Inquiry: The University of Arkansas Undergraduate Research Journal

Volume 9 Article 15

Fall 2008

Relative Searching using an Ordered Token List

Anthony Rosequist
University of Arkansas, Fayetteville

RELATIVE SEARCHING USING AN ORDERED TOKEN LIST

By Anthony Rosequist
Department of Computer Science and Computer Engineering

Faculty Mentor: Dr. Haiying (Helen) Shen
Department of Computer Science and Computer Engineering

Abstract

Many organizations have large amounts of information, such as consumer data, that need to be processed. Traditional searching algorithms only attempt to find exact matches to particular queries. This is undesirable when data are missing, outdated, or inaccurate. Therefore, a new type of search must be developed to locate records that are considered "interesting" to the user. This research paper examines past attempts to solve this problem and explores a new method involving ordered token lists to achieve this goal. The algorithm was developed, implemented, tested, and optimized.

1. Introduction

As technology improves the ability to gather information, the quantity of data significantly increases. A prominent example is consumer records, which consist of segments of information associated with individual consumers. In addition to traditional exact match searching, businesses are interested in searching methods that find similar, relevant matches. For example, a company may have the following source records:

Ann | Johnson | 16 | Female | 248 | Elm | St
Joe | Anderson | 25 | Male | 512 | 2nd | St
Jessica | Smith | 16 | Female | 716 | Main | St
Samantha | Anderson | 28 | Female | 248 | Oak | Dr

When, the following query is entered, all similar records are desired:

Jessica | Johnson | 16 | Female | Main | St

The algorithm should return Jessica Smith's record first, and Ann Johnson's record second, since they are the most similar records to the query. Through this example, we can see the importance of not requiring a strict match, since differences may be caused by missing, outdated, or inaccurate data. Jessica Smith may have been married since data was last gathered, explaining a change in her last name, or it is possible that Ann Johnson's middle name is Jessica and she recently moved to another street. However, the chance that the remaining source records are similar is extremely slim. Therefore, it is reasonable to allow a certain degree of dissimilarity between a source record and query record and still consider them relevant.

The rest of this paper is structured as follows. Section 2 reviews representative data searching approaches. Section 3 presents a search method based on an ordered token list, including its algorithms and some experiment results. Section 4 describes three strategies to improve the performance of the basic algorithm, and the performance of the strategies. It also shows the performance of the optimized algorithm in comparison with the basic algorithm. Section 5 recommends future research directions.

2. Related Work

Previous efforts have contributed to solving this problem. The simplest solution is linear searching, which involves comparing the query record to each source record using a distance function. This is extremely inefficient.

Andoni and Indyk proposed a method called Locality Sensitive Hashing (LSH) [1]. The basic premise behind LSH is to use hashing to place similar items into the same hash buckets. Then, for the query items, the correct bucket is determined and searched, so the domain of items needing to be examined is greatly reduced. LSH is accurate in locating most similar records, but requires too much time and memory to be practical in most real-world applications [2].

Other projects have improved the efficiency of LSH through various means. One method was to use the Lempel-Ziv-Welch (LZW) algorithm, which is a string compression technique [3]. Combining LSH with LZW was successful in improving the time and memory consumption of the original algorithm, but these deductions were still not significant enough for practical implementations [4].

Another method of achieving the same goal as LSH is using min-wise independent permutations to reduce the refinement stage of LSH. This is significantly faster than LSH, although the accuracy of the returned data is somewhat diminished, since the number of false positives is increased. False positives are located records but are not actually records similar to a query. This method is currently in development, and attempts are being made to increase its accuracy [5].

The algorithm proposed in this paper differs from these methods. Instead of reducing the search domain, it directly points to matches with similar tokens in a time-efficient manner using an ordered token list.

3. Ordered Token List Search (OTLS)

The algorithm requires two input parameters: a set of source records and a set of query records, denoted by S and Q, respectively. S contains records $s_1, s_2, \ldots, s_n$ and Q contains records $q_1, q_2, \ldots, q_m$. The records in both sets consist of a series of tokens, separated by a common delimiter. Many of the records will share common tokens, so it is useful to define a third set, T, which contains all of the unique tokens that exist in

the source records, denoted as t_1, t_2, …, t_j, in alphabetical order. The T is the ordered token list, which is the main structure used to allow efficient searching in this process.

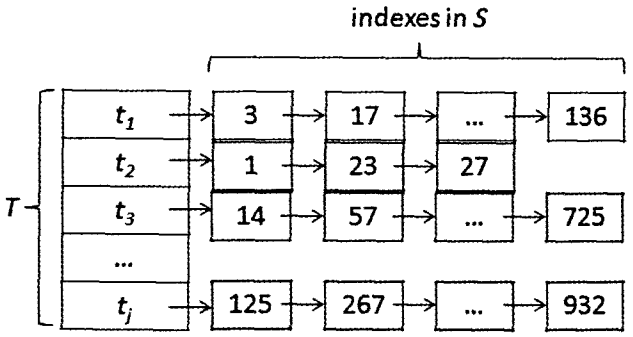

Figure 1. **Processing source records into ordered token lists.**

The first step in the algorithm is to assign a list to every token within T. These lists will hold the indexes, in ascending order, of all source records which contain that token, as shown in Figure 1. From this point on, all source records are known by their index within the source list rather than their actual string value.

That is all the processing that must be done to the source data. After building the token lists, the algorithm begins processing query records. For each record, the lists that are associated with each token are conjoined to form a *results* list for each query, as shown in Figure 2. This list holds the index of a source record each time it shares a token. Using these data, an algorithm can calculate how many tokens the query has in common with each source record and return the values accordingly.

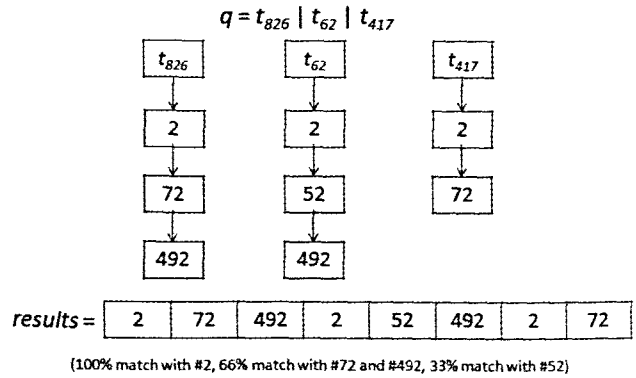

Figure 2. **Processing a query record.**

The results list will hold every source record that has anything in common with the query record. In practice, a threshold value, such as 60%, is set to ensure that trivial records are not returned, since the output should only consist of all relevant and interesting matches. Algorithm 1 shows the pseudo-code of the basic ordered token list search algorithm.

Algorithm 1: Pseudo-code of Ordered Token List Search algorithm

```
(1)  create an alphabetized list of tokens
(2)  for each source record i do
(3)  for each token in the source record do
(4)  add i to the Linked List for the token
(5)  end for
(6)  end for
(7)  for each query record i do
(8)  create new array, results
(9)  for each token in the query record do
(10) add the Linked List for the token to results
(11) end for
(12) sort the elements of results by number of occurrences
(13) output results
(14) end for
     end
```

4. Performance Evaluation and Improvements

Ordered Token List Search will return every record that is relevant, sharing any tokens in common (above a particular threshold, defined by the user). Therefore, it will return no false positives, and it will miss no true positives. In this sense, the algorithm is completely effective.

However, time requirements also needed to be considered. To test Ordered Token List Search, records reflecting the nature of real world consumer data were provided by Acxiom Corporation. The Ordered Token List Search algorithm processed 423,801 source records and 10,000 query records in 2,910.98 seconds (about 48.5 minutes) on a Windows PC. This was significantly longer than expected, making this simple implementation unusable for real-world purposes.

To address this problem, we further improved the basic Ordered Token List Search algorithm to enhance its efficiency in terms of query latency. Specifically, we integrated three strategies into the basic method: binary search, list merging, and token list serializing. Instead of using sequential search to locate all related records in the ordered token list T, binary search is adopted in this step. After relevant records in T are located, the list merging method is used to speed up the process of locating actual similar records. The details of the three strategies are introduced in the following sections. Finally, rather than building an ordered token list every time for data queries, a token list serializing method is developed to build a static ordered token list to save list construction time.

4.1 Binary Search

A timing analysis was used to determine bottlenecks in the code. As Figure 3 shows, a great majority of the time is spent filling the token lists with the right source indexes.

Upon further investigation, one particular section of the code can be seen as the primary performance bottleneck. Filling the token lists requires two operations: finding the list associated with the current token and appending the source

record index to the end of it. Since appending to the end of a list can be done in constant time, the main waste in the program is the time taken to find the list associated with the current token. Fortunately, this can be greatly improved.

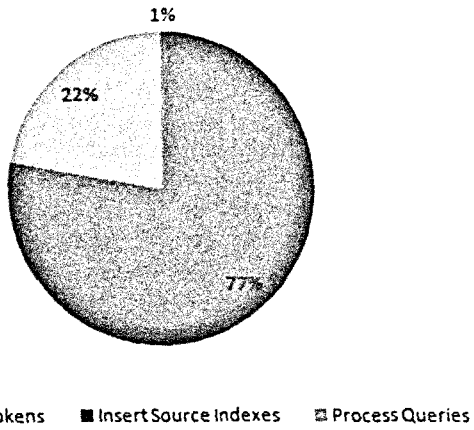

Figure 3. **Runtime of the original implementation.**

The simple implementation of the algorithm uses linear searching: It steps through each token until it finds the correct one. However, the token list being used is in alphabetical order, so the time to find the correct list can be reduced. The linear search was replaced with a binary search. Instead of starting at the beginning, the search begins at the middle of the list, and is able to cut the search space in half after each iteration. This decreases the asymptotic runtime of this particular operation from $O(n)$ to $O(\log n)$, a substantial improvement. Augmenting OTLS with binary token search reduced total runtime to 617.65 seconds (about ten minutes), a decrease of almost 80%, significantly impacting the actual time complexity of the algorithm.

4.2 List Merging

Although 10 minutes is a respectable runtime for 10,000 queries, further refinement was possible. Performance profiling identified that the query processing subroutine accounted for almost 97% of the total runtime, as shown in Figure 4.

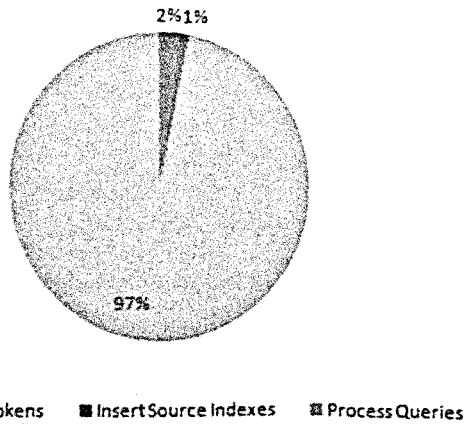

Figure 4. **Runtime of "binary search" implementation.**

To understand why query processing takes such a large amount of time to complete, a deeper explanation of the operation is necessary. Figure 5 shows the steps that a query record takes during processing, using the example from earlier in this paper. For each token, the token list is found and appended to the end of the *results* list. Once all of the token lists have been gathered, the *results* list is sorted in ascending order. From this, the indexes that appear most often (such that their percentage of similarity is above the threshold) are calculated and returned.

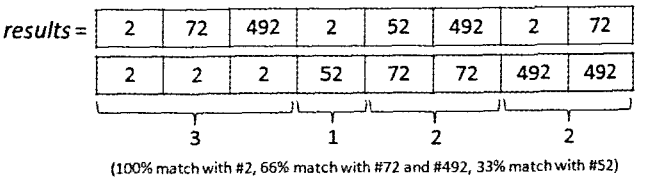

Figure 5. **Determining the closest matches of a query.**

The largest amount of time in this subroutine was spent sorting the *results* list. Since any source record that shares *any* tokens in common will appear in the list, it is expected to be large, requiring a substantial amount of resources to sort. However, this can be improved by noting that the token lists are already sorted in ascending order. Therefore, the *results* list can be sorted while it is being built instead of waiting until the end. This is essentially the "merge" routine that is used during a merge sort operation, so a simple implementation was used. By changing the method of merging lists, the entire query file was processed in only 21.78 seconds.

4.3 Token list serializing

Twenty seconds is certainly a reasonable runtime for an operation of this magnitude; however, some corporations may process millions of records at a time and would benefit from an even faster execution time.

Looking at the results from Figure 6, it can be seen that building the token list is now the bottleneck routine in the algorithm. Before any search, it must read the entire source file and return a sorted array of all unique tokens. An approach for

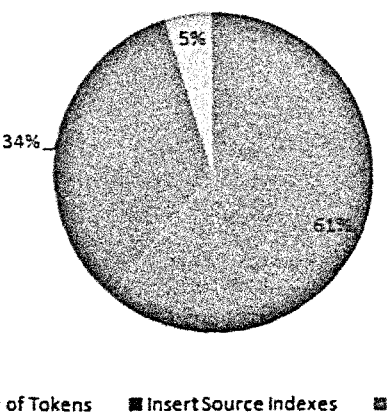

Figure 6. **Runtime of "merging lists" implementation.**

Inquiry: The University of Arkansas Undergraduate Research Journal, Vol. 9 [2008], Art. 15

reducing this time is to serialize the list, saving it to the user's computer, so it does not need to be generated every time. This is most optimal for static databases, where there will be few, if any, changes to the source files. However, even if the database is dynamic, changes may be done relatively quickly, since it is simply a matter of adding the tokens that do not already exist. The only time when this will not be practical is if there will be millions of source records added continually, a scenario unseen in most real-world applications of this method.

A separate file was created to hold the tokens, which were simply placed on separate lines in alphabetical order. Instead of generating this token list every time, the program simply loads this file into an array to make it usable. This dramatically reduced the overall runtime of the application. It takes around thirty milliseconds to load the token file, as opposed to thirteen seconds to build the token list previously. This lowered the total time to around 8.5 seconds.

4.4 Further Optimization

If needed, the program could be improved even more. The implementation that was used has overhead in object creation, method calling, and other operations that slightly added to the runtime. A final production version would need to reduce this overhead as much as possible; however, it would not reduce the asymptotic runtime and the results will vary across multiple systems, so it was not pursued further for this project. It is estimated that it would possibly reduce the runtime in this example another 0.5 seconds.

4.5 Performance Comparison

The Ordered Token List Search method of data searching has proven to be very effective. Due to the inflexible structure of the list orderings, it is impossible for the algorithm to miss a related entry. Similarly, it is impossible for it to identify false positives as matches, making it entirely accurate.

The Ordered Token List Search is also very efficient. Although the original implementation took a significant amount of time to complete, some optimizations were made that substantially reduced the time consumption. As can be seen in Figure 7, the final version runs in a fraction of the time that the original did, and Figure 8 gives a closer look at the last three implementations. Being able to match 10,000 records with their closest matches in approximately eight seconds on a standard system is reasonable for most environments.

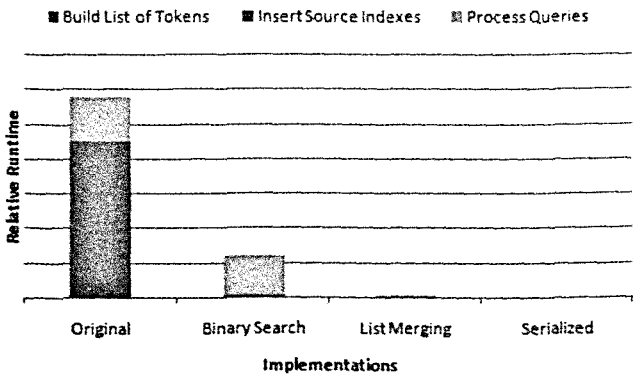

Figure 7. **Comparative runtimes of different implementations.**

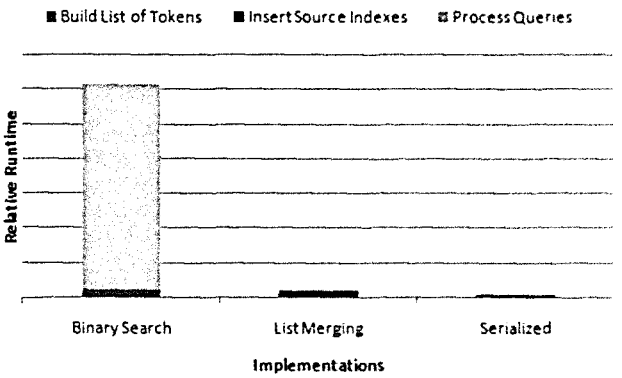

Figure 8. **Comparative runtimes of the later implementations.**

5. Conclusion and Future Directions

As technology to gather information improves and becomes more widespread, more advanced searching algorithms are necessary. Similarity data searching, instead of exact match searching, is increasingly needed by many organizations having large amounts of information. This paper proposed a data searching algorithm based on an ordered token list. This algorithm is able to locate not only exact matching data but also data similar to a query. Furthermore, it has high accuracy in that it returns all similar data without missing any false positives. To further improve the efficiency of the basic ordered token list search algorithm in terms of query latency, we integrated three strategies into the basic method: binary search, list merging, and token list serializing. Performance testing results show the superiority of the optimized searching algorithm in comparison with the basic algorithm.

Ordered Token List Search is an accurate, fast method of achieving this goal, running significantly faster than other routines. In the future, features can be added to improve the usefulness of the base algorithms in particular situations, such as adding weights to particular tokens (for example, matching a last name is more relevant than matching a person's state of residence) or relative comparisons of tokens (someone that is one year older is a closer match than someone that is eighty years older than the query). These would require specific knowledge about the records and we intend to address this challenge in our future work. However, the low runtime of this method allows for additional features to be added without significantly reducing performance, providing another significant benefit.

Acknowledgements

I thank my mentor, Dr. Haiying Shen, and her graduate research assistant Ting Li for their help with this project. I also want to express deep gratitude to Dr. Tom Schweiger, Acxiom

Corporation, for his insights, and Acxiom Corporation for financial support. This work was previously published [6].

References

[1] Andoni, Alexandr and Piotr Indyk, "Near-Optimal Hashing Algorithms for Near Neighbor Problem in High Dimensions," *Proceedings of the Symposium on Foundations of Computer Science*, Berkeley, CA, October 2006.

[2] Shen, Haiying, Ting Li, Felix Ching, and Ze Li, "A Study of Locality Sensitive Hashing: Its Advantages and Disadvantages for Nearest Neighbor Searching," *Acxiom Laboratory for Applied Research Conference*, Conway AR, March, 2008.

[3] Welch, Terry, "A Technique for High-Performance Data Compression," *IEEE Computer*, June 1984.

[4] Shen, Haiying, Ting Li, Ze Li, and Felix Ching, "Exploring Efficient and Effective LSH-based Methods for Data Prospecting," *Acxiom Laboratory for Applied Research Conference*, Conway AR, March, 2008.

[5] Shen, Haiying, and Ting Li, "Efficient Similarity Search Based on Locality Sensitive Hashing," *Acxiom Laboratory for Applied Research Conference*, Conway AR, March, 2008.

[6] Shen, Haiying, and Anthony Rosequist, "Relative Searching using an Ordered Token Method," *Mid-South Conference of the Consortium for Computing Sciences in Colleges*, Russellville, AR, April 2008.

Mentor Comments

Dr. Haiying Shen describes the way in which Anthony Rosequist's research and article represent the quality of outcomes generated by a combination of team effort and individual initiative.

Anthony Rosequist is an undergraduate student in the Computer Science and Computer Engineering Department, and one of our department's top students. I am the research mentor for his research work titled "Relative searching using an ordered token list." This research work has already produced two papers co-authored by Anthony, published in ACM/SIGCSE CCSC Mid-South'08 and the International Conference on Data Mining '08. Anthony's research is part of a broader project "hash-based proximity clustering for neighbor search in Acxiom database". The members of this project group include Anthony Rosequist, another undergraduate student, and a Ph.D. student. This project is designed to study the effectiveness of locality sensitive hash function (LSH) on data searching. First, this project designed a method using LSH for data searching. Second, this project developed a simulator to test the performance of the LSH-based data searching method. Third, the performance of LSH on data searching has been analyzed, and new methods including the token-list method to improve the LSH-based method have been explored and developed. Experimental results demonstrate the superiority of the proposed methods compared with the LSH-based method with regards to memory and time consumption. Anthony has been working on the project. In each step of the project, he not only accomplished his assigned work independently, but also cooperatively worked with others. More importantly, he implemented the token-list methods by himself.

This research work focuses on data searching in a massive database. It thoroughly investigates the current data searching methods, and proposes the token-list method to achieve enhanced data searching in terms of efficiency and effectiveness. The token-list method has significant impact to our society defined as the "Information Society", in which tremendous growth of information generates an increasing need for an efficient data searching method. In addition, this research work has many technical merits and significant contribution to the computer science and computer engineering area. This research provides critical insight into data searching, which is expected to have significant impact on data processing research. The outcome of this research is expected to serve the data processing community as a vehicle to conduct further research and experiments, and will advance the state of the art in data processing research area.

Based on the originality, novelty and contribution of Anthony Rosequist's research work, I highly recommend this work for publication in Inquiry.