Coreferences in REL

Flavio Hafner

Netherlands eScience Center

January 20, 2023

Contents

1 Introduction

REL is a software for entity linking. Entity linking consists of three steps: mention detection, candidate selection, and entity linking. Before entity linking, the current version of REL looks for coreferences in the detected mentions. Consider the example text: "This sentence is about Jimi Hendrix. The music of Hendrix is popular." Here, the second mention "Hendrix" refers to the entity "Jimi Hendrix" in the preceding sentence. Now, for each mention m_i, REL checks whether there is another mention m_j that contains mention m_i as a separate word. Denote the set of mentions m_j that m_i refers

to by R_i. If R_i is a singleton, REL replaces the original candidate entities of m_i with the candidate entities of m_j. If R_i contains multiple elements, the union of the unique candidate entities for all mentions $m_j \in R_i$ are assigned as candidate entities for m_i. Moreover, for any candidate entity that appears in more than one mention $m_j \in R_i$, their $p(e|m)$ scores are aggregated for the mention m_i.

Coreference search can make entity disambiguation more accurate if the original candidates for mention m_i are more noisy than the selected candidates for m_j. But coreference search also slows down REL for large data sets because it has quadratic time complexity.

I extend REL so that the user can choose from three options on how to deal with coreferences. The first option is "all" which corresponds to the current default of comparing all mentions with all other mentions. The second option is "off" that does not look for coreferences at all. It should be the most efficient. The third option is "lsh" that first applies locality-sensitive hashing (LSH) to all mentions, and then only searches for coreferences among mentions that are clustered together by LSH. If LSH is efficiently implemented, it could strike a balance between efficiency and effectiveness for datasets with many mentions.

To assess the performance (efficiency and effectiveness) of the options, I run REL with the three options on three data sets: on the AIDA data to check effectiveness, and on synthetic AIDA data—stacking the mentions from AIDA multiple times—to check the time complexity with larger datasets. The purpose of the last run with the msmarco data is to check time complexity on real-world data. I try to check the effectiveness on these data by checking the fraction of mentions that are different between the two approaches.

In what follows, I define coreferring mentions as mentions like the mention “Hendrix” above, and non-coreferring mentions as all other mentions.

The calculations are run on a Dell XPS 13 laptop, using Flair and Wikipedia 2019 for mention detection. I set the limit number of documents from AIDA to 500, which processes all documents. For the moment, I hard-code the settings for LSH in REL. The precision-recall tradeoff in LSH means that a higher recall for LSH results in a longer running time of coreference search because, for each mention, `with_coref` checks more mentions. The current settings are set to maximize the F-score for LSH on the coreferring mentions in AIDA.

coref option	Mention detection				Entity linking		
	Precision	Recall	F1	Time	Precision	Recall	F1
"all"	97.9%	62.9%	76.6%	3.12 sec.	62.1%	39.9%	48.6%
"lsh"	97.9%	62.9%	76.6%	3.69 sec.	62.2%	39.9%	48.6%
"off"	97.9%	62.9%	76.6%	3.21 sec.	62.1%	39.9%	48.6%

Table 1: Performance on AIDA data—all mentions. Setup: processing all documents in AIDA data set with wikipedia 2019.

2 Results

2.1 Performance on AIDA data

Table 1 shows precision, recall and F1 scores on the AIDA data sets (all documents) with the three options. The three options do barely differ. The timing results show that "lsh" is the slowest option, and this is because setting up LSH has some overhead that dominates the value of reduced coreference search after clustering.

To compare to Erik's results (table 1 in his document), I ran REL also only with 50 documents. The effectiveness results are very similar to his results. The efficiency results, however, differ—in Erik's case, ED takes more than twice as long than in my case. I am not sure why this is.

Because only a minority of the mentions in the AIDA data are coreferences, I now check the performance for coreferring mentions only. To do this, I classify the gold mentions by whether they are coreferences or not using the same method as in REL, and then compare precision, recall and F1 scores only on these mentions. The results are in table 2.

First, using option "all" , we can compare the results from the restricted sample with the results from the full sample of mentions. We see that recall for mention detection for coreferring mentions is much worse than for non-coreferring mentions—less than 40 percent of coreferring gold mentions are detected, while 60 percent of all gold mentions are detected. This could be (1) because coreferences are harder to detect, or (2) because some detected mentions are not labelled as coreferences because the mention they are referring to is not detected. I think case (1) is more likely. The tables also indicate that in the ED step, REL has a higher precision but a lower recall for coreferring mentions than non-coreferring mentions.

Second, comparing effectiveness of ED for the three options, we see that "lsh" performs better than "off" but worse than "all".

coref option	Mention detection				Entity linking		
	Precision	Recall	F1	Time	Precision	Recall	F1
“all”	90.7%	36.7%	52.3%	—	78.8%	31.9%	45.4%
“lsh”	90.7%	36.8%	52.4%	—	78.1%	31.7%	45.1%
“off”	90.7%	36.9%	52.5%	—	76.8%	31.3%	44.4%

Table 2: Performance on AIDA data—coreferring mentions. Setup: processing all documents in AIDA data set with wikipedia 2019. The timing is not reported because the results are taken from the same run as the results in table 1.

2.2 Time complexity

To assess time complexity of the different options, I create a synthetic AIDA data set: For each document, I repeat the detected mentions multiple times to increase the size of the input, and then compare the running time for entity linking as a function of the size of the input. Figure 1 shows the results.

The panel on the left compares option "off" with option "all". We see that the quadratic time complexity of "all" only starts to impact the overall running time when the number of mentions is higher than around 1500 mentions. This is because the `__predict` function in REL has a relatively large overhead that dominates the running time of coreference search when the number of mentions is small (further evidence is available if necessary). But for an input of 10000 mentions, option “all” takes about twice as long as option “off”.

The panel on right compares option “all” with option “lsh” and shows that the time complexity for option “lsh” is similar to “all”, except for large documents where it “lsh” is perhaps better than “all”. To confirm this, I would need to run another test with a larger synthetic data set.

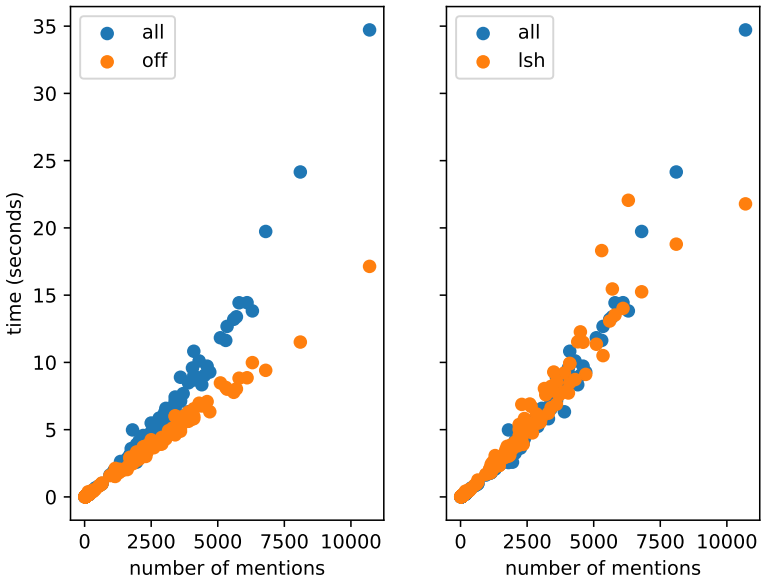

Figure 1: Time complexity on synthetic AIDA data. Left: comparing "all" and "off". Right: comparing "all" and "lsh". The unit of observation is one document stacked n times, $n \in \{5, 50, 100\}$.

I also profile the execution time of each function in entity disambiguation for the synthetic data. Figure 2 shows the the time taken by the function `with_coref`, which is the main function for coreference resolution. The figure suggests a much better time complexity of "lsh" compared to "all", starting at a document size with around 6000 mentions.

Comparing figures 1 and 2 for the largest document, we see that switching from "all" to "lsh" shortens running time by about 15 seconds (figure 1), while figure 2 suggests that `with_coref` runs 25 seconds faster. It is unclear what explains this difference.

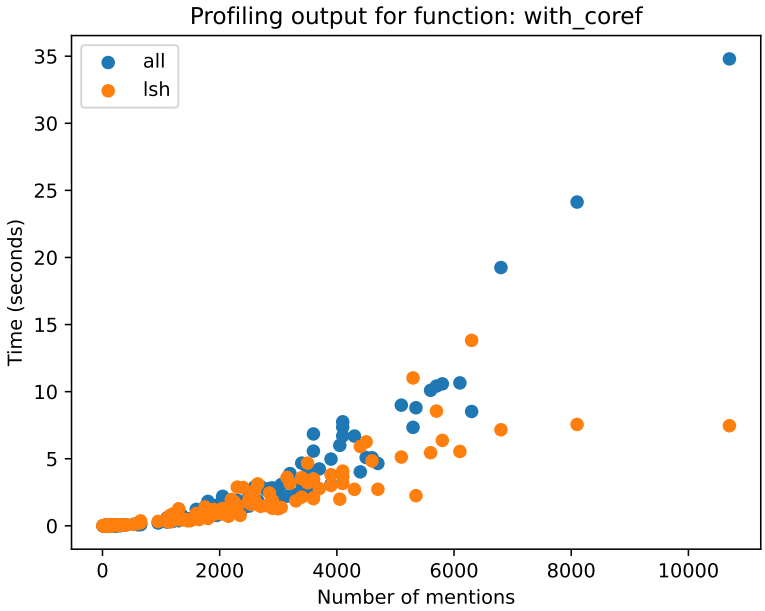

Figure 2: Profiling the function *with_coref* with synthetic AIDA data. See the notes for figure 1 for details on the sample.

2.3 Performance on msmarco data

2.3.1 Effectiveness

To get an idea of recall and precision despite not having a ground truth, I compare the linked entities for mentions with options "off" and "lsh" to the linked entities with option "all". During the run with "all" I also label mentions as whether they are coreferring or not. This allows me to focus on these mentions in the following analysis.

Approximating recall One can compare whether some mentions that are linked to an entity with "all" are not linked with the other options. The options "off" finds 0.36 percent fewer coreferring mentions than "all". "lsh" finds 0.10 percent fewer coreferring mentions than "all". This suggests that recall is very similar for all options, and that "lsh" is slightly better than "off".

Approximating precision Now I focus on the mentions that are linked. I start by comparing the fraction of mentions that have the same predicted entity by aggregating across all documents. Table 3 shows the results. Nearly all mentions that are not

	"off"			"lsh"		
coreferring mention	count	mean	std	count	mean	std
no	247754	0.99	0.056	247754	0.999	0.031
yes	8767	0.40	0.49	8767	0.61	0.48

Table 3: Fraction of mentions for options "off" and "lsh" with the same linked entity as in option "all".

coreferring have the same linked entity. But among coreferring mentions, when using "off" only 40 percent of mentions have the same linked entity. This implies a substantial variability in the precision of the linked entities for coreferring mentions. But the lack of ground truth prevents from drawing conclusions about whether "off" is better than "all". When using "lsh", 61 percent of mentions have the same linked entity.

Now I compare the overlap of predicted entities by document. Overlap is the fraction of mentions in a document that have the same predicted entity. Figure 3 plots the overlap separately for all mentions, coreferring mentions, and non-coreferring mentions. The x-axis is the total number of linked mentions in a document. All panels show the data for documents with at least five coreferring mentions.

The figure shows considerable variation in the overlap of the predicted entities overall, stemming from variation in the overlap for the coreferring mentions. The data do not allow to draw conclusions about whether there is a relation between the number of linked mentions in the document and the overlap in the predicted entities for coreferring mentions.

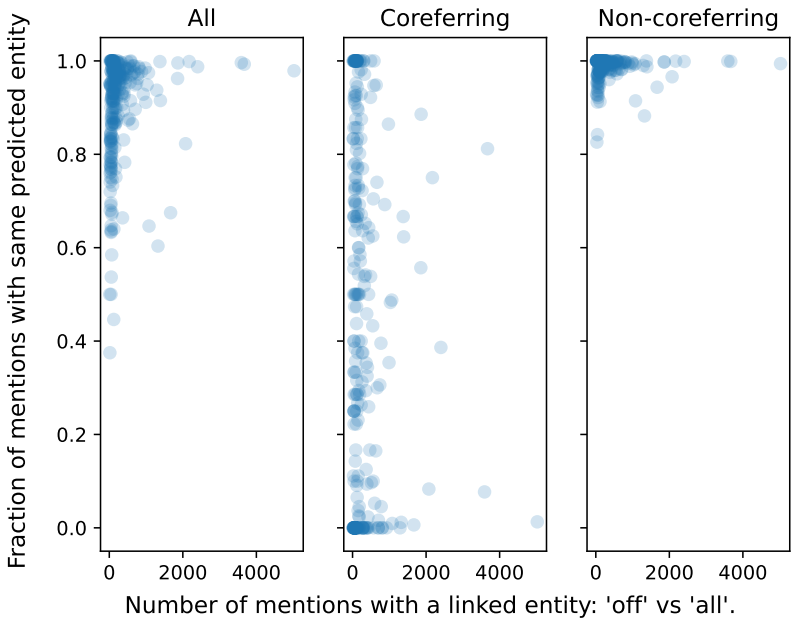

Figure 3: Overlap in predicted entities, by mention type. The unit of observation is the document.

Figure 4 shows the same data, comparing “lsh” with “all”. In line with the aggregate results from table 3, “lsh” has a higher overlap for coreferring mentions.

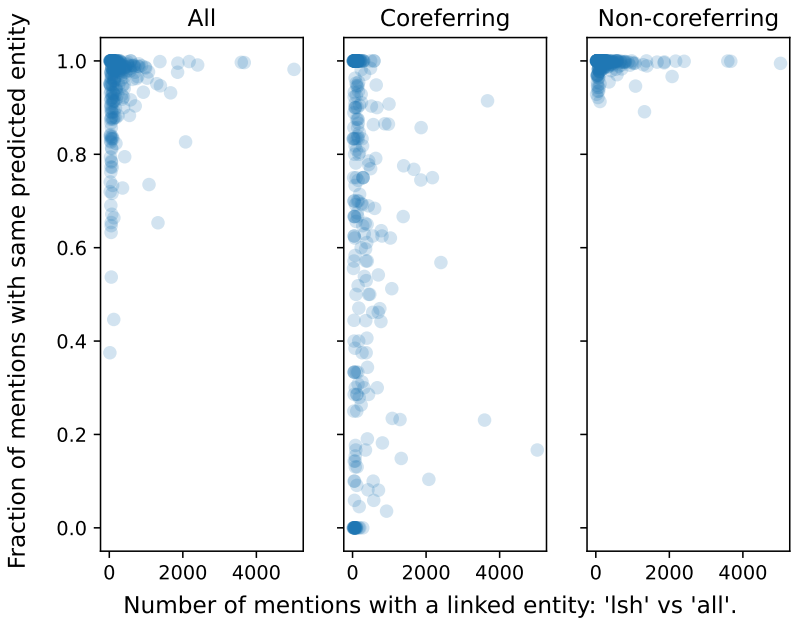

Figure 4: Overlap in predicted entities, by mention type. The unit of observation is the document.

2.3.2 Efficiency

I now compare the running time for the options. Figure 5 shows the running time of entity linking as a function of document size; it confirms the findings on the time complexity with the synthetic AIDA data: the running time for "off" and "all" is not very different for documents with a few thousand mentions. But for the largest document with 15k mentions, option "all" takes about 40 seconds while "off" takes 25 seconds. The figure shows that "lsh" has a similar time complexity as "all" except for the largest document, where "lsh" is faster.

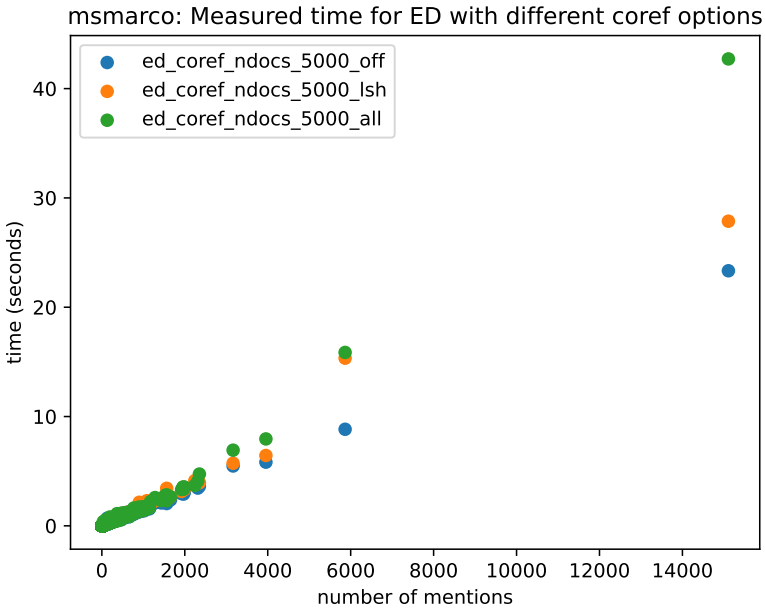

Figure 5: Time complexity on msmarco data for options "all" and "off". The unit of observation is the document.

3 Conclusion

3.1 Summary

1. For small input data (up to around 2000 mentions), the coreference search is inconsequential for running time. Effectiveness is highest when using option "all".

2. For large input data, efficiency is a problem when doing a full coreference search. Particularly for documents with many coreferring mentions the choice of the option could matter, depending on how strongly coreference search impacts effectiveness in the ED step.

3. The results for effectiveness of ED for coreferring mentions perhaps suggest that recall is similar for either "off" or "all". But the results for precision are not clear-cut. The results from AIDA show similar precision for either option, but the results from msmarco suggest a possibly large impact on precision between options "off" and "all". The option "lsh" is similarly effective as "all" but less efficient for the AIDA data; for the msmarco the data suggest it may speed up ED for documents

with 10k mentions or more, even though there is only one such document in the msmarco data set.

4. The time complexity for "lsh" is better than "all", but only for documents with at least 6000 mentions because of the overhead of how LSH is currently implemented. I am nearing the limit for optimizations with my from-scratch solution; external alternatives such as faiss (link) may be faster.

5. An important reason for low recall for coreferring mentions is the mention detection, not the disambiguation, but these mentions are a small part of the total number of mentions in the AIDA data. And some of this may be resolved with the update for MD from Erik (which I have not yet integrated in my version of REL).

3.2 Implications

1. The coreference search option "all" limits the scalability of REL for documents with 10k mentions or more.

2. If the current implementation of coreference search is integrated in the main branch of REL, one can consider an automatic switch to either of the types of coreference search when the number of mentions is large enough, and raise a respective notification to the user.

3. LSH may be useful for a use case with large documents with possibly many coreferring mentions (but I do not know what kind of documents these could be).

4. One could explore the following improvements:

    - Using an external dependency to reduce overhead
    - Automatically choose the parameter values for LSH, depending on the dataset. In particular, the band length can be chosen as $log(N)$, where N is the number of mentions. For LSH, this would give an approximate running time of $O(log(N))$.
    - Further select the candidates by the distance to the query point. Ie, select candidates that are sufficiently close, or fix a number of k candidates. This could limit the computing time of the coreference search by discarding very unlikely candidates. As I see it, it is not too much work to implement.