Giving Them What They Want: Search Strategies for Electronic Dictionaries

Michal Boleslav Měchura
Dublin City University

This paper deals with how humans search electronic dictionaries. It raises the point that users often make dictionary searches with misspellings, with inflected words copied and pasted from elsewhere, with complete sentences or fragments thereof, and with other kinds of low-quality input, and suggests methods for dealing with such phenomena in a preemptive manner. The issues addressed include searching with inflections, dealing with multi-word items, misspelling detection and text normalization. Additionally, the value of log files is emphasized as a source of information on user behaviour.

Introduction

When a computer user visits an online dictionary on the Web or launches an electronic dictionary from a CD-ROM, the interface they are presented with is usually a text box. The user types one or more words in the box and, upon clicking a *search* button, a flurry of activity ensues at the back-end for a few seconds, eventually returning a listing of search results in some form. This paper deals with what happens during that short span of time between the clicking and the results appearing.

At first impression, the process of consulting an electronic dictionary would seem straightforward and analogous to a printed dictionary: a user asks for a word and, if the word matches one of the headwords in the dictionary, the software will present the appropriate dictionary article on screen. In reality, the process is not always this simple. Often, users search dictionaries for things that cannot be found in them, or cannot be found in the form in which they typed them. Dictionary users are known to search indiscriminately for all sorts of material including misspellings, inflected words, various types of multi-word items, phrase fragments, even whole sentences—in other words, things that cannot easily be matched with any of the headwords of a dictionary. A smart search strategy is needed to resolve such requests.

Before we go any further, it has to be mentioned that one common strategy for dealing with this challenge is not to deal with it at all. Many dictionaries online today fail to return any useful results when searched for anything other than neatly lemmatised words. Such an approach is hard to justify. Electronic dictionaries are increasingly becoming marketable software products like any other, and their popularity or otherwise is often determined by concerns of usability and user friendliness, in addition to the actual dictionary content.

It is therefore a fact of life that when preparing an electronic dictionary on CD-ROM or on the Web, we cannot expect the users to behave as "ideal" dictionary users. On the contrary, we must expect them to search for all kinds of "noisy" textual input which is going to be difficult to match against the dictionary's content.

Log file analysis

In the past few years, studies have begun to appear in which the log files collected by dictionary websites are exploited to analyse user behaviour, mainly to determine which new entries need to be added to the dictionary in the future (see for example de Schryver et al. 2006, Bergenholtz and Johnsen 2005, de Schryver and Joffe 2004). Another way to exploit log files is to look for clues and ideas on improving the search algorithm. When reviewing a search log, it is instructive to look at searches that returned zero results and to try and find the reasons for the failures. In many cases, the word or expression simply is not present in the dictionary and there is nothing a search algorithm can do. In many other cases, the item is in the dictionary but the user has

misspelled it, has failed to de-inflect it, or there is some other problem which could have been resolved automatically during search, if only the search algorithm had been smarter.

To give just one example for many, I have observed (through log files) a user of the *focal.ie* terminology dictionary who searched for the phrase *radom sample*, obtaining no results, then trying only *radom*, again obtaining no results, and eventually giving up, apparently unaware of the misspelling. Had the search algorithm been able to detect misspellings, it could have recognized *radom* as a possible misspelling of either *random* or *radon,* sparing the user some frustration and the dictionary some loss of user confidence.

Search techniques

The rest of this paper will review some techniques and ideas which can be employed in search algorithms to tackle some of the challenges of "messy" user input, including inflection awareness, misspelling detection, and the handling of multi-word items. Some of the techniques presented here are based on my own experience while designing the *focal.ie* dictionary, some are simply suggestions for further discussion. Either way, what follows is a rather eclectic collections of ideas that may or may not lead to the desired result, but will hopefully be worthy of consideration for anybody designing an electronic dictionary.

Inflection awareness

An inflection-aware search algorithm is able to recognize the connection between a user's inflected input and a corresponding uninflected headword in the dictionary. Inflection awareness may also be needed for intelligent handling of multi-word items, which will be discussed later.

Some languages, such as English, have relatively simple and regular morphologies. A small set of rewrite rules, such as stripping the *-ing* and *-ed* endings to try and find the infinitive of a verb, will generally suffice and will achieve a good coverage of most of the language's inflection phenomena. Thus, when a user searches for *appealed* and such a word does not exist in the dictionary, the algorithm may automatically strip the final morpheme and attempt to search for *appeal* instead. In most cases, a simple approach like this constitutes a good morphology guesser for English. In cases when the rewrite rules produce an invalid guess, such as *speed* → *spe* or *crossbred* → *crossbr*, the search will simply fail as normal and the user will never even see the invalid guess.

Many other languages have complex or irregular morphologies, and the rewrite approach will not work. This is also the case for irregular English inflections, such as *leave* → *left*, *ox* → *oxen*, rare as they may be. In many languages, a morphological analyser, generator or guesser may already be available from a third party and it may be possible to incorporate such a module into the search algorithm. It is useful to check with linguistics departments in universities and research institutions, and with commercial language technology companies. When such a software module is not available or when it is not practicable to use one, a data-oriented solution may suffice. It may be possible to include in the dictionary database (invisibly to the user) a list of all inflected forms of all lemmas, or perhaps just a shorter list of the most frequently occurring forms of the most frequently occurring lemmas. Such lists can sometimes be downloaded from the Internet and can be used freely for non-commercial purposes, such as the *e_lemma.txt* list for English (Someya 1998), can be culled from a lemmatized corpus, or generated by a morphological module on a once-off basis, and then consulted automatically during each search. Even when this is not possible, the lexicographers may have already provided some inflected forms in the dictionary itself, for example the entry for *leave* may mention the irregular inflected form *left*. It should be possible to include this data in the search process to achieve at least some coverage of the language's inflectional behaviour.

For the sake of completeness, let it be mentioned that inflection awareness can be approached from a different angle altogether. One can choose to ignore the inflection rules of the given language, and instead perform matching purely on similarity, using fuzzy matching algorithms such as those common in misspelling detection (see below). This is common in translation-memory tools such as SDL Trados where each sentence to be translated is matched against a database of previously translated sentences. However, such a technique would fail on irregular

inflections where there is no similarity between forms of the same word, such as the Irish verb *abair* "say" and its past tense *dúirt* "said".

Multi-word items

Many dictionaries contain large numbers of multi-word items with lemma-like status. While building the *focal.ie* terminology dictionary, it has transpired that users search for these differently than for single-word entries. Users rarely type multi-word strings in their entirety. Instead, they often type just one of the words, and then rely on the search algorithm to offer a listing of multi-word items where this word occurs. For example, a user types *identity* but is really looking for *identity theft*.

This need is straightforward to meet in a weakly inflecting language such as English. When dealing with a strongly inflecting language, however, the algorithm must be inflection-aware. For example, when searching for the Irish word *monarcha* "factory", users may expect to be offered terms such as *oibrí monarchan* 'factory worker' and *limistéar monarchana* "factory area" in which the lemma appears with various inflections and mutations.

This can be approached in two ways. One way is to equip the search algorithm with an ability to generate the inflected and mutated forms of each lemma in real time, or to have access to a pre-generated list of those. For example, when a user searches for *monarcha*, the search algorithm will generate all possible forms of the lemma (*monarchan*, *mhonarchan*…) and then locate all multi-word items that contain either of these. It does not matter if the algorithm over-generates (generates invalid forms as well valid ones) because the user will never see the invalid forms—unless, of course, they happen to be valid forms of some other lemma. Another possibility is to parse and lemmatize each multi-word item beforehand and use the output as an index. For example, when a dictionary editor creates an entry for *oibrí monarchan* "factory worker", the system may parse the term into the lemmas *oibrí* "worker" and *monarcha* "factory", store these on an index in the database, and then consult the index during multi-word searches.

Another feature worth having in a search algorithm is awareness of word boundaries. When a user searches for *verse*, it would be irrelevant to offer matches such as *adverse* or *overseas* where *verse* occurs as a mere string of letters but not as a word. A smart search algorithm will exclude such false matches, and offer only valid matches such as *free verse* and *verse dialog*. However, some compromises may need to be made in this area. A compound word like *lunchtime* contains the lemmas *lunch* and *time* but they are not demarcated orthographically in any way and the compound would therefore be filtered out when searching for *lunch* or *time*. Languages differ in how frequently such compound words occur in them, and in some cases the search algorithm may need access to a wide-coverage parser to achieve satisfactory multi-word matching. Another solution is to ask dictionary editors to manually parse and index such entries, effectively telling the system that *lunchtime* can be broken down into *lunch* and *time*. Such valid decompositions would be placed on an index and consulted during search.

A different set of challenges exists for the minority of users who do search for multi-word expressions in full. Often, the multi-word expression does not exist in the dictionary, but the individual words do and the user may appreciate to see those. Again, it may be useful for the search algorithm to try to parse and lemmatize the search string, effectively to break the search string into smaller strings, and search for those individually. The same technique can be used to satisfy users who search for complete sentences or phrase fragments. Even though such items should not be searched for in a dictionary by traditional conventions, the users will probably welcome if the dictionary offers at least partial matches for individual words.

In some cases, splitting the user's multi-word input into individual words and searching for all possible combinations will yield useful results. If a user searches for the phrase *disability and unemployment benefits* (which she has perhaps copied and pasted from a document) and no such phrase exists in the dictionary, the search algorithm may try to search for all possible combinations of the individual words (perhaps excluding stop words like *and*) and find shorter phrases such as *disability benefit* and *unemployment benefit*. Nonsensical guesses, such as *benefit disability*, will fail as normal and the user does not need to see them. Note, however, that this

technique has the potential to return irrelevant results—such as when a search for *travel time* returns *time travel*. Still, a potentially irrelevant result is arguably better than no result at all.

Misspelling detection

Misspellings occur very frequently in online searches generally, on search engines as well as in dictionaries, and users are known to appreciate when the search algorithm offers corrections and suggestions, similar to the spellchecking facility in a word processor. One way to detect spelling mistakes in dictionary searches is to make use of a list of known common misspellings. Such lists are available for many languages and can sometimes be downloaded from the Internet for free.

When a search returns no results and the search term is not a common misspelling, there are techniques that can be used to try and find similar words in the dictionary. What constitutes a similar word is a question without an exact answer. Spellcheckers typically locate similar words by calculating the Levenshtein distance between the word the user has typed and words in the lexicon. A Levenshtein distance (also called edit distance) between two words is the smallest number of edits that need to be made in order to transform one word into the other. For example, the distance between *sitten* and *sitting* is 2: first change *e* to *i*, then add *g*. The lower this number, the more similar the two words are to each other and the more is one likely to be a misspelling of the other.

A technique such as this can be used in dictionary searches to compare the user's input to words in the dictionary and to suggest matches on the bases of similarity. It must be noted, however, that the Levenshtein distance technique and other fuzzy search algorithms may not scale well to large lexicons. Calculating the Levenshtein distance between a user's input and each word in the dictionary can be prohibitively slow in large dictionaries. Some compromises will probably have to be made when implementing such a feature, perhaps involving some form of pre-processing and indexing of the dictionary data.

Text normalization

Issues of punctuation and spacing sometimes get in the way of finding exact matches for a user's input. Online dictionary users frequently search for strings of text that include different types of hyphens and dashes, different types of quotation marks, trailing spaces, multiple spaces between words, and so on. A smart search algorithm will detect such elements and will normalize the input before the actual search.

Text normalization is the process of removing variation and reducing the text to a single format. There are almost twenty different characters representing different types of dashes and hyphens in the Unicode standard, and more than ten different quotation marks (Korpela 2006: 417-421). Text normalization removes all these and replaces them with just a single type of dash and a single type of quotation mark, as well as reducing all instances of whitespace (sequences of spaces and tab characters) into a single space. As a principle, the text that dictionary editors input and the text that users search for should be normalized in the same way to ensure that a match is possible.

Language selection

While preparing to launch the *focal.ie* terminology database, we noticed that our beta users frequently forgot to select the correct source language while searching. Users were searching for Irish words but leaving the language selection in its default position of English, and so on. Fortunately, it is possible to eliminate language selection completely, and to search all languages simultaneously. In most cases the search will return results in one language only, and so it is redundant to ask the user. In the small number of cases when a search returns results in more than one language (for example English *bean*, Irish *bean* "woman"), the *focal.ie* website groups them by language and allows the user to switch between them.

This strategy may not be applicable to all language combinations, especially not to pairs of closely related languages with a lot of shared words, such as Czech and Slovak, where searches would return results in both languages too often. In such cases, it may be a good idea to search in other languages only if a search in the user-selected language fails.

Conclusions

This paper has provided a small selection of techniques which designers of electronic dictionaries may use to make the search experience more user-friendly. Combining all these techniques into a single, smart search algorithm is not trivial. A user's search request may contain a combination of several complicating factors, for example it may be both misspelled and inflected. The search process can be viewed as a sequence of smaller processes, each feeding into the other: first it normalizes the textual input (removing double spaces and so on), then breaks it into individual words, then detects potential misspellings, and so on. The result is that the whole process can be quite complicated and individual searches can last rather long. In reality, users expect their dictionary searches to be not only smart but also fast. It is important to make good use of database indexing techniques, to understand the performance implications of each of the steps involved in the search process, and to be willing to leave features out if their performance cost is too high.

Last but not least, search is not the only way to access an electronic dictionary. Some users may not be looking for anything in particular, and providing an alphabetical browsing feature similar to a printed dictionary may satisfy such users better. On the other hand, some users will become "power users" and will need an advanced search facility for searching with wildcards, searching by part of speech, limiting searches to particular domains (such as: search only in mathematical terminology) and other features that a casual user would not be interested in.

In summary, the main mission of this paper has been to illustrate that the interaction of humans with dictionaries is far from straightforward, even more so when electronic media are concerned. Careful analysis of user behaviour and careful design of the search algorithm are necessary if a high level of user satisfaction is to be achieved

References

Bergenholtz, H.; Johnsen, M. (2005). "Log Files as a Tool for Improving Internet Dictionaries". *Hermes, Journal of Linguistics* 34. 117-141.

de Schryver, G.-M.; Joffe, D. (2004). "On How Electronic Dictionaries are Really Used". In *Proceedings of the Eleventh EURALEX International Congress*. Lorient: Université de Bretagne Sud. 187-196.

de Schryver, G.-M.; Joffe, D.; Joffe, P.; Hillewaert, S. (2006). "Do Dictionary Users Really Look Up Frequent Words?—On the Overestimation of the Value of Corpus-based Lexicography". *Lexikos* 16. 67-83.

focal.ie Irish National Terminology Database [online]. *http://www.focal.ie/* [Access date: 15 March 2008].

Korpela, J. K. (2006). *Unicode Explained*. Sebastopol: O'Reilly

"Levenshtein distance". In Black, P. E. (ed.). *Dictionary of Algorithms and Data Structures* [online]. U.S. National Institute of Standards and Technology. *http://www.nist.gov/dads/HTML/ Levenshtein.html* [Access date:15 March 2008].

SDL Trados [online]. *http://www.translationzone.com/en/products/sdltrados2007/* [Access date: 15 March 2008].

Someya, Y. (1998). *e_lemma.txt* [online]. *http://www.lexically.net/downloads/e_lemma.zip* [Access date: 15 March 2008].