Longest–commonest Match

Adam Kilgarriff[1], **Vít Baisa**[1,2], **Pavel Rychlý**[1,2], **Miloš Jakubíček**[1,2]

[1]Lexical Computing Ltd., Brighton, United Kingdom
[2]Natural Language Processing Centre, Masaryk University, Faculty of Informatics, Brno, Czech Republic
{vit.baisa,pavel.rychly,milos.jakubicek}@sketchengine.co.uk

Abstract

Finding two-word collocations is a well-studied task within natural language processing. The result of this task for a given headword is usually a list of collocations sorted by a salience score. In corpus manager Sketch Engine, these pairs are extracted from data using a word sketch grammar relation rules and log-dice statistics resulting in a sorted list of triples <headword, grammar-relation, collocate>. The longest–commonest match is a straightforward extension of these two-word collocations into multiword expressions. The resulting expressions are also very useful for representing the most common realisation of the collocational pair and to facilitate the interpretation of the raw triplet because sometimes, for such a triple, it is not clear from what texts it comes. We present here an algorithm behind the longest–commonest match together with a simple evaluation. The longest–commonest match is already implemented in Sketch Engine.

Keywords: multiword expresion; collocation; word sketch; Sketch Engine

1. Introduction

The prospects for automatically identifying two-word multiwords[1] in corpora have been explored in depth, and there are now well-established methods in widespread use[2]. But many multiwords are of more than two words and research into methods for finding items of three and more words has been less successful (Kilgarriff et al., 2012). Here we introduce a method for finding salient multiword expressions based on collocations—word sketches (Kilgarriff et al., 2004). The resulting multiword expressions are also very useful when it is not clear from what texts a collocation pair comes, e.g. <flame$_n$, object-of, put$_v$>, <love$_v$, object, neighbor$_n$>, etc. The longest–commonest match is therefore also a representative expression for collocational pairs. In the next section we describe the longest–commonest match, the algorithm and a rationale behind it. Then we present a small scale evaluation of the algorithm which was done on an English corpus and a set of collocation pairs. In the fourth section we discuss some issues with finding the longest–commonest matches and in the fifth section we propose some possible improvements of the algorithm.

[1] We use 'multiwords' as a cover-all term to include collocations, colligations, idioms, set phrases etc.
[2] (Church and Hanks, 1990; Pearce, 2002) and others.

2. Longest–commonest match

In this section we describe an algorithm for identifying candidate multiwords of more than two words called the *longest–commonest match* (LC match; in the previous works we have used the terms commonest match or commonest string). It starts from a two-word collocation, as identified using well-established techniques (dependency-parsing, followed by finding high-salience pairs of lexical arguments to a dependency relation) (Kilgarriff et al., 2004). We then explore whether a sufficient proportion of all collocation examples is accounted for by a particular string—the longest–commonest match.

The two-word collocations from which we start are triples: <lemma1, grammar-relation, lemma2> for example <drink$_v$, object, tea$_n$>. The lexical arguments are lemmas, not word forms, and are associated with word class, here n for noun, v for verb. The corpus instances that will have contributed to giving a high score include "They were drinking tea." and "The tea had been drunk half an hour earlier." The first argument may be to the right, or to the left, of the second. It depends on a particular grammar relation which is described in word sketch grammar rules.

If a particular string (consisting of word forms, not lemmas) accounts for a high proportion of the corpus instances, it becomes a candidate multiword-of-more-than-two-words. We want the string to be common and we want it to be long. Hence the name. We find the longest–commonest match as follows:

Input: two lemmas forming a collocation pair, and N hits for the pair in a given corpus; parameters: proportion p (1/4), minimum frequency minf (5) and minimum number of hits minhits (10).

Initialization: initialize the match as, for each hit, the string that starts with the beginning of the first of the two lemmas and ends with the end of the second. If the initial number of hits is less than minhits then return empty string, i.e. there is no LC match for a given lemmas.

For each hit, gather the contexts comprising the match, the preceding three tokens (the left context) and the following three tokens (the right context).

1. Count the instances of each unique string. Do any of them occur more than p × N?
2. If no, return empty string.
3. If yes
   (a) Call the most frequent string **LC match**
   (b) Look at the first tokens in its right and left contexts (max 3 positions), if we cannot expand farther, return **LC match**
   (c) Do any of the expanded strings occur more than p × N times?
   (d) If no, return the current **LC match**.

(e) If yes:
   i. Assign the most frequent expanded string to **LC match**.
   ii. Go to 3.b.

If there are no strings meeting the thresholds, there is no **LC match** (it is empty). Since LC match is extracted from corpus examples it consists from word forms not from lemmas.

An earlier version of this work was presented at EURALEX 2012 (Kilgarriff et al., 2012). We present it here again because it was only covered very briefly, and in the meantime we have developed a version of the algorithm that works very fast even for multi-billion word corpora, and is fully integrated into our corpus query system Sketch Engine, see Figure 1. It is a word sketch table for the headword put (verb). The first column contains collocates, the second column contains grammar relations, the third and fourth columns contain frequency and salience score and the last column contains LC matches.

put *(verb)* **British National Corpus + Commonest Match freq = 67,367**

down	*part_trans*	2,534	9.94	put down
forward	*modifier*	1,720	11.56	put forward
up	*part_intrans*	1,176	7.83	to put up with
just	*modifier*	832	8.77	just put
in	*part_intrans*	796	8.88	put in
pressure	*object*	519	8.20	put pressure on
then	*modifier*	438	8.05	and then put
head	*object*	387	6.92	put his head
thing	*object*	382	6.57	put things
end	*object*	327	6.80	to put an end to
off	*part_intrans*	320	7.76	be put off
place	*pp_in-p*	241	6.61	put in place
right	*np_adj_comp*	217	7.96	put it right
phone	*object*	213	7.47	put the phone down
hand	*part_out-a_obj*	167	5.92	put out a hand
lot	*object*	147	6.30	put a lot of
use	*pp_to-p*	145	6.33	put to good use
kettle	*object*	142	7.15	put the kettle on

Figure 1: Integration of the longest–commonest match in Sketch Engine

Comment on Figure 1 In some cases, the LC match is simply a bigram of adjoint collocates: put down, put in, etc. Sometimes the two collocates are separated by a token thus producing a trigram: put his head, put in place. This may occur when a headword is a phrasal verb with an object (put in place). In the example there are also 4-grams, e.g. put the phone down. It again captures phrasal verb and this time the object comes together with the determiner. It

results directly from the LC match algorithm that these "examples" are the most frequent realisations of the collocation pairs.

Implementation We have implemented the longest–commonest match in Python language and integrated it into Bonito/manatee corpus manager (Rychlý, 2007). The script is run only once at the time of corpus compilation and the resulting longest–commonest matches for each collocation pair are saved into word sketch data index files. That is why it is immediately available when showing word sketch data (as in Figure 1). The downside is that we need to set the parameters p, minhits, minf before the corpus compilation process. To compute and show LC matches with different settings, we need to process the whole corpus again and store the found matches in separate index files.

3. Evaluation

To overcome the issue of pre-setting the parameters, we designed a simple evaluation of various settings to find out what is the best combination of the parameters. We were most interested in the proportion, parameter (p). Other parameters (minhits, minf) are good for controlling coverage of the output and for limiting the time needed for computing LC matches for all collocation pairs in a corpus. The width of the token context (3 to the left and to the right) is not adjustable, but it could be another parameter available for tuning. Nevertheless we have decided to compare results for various settings of the only parameter, p.

Since this is not a classification task, it is not possible to measure the standard metrics precision and coverage. We have let two annotators decide for a set of 500 LC matches (extracted from SkELL corpus (Baisa and Suchomel, 2014)) which are good (helpful, well-formed, informative) and which are wrong but the definition of what is good and wrong was hard to agree on. Instead, we extracted LC matches for various settings of the proportion parameter p and let two annotators compare the resulting LC matches. The features were the same as before. Is one LC match a better example for a collocation pair? Is one LC match more informative, explanatory and understandable than other matches? The difference was that annotators were comparing three LC matches instead of telling yes or no for particular LC matches. The agreement was much better for this variant. For the results, see Table 1.

Two annotators (A1, A2) were provided with 102 randomly selected collocation pairs (examples below) together with three LC matches where the proportions (parameter p in the algorithm) were 0.5, 0.25 and 0.16 (columns LC match 1, 2 and 3, respectively). Their task was to select the most helpful LC match for understanding the collocation pair (first three columns). When two columns were the same, both column numbers were used in the annotation (last two columns labelled with annotator's indication). The most frequently favoured LC match (61%) was the least restrictive ($p = 0.16$) which means that in general, the length was preferred against the commonness of the strings. LC match 2 has been selected in 58% of

Headword	Relation	Collocate	LC match 1	LC match 2	LC match 3	A1	A2
love-v	modifier	personally-a	I personally love	. I personally love	. I personally love	1	1
calorie-n	object-of	need-v			calories needed	3	3
flame-n	object-of	put-v		put the flames out	put the flames out	23	23
vision-n	modifier	limited-j	limited vision	limited vision	limited vision .	12	12
meeting-n	modifier	joint-j	joint meeting	a joint meeting	a joint meeting of the	2	3
classroom-n	modifier	virtual-j	virtual classroom	virtual classroom	a virtual classroom	3	12
unofficial-j	modifies	symbol-n	unofficial symbol of	an unofficial symbol of	an unofficial symbol of	23	23
worthwhile-j	adj-comp-of	seem-v		seems worthwhile	seems worthwhile to	3	3
climb-v	modifier	gradually-a		gradually climbing	gradually climbing	23	23
delicate-j	modifies	matter-n	delicate matter	a delicate matter	a delicate matter	23	23

Table 1: Example of lines from evaluation data together with annotators' choices.

cases and LC match 1 (the most restrictive `p`) in 33% of cases. Mind that it was not a simple classification but rather the assignment of (multiple) labels to the LC matches (columns). That is why the percentages do not sum up to 100%. There was 67% agreement between the two annotators.

Evaluation data We have used a random sample from the dataset used in (Kilgarriff et al., 2014). The dataset[3] contains only verbs, nouns and adjectives as headwords in the English language. Here we include some examples of collocation pairs from the gold standard dataset (headword, collocate): (average_j, age_n), (black_j, hole_n), (circuit_n, short_j), (delicate_j, ecosystem_n), (empty_j, bin_n), (free_j, lunch_n), (global_j, crisis_n), (harp_n, player_n), (inject_v, vaccine_n), (kid_v, entirely_a), (love_v, genuinely_a), (metal_n, galvanized_j), (operational_j, remain_v), (past_j, participle_n), (root_v, firmly_a), (slow_v, abruptly_a), (tempting_j, extremely_a), (unofficial_j, biography_n), (virulent_j, campaing_n), (weed_n, grow_v), (worthwhile_j, highly_j).

4. Discussion

The evaluation helped us to discover some issues which we need to address. The most obvious is the punctuation being part of LC matches which was never preferred by annotators. It would be straightforward to strip it from the LC matches, nevertheless we are not sure if this is desirable. Sometimes it might be helpful to know that some phrases contain a comma or a full stop. It might help users understand that a certain phrase is used usually at the end of sentence (or at the beginning as the first example from Table 1 indicates) or that it is separated from the rest of the sentence by a comma.

Since the algorithm is language-independent (once we have a list of collocation pairs), adding a language-dependent list of punctuation to be removed from LC matches would spoil this desired feature. But a simple approach usable for most European languages would be simply to strip all commas, semicolons, full stops, exclamation and question marks. The punctuation

[3] Available for download: `http://www.sketchengine.co.uk/documentation/wiki/CorpEval`

would be removed only from the beginning and the end of a LC match as a punctuation mark within an LC match will have an obvious interpretation.

It is also clear that any match is preferred against an empty LC match. As for finding multiword expressions, empty matches decrease coverage which is not a big issue; but regarding the second goal of a LC match it surely decreases understanding of the original collocation pair. In other words, it is always helpful to have at least the collocation pair in the most common order (see examples in Table 1: limited vision, joint meeting, etc.) than to rely only on the original collocation pair. Thus it is reasonable to use a rather less restrictive parameter p.

The original combination of parameters proved to be solid. We found that using a somewhat less restrictive parameter p yields slightly better results but the difference is too small (3%) for us to change the default settings currently used in Sketch Engine.

5. Further work

Based on the evaluation and on a brief error analysis of the algorithm, we want to explore a few possible improvements of the algorithm in the future.

First, in some cases, LC matches were skewed by many occurrences of a string within one specific document. It could be treated by filtering input concordances to contain one (e.g. the first one) hit per document. This filter is already implemented in Sketch Engine.

In general, the algorithm suffers when duplicate documents are present in a corpus. This is addressed by de-duplication phase when building such corpus and has been treated in (Pomikálek, 2011). Sketch Engine uses procedures described in the PhD thesis.

Second, the current algorithm works with parameters which are fixed for all concordances / collocation pairs. It is to be evaluated whether making the parameters relative to concordance size (N input hits) would help.

Another improvement to the algorithm efficiency would be sampling of input concordances. The time complexity of the algorithm is roughly linear to the length of the input (concordance with N lines). For very large concordances (concordance for collocation pair take$_v$, place$_n$ has almost 1 million hits in corpus enTenTen12) it would be reasonable to use a random sample of such concordances. The question is whether the sample should have a fixed size or if the size should be (again) relative to the size of the original concordance. Despite the resulting LC matches being thought to be the same it is necessary to try and evaluate it. The sampling is also already available in Sketch Engine.

It was not mentioned earlier but the algorithm does not depend on collocation pairs. It is simply applicable for any concordance, meaning that for any search in a corpus, we can

compute (on-the-fly) the longest–commonest match or the longest–commonest KWIC as a generalized and expanded representation of the original corpus search query. It could be a handy feature to provide such generalized KWIC for all searches in Sketch Engine but again, we would need to evaluate its contribution based probably on user feedback.

6. Conclusion

We believe that the LC match will improve understanding of sometimes cryptic collocation pairs (triples) as available in Sketch Engine. The resulting strings are also salient multiword expressions despite the fact that it is not straightforward to properly evaluate the quality of these multiwords.

7. Acknowledgement

This paper was published posthumously for Adam Kilgarriff died on Saturday, May 16th, 2015. He has been working on this paper even in his later days while undergoing a palliative chemotherapy. We dedicate this paper to him, as the originator of the longest–commonest match.

Adam Kilgarriff (12 February 1960 – 16 May 2015)

This work has been partly supported by the Ministry of Education of CR within the LINDAT-Clarin project LM2010013 and by the Grant Agency of CR within the project 15-13277S. The research leading to these results has received funding from the Norwegian Financial Mechanism 2009–2014 and the Ministry of Education, Youth and Sports under Project Contract no. MSMT-28477/2014 within the HaBiT Project 7F14047.

8. References

Kilgarriff, A., Rychlý, P., Kovář, V., & Baisa, V. (2012). Finding multiwords of more than two words. In Fjeld, R. V. & Torjusen, J. M., editors, *Proceedings of the 15th EURALEX International Congress*, Oslo, Norway. Department of Linguistics and Scandinavian Studies, University of Oslo, pp. 693–700.

Baisa, V. & Suchomel, V. (2014). SkELL: Web interface for english language learning. In *Eighth Workshop on Recent Advances in Slavonic Natural Language Processing*, pp. 63–70.

Church, K. W. & Hanks, P. (1990). Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1), pp. 22–29.

Kilgarriff, A., Rychlý, P., Jakubíček, M., Kovář, V., Baisa, V., & Kocincová, L. (2014). Extrinsic corpus evaluation with a collocation dictionary task. In N. C. C., Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., & Piperidis, S., editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA), pp. 454–552.

Kilgarriff, A., Rychlý, P., Smrž, P., & Tugwell, D. (2004). The Sketch Engine. In Williams, G. & Vessier, S., editors, *Proceedings of the 11th EURALEX International Congress*, Lorient, France. Université de Bretagne-Sud, Faculté des lettres et des sciences humaines, pp. 105–115.

Pearce, D. (2002). A comparative evaluation of collocation extraction techniques. In *Third International Conference on Language Resources and Evaluation, LREC'02*, pp. 1530–1536.

Pomikálek, J. (2011). Removing boilerplate and duplicate content from web corpora. *PhD en informatique, Masarykova univerzita, Fakulta informatiky.*

Rychlý, P. (2007). Manatee/bonito-a modular corpus manager. In *1st Workshop on Recent Advances in Slavonic Natural Language Processing*, pp. 65–70.