Introduction

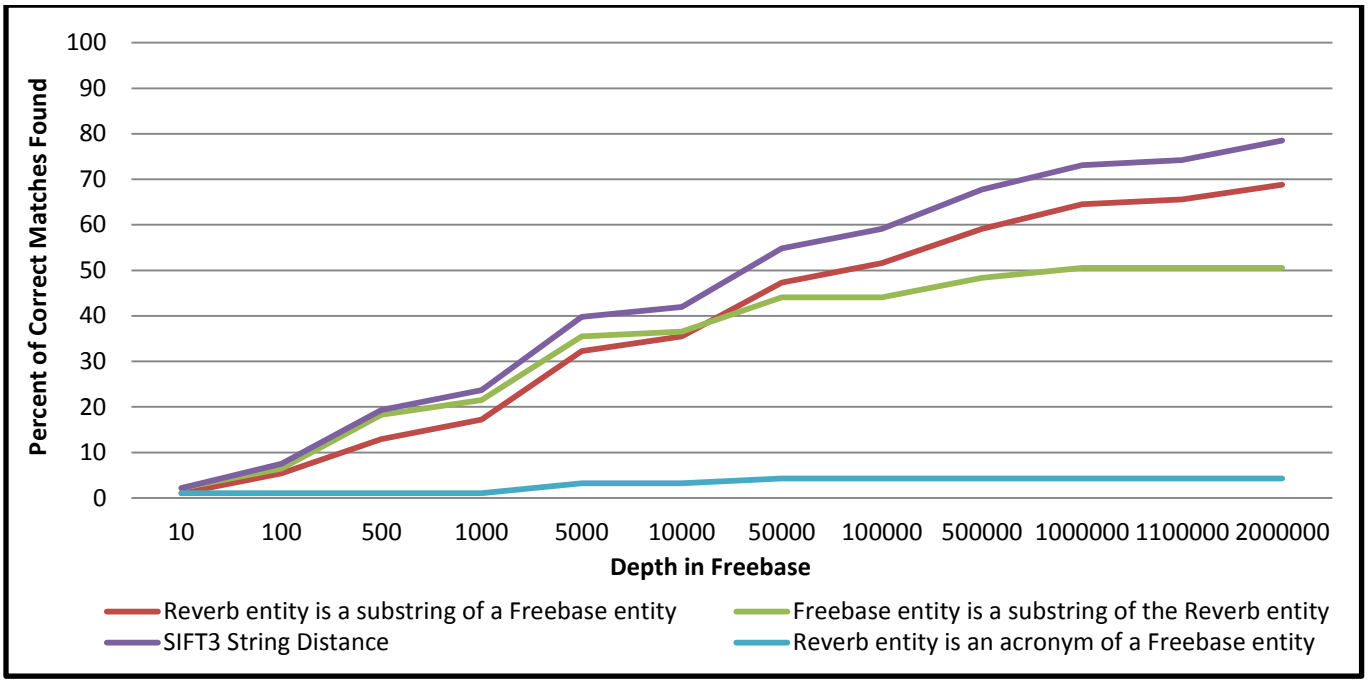
This system's main purpose is making the entity disambiguation task of connecting Freebase entities with Reverb Entities scalable. The existing system uses context and prior knowledge to rate matches between the two ontologies; however, this is a very slow process, and initial matches need to be provided to the context disambiguation system in order to improve performance to a point where it is practical. Without the initial matching system described in this paper, the context disambiguation system would be too slow to be considered practical.

In addition to an improvement in speed, this system also provides rated matches so that the score of the string match can be taken into account when determining the final score. The idea behind this is that entity names that have a higher string match are more likely to be actual matches. For example "US" is more likely to match "United States" than to match "donuts".

It could be asked why a system such as this needed to be designed, instead of using an existing string search engine such as Lucene. This is because Lucene does not provide the performance we are looking for. Lucene usually returns matches within 20ms. Lucene is still used in the system, but shallower string matching, which is much faster, is used more often.

Initial Approach

The initial approach to solving this problem iterated over all entities in Freebase and it returned all entities that met the matching criteria. This approach was ultimately scrapped, but it provided valuable intuitions for the final solution to the problem.

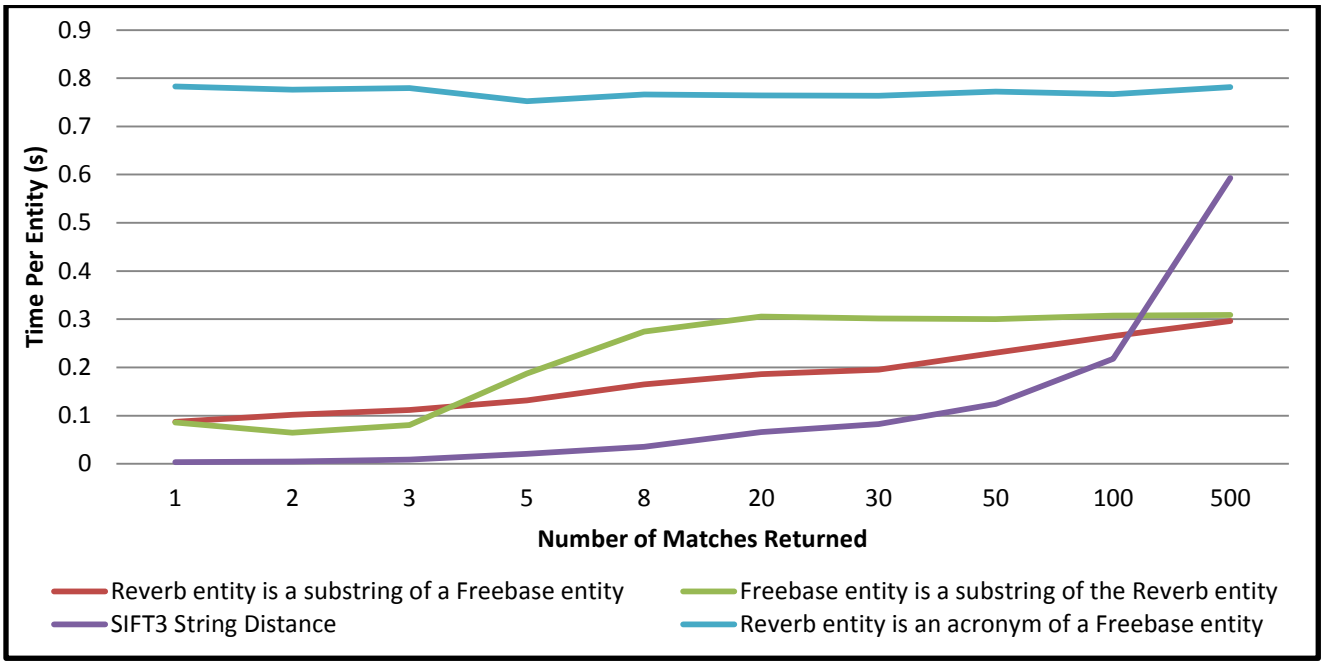


Graph 1: Each line signifies the recall of each individual matching method standing on its own.

The matches were then ordered by inlink count (prominence). The matching criteria were as follows:

- Substring Matching
  - *The Reverb entity is a substring of a Freebase entity*
  - *A Freebase entity is a substring of the Reverb entity*

- Acronym Matching
    - *The Reverb entity is an acronym of a Freebase entity*: One intuition provided by this experiment is that very little is gained by using this matching technique to higher depths in Freebase. As displayed in graph one, the recall for the news dataset did not increase at all after going deeper than the 50,000th entity in Freebase. To take advantage of this, acronym matching is only used for Freebase entities that have more than 20 inlinks, which creates a threshold that includes around the top 300k entities in Freebase. This makes sense because entities with very low prominence are less likely to have an acronym associated with them. For this approach, it helped in two ways: it filtered out many low quality matches and it improved the speed of the overall algorithm.
    - *A Freebase entity is an acronym of the Reverb entity*: This criterion was discarded, because it made precision worse. This is because most of the entities in Freebase are full names, and not acronyms. Instead of providing more good matches, it pushed some desired matches down by providing poor matches that may have higher prominence than the desired matches. **(EXAMPLE)**
- The String Distance between the Freebase entity and the Reverb entity is below a threshold
    - *SIFT3* was used to compute string distance, which provides an approximation of the Levenshtein Distance. Computing the exact string distance slowed down performance too much.



Graph 2: Speed of each matching criterion. The reason acronym matching is so high and constant, is because it would have to search much of Freebase before returning (since it could never find more than a few matches in the top hundred thousand), which is why putting in the threshold helps performance.

There were several problems with this approach. The biggest problem was speed. Iterating over all entities used very little memory, since a file could be scanned from disk; however, it took over a second to find matches for one Reverb entity, which is far too slow.

Another inefficiency is that full substring matching was not necessary to find high quality matches. Most often, full word matches are higher quality than partial word matches. For example, full substring

matching for “Chirstchurch” provided the entity “Christ”, which not a high quality match. Matches such as this pushed higher quality substring matches such as “Christchurch, Dorset”.

Final Approach

What are the new matching criteria?
Why using hash tables helped (speed) and hurt (memory usage).

Refinements:

- Cleaned Strings Matching -> remove s and parens
- Partial Substring Matching
- Wiki Aliases Matching
- Scoring Algorithm
  - Why only ordering on prominence didn’t work
  - How scores were assigned
- Lucene Matching:
  - When to use it (threshold)?
  - How much to decrease Lucene match scores by?

For Each Refinement

- Why add it in (case it helped)?
- What did it do to speed & memory usage?