OPEN ACCESS

International Journal Of Modern Engineering Research (IJMER)

Enhanced Methodology for supporting approximate string search in Geospatial data

Ashwina.R[1], Mrs.T.Megala[2]

[1, 2] *(MCA-III year, MCA. Assistant professor, Sri Manakula Vinayagar Engineering College, Madagadipet, Pondicherry, India)*

Abstract: *In recent years many websites have started providing keyword search services on maps. In these systems, users may experience difficulties finding the entities they are looking for if they do not know their exact spelling, such as a name of a restaurant. In this paper, we present a novel index structure and corresponding search algorithm for answering map based approximate-keyword in an Euclidean space so that the users get their desired results even though they have typos in the keyword. This work mainly focuses on investigating range queries in Euclidean space.*

Keywords: *Euclidean space, Approximate keyword, typos, LBAK.*

I. Introduction

Geospatial information is information that refers to the position of an entity on the Earth, and includes information such as street, city, and national borders as well as longitude and latitude coordinates. There are several local-search websites, such as Google Maps, Yahoo! Local, Bing Maps, Yellow Pages, and MapQuest Maps. At such a website, a user might look for a restaurant called "Chaochi" close to San Jose in California. The website returns the restaurants close to the city that match the keywords. Users often do not know the exact spelling of keywords. For example, the user may mistype a query as (chochi restaurant) near (San Jose, CA) when looking for the restaurant Chaochi. Similarly, a user could mistype the word "restaurant" and submit a query: (resturnt) near (San Jose). Approximate string search is necessary when users have a fuzzy search condition, or a spelling error when submitting a query. It is important to find relevant answers to such mistyped queries. Unfortunately, most existing location-based systems do not provide correct or approximate answers to a query even with only a single typo.We conducted various experiments to show how several systems behaved for three mistyped variations of the query "Chao chi restaurant" as of May 5, 2014.

In most cases, the search engines either returned an empty answer or gave irrelevant results. Both Google and Yahoo returned "We could not find the desired results". In this paper, we study how to solve this problem by supporting approximate keyword search on spatial data. Given a query with keywords and a spatial location, we want to find the location with those keywords, even if those keywords do not match exactly. Thus we can find relevant objects for the user even in the presence of typos in the query or data.

TABLE I. Results of Local-Search Engines for Mistyped Queries (as of May,5 2014)

Search Engine	**Results of mistyped queries**		
	Chochi restaurant	***Chooch i restaurant***	***Chauchi restaurant***
Yahoo!	Could not find the desired results	Word misspelt	misspelt
Google	Could not find the result	Misspelt	mispelt

Notice this approach is more powerful than the approach of suggesting an alternative query (the "Did you mean" feature used by many systems). The latter can only suggest a new query, while the new approach can find approximate answers, even if the answers' keywords are only similar to those of the query.

BACKGROUND WORK

Euclidean plane or space is as a set of points satisfying certain relationships, expressible in terms of distance and angle. It supports range queries. For example: It retrieves places within a particular range or distance.

a. R TREE

R tree indexing is used for searching string queries in Euclidean space. They are actually data structures used for spatial access methods for indexing multi –dimensional information such as geographical co-ordinates. But doesn't exactly support for approximate string search.
Defining the similarity between 2 strings was a major issue since computing the edit distance between 2 strings has a quadratic complexity (to the length of the string). Several string similarity comparisons lead to higher CPU cost (overhead).

Suppose a query string that does not have any similar strings within its query range leads to minimum query cost. But in R tree, it visits all the index nodes within query range leading to unnecessary IO overhead. R tree solution could suffer from unnecessary node visits which lead to higher IO cost (overhead). The fundamental issue here is that the possible pruning power from the string match predicate is completely ignored by the R tree solution. Since the pruning power was completely ignored by the R tree a combined approach that prunes simultaneously on string match predicate and spatial predicate was proposed.

b. MHR TREE

MHR tree is R tree augmented with the min- wise signature and the linear hashing technique. The min – wise signature for an index node u keeps a concise representation of the union of q-grams from string under the sub tree of u. The pruning functionality of such signatures was analyzed based on the set resemblance between the query string and the q grams from the sub trees of index nodes.

In this structure similar strings are retrieved, points that do not satisfy the spatial predicate are pruned in post processing.

But the size overhead of these signatures is very small. Less space so can't increase query time. The method misses query answers due to probabilistic nature of the signatures.

1. AK TREE

AK (Approximate Keyword) tree is used to return approximate results rather than exact results. It can answer queries such as find **chochi restaurant near San Jose**. Notice that chochi is misspelled but the AK-Tree can still find useful answers. In short, the AK-Tree answers queries with a spatial component and a keyword component, where the keywords don't need to match exactly but approximately. The main idea of the basic index is to augment a tree-based spatial index with capabilities for approximate string search and keyword search. We use approximate string search to identify for each keyword those strings that are similar. Once we have identified similar keywords, we use the keyword-search capability to prune search paths. To support keyword search we choose some nodes to store the union of keywords contained in objects of their sub tree. The indexing structure used is R *tree which is an enhanced version of R tree.

1.2 PROBLEM FORMULATION

The problem of approximate keyword search on spatial data can be formulated as follows. Consider a collection of spatial objects obj1,...,ojbn, each having a textual description (a set of keywords) and a location. A spatial approximate- keyword query Q = (Qs,Qt) consists of two conditions: a spatial condition Qs such as a rectangle or a circle, and an approximate keyword condition Qt having a set of k pairs {(w1,δ1),(w2,δ2),...,(wk,δk)}, each representing a keyword wi with an associated similarity threshold δi. The similarity thresholds refer to a similarity measure such as edit distance, Jaccard, etc., which could be different for each keyword. Our goal is to find all objects in the collection that are within the spatial region Qs and satisfy the approximate keyword condition Qt. We focus on conjunctive approximate keyword queries; thus, an object satisfies the approximate keyword condition if for each keyword wi in Qt, the object has a keyword in its description whose similarity to wi is within the corresponding threshold δi.

1.2.1 ALGORITHM

We discuss the search procedure for answering spatial approximate-keyword queries. We classify the AK- tree nodes into three categories:

- S-Nodes
- SA-Nodes
- SK-Nodes

S-nodes: Do not store any textual information such as keywords or approximate indexes, and can only be used for pruning based on the spatial condition.

SA-Nodes: Store the union of keywords of their sub tree, and an approximate index on those keywords. We use SA-Nodes to find similar keywords, and to prune sub trees with the spatial and approximate keyword conditions.

SK-Nodes: Store the union of keywords of their sub tree, and prune with the spatial condition and its keywords. Note that we must have previously identified the relevant similar keywords once we reach an SK-Node.

a. ALGORITHM OUTLINE

Let Q be a query with a spatial condition Qs and an approximate-keyword condition Qt with k keywords. The algorithm traverses the tree top-down and performs the following actions at a tree node depending on the node type. At an S-Node, the algorithm only relies on the spatial information of the node to decide which children to traverse. Once the algorithm reaches an SA-Node, it uses the node's approximate index to find similar keywords. For each keyword wi in Qt, the algorithm maintains a set of key- words Ci that are similar to wi according to its similarity threshold δi. Assuming we use the AND-semantics, a node is pruned if one of the query's Ci sets is empty. Otherwise, we propagate the list C = C1,...,Ck downward and use it for further pruning. In particular, at each SK-Node n, for each keyword wi in Qt, the algorithm intersects its similar keywords Ci propagated from the parent with the stored keywords of n. The node n can be pruned if one of the similar keyword-sets Ci has an empty intersection with the node's keywords. Otherwise, the algorithm passes those in-tersection sets of similar keywords to n's children for further traversal. At a leaf node, the algorithm adds to the answer set all the node's objects that satisfy the condition Qs and have a non-empty intersection between their keywords and each of the propagated similar-keyword sets from the query.

Pseudo-code: Let us examine the pseudo-code for AK- Search in Algorithm 1. Note that the algorithm invokes several helper procedures, e.g. InitSimilarKeywordSets, Proc- SNode, etc., defined in Algorithms 2, 3, 4 and 5. The input of Algorithm 1 is a query Q = (Qs,Qt) and an AK-tree root r. We initialize a list C of k similar-keyword sets (line 2), where k is the number of keywords in Qt. We maintain a stack S to traverse the tree. Initially, we push the root r and the list C to the stack (line 4). We start traversing the tree by popping the pair (n, C) from the stack (line 6). If n is not a leaf node, then all n's children that satisfy the spatial condition will be investigated (lines 7-9). Depending on the type of the node we invoke a helper procedure to process it (lines 10-18): For an S-Node (lines 11-12), we only rely on the spa- tial condition to do pruning, and push the pair (ni, C) to the stack S (within Algorithm 3). For an SA-Node (lines 13-14), we use its approximate index to find similar key-words for each query keyword as shown in Algorithm 4. We call GetSimKwds (wi, δi) to get wi's similar keywords, for i = 1,...,k, and store them in wi's corresponding similar- keyword set Ci. If at least one similar keyword is found for each keyword in Qt, then the pair (ni, C) is pushed to the stack S for future investigation. For an SK-Node (lines 15- 16), we compute the intersection Gi between ni's keywords and the similar-keyword set Ci. If all the intersection sets are not empty, then the pair (ni, G) is pushed to S to be examined later (Algorithm 5). Finally, when we reach a leaf node, we add its objects that satisfy the two conditions Qt and Qs to the results.

Algorithm 1. Ak Search

```
Input: A query Q = hQs,Qti, with Qt having k pairs {(w1,δ1),...,(wk,δk)} associating a keyword wi
with its similarity threshold δi;
                    An AK-tree root r;
Output: A set R of all objects satisfying Qs and Qt;
1  Result set R ← ∅;
2  C ← InitSimilarKeywordSets(r, Qt);
3  Initialize an empty stack S;
4  S.push (r,C);
5  while S = ∅ do
6    (n,C) ← S.pop();
7    if n is not a leaf node then
8      for each child ni of n do
9        if ni does not satisfy Qs then continue;
10       switch ni.type do
11         case S-Node:
12             S ← ProcSNode(ni, Qt, C, S);
13         case SA-Node:
14             S ← ProcSANode(ni, Qt, C, S);
15         case SK-Node:
```

```
16                S ← ProcSKNode(ni, Qt, C, S);
17            endsw
18        endsw
19     end
20 else // leaf node
21    foreach object oi of n do
22     if oi satisfies Qs and Qt then
23       R.add(oi);
24     end
25    end
26   end
27   end
28  return R
```

Algorithm 2. InitSimilarKeywordSets

```
Input: Root r of an LBAK-tree;
       Qt = {(w1,δ1),...,(wk,δk)}
Output: A list of similar-keyword sets
        C = C1,...,Ck;
 1 C ← {∅,∅,...,∅} // k empty sets
 2 return C
```

Algorithm 3: ProcSnode

```
Input: S-Node n;
       Qt = {(w1,δ1),...,(wk,δk)};
       Similar-keyword sets C = C1,...,Ck;
       Stack S;
 Output: Stack S;
 1 S.push (n,C);
 2 return S
```

Algorithm 4: ProcSANode

```
Input: SA-Node n;
       Qt = {(w1,δ1),...(wk,δk)};
        Similar-keyword sets C = C1,...,Ck;
        Stack S;
       Output: Stack S;
       1 for i=1 to k do
       2 Ci ← n.GetSimKwds(wi, δi);
        3 end
        4 if all Ci's = ∅ then
       5 S.push (n,C);
       6 end
       7 return S
```

Algorithm 5: ProcSKNode

```
Input : SK-Node n;
       Qt = {(w1,δ1),...(wk,δk)};
       Similar-keyword sets C = C1,...,Ck;
       Stack S;
       Output: Stack S;
       1 for i=1 to k do
       2 Gi ← n.keywords ∩ Ci;
       3 end
      4 if all Gi's = ∅
      then 5 G ← G1,G2,...,Gk;
```

```
6 S.push (n,G);
7 end
   8   return S
```

II. Demonstration Description

The system provides an interface similar to existing local-search on maps. The interface has a map and two input boxes, one for textual keywords and one for a location. The map is using the Google Maps API, and can display the search results for a query. We also use the Google Maps Geocoder API to obtain the latitude and longitude of the entered location. When a user clicks the search button, the objects satisfying the query conditions will be shown as red markers on the map. There are options to zoom in and out. The below screen shots are taken from our project.

Fig.1 A screen shot of our system with a mistyped keyword "Calagaeri city".

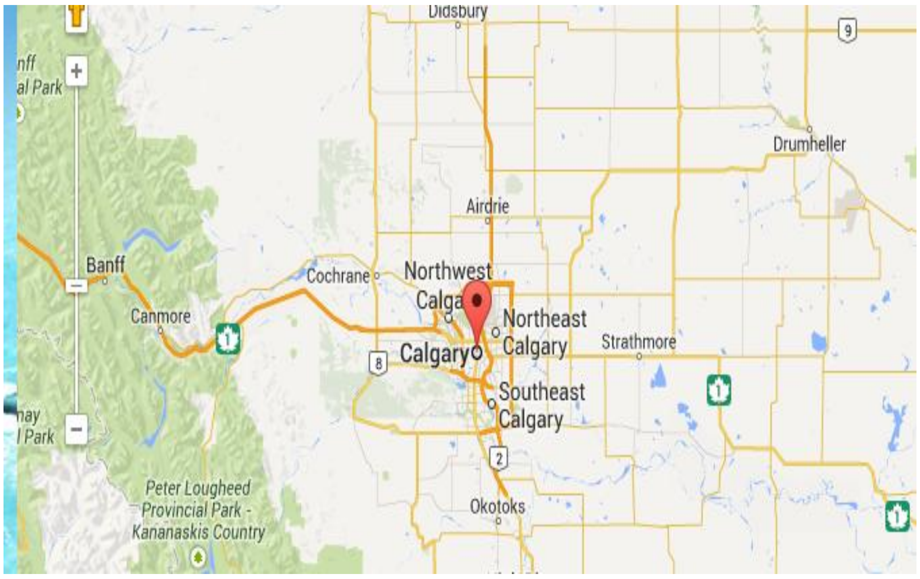


Fig. 2 Location displayed even though there were errors in the keyword for Chochi restaurant

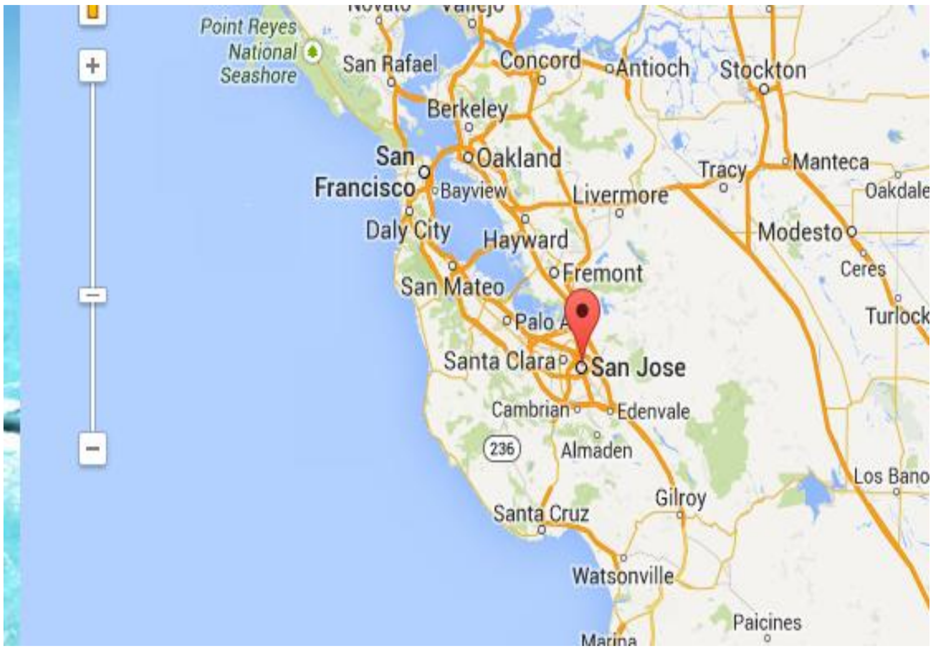
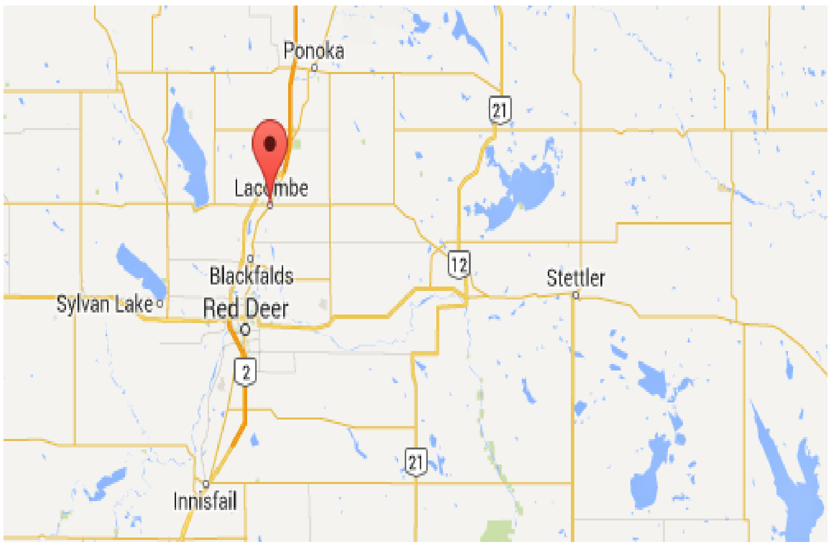


Fig.3 A screen shot of our system with a mistyped keyword ”Lacoomeee city”.



FFig.4 A screen shot of our system with a mistyped keyword”Tibessa city”.

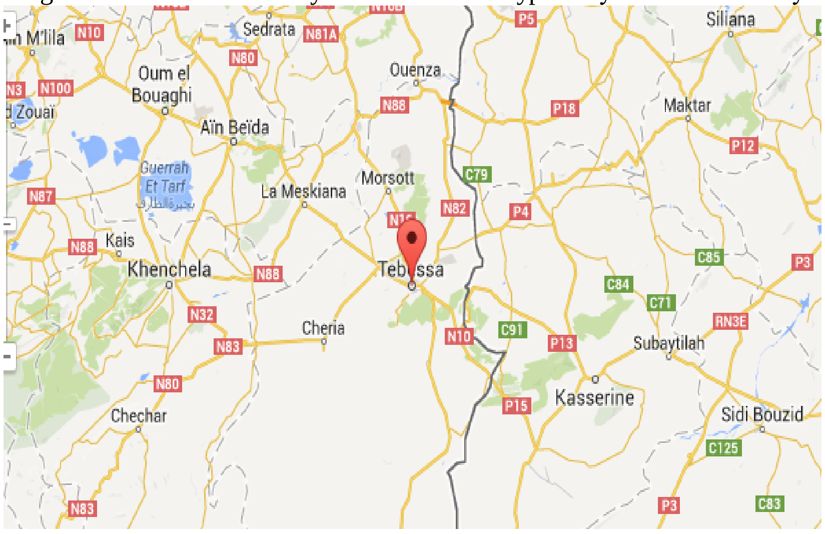
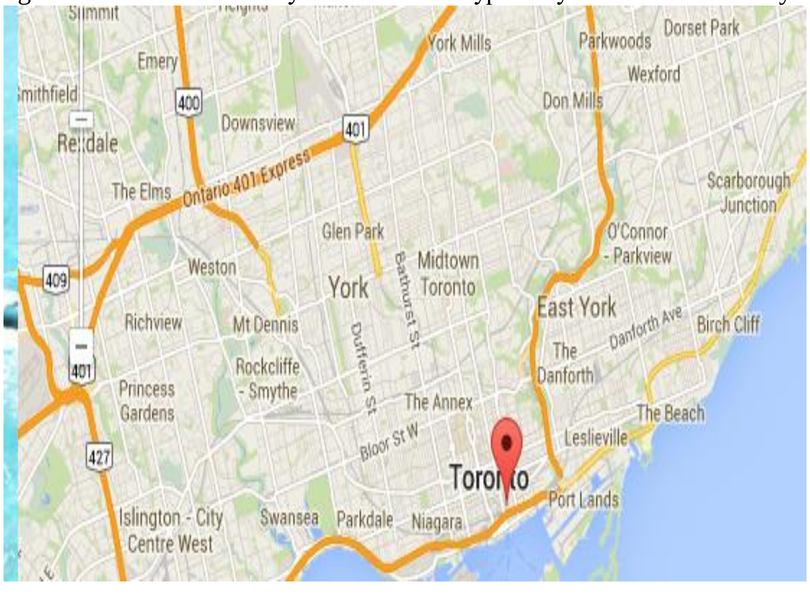


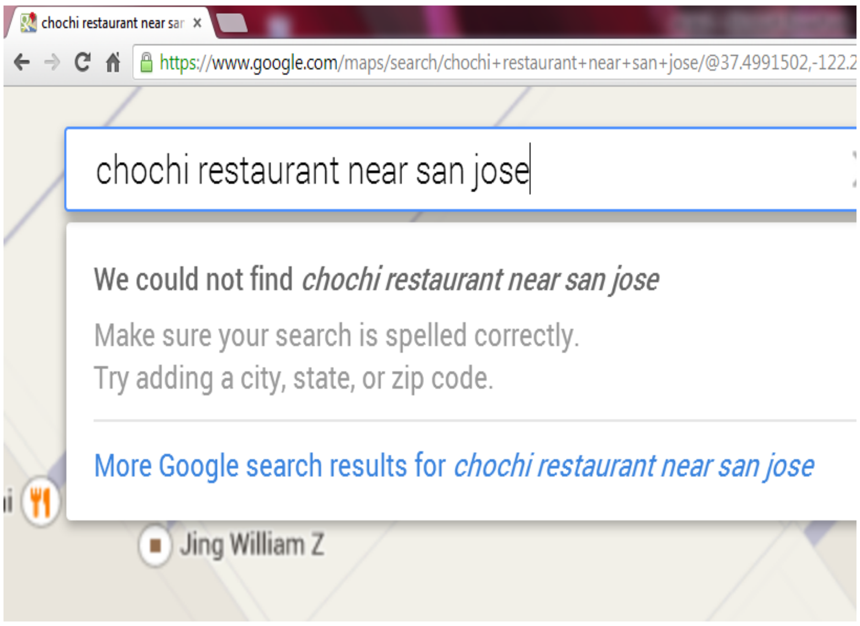
Fig.5 A screen shot of our system with a mistyped keyword ”Nunavote city”.



a.3. Comparison

Our system was compared with other several search engines. It is proved from the Figures 1,2,3,4 & 5 that the problem of approximate string search has been solved by 80 %. When other location based search engines were unable to retrieve approximate results when a user entered a mis spelt keyword,our system retrieved accurate results even though the user entered a wrong keyword.

Fig.6 A screen shot of the search engine Google map retrieving with a mistyped keyword "Chochi restaurant near san jose".



chochi restaurant near san jose

https://www.google.com/maps/search/chochi+restaurant+near+san+jose/@37.4991502,-122.2

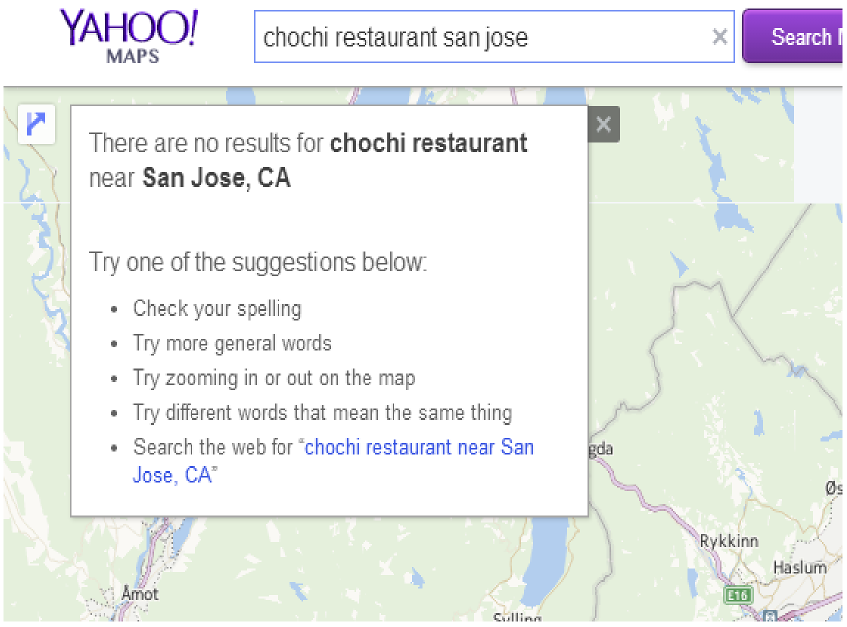
chochi restaurant near san jose

We could not find *chochi restaurant near san jose*

Make sure your search is spelled correctly.
Try adding a city, state, or zip code.

More Google search results for *chochi restaurant near san jose*

Fig. 7. A screen shot of the search engine Yahoo map retrieving the keyword "Chochi Restaurant near San Jose"



chochi restaurant san jose

There are no results for **chochi restaurant** near **San Jose, CA**

Try one of the suggestions below:

- Check your spelling
- Try more general words
- Try zooming in or out on the map
- Try different words that mean the same thing
- Search the web for "chochi restaurant near San Jose, CA"

The above screen shots show that still several search engines lack approximate string search. The figure 2. Shows the result of my system giving the exact result for the keyword "chochi restaurant san jose". Thus it is proved from the above results that our system provides approximate results even though the user has mistyped or misspelt the keyword.This proves that our approach works better when compared to other systems.

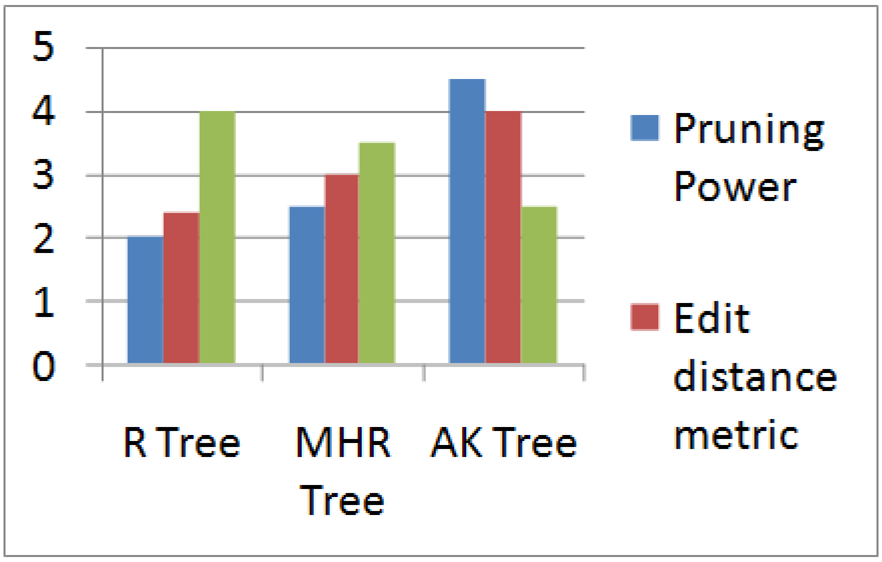


Fig. 8. Comparison with various tree and their metric results

The above graph describes that R tree needs a higher IO and CPU cost whereas it has been proved to be low in AK Tree. The edit distance metric for R tree and MHR tree is less when compared to AK tree and this proves that AK tree supports better pruning than any other tree.

III. Conclusion

Thus the use of AK tree reduces the query time. Unnecessary CPU and IO overhead is completely reduced with the use of R * tree. The usage of R * tree incurs a minimum query cost. A combined approach that satisfies both string match predicate and spatial predicate solved the problem of pruning resulting in an accurate string search. Comparisons from other system prove how efficient our system works.

REFERENCES

[1] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. In SIGMOD, pages 13–24, 1999.
[2] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The R∗- tree: an efficient and robust access method for points and rectangles. In SIGMOD, pages 322–331, 1990.
[3] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min- wise independent permutations (extended abstract). In STOC, pages 327–336, 1998.
[4] G. Navarro. A guided tour to approximate string matching. ACM Comput. Surv., 33:31–88, 2001.
[5] Hariharan, R., Hore, B., Li, C., Mehrotra, S.: Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In: SSDBM, p. 16. (2007)
[6] Li, C., Lu, J., Lu, Y.: Efficient merging and filtering algorithms for approximate string searches. In ICDE, pp. 257–266. (2008)
[7] Felipe, I. D., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In ICDE, pp. 656–665. (2008)
[8] Yao, B., Li, F., Hadjieleftheriou, M., Hou, K.: Approximate string search in spatial databases. In ICDE. (2010)
[9] A. Guttman. R-trees: A dynamic index structure for spatial searching. In B. Yormark, editor, SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984, pp. 47–57. ACM Press, 1984.
[10] K. S. McCurley. Geospatial mapping and navigation of the web. In WWW, pp. 221–229, 2001.
[11] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In ICDE, pp. 688–699, 2009.
[12] D. Zhang, B. C. Ooi, and A. K. H. Tung. Locating mapped resources in web 2.0. In ICDE, pp. 521–532, 2010.
[13] J. Zobel and A. Moffat. Inverted files for text search engines. ACM Comput. Surv., 38(2):6, 2006.
[14] E. Ukkonen. Approximate string-matching with q-grams and maximal matches. Theor. Comput. Sci., 92(1):191–211, 1992.
[15] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In ICDE, pages 688–699, 2009.