Improving Ontology Alignment Accuracy with Detailed Feature Extraction Tools

Josh D. Dispoto

***Abstract*—Ontologies in the biomedical field have massive collections of information that are difficult to align and compare. Outlined in this paper are strategies to analyze current state of the art ontology alignment algorithms by analyzing results and performance with both small- and large-scale ontologies. Results of testing various sets of ontologies against a few named Ontology alignment tools is also included. The ultimate goal will be the modification and optimization of the Stoutenburg algorithm with Branch and Bound in order to produce superior results in precision and run time for large scale ontology alignment.**

I. INTRODUCTION

IN recent years, several algorithms for ontology alignment have been developed. The algorithms are needed for the integration and optimization of the many ontologies now available on the web [1]. Similarly, the size and scale of the ontologies available, as well as the potential alignments between said ontologies has grown substantially [2].

Ontologies are always being updated and revised. Ontology alignments also occur much more. These actions of ontological expansion inevitably lead to larger and larger ontologies[1]. Also, creating large ontologies takes time, funding, and can often be very difficult to create [3]. The clearly visible advantages to aligning ontologies manually include highly accurate alignment results. Therefore, to reduce the time of aligning ontologies, a highly accurate alignment program can be used.

The intention of this paper is to describe the approach used to extract and use highly detailed features from ontologies as a basis for alignment. Also, a detailed description of the algorithm currently in development that utilizes the aforementioned feature extraction tools.

II. PROBLEM STATEMENT

The goals of this project include: determining new, detailed features to extract from ontology concepts to form a basis for ontology alignment as well as developing an ontology alignment algorithm that will not only accurately extract the features but also use them to determine highly accurate alignments amongst ontology concepts.

The vast amount of information included within the ever-growing ontologies today has become increasingly difficult to analyze and compare. The algorithms being used today to align ontologies are all unique in their own strategies to determine matching ontology concepts. However, some more than others rely on an approach that may address reducing the run time of ontology alignment slightly more than accuracy[4]. Similarly some programs have sacrificed accuracy to improve specific concepts of their algorithm. The end result is the primary focus, however. Without accurate results, the whole purpose of ontology alignment falls from view. In order to uphold the highest standard of quality in information present in ontologies, the highest level of accuracy must be achieved in the process of aligning ontologies.

III. RELATED RESEARCH

Among the several ontology alignment algorithms and tools, different methods for alignments have been developed. These automated systems utilize a broad range of alignment approaches such as logical axiom similarity processing mentioned in [5] and similarity grouping followed by detailed relationship comparisons as in [2].

However, the Stoutenburg algorithm steps away from similarity-based alignments and uses the knowledge of upper ontologies and applies support vector machine (SVM) technology to produce non-equivalence relation alignments[4].

The Stoutenburg approach uses WordNet[1] and OpenCyc[2] to analyze concepts of the ontologies being aligned [4]. These tools extract features within the ontologies and organizes them appropriately. The resulting classifications are passed through a Support Vector Machine which determines proper alignments [4]. However, it fell short in execution time pushing upwards of 96 hours to align ontologies [4].

In response, paths were taken to decrease execution time of the Stoutenburg algorithm which led to the Branch and Bound algorithm. As described in [4] the Branch and Bound algorithm uses a concept of one ontology and determines semantic closeness to concepts in a second ontology; if it was determined to not have semantic closeness, the concept of ontology will be taken out as well as its children, otherwise the concept pair would continue to be aligned. Also, synonymous, hyponymous, and hypernymous relations are used to determine semantic closeness. Based on the results in [4] hyponymy yielded the best results. This later approach, however, yielded less accurate results and as a result still has room for improvement[4].

Altogether, there have been many approaches to ontology alignment. However, only recently have more approaches incorporated ways to handle large ontologies. Among the small number of programs accounting for scalability, the Stoutenburg branch and bound algorithm shows the most promising preliminary results in reducing run time. This

Paper submitted July 30, 2010. Submitted as an initial proposal draft as part of the REU NSF Internship program.

[1] http://wordnet.princeton.edu/
[2] http://www.cyc.com/opencyc

further exemplifies the need for higher accuracy in ontology alignments as the topic of scalability is an area of increasing focus. The accuracy of alignments must be maintained and improved for greater end results.

IV. FEATURE EXTRACTION

When addressing the aspects that might increase accuracy, we focused on the features being extracted from each ontology concept. Based on the prior work mentioned in [4] we found that the features being extracted included the following:

- Linguistic-based
- Pattern-based
- String-based
- Extrinsic-based
- Bag-of-words-based

Linguistic-based features as mentioned in [4] include those meaningful prefixes in strings that might indicate a subclass relationship because the algorithm in [4] uses an SVM to find class correspondence. Pattern-based features are similar to linguistic features in that they are simply patterns that arise in ontology concept names.

String-based features are those that include sub-string comparisons[4]. Finding that some ontology class pairs had the same ending words made for some meaningful data.

Bag-of-words-based features consist of the scoring of synonyms, hyponyms, and hypernyms among ontology concept pairs[4].

Finally, Extrinsic-based features are those concept pairs that hold syntactic or semantic relationships based on outside knowledge sources such as WordNet and OpenCyc as mentioned earlier[4].

Altogether these features are essential for accurate ontology alignments. All the preceding features were used in their own specific way in [4] to retrieve results using the Stoutenburg algorithm.

However, we used this knowledge to create our own specific set of features that we use in our own specific implementation of an ontology alignment algorithm.

V. CONCEPT

Based on the domain we will be focusing our efforts – biomedical ontologies – we found that there is a lot of information within the class names and labels of the ontologies. Our idea is to extract this information and use it as a basis for the concept alignment within the ontologies.

More specifically, the linguistic value of each class name or label is our focus. For example, the following label found in the Gene Ontology holds several key words that accurately describe what that specific class represents:

"The chemical reactions and pathways involving amino acids containing sulfur, comprising cysteine, homocysteine, methionine and selenocysteine."

Obviously there is a lot of information within the label that gives a vivid representation of the ontology class. However, while the linguistic-based features mentioned in [4] focused only on prefixes for certain relationships, our method uses not only prefixes, but also the root and suffixes of those keywords.

With all the affixes and roots compiled as a large knowledge base for comparisons, we can make more accurate results because of the highly specific information we extract.

However, there is still the hurdle of designing an algorithm that will accurately extract the linguistic features we seek.

VI. AFFIX/ROOT EXTRACTION

The goal of this feature extraction API is to take in a string word and extract all the prefixes, suffixes, and root from the word which can then be used by the user for whatever reason they may have for it.

We utilize character manipulation to complete this process. We will be using the word "invisible" as an example to provide a visual reference for how this algorithm works.

The extraction algorithm goes in the following order in extracting information from the word:

(1) extract suffix
  (1.1) extract additional suffixes
(2) extract root
(3) extract prefix
  (3.1) extract additional prefixes

Therefore, to begin with the algorithm strips off one character at a time from the front of the string and compares the newly truncated string against a large collection of suffixes that have been compiled over time into one file.

Figure 1 details the process through a more visual representation.

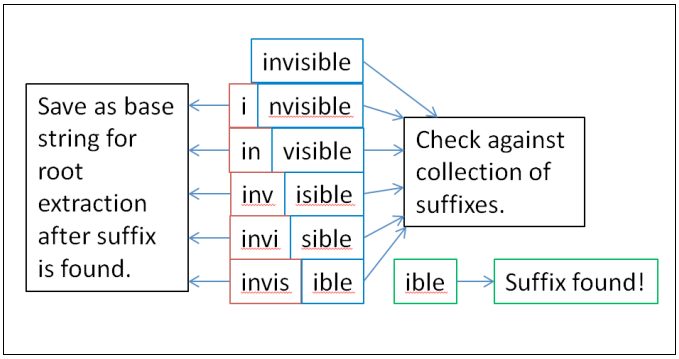

Figure 1: Suffix extraction process

The suffix extraction process iterates until no more suffixes are found. This ensures that those words that have multiple suffixes are taken care of. However this iterative process can also cause inaccurate extractions but that is detailed later.

Following suffix extraction, the saved base string – in this case the "invis" string – is passed to the root method to extract the root of the word, if there is one.

The root extraction process mirrors that of the suffix

extraction except that there are no iterations, there is only one pass through the string to determine a root. Figure 2 shows the continued extraction process of the word “invisible”.

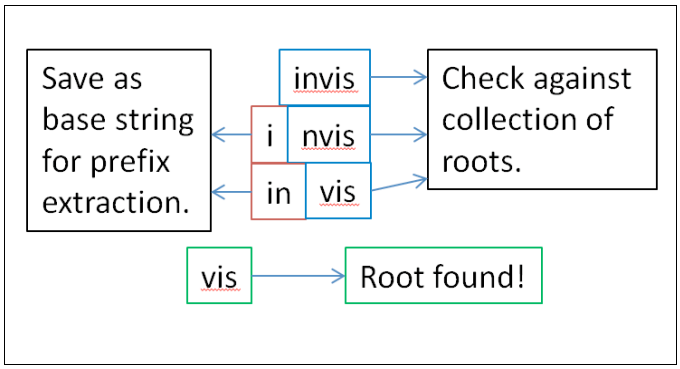

Figure 2: Root extraction process

Again, the root goes by the same process as the suffix extraction process, just without iterations. In some cases, however, there may be no root found. This may be caused because the root within the word is not in the collection of roots or it may be because there is no root in the word and the word consists of only prefixes and suffixes. Either way, the next step is to extract the prefix of the word.

The prefix extraction process differs slightly. Instead of stripping characters from the front of the string, the prefix extraction algorithm takes characters from the end of the string. This way, in case no root was found and there were excess characters on the end of the string, it will not interfere with the prefix process comparing the string to the collection of prefixes. Figure 3 shows the final results of the extraction process of “invisible”. However, to further detail how the prefix extraction algorithm works, see Figure 4.

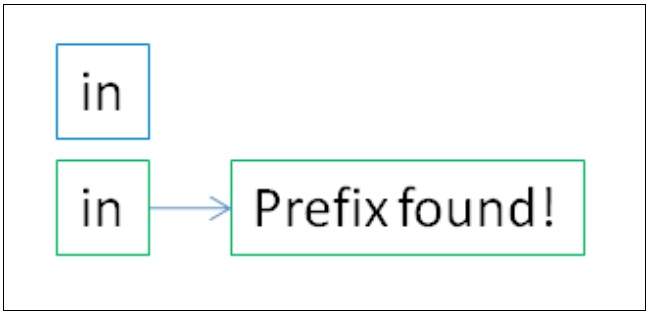

Figure 3: Prefix extraction process

The string “in” that remained was quickly checked and found within the collection of prefixes and so the extraction process for “invisible is complete.

For the following example in Figure 4, we will be using the word “covalently”. Assume that the suffix “ly” has already been stripped from the string and that no root was found when checked through the root extraction process.

In this case, the left behind string will be iterated again until no prefix is found. Also, just like the suffix extraction process the iterations can cause inaccuracies to occur. However there have been steps taken to error check the validity of the end results which will be mentioned later.

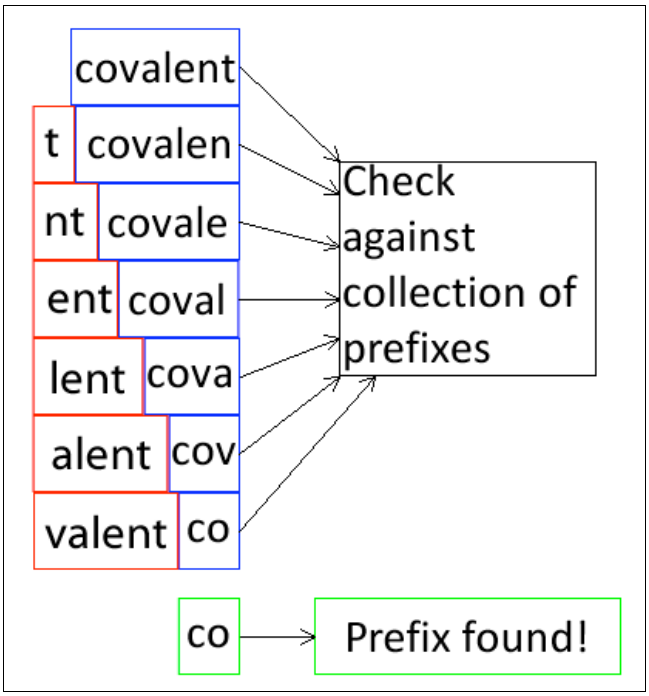

Figure 4: Complete Prefix extraction process

After all suffixes, prefixes and the root have been extracted the process is nearly complete. For the most part these three methods will accurately extract all the information from the string and will be available to the user to do with as they wish.

All that remains are some error checking procedures that handle some of the inaccuracies that may occur.

VII. ERROR CHECKING

Language, especially the English language, has many exceptions that have the potential to fool the extraction algorithm detailed above. Therefore we have implemented a few error checking algorithms to ensure the accuracy of the affix and root extraction process.

To begin, before the string is passed through any extraction algorithms it is checked for proper spelling and whether or not it is an accurate word in a large dictionary file. Currently the file has over 180,000 word entries that provide an adequate baseline. There is still room to expand the dictionary with more biomedical terms that will cover much of the domain we will be testing in.

Once the word has passed the spell-checking test, it moves through the extraction methods and based on the ending results of the extraction process will be put through further error checking algorithms.

There are special cases that can cause double checking to occur. Some of the cases that can cause inaccuracies in the extraction process include:

- Too many suffix extraction iterations that strip away substrings that are not accurate suffixes

- Ex. “covalently” on the first run through will find “ly”, “ent”, and “al” as suffixes.
- “ly” is the only suffix in this word

- Inaccurate root extraction
  - Ex. In the case of “” the root was extracted as “” when it should accurately be “”.
- No root found
  - Leaves behind remaining string as in Figure 4
- Too many prefix extraction iterations that identify incorrect prefixes if no root was found.
  - Ex. If no root was found and the prefix extractor stripped out a registered prefix still included in the extra substring

The first error check algorithm is used if there is a string left over from the extraction process as in Figure 4. The algorithm takes the left over substring and begins reconstructing the word by adding the last suffix found, one character at a time.

An example of this process can be found in Figure 5. Assume in Figure 5 that the word “covalently” is used. Also, assume that the suffix iteration extracted incorrect suffixes as mentioned above in the first bullet of inaccuracy cases and that the remaining string consists of only “v” because the prefix was extracted as well.

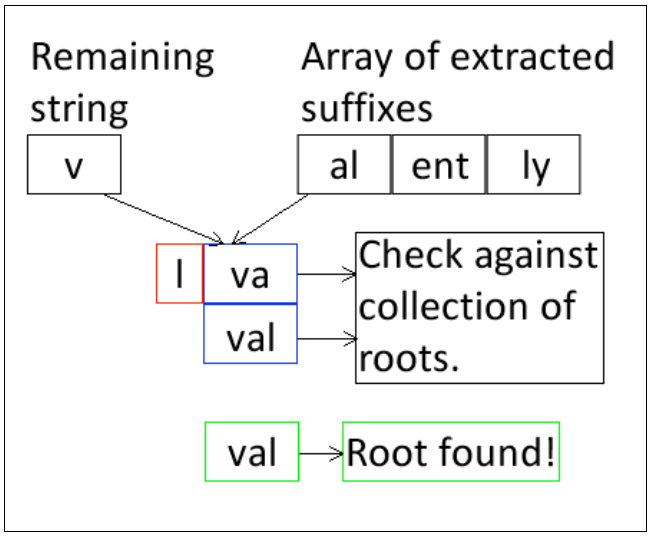

Figure 5: Error check - phase 1 process

If the error check finds no roots after reconstructing the first time, it will continue checking by taking the reconstructed string (in this case “val”) and adding the latest prefix that was extracted. Therefore, the string would be further reconstructed to “coval”. This substring is then checked against the prefix collection to make sure that the correct prefix was extracted and that the over-iteration of suffix extractions did not cause an incorrect prefix to be found.

Also, during this initial error-check, if any inaccuracies are found, the prefix, root and suffix information will be altered immediately to reflect the correct extractions.

Once the first error check is complete the next error case is reviewed. The occurrence may arise in which there is no string remaining after all extraction processes have completed but there are no prefixes found. If this is the case and there are more than three suffixes that have been extracted, a new error check is performed. The first step it takes is to send the root word through the prefix extraction to ensure that there was not a premature extraction that caused no prefixes to be found. After finding finishing the prefix extraction method, this method reconstructs the string starting with the root and adding each suffix in order. Each time a suffix is added to the remainder string, it will be passed through the root extraction method to ensure accuracy in the root word.

After all checks have concluded, the proper changes are made and all error checking is complete.

VIII. IMPLEMENTATION

While the extraction algorithm may seem to accurately extract linguistic features from words, we could only do so much to test its level of effectiveness. We created several test cases to ensure the accuracy of the extraction algorithm. However, we decided to test it further by developing a unique ontology alignment program that utilized this tool.

The program is separated into four sections based on the information we would use to determine equivalence between ontology classes:

- Ontology class labels
- Affix/root definition closeness database
- The ontologies
- Alignment process

The information we will be focusing on to utilize the extraction algorithm are the labels for each ontology class. As described in Section V. CONCEPT, the labels of an ontology class hold a large amount of information that, if extracted properly, can serve to be an excellent source for highly accurate ontology alignments.

The ontology class labels section of the program holds the entire affix and root information as well as the definition closeness data from the database which will be covered soon.

The class labels code takes in the label string and separates the string into keywords. The keywords are then passed through the extraction algorithm where all the affix and root information is saved for later use. Once all the affix and root information has been collected, they are all passed through a database specific identification numbers are collected in return for use in the alignment process.

The database is a large knowledge base similar to that of WordNet and other defining database tools. The database holds synsets of data that carry an identification number. Each synset also holds a definition in which specific affixes or roots can be linked to. For example, the synset with the definition “one” also has specific affixes linked to it such as “uni” and “mono” which hold the same definition. This specific synset may hold the identification number of “10”. Therefore, if a synset of similar definition is present in the database such as

"single" the identification number would be close to the "one" synset and possibly be "11" (See Figure 6). The closeness of definitions is the basis for our alignment process.

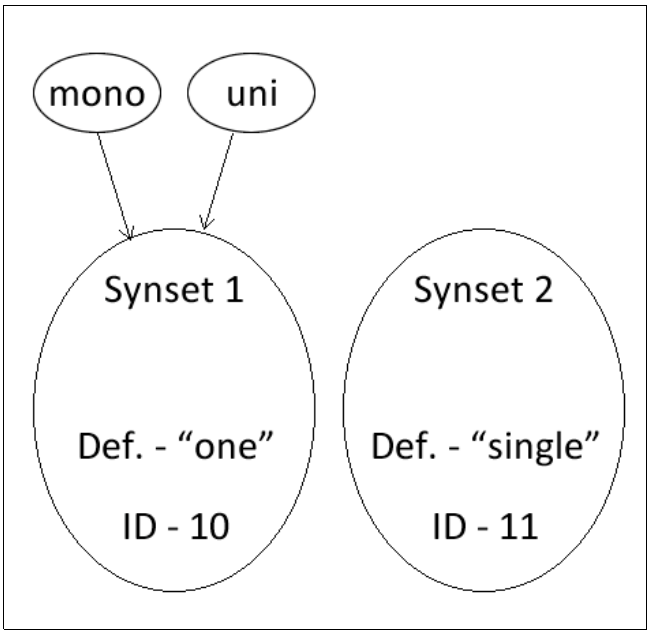

Figure 6: Database layout example

The next section includes the ontologies themselves. Two ontologies are passed two the alignment process to begin. Therefore, each ontology object holds all the class objects within that ontology as well as the class label objects. The ontology object also can return the specific URI of a class for alignment output.

Finally, the alignment process uses all the aforementioned information to determine equivalence and similarities between concepts from each ontology.

As described earlier, each class holds a label and within each label object is a collection of synset ID numbers from the definition database. The alignment process compares each label with the labels in the second ontology by calculating an accuracy value. Figure 7 shows how the process works for each comparison.

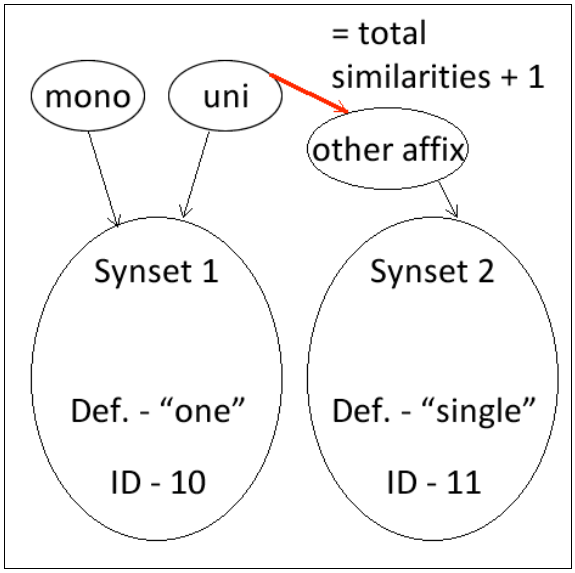

Figure 7: Synset Comparison and scoring

For every synset ID in Label 1 that is similar to a synset ID in Label 2, a counter is incremented and by the end of all the comparison the total calculated similarities is divided by the total number of comparisons made. A decimal such as this is calculated for all the prefix, suffix and root synset ID comparisons. Next, all the decimal values are added together and averaged for a final accuracy value.

The final accuracy is checked against a predetermined accuracy threshold. If the result is higher than the threshold, the both class URIs are printed to a text file with the accuracy value all separated by a semicolon. The results below the threshold point can also be printed if the user decides to use the information.

Also, a timer has been implemented to record the run time of the alignment process. The run time is displayed in the results both above and below threshold in milliseconds.

In conclusion, the alignment program utilizes the linguistic features derived from the word extraction algorithm in hopes of highly accurate results and excellent alignment results.

References

[1] S. K. Stoutenburg. *Acquiring advanced properties in ontology mapping.* (2008). PIKM '08: Proceeding of the 2nd PhD workshop on Information and knowledge management. pp. 9-16.

[2] Wei Hu, Yuzhong Qu, Gong Cheng. (2008). Matching large ontologies: A divide-and-conquer approach. *Data & Knowledge Engineering*. Vol. 67 (1). pp. 140-160

[3] R. Freckleton. (2010). *Ontology Alignment Using Automatic Machine Learning Techniques.*

[4] S. K. Stoutenburg, "Advanced ontology alignment: New methods for biomedical ontology alignment using non-equivalence relations," PhD. Dissertation, University of Colorado at Colorado Springs, 2009.

[5] M. Ehrig, Y. Sure. (2005). *Ontology Mapping by Axioms (OMA)*. Professional Knowledge Management: Lecture Notes in Computer Science. Vol. 3782/2005. pp. 560-569