Secured Diversified Data set Selection From Dynamic Data

K. Suresh¹, Ch. Nanda Krishna²
¹Student, M.Tech, Vr Siddartha Engineering College, Kanuru, Vijayawada
²Assistant Professor, M.Tech, Vr. Siddartha Engineering College, Kanuru, Vijayawada

Abstract

Once the data is created, the data that it contains doesn’t change is known as static data (ex: newspaper). Where as the data can be change and update in the dynamic data (ex: website). A lot of information available online creates the need for developing methods towards selecting and presenting to users representative result sets. Diversification has recently attracted considerable attention as a means of increasing user satisfaction in recommender systems, as well as in web and database search. In this paper on the problem of selecting the k-most diverse items from a result set. Whereas previous research has mainly considered the static version of the problem, in this paper, exploit the dynamic case in which the result set changes over time, as for example, in the case of notification services.

The proposed approach is based on cover trees and supports dynamic item insertion and deletion and provides theoretical bounds that characterize the quality of cover tree solution with respect to the optimal one. Since results are often associated with a relevance score, to extend the proposed approach to account for relevance.

Keywords: Indexing methods, selection process, information filtering, search process, similarity measures.

INTRODUCTION

Dynamic data or transactional data denotes information that is asynchronously changed as further updates to the information become available. The opposite of this is persistent data, which is data that is infrequently accessed and not likely to be modified. Dynamic data is also different from streaming data, in that there is no constant flow of information. Rather, updates may come at any time, with periods of inactivity in between. Difference between static and dynamic data is that once static data is created, the data that it contains doesn’t change, whereas the data can change and update in dynamic data [1]. An example of static data, is a newspaper, as once it has been printed, the information on it cannot be updated, whereas an example of dynamic data, would be a website, as that can be updated as and when needed.

Websites are classified as dynamic sources of data, because the information on the website can be constantly updated, for example if a sports website shows live football scores, it can be updated as and when goals have been scored thereby providing almost real-time information [4]. Another reason why websites are classed as dynamic sources of data, is because users can interact with the site, e.g. If a website is that of football clubs, then a user can request new information about ticket availability for an upcoming game.

A lot of information available online creates the need for developing methods towards selecting and presenting to users representative result sets.

To this end, result diversification has attracted considerable attention as a means of increasing user satisfaction. Result diversification takes many forms including selecting items so that their content dissimilarity, novelty or topic coverage is maximized [1].

Most previous approaches to computing diverse sets rely on greedy or interchange heuristics. Greedy heuristics build a diverse set incrementally, selecting one item at a time so that some diversity measure is maximized, whereas interchange heuristics start from a random initial set and try to improve it.
Despite the considerable interest in diversification, most previous research considers the static version of the problem, i.e., the available items out of which a diverse set is selected do not change over time. In this paper, mainly focus on the dynamic diversification problem, where insertions and deletions of items are allowed and the diverse set needs to be refreshed to reflect such updates [12]. The dynamic problem was address using a greedy heuristic and in using an interchange heuristic. Here, the proposed approach is an index-based approach.

The proposed solution is based on cover trees. Cover trees are data structures originally proposed for approximate nearest-neighbor search [11]. They were recently used to compute medoids and priority medoids. An index based approach was also followed for the static version of the problem. The proposed index exploited a Dewey-encoding tree and can be used only with a specific diversity function on structured data. The proposed approach is more general and can be used with any diversity function.

Motivated by popular proactive delivery paradigms, such as news alerts, RSS feeds and notification services in social networks, where users specify their interests and receive relevant notifications, we also consider the continuous version of the problem, where diversified sets are computed over streams of items.

To avoid overwhelming users by forwarding to them all relevant items, we consider the case in which a representative diverse set is computed, instead, whose size can be configured by the users.

The proposed incremental algorithms produce results of quality comparable to that achieved by re-applying the greedy heuristic to re-compute a diverse set, while avoiding the cost of re-computation [11]. Using cover trees also allows the efficient enforcement of the continuity requirements. Furthermore, multiple queries with different values of $k$ can be supported.

In many cases, the items in the result set are associated with a relevance rank. To extended the proposed approach to support the computation of diverse subsets of ranked sets of items [5]. First show how to incorporate relevance in the diversity function used to build the cover tree. In addition, to allow for dynamically tuning the relative importance of relevance and diversity, introduce an alternative solution based on weighted cover trees along with appropriate algorithms [6]. Finally, note that a recent line of research focuses on combining relevance and diversity by viewing diversification as a top-$k$ problem. In such cases, threshold algorithms are used for selecting diverse items aiming at pruning a portion of the candidate items. Such approaches assume the existence of indices to provide sorted access to items, e.g., based on relevance or their distance from a given item in this paper aim at constructing indices that will guide the selection process.

- Propose indexing based on cover trees to address the dynamic diversification problem along with continuity requirements appropriate for a streaming scenario,
- Present a suite of methods with varying complexity that exploit the cover tree for the MAXMIN problem and provide bounds for the achieved diversity with regards to the optimal solution,
- Extend the cover tree and algorithms for selecting items that are both relevant and diverse and
- Experimentally evaluate the efficiency and effectiveness of proposed approach using both real and synthetic datasets

**PROPOSED METHOD**

2. 1 Index- Based Diversification

To compute diverse sets in a dynamic setting, we rely on a tree structure, called cover tree, to index the items in $P$. In this section provide a formal definition of the cover tree along with algorithms for constructing cover trees appropriate for the diversification problem. Online shopping sometimes offer good payment plans and options for customers. Customers can decide their payment date and amount in their own preference and convenience.
2.2 Cover Tree

A Cover Tree (CT) [8] for a set P is a leveled tree where each level is associated with an integer l which increases as we move up the tree. Each node in the tree is associated with exactly one item p ∈ P, while each item may be associated with multiple nodes, but with at most one at each level. In the following, when clear from context, to use p to refer to both the item p and the node associated with p at a specific level.

Let C_l be the set of items at level l and \( \ell_{\text{min}} \) and \( \ell_{\text{max}} \) be respectively the levels of the root of the tree and the leaves. A cover tree of base b, b > 1, is a tree that obeys the following invariants

1. **Nesting:** For all levels l, \( \ell_{\text{min}} < l \leq \ell_{\text{max}} \), C_l ⊆ C_{l+1}, i.e., once an item p appears in the tree at some level, then there is a node associated with p at every lower level.

2. **Separation:** For all levels l, \( \ell_{\text{min}} \leq l \leq \ell_{\text{max}} \), and all distinct p_i, p_j ∈ C_l, it holds that, \( d(p_i, p_j) > b^l \).

3. **Covering:** For all levels l, \( \ell_{\text{min}} \leq l < \ell_{\text{max}} \) and all \( p_i \in C_{l+1} \), there exists a \( p_j \in C_l \), such that, \( d(p_i, p_j) \leq b^{l+1} \) and the node associated with \( p_j \) is the parent of the node associated with \( p_i \).

The CT was originally proposed with base b = 2. In this paper, we use a more general base b, b > 1. Generally, larger base values result in shorter and wider trees, since fewer nodes are able to “cover” the nodes beneath them. The value of b determines the granularity with which we move from one level to the next, i.e., how many more items become visible as we descend the tree [13]. Due to the CT invariants, if an item p appears for the first time at level l of the tree, then p is a child of itself at all levels below l. This observation provides us with a more space-efficient representation of the CT achieved by coalescing all nodes whose only child is a self child. We call this representation explicit. The explicit representation of a CT for a set P with n items requires \( O(n) \) space [8]. Although we use an explicit representation in our experiments, for ease of presentation, we shall use the full implicit representation when describing the algorithms. Next, we first present an algorithm for batch constructing a CT tailored for the MAXMIN problem. Then, consider an incremental construction of a CT appropriate for dynamic environments.

2.3 Cover Tree Construction

Given a set P of items, we build an appropriate CT for P using a bottom-up approach. First, we construct the lowest level that includes all items in P [16]. Then, given a level \( \ell \) to build the next level \( \ell + 1 \), for select items from level \( \ell \) whose distance is larger than \( b^{\ell+1} \) (so that the separation invariant is maintained), as long as such items exist. To construct a CT whose items at each level are as far apart from each other as possible, follow a greedy approach in selecting which items from C_l to include in C_{l+1}. 

Figure 2: Example of the top 10 levels of a cover tree for a set of items in the 2-dimensional Euclidean space. Bold points represent the items (i.e., nodes) at each level, moving from lower levels to the root level, move from left to right [2].
Figure 4: Complete block diagram of cover tree construction

Specifically, to start by selecting the two items in $C_\ell$ that are the farthest apart from each other and continue by selecting the item that has the largest minimum distance from the items already selected [10]. The remaining items at $C_\ell$ are assigned a parent node from $C_{\ell+1}$ so that the covering invariant holds. To reduce the overlap among the areas covered by sibling nodes, we assign each node to its closest candidate parent. We call this step nearest parent heuristic. Clearly, from the way the tree is constructed, $C_{\ell+1} \subseteq C_\ell$, thus the nesting invariant also holds. Call the tree constructed using this procedure, Batch Cover Tree (BCT) [1].

Algorithm:

Step 1: level $l = \log b(\min(b(p,q))$
Step 2: $T.C_1 = \text{NULL}$ ($C_1$ is a position of tree $T$)
Step 3: then for all $p$ is belongs to the set $P$ then
Step 4: $T.C_1 = T.C_1 \cup \{p\}$
Step 5: end for
Step 6: while $T.C_1 > 1$ then
Step 7: then $T.C_{l+1} = \text{NULL}$
Step 8: then candidates (items) = $T.C_1$
Step 9: $P_*,q_* = \max (d(p,q))$
Step 10: $T.C_{l+1} = T.C_{l+1} \cup \{p_*,q_*\}$
Step 11: Candidates = Candidates\{p_*,q_*\}
Step 12: While Candidates = NULL then
Step 13: Candidates = Candidates\{p: \exists q \in T.C_{l+1} with d(p,q) \leq b^{l+1}\}
Step 14: $p_* = \max (d(p_*, T.C_{l+1}))$
Step 15: $T.C_{l+1} = T.C_{l+1} \cup \{p_*\}$
Step 16: Candidates = Candidates\{p_*,q_*\}
Step 17: end while
Step 18: for all $p \in T.C_l$
Step 19: $q_* = \min(d(p,T.C_{l+1}))$
Step 20: make $q$ parent of $p$
Step 21: end for
Step 22: $T.C_* \leftarrow T.C_{l+1}$
Step 23: $l \leftarrow l+1$
Step 24: end while
Step 25: return $T$

RESULTS:

Initially run the main project using java programming in order to open the main window. In this housing detail the contents are admin, register, apartment details, selection process, user needs, and searching process. In this home screen for both user and admin it comprises with admin login and user login. In this if you click on the admin, give the user name and password for both user login and admin login. If you click on the register button a form will be displaying with all details of new user registration.

After click on the registration the page will be displayed. The registration form can contains the some fields like name, age, address, contact, username and password (ex: name: suresh, age: 24, address: Guntur, contact: 9989447460, username: suresh, password: 12345). To fill all the details of user registration or to insert the data in table registration form. After giving the data click on the register button the data will be saved in the apartment database.
Whenever click on the admin button in the home screen the page will be displayed like this. This login screen for both user and admin and selects the data in table registration from and login of admin apartment database. It contains the username and password for both user login and admin login. If a user (already registered user) login, give the username and password of user and then click on the user login. Admin can be login with their user name and password and then click on the admin login.

Executes the Query:
```
select * from register where username='" + username + '" and password='" + password + '"
```

Whenever the admin or user login and click on selection process the above screenshot will be displayed. It can displays the user needs and apartment pictures based on selected by the user. Housing details can be shown like his way type, location, facility and availability. In this admin can use delete query for deleting the already saved data. An apartment picture shows the different types of apartment images like duplex, semi duplex, classic six and triplex etc. Admin can click on the display button all the saved records can be displayed.

**CONCLUSION:**
Result diversification has attracted considerable attention. However, most current research addresses the static version of the problem. In this paper, I studied the diversification problem in a dynamic setting where the items to be diversified change over time. The proposed method is an index-based approach that allows the incremental evaluation of the diversified sets to reflect item updates. The proposed solution is based on cover trees. Here the proposed Greedy Cover Tree Algorithm for data set selection, updation and insertion with incremental insertion and deletion updates is observed and the results are shown with a secure motive.

**REFERENCES:**


streaming-based approach,” in *Proc. 34th Int. SIGIR*, Beijing, China, 2011.


