Fast Identification of Candidate Entities for Context Disambiguation

Ryan Oman

Introduction

The problem this system solves is as follows: Given an entity from a ReVerb [ReVerb, Turing Center] tuple, quickly come up with k Freebase [Freebase, Metaweb Technologies] entities that are likely candidate matches based purely on string matching that can then be processed by a context disambiguation system. For example, for the tuple "(Clinton, visited, New York)", top candidate entities such as Bill Clinton and Hillary Clinton should be returned. This system employs a variety of matching techniques such as exact string matching, exact cleaned string matching, full word substring matching, Wikipedia alias matching, acronym matching, and Lucene spell correction matching [Apache Lucene]. The main difference between this system and a similar system like SecondString [Cohen, CMU] is a large increase in speed. A system like SecondString deals only with approximate string matching, while this system attempts to find full word matches and only uses approximate matching if that fails. On a test set of 93 unique terms from real news articles, our best system is able to return the correct match in the top 5 matches 81% of the time at a speed of 150 ReVerb entities per second. This is around a 40% improvement over the naïve approach of using only exact string matches and ordering on prominence.

Naïve Approaches

A naïve approach to solving this problem would be using only exact string matches and ordering on matches; however, this misses many matches such as: "US" matching "United States", "Obama" matching "Barack Obama", and "Gadafy" matching "Muammar al-Gaddafi". When we evaluated this approach, we found that it achieved 42% accuracy as shown in Figure 3.

Another simple approach would be to just take substring matches, ordering on prominence. When we evaluated this approach coupled with exact string matching and prominence ordering, we found that it achieved 58% accuracy. Used alone, it provided only 20% accuracy. The reason it is so low is because pure substring matching provides many low quality matches that push the desired matches down past the top 5 matches returned. Also this method still misses matches such as "US" matching "United States" and "Gadafy" matching "Muammar al-Gaddafi".

These simple approaches are able to correctly match some simple types of matches, but they miss more complex types of matches. Table 1 enumerates all different types of matching that a system might encounter when mapping Strings to their corresponding Freebase entities.

		Example Case		
	Match Type	**ReVerb String**	**Freebase Entity**	**Handled?**
1	Exact match between arg1 and Freebase entity	Bill Clinton	Bill Clinton	Yes
2	Exact match between cleaned arg1 and cleaned Freebase entity	Titanic	Titanic (1997 film)	Yes
3	Arg1 is a substring of the Freebase entity	Feb	February	Some
4	Freebase entity is a substring of arg1	beautiful Montana	Montana	Some
5	String overlap between arg1 and Freebase entity	President Clinton	Bill Clinton	Yes
6	Arg1 is an abbreviation of a Freebase entity	G.E.	General Electric	Yes
7	Misspelling match between arg1 and Freebase entity	Gadafy	Muammar al-Gaddafi	Yes
8	Prominent Alias between arg1 and Freebase entity	America	United States	Yes
9	Obscure Alias between arg1 and Freebase entity	The Red Planet	Mars	No

Table 1: Outlines the cases handled by our system

Key Components in System

In addition to the components listed below, we also incorporated a scoring algorithm for ordering matches. The naïve approaches ordered only on prominence, but this approach doesn't take into account the fact that some matching techniques yield higher quality matches than others. For example, an exact string match is higher quality than a single word-overlap match. All of the matching techniques listed below use case insensitive matching.

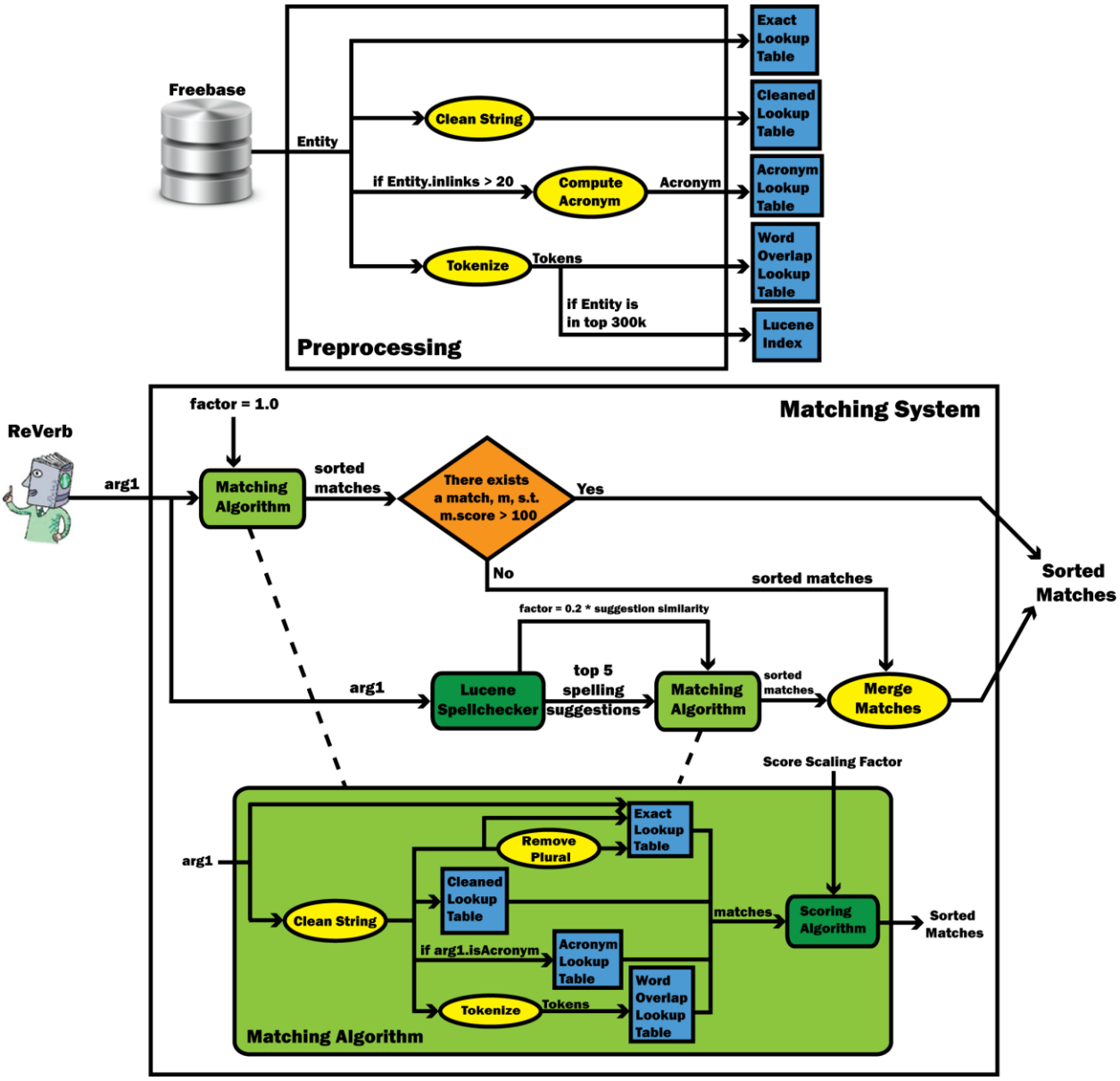

Figure 1: This chart outlines the architecture of our system. The matching system outputs only one set of sorted matches, depending on if Lucene is used or not.

Scoring Algorithm: The scoring algorithm assigns a weight to each type of matching, and the score is computed by adding all of the weights together. The weights are on a scale of 0 to 100 For example, if a Freebase entity has an exact string match, 100 points would be added to its overall score. Every match's score is initialized to be the normalized inlink count. Normalizing the prominence is necessary to prevent the prominence from affecting the score too much.

Attempts were made to auto-tune the weights for each match using hill climbing and linear regression, but time constraints prevented us from being able to fully take advantage of auto-tuning. Auto-tuning yielded many different combinations of weights that had the same accuracy on the dataset. These were similar to our hand-tuned weights with some adjustments that would help generalize on a dirtier dataset. Figure 2 shows the distribution of "reasonable" matches based on the scores on a dirty dataset. We hand labeled 465 matches as either a possible match or not. The reason that matches with scores below 40 have an abnormally high match rate, is because many of the ReVerb arg1s that returned those matches were ambiguous, meaning they could match many different Freebase entities, and one clear match wasn't evident. Other than the low scoring matches, matches with higher scores are generally more likely to "reasonable" matches than those with lower scores.

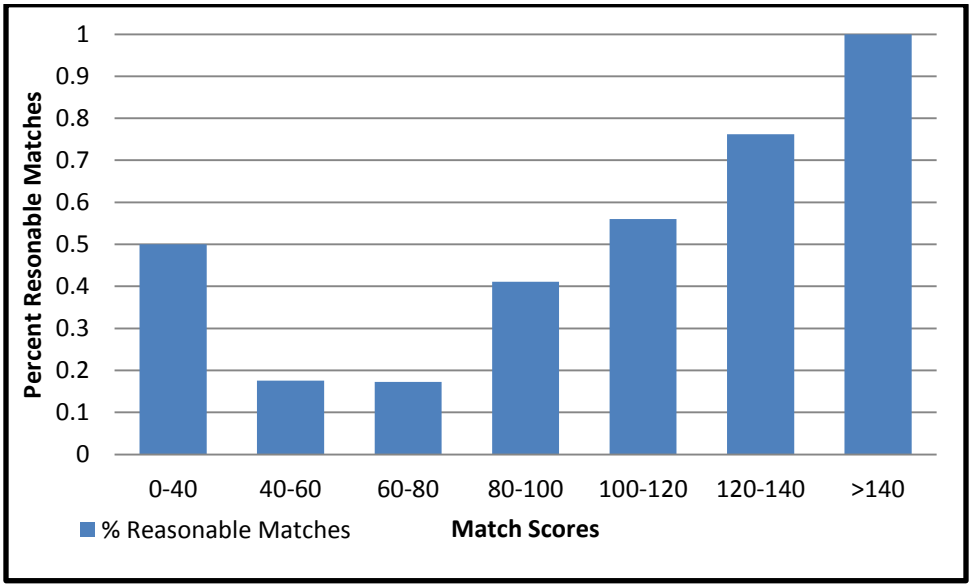

Figure 2: This figure shows the percentage of matches with scores between two thresholds that are "reasonable" matches (they were hand labeled). This data was collected from a larger ReVerb dataset, which was extracted from web text.

Total Score = NormalizedInlinks + ScalingFactor * (ExactMatchWeight + CleanedMatchWeight + WordOverlapMatchWeight + AbbreviationMatchWeight + WikiAliasMatchWeight)

NormalizedInlinks = (ln(inlinks) / ln(max_inlinks)) * 100
ExactMatchWeight = ExactMatchExists ? 100 : 0
CleanedMatchWeight = CleanedMatchExists && !WordOverlapMatchExists && !ExactMatchExists && AbbreviationMatchExists ? 100 : 0
WordOverlapMatchWeight = WordOverlapMatchExists && !ExactMatchExists && !AbbreviationMatchExists ? WordOverlapCount * 20 : 0
AbbreviationMatchWeight = AbbreviationMatchExists && !ExactMatchExists ? 80 : 0
WikiAliasMatchWeight = WikiAliasMatchExists ? 90 : 0

1. *Exact String Matching*: Uses hash table stored in memory for fast lookup.
    a. Handles match type #1 in Table 1. Generally provides a few very high quality matches such as “Bill Clinton” matching “Bill Clinton”
    b. *Scoring*: If a Freebase entity is an exact string match, 100 points are added to its score. An exact match is the best type of match, which is why it is assigned the highest weight possible.
2. *Exact Cleaned String Matching*: Freebase entities are cleaned, then loaded into a hash table for fast lookup of a cleaned ReVerb string. Cleaning an entity involves removing commas, periods, quotes, html special characters, and removing parts of string in parentheses (“Titanic (1997 film)” would become “Titanic”)
    a. Handles match type #2 in Table 1, which also includes cases like “Great Britain&rsquo” matching “Great Britain”.
    b. *Scoring*: If a Freebase entity is an exact cleaned string match, 100 points are added to its score. This was chosen as the same as an exact string match because cleaned strings are generally about the same quality as exact string matches. A match is only considered to be a cleaned string match if it is not an exact string match.
3. *Wikipedia Alias Matching*: Uses a list generated from Wikipedia of different articles that refer to the same entity in Wikipedia. For example, the article with the title “America” refers to the article with the title “United States”. Aliases are cleaned as in (2) and loaded in a hash table for fast lookup of a cleaned ReVerb string.
    a. Handles match type #8 in Table 1, which also includes cases like “Democrats” matching “Democratic Party (United States)”. Match type #7 in Table 1 is not handled because it is likely to happen that often and obscure aliases aren’t included in our Wikipedia alias file.
    b. *Scoring*: An exact alias match generally provides a very high quality match, but still not quite as good as an exact string match, which is why these matches get a weight of 90.
4. *Word-Overlap Matching*: Determines how many words overlap between two strings. This method is better than pure substring matching for two reasons. One is that substring matching is much slower than this technique. For this method, a hash table can be loaded with each word for fast lookup of a cleaned ReVerb string, while in substring matching; many string comparisons need to be made for each match. Another reason is that this method returns higher quality matches. Full word matches are generally better than partial word matches. Using only word-overlap matching achieves 57% accuracy as shown by Figure 3, which is a 37% improvement over substring matching.
    a. Handles match type #5 and some of #3 and #4 in Table 1, which also includes cases like “William Jefferson Clinton” matching “William Clinton”.
    b. *Scoring*: For each word that overlaps between the ReVerb string and the Freebase entity, 40 points is added to the score. Most of these match types only have one word overlapping, which generally gives just a 40 point boost. This score makes sure that even a word-overlap match that has high prominence will still most likely be lower than the other types of matching, which generally give higher quality results. A match is only considered to be a word-overlap match if it is not an exact string match or abbreviation match.
        i. In order to differentiate between other word-overlap matches, a word-overlap match is also multiplied by a factor of how rare of a word it is that is matched. Word frequencies were pulled from a 9 million tuple ReVerb extraction. A words factor is 1.0 if it is not in this extraction and the factor is (ln(max_freq) – ln(word_freq)) / ln(max_freq) if it is in the extraction. This helps cases like “Distinguished radiographers” matching to a Freebase entity with “radiographers” in it instead of one with “distinguished” in it.

   ii. Another matching heuristic used to differentiate between word-overlap matches is that if a word in the ReVerb arg1 matches an entire Freebase entity it is given a score of 60 instead of 40. This would be a case like arg1 of "beautiful Montana" matching the Freebase entity "Montana".
   iii. A stop word list from [WebConfs] is also used for word-overlap matching. This prevents matches like "The Brotherhood" matching "Harry Potter and the Half Blood Prince".
5. *Abbreviation Matching*: If a ReVerb string, or any word within a ReVerb String, looks like an acronym, then it is matched to the acronyms of all Freebase entities with at least 20 inlinks, where inlinks correspond to prominence [Grounder, 2009]. A string looks like an acronym if it is composed of all capital letters. It could also have periods in between the letters, but is not required to. The acronyms for Freebase entities with more than 20 inlinks are computed prior to matching and loaded into a hash table for fast lookup. Freebase entities with fewer than 20 inlinks generally do not have prominent acronyms associated with them. On our dataset, all of the acronyms' correct matches were found in the top 50,000 Freebase entities. The 20 inlink threshold includes around 300,000 of the top entities in Freebase.
   a. Handles match type #8 in Table 1, which also includes cases like "U.S." matching "United States and "CMU" matching "Carnegie Mellon University"
   b. *Scoring*: An abbreviation match has a weight of 80 points. This is because even though filtering out the less prominent Freebase entities for acronym matching removes many low quality matches, exact string matching and wiki alias matching still yields higher quality matches. A match is only considered to be an abbreviation match if it is not already an exact string match.
6. *Non-Plural Matching*: If a ReVerb string ends with an "s", it is stripped off and the stub is used to query against the exact string hash table, the substring hash table, and the abbreviation matching table (if its stub looks like an acronym). Plural entities are not handled by the Word-Overlap method, so a special case had to be made for them.
   a. Handles some of match type #4 in Table 1, which includes cases like "Egyptians" matching "Egyptian"
   b. *Scoring*: These matches receive the same score as they would if they were normal exact string or substring matches, since most words with a trailing "s" stripped off are non-plural forms or are nonsense words.
7. *Lucene Matching*: This type of matching gets spelling suggestions from a Lucene index based off of all of the words used in Freebase entities. Lucene performance is much slower than other types of matching as shown in Figure 3, which is why it is only used if there are no high scoring matches returned from the other types of matching. If there is not at least one entity with a score over 40 points, Lucene will get spelling suggestions and then all of the matching techniques listed above are performed on the top 5 matches. Any duplicate matches are ignored. Only the top 3 matches returned from Lucene are actually used. This is because Lucene returns a lot of matches that aren't necessarily high quality, and they can push out all of the word-overlap matches from the top 5. By returning only the top 3, good Lucene matches could still be taken account for, but so will word-overlap matches.
   a. Handles match type #7 in Table 1, which includes cases like "Gadafy" matching "Muammar al-Gaddafi" and "Democrats" matching "Democratic Party (United States)". This match type is not as common as the others in our news dataset, but it would be on a pure web extraction since there are more likely to be misspellings in random web text than in a news article.

b. *Scoring*: Lucene matches' weighted match scores (excluding the prominence score) are multiplied by two factors. The first is the similarity score that Lucene returns with each spelling suggestion, which is between 0 and 1. The second is a constant Lucene scoring factor of 0.2. Lucene provides many low quality matches, and these two factors, used in combination, pushes down less prominent entities below any word-overlap matches, which are generally better matches even than exact Lucene matches. Prominent entities still get pushed above other normal matches because the normalized prominence score is not affected by the Lucene weighting factor.

Evaluation

We used a test set of actual new articles created from queries to Yahoo Boss's news interface [Yahoo Boss]. We queried "Watson" and "Jeopardy", "Wael" and "Egypt", and "the". Each of the 3 queries returned several hundred news articles. We then ran ReVerb on these news articles, and extracted out the top 50 most frequent arg1 entities from each set. This is a small dataset, and good for experimenting with which ReVerb arguments might not be in Freebase, and what kind of matching algorithms are needed to correctly match with entities from news articles. This dataset provided 150 ReVerb arg1s, of which 122 were unique entities. Duplicate entities were filtered out by hand. Of those 122 unique ReVerb entities, 93 had identifiable entities in Freebase. The 93 that we could ID in Freebase are the arg1s we used for evaluation.

We then manually labeled the ReVerb entities by examining the articles they were pulled from and matching them to a Freebase entity. ReVerb entities that did not have matching Freebase entities were not used when computing overall accuracy, since they would always be missing. ReVerb entities with multiple possible matching Freebase entities were not used because they represent extraction errors by ReVerb. Our system still provides matches for those entities, but it is difficult to say whether or not it is providing the correct matches since there are multiple possible correct answers.

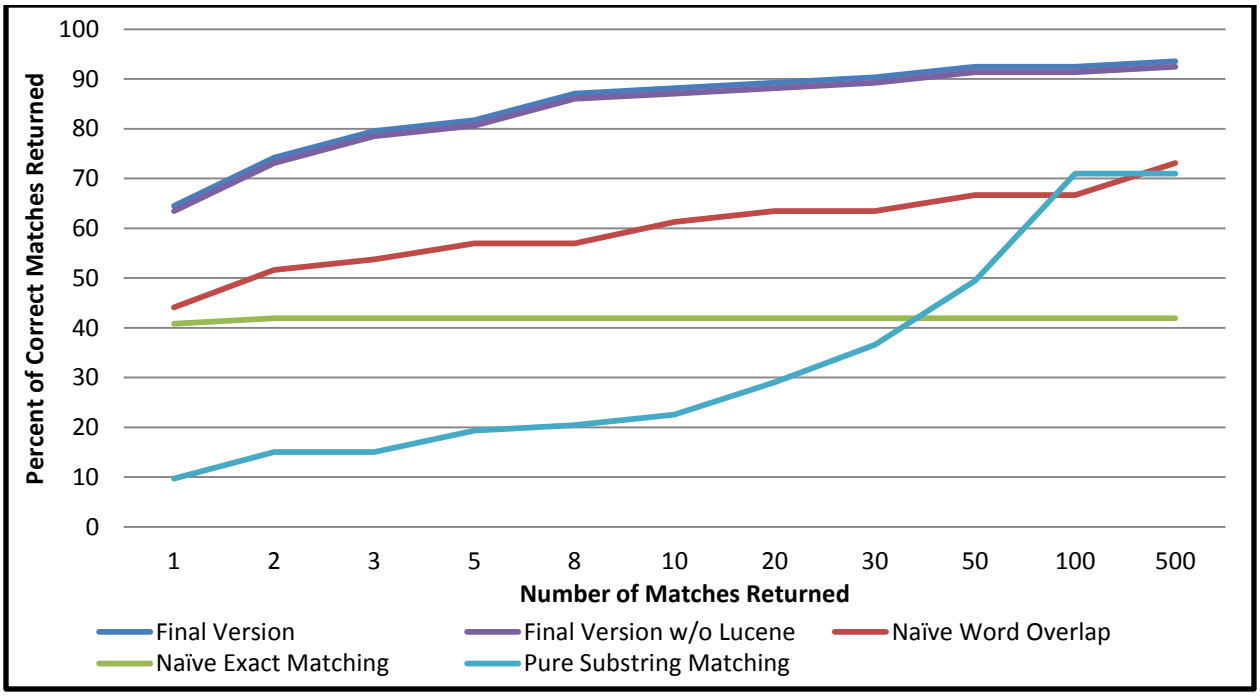

Figure 3: This figure shows the improvement using more than one matching technique provided. It also shows the slight improvement using Lucene matching provided at the slight cost of earlier accuracy (in the top 3 matches).

We then tested our matching system on this set of entities and were able to achieve 81% accuracy for the top 5 matches as the dark blue line shows in Figure 3. In other words, 81% of the time, the correct match was in the top 5 matches. Missed matches were mostly because the actual matches had very low prominence. For example "Carney" is missed because there are many arg1s with "Carney" in them with higher prominence than the correct match "Jay Carney".

As displayed by Figure 3, the more matches that are returned, the higher accuracy that the system achieves; however the context matching system that these matches are passed off to cannot handle a large number of matches for two reasons. One is that providing too many matches makes it more difficult to determine which one is the correct entity from the context. The other is that the context matching system is slow, so providing even a few more matches hurts performance significantly.

We were able to process an average of 240 ReVerb entities per second, not including pre-computation. The system takes a minute or two to set up the hash tables used for matching and load Freebase into memory. While running, the system takes around 4GB of memory (plus the size of the ReVerb corpus), which is a rather large amount, but it is manageable by a cluster of servers.

Tradeoffs

When deciding whether or not to add a feature to the system we considered two criteria: how much it slowed the system down, and how it affected the top 5 results.

String distance and substring matching, which are implemented by Lucene matching, were not included as a major part of the system (Lucene is only used when no other methods yield high quality matches) because they were much slower than the other methods and because they also bloated the top 5 matches with many irrelevant matches. For example “Khaled” matches to “The Persian Gulf” because

"Khaled" has a similar string distance to "Khaleej" which is a Wikipedia alias for "The Persian Gulf". Substring matching also provides many low quality matches, seeing as it only achieved 20% accuracy when used alone.

Exact string matching and word-overlap matching were included as necessities for many types of matches. When using hash tables, these methods are also very fast. One downside of these methods is that they take up a lot of memory, since both methods create several million entry hash tables. Cleaned exact string matching, Wikipedia alias matching, and abbreviation matching also created several million more hash table entries, but they were also very fast for lookup as Figure 3 shows. They also provide many high quality matches, especially Wikipedia alias matching. Exact string matching, cleaned exact string matching, Wikipedia alias matching, and abbreviation matching account for 2.1% of the overall time spent matching per entity. Word-overlap matching accounts for 46% of the time spent matching. This is mainly because it provides 98% of the total potential matches. Many of those matches are not high quality matches, and have lower match scores than matches retrieved through other methods.

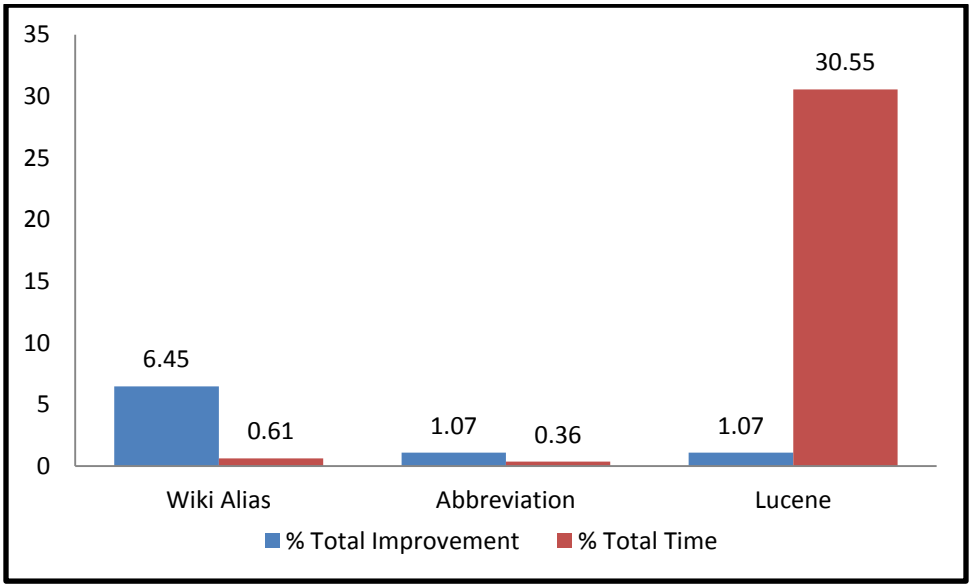

Figure 4: This figure graphs the tradeoffs of certain features. The percent total improvement is based on removing only that feature from the final version of the system, and the percent of total time is based on the total time used for the combined final version of the system. Word-overlap and exact matching are not shown here because they are the baseline matching criteria and removing them hurts overall speed and accuracy because it causes the system to rely more on Lucene matching.

As Figure 4 shows, Lucene matching took the largest majority of the time spent matching at 31%, even though it was only used on 60% of the dataset. Recall that Lucene is only used when a ReVerb string does not have a match with a score of 40 or greater. Since there weren't many ReVerb entities that this method helped, so if it was necessary to speed up the system, the threshold that determines when Lucene is used could be lowered so that it would only be used in a case where there were only very low scoring matches.

Misspelled arg1	Correct Freebase Entity	Successfully Matched?
Gadafy	Muammar al-Gaddafi	yes
Americna	United States	yes
Lucnene	Lucene	yes
Titatic	Titanic	yes
Obma	Barrack Obama	yes

Goooogle	Google	yes
Mobarak	Hosni Mubarak	no
Isrial	Israel	no
Abrham	Abraham Lincoln	yes
Lincon	Abraham Lincoln	no
Compusters	Computer	yes
Versizon	Verizon	yes

Table 2:This table gives some examples of misspelled arg1s that we generated and whether or not they were successfully matched to the desired Freebase entity.

Despite Lucene taking up a significant amount of time, it is still an essential part of the matching system, especially for ReVerb extractions from noisier web text. As Table 2 shows, our system correctly matches 75% of a set of 12 misspelled ReVerb arg1s because Lucene was included. Lucene was chosen to be included in the system for this reason, and so these misspelled matches would not be ignored.

Conclusion

Some surprising results that we learned during this project were that substring and string distance matching methods were not very effective when used alone, while Wikipedia alias matching, exact string matching, and word-overlap matching performed very well. The reasons that the stricter matching performed better than the weaker substring and string distance matching are because most of the entities that ReVerb extracts are not misspelled and are usually contain whole words that match to a Freebase entity. Many times when words seem to be misspelled or shortened, the Wikipedia aliases will still provide the correct matches for those when word-overlap and exact matching may have missed them. This is because when an alias/shortening appears online, especially in new articles, there is a good chance it also appears on Wikipedia, and will then appear in for our alias matching.

Some possible extensions to this system include the following:

- Speeding up word-overlap matching: Currently, word-overlap matching returns far more matches than is necessary, which is part of the reason why it takes so long. To speed it up, we could divide the hash table lookup into *k* buckets, ordered by prominence, and only search the next hash table if there haven't been a sufficient number of matches found so far.
- Performance upgrades: Finding a way to filter out low quality matches before computing the scoring/sorting algorithm would help speed. Better allocation and management of the data structures used could help reduce memory usage and possibly speed up pre-computation speed.
- For news articles it would be useful to have a preprocessor that could resolve arg1s like "Obama", to "Barack Obama" if "Barack Obama" is mentioned earlier in the article.
- A lot of ReVerb extractions have descriptor and determiner words in them. Matching would be improved if those were treated differently than nouns such as filtering them out, or weighting a word match on an adjective less than one on a noun. Perhaps this could be achieved using a POS tagger.
- Freebase has a list of synonyms that were not included in this matching system for performance reasons. This list may be able to catch more matches like the wiki-alias matching.
- Acronyms usually only correspond to companies, organizations, etc. Freebase tags entities into different categories like these. Acronym matching would be improved if it was only to entities that are in these categories.

By showing an improvement in the overall accuracy of around 40% from the naïve matching method of exact string matching, this system is a valuable addition of systems for mapping textual extractions to

ontologies by quickly providing the initial matches, which helps such systems improve overall performance.

References

[Apache Lucene] – www.lucene.apache.org

[Grounder, 2009] – *Scaling Wikipedia-based Named Entity Disambiguation to Arbitrary Web Text* – Anthony Fader, Stephen Soderland, and Oren Etzioni. WikiAI (IJCAI workshop), 2009.

[Freebase, Metaweb Technologies] – www.freebase.com

[ReVerb, Turing Center] – *Identifying Relations for Open Information Extraction* – Anthony Fader, Stephen Soderland, and Oren Etzioni. EMNLP, 2011. www.reverb.cs.washington.edu

[Cohen, CMU] – William Cohen, SecondString, Carnegie Mellon University, http://secondstring.sourceforge.net/

[WebConfs] - http://www.webconfs.com/stop-words.php

[Yahoo Boss] – www.developer.yahoo.com/search/boss