Efficient Algorithms for Approximate Member Extraction

Pavan Basheerabad
pb14e@my.fsu.edu

Swapnil Kharche
sdk14e@my.fsu.edu

Project Type: Implementation

1 Problem Definition

Given a dictionary R of strings and a similarity threshold $\delta \in [0, 1]$, then a query M is submitted. The goal of Aprroximate Member Extraction is to extract all M's substrings m, such that there exists some r ∈ R satisfying $Sim(m, r) \geq \delta$. The approximation is constrained by a similarity function (such as Jaccard similarity and Edit distance) and setting an appropriate threshold will allow slight mismatches between the strings from given input query and strings in dictionary.

1.1 Summary

The project implementation is about finding the approximate match for the substring present in the document with dictionary of strings. If we have a document and we want some information from that document such as entity names, addresses, and conferences, then the extraction will be approximate basis i.e. allowing slight mismatches in the substrings. For instance, if we have a list of all conferences such as 'ACM SIGMOD' and 'CIKM Conference' in the dictionary and we want to extract all the conference names from the given document. The goal is to get the matchings such as 'CIKM international Conference' even though they do not exactly match the entry 'CIKM Conference' from our dictionary. A two step approach is followed for approximate member extraction. In the first step, the strings from the dictionary which are very different from the substring are filtered out. We made use of similarity measures like Jaccard Similarity and Edit Distance by setting a threshold δ to filter out the data. In the second step, each candidate string is checked to decide whether it should be extracted or not.

2 Related Work

The algorithmic nature of the AME problem mainly depends on the type of errors considered and the solutions range from linear time to NP-complete [4]. The problem can be approached by Dynamic Programming, Automata, Bit parallelism and Filtering Algorithms. We focus on solving problem of token-based AME problem by the use of Filtering algorithms. The problem can be reduced to set similarity join operation [1, 2, 5, 6]. Set similarity join problem is solved in [1] where the set similarity distance is converted into hamming distance between binary vectors and studying the number of shared segments of the divided vectors.

AME was drawn closer by both deterministic and indeterministic approaches in the writing. The deterministic methodology has been proposed with regards to d-queries: for a given input string , gives back all the dictionary entries whose distance to is inside [0, d]. The solutions proposed by Yao and Yao [10] and Brodal and Gasieniec [11] proficiently address the issue by using tree-like data structures for small values of d. Manber and Wu [9] selected to produce every single conceivable variety of every dictionary string, and afterward embed all varieties into Bloom filters. This approach additionally functions well for small d values, but for large distance threshold the generation of varieties gets to be computationally costly. In the indeterministic approaches, all substrings of the document are enumerated and several algorithms have been proposed under the filtration-verification structure [1, 2, 3, 7, 8]. In the filtration step, a substantial rate of substrings are filtered out using different sorts of signature schemes [1, 2, 5, 18], or inverted index list [18], or both of them [17]. In the verification step, the similarity between substrings and references in each remaining candidate pair is computed to figure out whether it qualifies as an AME result. Similarity functions can be categorized as token-based and character-based when the strings are regarded as sets of tokens and sequences of characters respectively. There are two methods which can be used for filtration - one is using inverted index on dictionary and other is based on the signatures generation [1, 2]. Currently, researchers are trying to combine these two methods. For example Mr. Wang

proposed a method called NGPP [3] for the efficient approximate entity extraction by using the edit distance similarity function.

When a similarity threshold is provided AME is set to 1, which degrades this problem into the exact match checking. In the exact match checking there are two well-known approaches, such as Aho-Corasick algorithm [12] and the Bloom Filter [13], are used more commonly. The Aho-Corasick algorithm recognizes all substrings that exactly match any part in the given dictionary in a single go through the input content by using a pattern matching machine or finite state machine. It develops a tree like machine which contains two types of connections, goto link and a failure link. Amid the lookup phase, all matches are listed by moving along the input, following the links and keeping the longest match. The Bloom Filter calculates the likelihood for a data string to be an individual from given dictionary by utilizing k different hash function to outline dictionary string to k array positions in a bit-array and the same set of hash functions to register k positions for the input data string. On the off chance that any piece on these positions is 0, the query substring is rejected. It may bring about false positives, yet never false negatives.

In the event that there is one and only part in the reference list, then the AME issue is decreased to another well studied problem, "Approximate String Matching," which focuses at finding a pattern string approximately in a content. Some good surveys are given in [4] and [14]. If we display every reference into a pattern, then the AME problem will be reduced to another well considered problem in the "String matching" literature, "Multipattern Matching" [15], which goes for finding all occurrences of patterns from a given set inside a record. A popular method for explaining the multi pattern coordinating issue is to construct a tree over all patterns as used in the Aho-Corasick algorithm. This can fundamentally reduce the number of comparisons between the substrings and patterns [16]. However, this procedure works for exact matches (with a threshold limit set to 1).

The motivation came from the work done by [7]. AME problem is solved using an Inverted Signature-based Hashtable (ISH), where the string and its signatures are encoded in to a binary matrix. However, there are some problems with the work carried out in [7]. First, the total weight of the signatures generated by the ISH is under certain lower bound which results in false negatives when the lower bound is too high. Second, the filtration process of ISH is an NP-complete problem. Simple Heuristics were used to solve the problem. As a result, it increases the number of false positives and overall performance is degraded. Third, similarity threshold is static meaning that the threshold value cannot be changed for different queries unless the index is generated again.

3 Technical Details

To perform the Approximate Membership Extraction, we need a dictionary. Dictionary is nothing but the list of conference names from the DPLB dataset. To get these conference names, we downloaded DBLP dataset from DBLP website [1]. The dataset was originally in xml format, so to get the data from the dataset we used parser and scraped the conference names from that. First we stored this dictionary in the MySQL database. We gave few text files as an input and stored the data from these files into a MySQL table. We implemented stemming to remove the stop-words from these input files before loading them to the database. We stemmed the data because the stop words such as: the, in, and etc. will not help for finding the approximate member. As mentioned in the paper, we first performed the filtration and then verification.

3.1 Filtration Framework

In the filtration i.e. pruning, all the irrelevant data is removed from the dictionary. Here, the dictionary is checked against the input data to find the irrelevant data. We have two similarity measures Jaccard similarity and Edit distance to perform filtering on the dictionary. The Jaccard similarity between any two strings m and r is calculated using the formula:

$$J(m,r) = \frac{m \cap r}{m \cup r}$$

where, weight (wt) used here is the Inverse Document Frequency (IDF) weight amd m, r are the strings as defined in the Problem definition.The Edit distance similarity is calculated using the formula:

$$ES(r,m) = 1 - \frac{ED(r,m)}{max(r,m)}$$

The foundation of the filtration is based on some necessary condition. The necessary condition used here is $Sim(m,r) \geq \delta$, that is, if some candidate evidence is real evidence, it must satisfy necessary condition. Note

[1] http://dblp.uni-trier.de/xml/

that, the necessary condition plays an important role in the whole framework. It ensures the correctness of the algorithms which we have used to find the approximate members. Again, it also determines how balanced the project framework is. There is tradeoff between the two phases of filtration-verification. For instance, if we considered all dictionary strings as the potential evidence, the most easy-going filtration approach for this is obviously "no filtration", which needs to scan the whole dictionary. In this case the time required for performing is less, but for the verification phase it is more. On contrary, if we make filtration more powerful, i.e. it produces no false positives and achieves less verification time; it will be expensive to perform such a filtering. As a matter of fact, we need to balance these two phases to achieve the best performance out of it.

For a given string S and the similarity threshold , we sorted all the tokens by their IDF weights in descending order (if the two tokens appear with the same weight, then those token are sorted according to their dictionary order), and then choose first few tokens to get subset Sig(S) (Sig(S) is the prefix signature set of string S), such that

$$\tau(s) = wt(Sig(s)) - (1 - \delta)wt(s) \geq 0$$

To ensure the uniqueness of the signature set we used k-signature set, where parameter K is nothing but the positive integer. When K is fixed we select k-signature set among all other signature sets as follows:

- Select the smallest signature set, when all signature sets are larger than k, or
- Choose a signature set whose size is exactly k, or
- Choose the largest one

When, k = 1, it is known as min-signature set and when k is infinity, string itself is the signature set.

3.2 Signature-based Inverted List

This algorithm is used to generate the inverted lists for the given strings. The algorithm takes an input dictionary of strings R, Similarity threshold δ and parameter k. The k-signature scheme is used to generate the signature sets for each substring r, where $r \in R$. The signatures are stored in variable Sig where for each token, $t \in Sig$, the $list[t]$ is merged with rid of r. The final list contains the signatures and string rids.

Algorithm 1 BuildSIL(R, δ, k)

```
1:
2: for each r ∈ R do
3:     Sig ← GenSig(r, δ, k);
4: /*The function GenSig(r, , k) generates signature for r under k-signature scheme. */
5:     for each ∈ Sig do
6:         list[t]=list[t] ∪ rid(r);//insert rid of r into list
7:     end for
8: end for
9: return list
```

For example, lets consider dictionary R = r1 = "canon eos 5d digital camera", r2 ="Nikon digital slr camera", r3 = "canon slr" and the weights are wt(5d, eos, slr, Nikon, canon, camera, digital) = (9, 7, 2, 2, 2, 1 , 1), then we have the signature set for each string in following table

rid	String	Signature Set
1	"canon eod 5d digital camera"	{"canon", "eos", "5d"}
2	"nikon digital slr camera"	{"nikon", "slr", "camera"}
3	"canon slr"	{"canon", "slr" }

Table 1: Signature sets of R's strings

Signature	String rids
5d	(1)
canon	(1), (3)
camera	(2)
eos	(1)
nikon	(2)
slr	(2), (3)

Table 2: SIL

3.3 Verification Algorithm

The algorithm is used to check the membership of each single substring m in document M. IT takes the parameters (M, δ, k, L) as input. When membership of string m needs to be checked, we compute the signature set of m, as { t1, t2, , tn }. Here, t denotes the token. Then we scan all the lists which were indexed by $list[t1], list[t2]$ $list[tn]$, to aggregate the weights of token ti to all the dictionary ids (rids) whose record contains the token ti as any of its signature. To record this aggregated weight array Sum[] can be used. The calculated aggregated weight of any rid is nothing but the exact value of $Sig(m) \cap Sig(r)$. The filtering condition checked here is

$$wt(Sig(m) \cap Sig(r)) \geq min\tau(m), \tau(r)$$

Any rid that satisfies this condition is stored for the later verification to determine whether it is one that makes substring m from input document a true member. Here, $\tau(m)$ is $wt(Sig(m)) - (1 - \delta)wt(m)$.

Algorithm 2 Verification(M, δ, k, L)

```
1: ResultSet ← φ ;// for storing approximate members
2:
3: for each m of Ms substrings (|m| ≤ L) do
4:     Sig ← GenSig(m, δ, k);
5:     Initialize Sum[];//for weight aggregating
6:     CandSet ← φ;// for storing candidate evidence
7:
8:     for each t ∈ Sig do
9:
10:        for each rid |in list[t] do
11:            Sum[rid] + =wt(rid);//aggregating weight
12:
13:            if Sum[rid] ≥ min{τ(m), τ(rid) } then
14:                CandSet ← CandSet ∪ { rid };
15:            end if
16:        end for
17:    end for
18:
19:    for rid ∈ CandSet do //verification
20:
21:        if Sim(m, r(rid)) ≥ δ then //true evidence found
22:            ResultSet ← ResultSet ∪ { m } ;
23:            break;
24:        end if
25:    end for
26: end for
27: return list
```

3.4 Similarity Threshold

Previous solutions to AME [1, 2, 7] supports only static similarity thresholds. The problem with this is, it is not possible to submit the query with other thresholds unless the filter is re-initialized. The SIL algorithm proposed above can overcome this problem by making little changes. In SIL, the problem of static threshold

is caused by prefix signatures. We need different signatures to build various filters for different δ.When the threshold gets lower, for any string s, the token t selected as a signature will also be in the signature set of s. Whenever the filter is initialized at a relatively low threshold, the *rids* of the strings should be included in some nodes on $list[t]$ and the nodes here are referred to as active nodes. The Theorem 3 of [17] gives the sufficient and necessary condition of min-signature. For any threshold value, all active nodes must form a prefix of the list. Scan can be stopped when an inactive node is found which results in enhancing the performance. When compared to the original SIL algorithm, applying this modification will require some additional space and additional sorting in the filter - building phase. The modified SIL solves the problem of static threshold problem for various similarity thresholds without visiting additional nodes or degrading query performance.

4 Evaluation

We ran our experiments on the dataset downloaded from the DBLP website. The original dataset consisted of 8000 conference titles. However, we reduced the size to 500 names for most of our experimental study. For the input query, we have provided 10 different documents which has conference names in addition to the other information. Example data from one of the query document is "*Base Endowment Inc VLDB, 2015 conference, is a non-profit organisation*" and example data strings from one of the rowid in dictionary dataset is "VLDB conference". Each token or string from the dictionary dataset and input query is either an English word or a number.

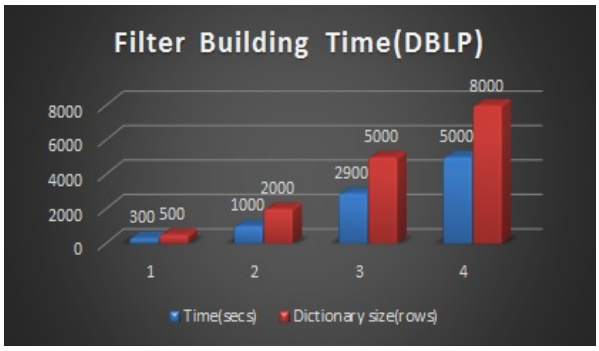

Figure 1: Filter Building using Jaccard Similarity (k=4, $\delta > 0$)

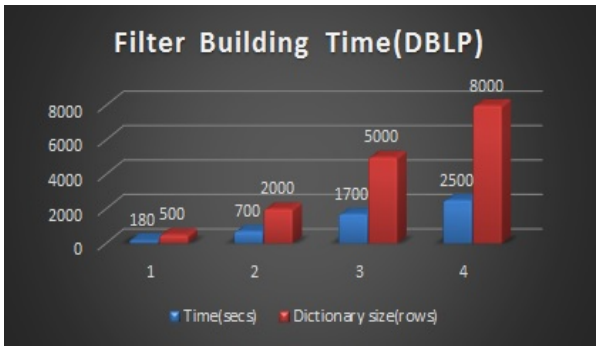

Figure 2: Filter Building using Edit Distance($\delta >$ 0.09)

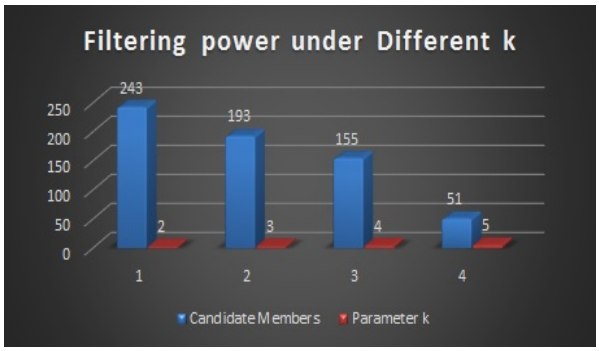

Figure 3: Filtration power using different k

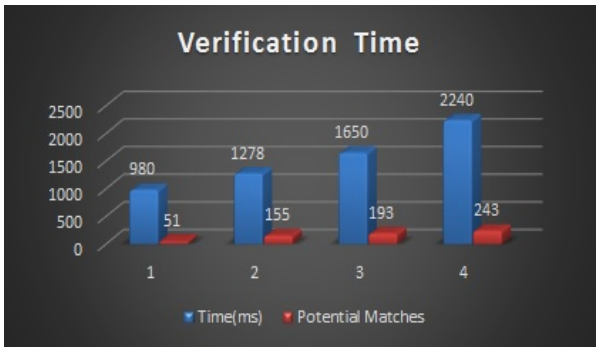

Figure 4: Verification time for different Candidates

Figures 1 and 2 show that Jaccard similarity measure takes more time to build the filters than Edit distance similarity. Since, in Jaccard similarity we are considering all the matches whose threshold is more than 0 and for Edit distance, we kept the threshold as $\delta > 0.09$.

Figure 3 shows that the number of candidate numbers decreases as the value of k increases. It means that there will be less number of potential matches when the value of k is higher. Figure 4 shows that the time taken for the verification of potential matches is directly proportional to the number of candidate members. Adjusting the parameter k and also similarity threshold will balance the filtration-verification power.

5 Conclusion

We studied Approximate Membership Extraction (AME) problem. In this problem, we identified all the strings from the input data that approximately matches the strings from the large directory. The characteristic of this scenario is most of the input strings do not match the dictionary members. Here, we implemented a compact filter which efficiently filters out large number of the strings which cannot match any member of the dictionary. The substrings which passes out the filter are then verified by checking the membership in the dictionary.

We implemented the filtration-verification framework and analyzed the problem of trading between the time requirement taken by filtration and verification phases. Here, we noticed that the filtration and the verification is inversely proportional to the each other in terms of time requirements, means if the filtration takes more time then the verification will be carried out in less time and vice-versa.

We implemented two similarity measures in this project such as Jaccard similarity and the Edit-Distance similarity. With the help of these similarity measures we filtered out the data.

6 Workload Division

The workload was pretty evenly distributed between the team members. The study of existing systems and choosing the algorithms for study was done by both team members. Pavan was responsible for implementation of the verification framework of the project and handling the MySQL database. He conducted the experimental study and analysed the results. The final report was written in Latex by Pavan.

Swapnil was responsible for gathering the Dataset using the DBLP parser and for the implementation of filtration framework of the project. In the filtration it covers staging of the data, here he loaded the data into the database and pruned the dictionary data which means removing the unnecessary data from the dictionary. Also, he implemented different similarity measures such as Jaccard similarity and Edit distances.

7 Source Code

`https://github.com/swapnilkharche/AME`

References

[1] A. Arasu, V. Ganti, R. Kaushik; Efficient exact set-similarity joins; In VLDB 2006.

[2] S. Chaudhuri, V. Ganti, and R. Kaushik; A primitive operator for similarity joins in data cleaning; In ICDE, 2006.

[3] W. Wang, C. Xiao, X. Lin, C. Zhang; Efficient approximate entity extraction with edit distance constraints; In SIGMOD Conference, 2009.

[4] G. Navarro. A guided tour to approximate string matching. ACM Computer Survey, 33(1):3188, 2001.

[5] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In VLDB, pages 491-500, 2001.

[6] S. Sarawagi, A.Kirpal, Efficient set joins on similarity predicates. In SIGMOD Conference, 2004.

[7] K. Chakrabarti, S. Chaudhuri, V. Ganti, D. Xin. An efficient filter for approximate membership checking. In SIGMOD Conference, 2008.

[8] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In ICDE, pages 257266, 2008.

[9] Manber, Udi, and Sun Wu. "An algorithm for approximate membership checking with application to password security." Information Processing Letters 50.4 (1994): 191-197.

[10] Yao, Andrew C., and Frances F. Yao. "Dictionary look-up with small errors." Combinatorial Pattern Matching. Springer Berlin Heidelberg, 1995.

[11] Brodal, Gerth Stlting, and Leszek Gasieniec. "Approximate dictionary queries." Combinatorial Pattern Matching. Springer Berlin Heidelberg, 1996.

[12] Aho, Alfred V., and Margaret J. Corasick. "Efficient string matching: an aid to bibliographic search." Communications of the ACM 18.6 (1975): 333-340.

[13] Bloom, Burton H. "Space/time trade-offs in hash coding with allowable errors." Communications of the ACM 13.7 (1970): 422-426.

[14] Navarro, Gonzalo, et al. "Indexing methods for approximate string matching." IEEE Data Eng. Bull. 24.4 (2001): 19-27.

[15] Navarro, Gonzalo, and Mathieu Raffinot. Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences. Cambridge University Press, 2002.

[16] Gusfield, Dan. Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge university press, 1997.

[17] Lu, Jiaheng, Jialong Han, and Xiaofeng Meng. "Efficient algorithms for approximate member extraction using signature-based inverted lists." Proceedings of the 18th ACM conference on Information and knowledge management. ACM, 2009.

[18] Chandel, Amit, P. C. Nagesh, and Sunita Sarawagi. "Efficient batch top-k search for dictionary-based entity recognition." Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on. IEEE, 2006.