Methods and apparatus for encoding of explicit morphological knowledge into a full-form lexicons compiled as finite-state automatons for usage in morphological guessing and stemming

A program is disclosed that converts an existing Directed Acyclic Word Graph (DAWG) morphological dictionary into a highly efficient morphological guesser without needing access to any linguistic knowledge.

Morphosyntactical characteristics of words (such as part-of-speech, plural/singular, ...) in economically significant Indo-European languages can be guessed by postfix pattern. Traditional guessers use explicit linguistic knowledge. In this paper we describe the method how to extract the knowledge from preexisting dictionaries (annotated lists of words), and how to efficiently use finite-state device for guessing morphosyntactic characteristics of out-of-vocabulary words while preserving the correct morphosyntactic information about the words in the dictionary.

DAWG is efficient finite-state device. Logically it is list of words (surface forms) provided with glosses. The basic idea is to create new DAWG dictionary from already existing DAWG dictionary, by inverting the sequence of characters in each word so that the ending (which is most likely to determine the word type) is examined first. This means that the part of speech and inflection can be determined by examining a minimal set of letters. Furthermore the algorithm which is presented will allow the size of the dictionary to be substantially reduced by pruning multiple nodes which lead to the same gloss and only storing nodes which represent paths for words that do not follow the normal default rules. Our initial trials indicate that 95-97% of the nodes in an unminimized DAWG can be eliminated by this method. Furthermore the algorithm to eliminate unnecessary nodes runs very quickly. Small scale trials have also indicated that it seems to make good guesses for out of vocabulary words.

This program is conceptually similar to that which is described in the paper by Jan Daciuk entitled "Treatment of Unknown Words", which was published in the Proceedings of the workshop on Implementing Automata WIA'99, Postdam, Germany, 1999. The main difference between this paper and the algorithm described here is:

1. Whereas Daciuk discusses generic Finite State Transducers (FSTs) which can have non-deterministic transitions and cycles, the algorithm described in this paper only discusses Directed Acyclic Word Graphs (DAWGs).
2. His rules will miss out some annotation possibilities (see his rules 6&7), while ours will not. However, this is more a tuning factor rather than an inherent advantage of our method.
3. Daciuk does not actually disclose an algorithm, he only lists the rules that the algorithm must obey.

Many search and categorization systems want to reduce the word in the text to their root or lemma form (e.g. *walk*, *walking* and *walked* would all get treated the same as *walk* for indexing purposes). This task is commonly referred to as lemmatization. The existing systems to do this fall into two categories,

- Dictionary based analyzers.
- Rule-based analyzers such as morphological guessers, stemmers.

Dictionary based systems use a complete dictionary of the language in question

and for each word encountered in the text, they consult the dictionary to find its corresponding root or lemma form (e.g. InXight http://www.inxight.com/products/core/). Stemmers and morphological guessers work similarly (also morphological guessers take into account mainly inflectional morphology and try to restore lemma, while stemmers take into account derivational morphology also and the result they produce is not necessary an orthographic word). Both of these devices use hand crafted rules to convert surface words to their root form (e.g. in English we might have a rule stating that any word ending in *ing* is a verb in the present perfect tense and the lemma form can be constructed by deleting the last three letters in the word). See http://www.tartarus.org/~martin/PorterStemmer/ for a description of the most well known stemming algorithm for English or see http://snowball.tartarus.org/ for an effort to extend this to more languages.

There are various advantages and disadvantages of both approaches. These are summarized by the following table:

Dictionary Based Systems	Stemmers and guessers
Advantages: ● Can recognize all irregular forms ● Will never over recognize	Advantages: ● They are relatively easy and cheap to implement. ● They can be fast and don't need to consume much memory. ● Will recognize newly coined words so long as they follow the standard inflection rules of the language (e.g. standard English rules would allow us to guess that *punged* was the past tense of the verb *punge* and even if we don't know the meaning we could easily categorise documents containing this word along with ones containing the word *punging*)
Disadvantages: ● The dictionaries are expensive to produce and maintain ● Even the most complete dictionary will not have adequate coverage for specialized domains and so many words will fail to be recognized ● They can be slow or if tuned for speed they will require a lot of memory	Disadvantages: ● Many words without a common ending will not be recognized ● They will sometimes over recognize (e.g. the example rule above will incorrectly categorize the noun *ring* as a verb with lemma *r*) ● Requires good knowledge of the language to tune the rules so that the best recognition rate is achieved.

What this idea describes is a way to alter a dictionary based system so that it can get the main benefits of guessers and stemmers (i.e. small data dictionaries and ability to process out of vocabulary words), while retaining the advantages of a dictionary based system (e.g. increased accuracy of recognition and greater ability to deal with irregular inflections). While in theory the system can be applied to any language, the

best results will be achieved for languages whose inflection rules involve adding inflectional endings at the end of the words (this includes almost all economically significant languages).

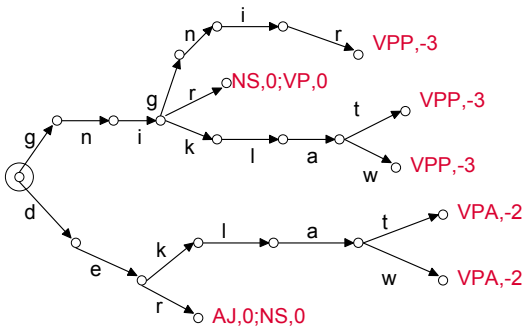
The principles captured by this idea are:

- Reverse the entries of dictionary DAWG and correspondingly reverse the order of letters in each look-up word. In formal terms of state-transitions space it means that the end of the look-up word is used as the initial state, the last letter is used for finding the first transition, etc. In most cases this will mean that we will on average have identified the part of speech after processing fewer letters than would be required if we were processing characters from the start of the word as is done in conventional dictionary look-up.
- Eliminate all nodes in the DAWG beyond those required for identifying inflectional endings except in the case of irregular words. The idea of eliminating unnecessary transitions has been tried before (e.g. the work by Daciuk) but our idea should perform better at finding the critical points since our dictionary entries are reversed and most languages have their inflectional marking at the end of words. In addition we are proposing a mechanism to allow recording or defaults and exceptions which we believe is unique.

To illustrate the algorithm, consider a simple dictionary containing only the following words:

Word	**Lemma**	**Gloss**	**Meaning**
walking	walk	VPP,3	A verb in the present perfect tense. It's lemma is found by deleting the last 3 characters of the word.
talking	talk	VPP,3	as above
red	red	AJ,0	An adjective. Its lemma is the same as the surface form.
walked	walk	VPA,2	A verb in the past tense. It's lemma is found by deleting the last 2 characters of the word.
talked	talk	VPA,2	as above
ring	ring	VPR,0; NC,0	This word has two possible interpretations. The first says that it is a present tense verb and the second says that it is a noun. In both cases the lemma is the same as the surface form
ringing	ring	VPP,3	A verb in the present perfect tense. It's lemma is found by deleting the last 3 characters of the word.

The first step is to compile this into a "reversed wordforms DAWG". The resulting DAWG is shown below:



The compilation of this DAWG is in principle no different from a normal build process. The only adaptations to our existing build program would be:

- Reverse the strings before adding to DAWG
- Revise the gloss creation logic to handle the subtly different gloss semantics (i.e. the cut and paste is applied to the end of the word which is now at the start of the DAWG)
- We do not carry out the minimization step, because this would complicate the task of eliminating redundant nodes.

This DAWG will behave exactly like our existing dictionaries with the only difference being that text must be processed in reverse order. i.e.:

1. Start at the initial node (marked by a double circle in the illustration)
2. Point at the **last** character in the word
3. Examine the character being pointed at to see if it matches any outgoing transition in the current node (The exact procedure for matching depends upon the way the node is being represented, but that is not relevant here.)
   - If a matching arc is found, make the node pointed to by the arc the current node, move the character pointer to the **preceding** character and then proceed to step 4
   - If no matching arc is found, then we fail to recognize the word.
4. Check if our pointer is before the start of the word i.e. on a space.
   - If we are not at the start of the word then to the go back to step 3
   - If we are at the start of the word return the gloss associated with the current node. However, if there is no gloss associated with the current node then we fail to recognize the word.

At this stage there is no real benefit of having the wordforms in DAWG reversed. The benefits of the reversing only become apparent when we start eliminating redundant nodes. However, to facilitate the eliminating of redundant nodes we need to introduce a different gloss with a slightly different semantics. I will call these glosses probable_glosses and I will distinguish them in the diagrams by placing them within brackets. The meaning of these glosses is as follows:

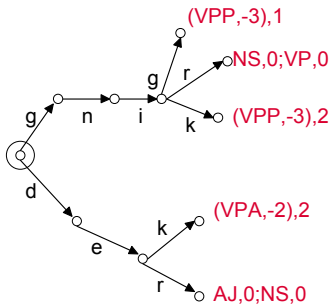
Normal_gloss => if we reach the start of the word at this node, then take this gloss.
Probable_gloss => if we fail to match at any point after this node, then take this

gloss. However, if we meet a subsequent probable_gloss it will supersede this one. Also this probable_gloss will have no effect if we terminate on a normal_gloss.

The revised rules for executing the DAWG are:

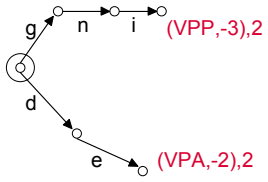
1. Start at the initial node (marked by a double circle in the illustration)
2. Point at the **last** character in the word
3. Set the probable_gloss variable to NULL.
4. If this node has any probable_gloss entry records this in our probable_gloss variable overriding any previous value.
5. Examine the character being pointed at to see if it matches any outgoing transition in the current node (The exact procedure for matching depends upon the way the node is being represented, but that is not relevant here.)
   - If a matching arc is found, make the node pointed to by the arc the current node, move the character pointer to the **preceding** character and then proceed to step 6
   - If no matching arc is found, then return the value of the probable_gloss variable as our result. If the probable_gloss variable is NULL this means that we fail to recognize the word.
6. Check if our pointer is at the start of the word
   - If we are not at the start of the word then go back to step 4
   - If we are at the start of the word return the gloss associated with the current node. However, if there is no gloss associated with the current node then we return the value of the probable_gloss variable. If this variable is set to NULL, this means we fail to recognize the word.

A visual inspection of our original DAWG shows that some of the paths can only lead to one possible gloss, therefore we can consider them redundant and change the DAWG to the form shown below. (A formal algorithm for collapsing these nodes will be given later in this document).

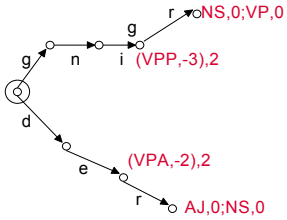


It is clear that if any of our dictionary words are matched against this DAWG, they will return the same gloss as in the original DAWG. However this DAWG is also similar to a stemmer, in that it will correctly recognize non-dictionary words. For example, you can verify that *sucked* and *sucking* will be successfully recognized as forms of *suck*. However, it will only correctly recognize VPP,-3 as the correct gloss for words ending in *ging* or *king* rather than the more general rule of handling all words ending with *ing* in the conventional rule set. But it is superior to the Porter stemmer because it recognizes some words which are exceptions to the rules (i.e. *red* and *ring*).

The following is the DAWG which performs equivalent to the Porter stemmer for our limited set of words (This DAWG implements only two rules, but the latest version of the English stemmer has a very large number of rules):



We can easily produce the following DAWG which combines the Porter stemming rules with recognition of the exception words *red* and *ring*.

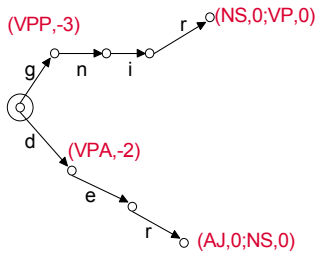


In real life, however the lexicon DAWG is too large to be visually examined and manipulated in this manner. Therefore we have invented the following algorithm which will safely remove all redundant nodes from our reversed DAWG. (Note: we might need refinement of this algorithm if we allow loops in our DAWG).

1. Add the following variables to each node and initialize them. (If it is easier to implement, this could also be implemented by means of a few arrays with space for an entry for each node).
   - possible_gloss (initial value = NULL)
   - confidence_factor (initial value = 0)
   - alternative_count (initial value = 0)
2. Set the current node to the initial node.
3. The action to take depends upon the type of the current node
   1. If this node is a final node (i.e. no outgoing arcs and contains a gloss), set the confidence_factor to 1, set the possible_gloss field to the same as the gloss, leave the alternative_count=0 and then return.
   2. If this is a non-final node (i.e. has outgoing states) then set the current node to the nodes pointed to each of the arcs in turn and then call this function (starting at step 3 above). When this has been done for all outgoing arcs it is OK to proceed to step 4.
4. Examine the possible_gloss and confidence factor variables each of the nodes pointed to by the arcs from this node, and calculate the sum of the confidence_factors for each of the possible_gloss values encountered.

5. Pick the possible_gloss value with the highest total confidence_factor and use it for this node.
6. Set the confidence_factor for this node to be the sum of the confidence factors for all of the child nodes which have the same possible_gloss value that we have just selected above.
7. Set the alternative_count to the total of the alternative_counts from each of the child nodes plus the total of the confidence_factor values from all nodes which have a different possible_gloss from the one we selected for this node.
8. Calculate the ratio of confidence_factor/alternative_count which is effectively the probability of our possible gloss being correct. If this falls below a threshold (e.g. 0.5) then we are in trouble??? ... this is an extension of the algorithm to stop it being to aggressive .... but do we need it??
9. For each of the target nodes which have alternative_count = 0 and the same possible_gloss as this one can now be safely eliminated.
10. For each of the target nodes that have the same possible_gloss as this node, we can remove the possible_gloss from the child node because it is redundant and it will allow space saving (Note: if we want to do minimization later this step is not safe)
11. Return and handle the parent node.

Running this algorithm on our original DAWG transforms it to the following DAWG



Because of our limited vocabulary we have ended up with a very aggressive recognizer which will treat any word ending in *g* other than *ring* as a VPP,3 gloss and will treat any word ending in *d* other than *red* as a VPA,2 gloss. However, if you run this algorithm on a reasonably complete dictionary it should result in a DAWG which recognizes all dictionary words and applies rules similar to the porter stemmer for all out of dictionary words encountered. In fact one of the benefits of this algorithm is that it can discover the correct rules from a dictionary without the need for manual tuning to the language by a linguistic expert. (Adapting the porter stemmer to a new language tends to be a substantial project).

Disclosed by International Business Machines Corporation