *ocular* are shown in Figure 2. Following each

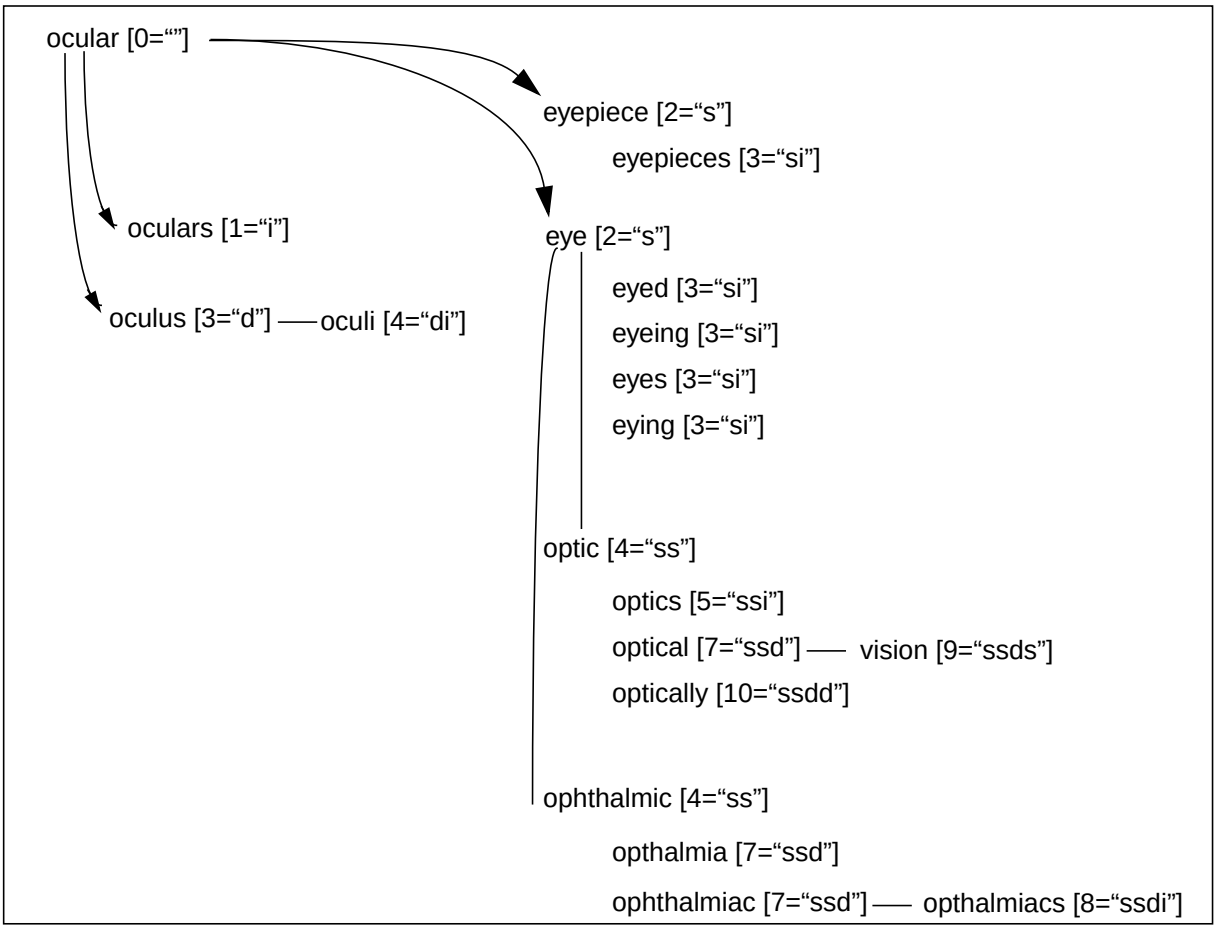

Figure 2. Variants for the generator *ocular*

variant is its variant distance score (a rough measure of how much it varies from its generator) and the history of how it was computed. For example,

- *oculus* (with variant distance 3 and history “d”) is simply a derivational variant of the generator *ocular*;
- *optical* (with variant distance 7 and history “ssd”) is a derivational variant of a synonym (*optic*) of a synonym (*eye*) of *ocular*; and
- *vision* (with variant distance 9 and history “ssds”) is a synonym of the derivational variant *optical* described above.

2.3 An LVG Approximation of the MetaMap Variant Generation Algorithm

Referring to Figure 1 it is a simple matter to construct a set of LVG flows roughly corresponding to the MetaMap variant generationalgorithm. It is important to note that flow options a (Generate acronyms/abbreviations) and y (Generate synonyms) are not part of LVG. In order to implement MetaMap's variant generation algorithm using LVG, such capabilities would have to be provided either by LVG or independently.

```
-fn -fs -fi -fbd -fbdy -fbdi -fbdyi
-fba -fbay -fbayi -fbai -fbad -fbady -fbadi -fbadyi
-fby -fbya -fbyai -fbyi -fbyd -fbydy -fbydi -fbydyi
```

The three lines of flows above correspond to the variants generated by the Generator, Acronyms/Abbreviations and Synonyms, respectively, of Figure 1. In addition to the flow options, it is necessary to use `lvg`'s `k` option to restrict inflectional and derivational variation to produce actual words. Ignoring the flows which do not yet exist, the currently available LVG approximation to MetaMap's variants for the generator *ocular* is:

```
lan@nls3> echo 'ocular' | lvg -fn -fi -fbd -fbdi -k i:1 -k d:1
ocular|ocular|2047|255|n|1
ocular|oculars|1024|128|i|2
ocular|oculars|128|8|i|2
ocular|oculars|1024|128|i|2
ocular|oculus|128|255|bd|3
ocular|oculi|128|8|bdi|4
```

The LVG results are the same as those generated by Generator in the MetaMap algorithm.

Simulating the generation of variants for the synonym *optic* of the synonym *eye* of the generator *ocular* produces:

```
lan@nls3> echo 'optic' | lvg -fn -fi -fbd -fbdi
optic|optic|2047|255|n|1
optic|optics|1024|128|i|2
optic|optics|1024|128|i|2
optic|optics|128|8|i|2
optic|optical|1|255|bd|3
optic|opt|128|255|bd|3
optic|optical|1|255|bd|3
optic|opts|128|8|bdi|4
```

In this case, LVG produces incorrect variants such as *opt*. MetaMap avoids generating *opt* by remembering the possible parts of speech for each variant generated and restricting subsequent generation accordingly.[1] In addition LVG does not generate *optically* since it does not recurse in generating variants, and it does not generate *vision* since it does not support synonyms.

3. Conclusion

LVG's current variant generation functionality is not sufficient to support MetaMap's variant generation algorithm. However, the following modifications would make it able to do so:

- the ability to access LVG submodules programmatically;[2]
- the implementation of the spelling variant and stop word[3] submodules;
- the addition of acronym/abbreviation and synonym submodules;
- the ability to specify recursion for a flow with corresponding changes to the flow type output field;[4] and
- the ability to compute parts of speech for eliminating incorrect variants during the generation process.[5]

1. Although it seemed possible that some combination of LVG options would prevent the generation of *opt*, Allen assures me that this is not the case because LVG does not use the inflectional information available from the SPECIALIST lexicon.

2. This modification is necessary only to achieve adequate performance and to shield users from the complex LVG interface; it does not affect functionality.

3. The stop word submodule would be used to approximate MetaMap's use of parsing information to ignore lexical items with little information content: prepositions, determiners, conjunctions and punctuation.

4. This modification is not essential but would relieve LVG users of controlling recursion themselves and also enhance performance.

5. This modification subsumes the need for a stop word submodule.

Despite LVG's current inability to support MetaMap's variant generation algorithm, its generality gives it the ability to support the development of many variant generation strategies.[1] The modifications suggested above would enhance that ability to develop realistic variant generation strategies similar to that of MetaMap.

1. A variant strategy development tool providing knowledgeable access to LVG would facilitate the development process by freeing the developer from knowing LVG interface details.