Diverse Set Selection over Dynamic Data

Marina Drosou and Evaggelia Pitoura

Abstract—Result 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, we focus 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, we exploit the dynamic case in which the result set changes over time, as for example, in the case of notification services. We define the CONTINUOUS k-DIVERSITY PROBLEM along with appropriate constraints that enforce continuity requirements on the diversified results. Our proposed approach is based on cover trees and supports dynamic item insertion and deletion. The diversification problem is in general NP-hard; we provide theoretical bounds that characterize the quality of our cover tree solution with respect to the optimal one. Since results are often associated with a relevance score, we extend our approach to account for relevance. Finally, we report experimental results concerning the efficiency and effectiveness of our approach on a variety of real and synthetic datasets.

Index Terms—Indexing methods. Selection process, Information filtering. Search process. Similarity measures.

1 INTRODUCTION

The abundance 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 [14].

Most previous approaches to computing diverse sets rely on *greedy* or *interchange* heuristics. Greedy heuristics (e.g., [31], [20]) build a diverse set incrementally, selecting one item at a time so that some diversity measure is maximized, whereas interchange heuristics (e.g., [30], [24]) 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, we 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. The dynamic problem was addressed in [13] using a greedy heuristic and in [25] using an interchange heuristic. Here, we propose an index-based approach.

Our solution is based on cover trees. Cover trees are data structures originally proposed for approximate nearest-neighbor search [8]. They were recently used to compute medoids [23] and priority medoids [9]. An index-based approach was also followed in [28] 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. Our approach is more general and can be used with any diversity function.

• *M. Drosou and E. Pitoura are with the Computer Science Department, University of Ioannina, Greece.*
E-mail: {mdrosou, pitoura}@cs.uoi.gr

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. We introduce a sliding window model along with *continuity* requirements. Such requirements ensure that the order in which the diverse items are delivered follows the order of their generation and that an item does not appear, disappear and then re-appear in the diverse set.

We focus on the MAXMIN diversity problem defined as the problem of selecting k out of a set of n items so that the minimum distance between any two of the selected items is maximized. The MAXMIN problem is known to be NP-hard [17]. We propose a suite of algorithms that exploit the cover tree to provide solutions with varying accuracy and complexity. We provide theoretical results that bound the accuracy of the solutions achieved with regards to the optimal solution. The most efficient algorithm achieves a $^{b-1}/_{2b^2}$-approximation of the optimal solution, where b is the base of the cover tree, whereas the most expensive algorithm, a pruned implementation of a greedy heuristic, a $^1/_2$-approximation. The most efficient algorithms just select items from a single level of the tree whose size is close to k and thus they achieve scalability with n.

Our 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. 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. We have extended our approach to support the computation of diverse subsets of ranked sets of items. We 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, we introduce an alternative solution based on weighted cover trees along with appropriate algorithms.

Finally, note that a recent line of research focuses on combining relevance and diversity by viewing diversification as a top-k problem ([7], [21], [19]). 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, instead, we aim at constructing indices that will guide the selection process.

In a nutshell, in this paper, we:

- 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 our algorithms for selecting items that are both relevant and diverse and
- experimentally evaluate the efficiency and effectiveness of our approach using both real and synthetic datasets.

This paper extends our previous work [15] with new diversification algorithms, among which an efficient implementation of the greedy heuristic that uses the properties of the cover tree to prune the search space. Also, we extend our approach to combine diversity with relevance.

The rest of this paper is structured as follows. In Section 2, we present our diversification framework and define the CONTINUOUS k-DIVERSITY PROBLEM. Section 3 presents the cover tree index structure, while Section 4 introduces algorithms for computing diverse items. Section 5 considers combining diversity with relevance. Section 6 presents our experimental results. In Section 7, we present related work, while in Section 8 a summary of the paper.

2 THE DIVERSIFICATION MODEL

There are many definitions of diversity. In this paper, we focus on a general form of diversification based on content dissimilarity. Next, we first provide a formal definition of the diversity problem and then introduce its continuous variation.

2.1 The *k*-Diversity Problem

Let $P = \{p_1, \dots, p_n\}$ be a set of n items. Given a distance metric $d : P \times P \rightarrow \mathbb{R}^+$ indicating the dissimilarity of two items in P, assume that the *diversity* of a set S, $S \subseteq P$, is measured by a function $f : 2^{|P|} \times d \rightarrow \mathbb{R}^+$. For a positive integer k, $k \leq n$, the k-DIVERSITY PROBLEM is the problem of selecting a subset S^* of P such that:

$$S^* = \operatorname{argmax}_{\substack{S \subseteq P \\ |S| = k}} f(S, d).$$

The choice of f affects the selection of items, even for a specific distance metric d. Two widely used functions are the *minimum* distance among the selected items and the *sum* of the distances of the selected items, formally defined as:

$$f_{\text{MIN}}(S, d) = \min_{\substack{p_i, p_j \in S \\ p_i \neq p_j}} d(p_i, p_j), \text{ and}$$

$$f_{\text{SUM}}(S, d) = \sum_{\substack{p_i, p_j \in S \\ p_i \neq p_j}} d(p_i, p_j).$$

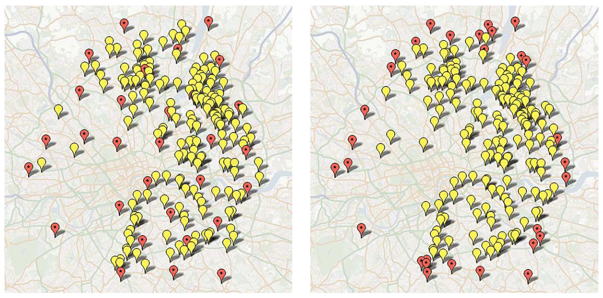

(a) MAXMIN. (b) MAXSUM.

Fig. 1: MAXMIN vs. MAXSUM for $n = 200$ and $k = 30$. Diverse items are marked with a darker (red) color.

The corresponding problems are called MAXMIN and MAXSUM. Intuitively, MAXMIN aims at discouraging the selection of nearby items, while MAXSUM at increasing the average pairwise distance among all items. An example is shown in Figure 1, which depicts the $k = 30$ most diverse, in terms of geographical distance, apartments for sale from a set of $n = 200$ available apartments in the London area retrieved from [5]. In general, MAXSUM tends to select items in the outskirts of the set P, whereas MAXMIN selects items that are more representative of P in the sense that they provide a better coverage of it. In the rest of this paper, we will focus on the MAXMIN problem that exhibits this desired property.

The MAXMIN Greedy Heuristic. The k-DIVERSITY PROBLEM is known to be NP-hard [17]. Various heuristics have been proposed, among which a natural greedy heuristic (Algorithm 1) has been shown experimentally to outperform the others in most cases ([13], [18]). The algorithm starts by selecting either a random item or the two items in P that are the furthest apart (line 1). Then, it continues by selecting the items that have the maximum distance from the items already selected, where the distance of an item p_i from a set of items S is defined as:

$$d(p_i, S) = \min_{p_j \in S} d(p_i, p_j).$$

It has been shown (e.g., in [27]) that the minimum distance of the set produced by the greedy heuristic is a $^1/_2$-approximation of the minimum distance of the optimal solution and that no polynomial algorithm can provide a better guarantee.

Algorithm 1 Greedy Heuristic.

Input: A set of items P, an integer k.
Output: A set S with the k most diverse items of P.

```
1: $p^*, q^* \leftarrow \operatorname{argmax}_{p,q \in P, p \neq q} d(p, q)$
2: $S \leftarrow \{p^*, q^*\}$
3: while $|S| < k$ do
4:     $p^* \leftarrow \operatorname{argmax}_{p \in P} d(p, S)$
5:     $S \leftarrow S \cup \{p^*\}$
6: end while
7: return $S$
```

2.2 The Continuous *k*-Diversity Problem

We consider the case in which the set P changes over time and we want to refresh the computed k most diverse items to represent the updated set. In general, the insertion (or deletion) of even a single item may result in a completely different diverse set. The following simple example demonstrates this. Consider the set $P = \{(4,4),(3,3),(5,6),(1,7)\}$ of points in the 2-dimensional Euclidean space and $k = 2$. The two most diverse items of P are $(4,4)$ and $(1,7)$. Assume that $(0,0)$ is added to P. Now, the two most diverse items of P are $(0,0)$ and $(5,6)$.

In many applications, new items are generated in a continuous manner and, thus, the set P changes gradually over time. For example, consider a user continuously receiving a representative, i.e., most diverse, subset of the stream of available apartments in her area. We would like to offer her a continuous view of the most diverse items in this stream.

We adopt a sliding-window model where the k most diverse items are computed over sliding windows of length w in the input stream. The window length may be defined either in time units (e.g., "the k most diverse items in the last hour"), or in number of items (e.g., "the k most diverse items among the 100 most recent items"). Without loss of generality, we assume item-based windows. We allow windows to not only slide but also jump, i.e., move forward more than one item. For windows of length w and a jump step of length h, $h \leq w$, consequent windows share $w - h$ common items (Figure 2). Two consequent jumping windows may correspond, for example, to the items seen by a user in two consequent visits to her RSS reader application. Between these two visits, some items have ceased to be valid, new items have been generated, while a number of older items remain valid. Note that, for $h = 1$, jumping windows behave as regular sliding windows, while for $h = w$, consequent windows are disjoint which corresponds to periodic behavior with a period of length w.

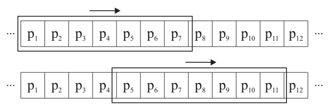
Formally, let $\mathcal{P}$ be a stream of items. We denote the i^{th} jumping window of $\mathcal{P}$ as P_i. The UNCONSTRAINED CONTINUOUS k-DIVERSITY PROBLEM is the problem of selecting a subset S_i^* of P_i for each P_i, such that:

$$S_i^* = \operatorname{argmax}_{\substack{S_i \subseteq P_i \\ |S_i| = k}} f(S_i, d).$$

Constrained Continuous *k*-Diversity Problem. Since users may expect some continuity in the diverse sets they receive in consequent retrievals, we consider additional requirements on how the diverse sets change over time.

First, we want to avoid cases where diverse items that are still valid disappear. This may lead to confusing results, where an item appears in one diverse set, disappears in the next one and then appears again. Thus, an item selected as diverse remains in the diverse set until it expires, i.e., exits the current window. The diverse set is complemented with new items that are diverse with regards to those previously selected diverse items that are still valid. For instance, in the apartments example, the user sees new items that are diverse with regards to other previously seen apartments that are still available. We call this the *durability* requirement.

Second, we want the order in which items are chosen as diverse to follow the order of their appearance in the stream.



p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11} p_{12}
p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10} p_{11} p_{12}

Fig. 2: Two consequent windows with $w = 7$ and $h = 4$.

This means that, once an item p is selected as diverse, we cannot later on select an item older than p. We call this the *freshness* requirement. This is a desirable property in case of notification services, such as news alerts and RSS feeds, so as to ensure that the diverse items selected to be delivered to the users follow the chronological order of their publication. Raising this requirement may result in out-of-order delivery which may seem unnatural to users.

Based on the above observations, we now formally define the CONSTRAINED CONTINUOUS k-DIVERSITY PROBLEM.

Definition 1: (CONSTRAINED CONTINUOUS k-DIVERSITY PROBLEM). Let $\mathcal{P}$ be a stream of items, P_{i-1}, P_i, $i > 1$, be any two consequent windows and S_{i-1}^* be the diverse subset of P_{i-1}. The CONSTRAINED CONTINUOUS k-DIVERSITY PROBLEM is the problem of selecting a subset S_i^* of P_i, $\forall i$, such that:

$$S_i^* = \operatorname{argmax}_{\substack{S_i \subseteq P_i \\ |S_i| = k}} f(S_i, d)$$

and the following two constraints are satisfied:

(i) $p_j \in (S_{i-1}^* \cap P_i) \Rightarrow p_j \in S_i^*$ (durability requirement),

(ii) Let p_l be the newest item in S_{i-1}^*. Then, $\nexists p_j \in P_i \backslash S_{i-1}^*$ with $j < l$, such that, $p_j \in S_i^*$ (freshness requirement).

3 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, we provide a formal definition of the cover tree along with algorithms for constructing cover trees appropriate for the diversification problem.

3.1 The 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 $p \in P$, while each item may be associated with multiple nodes, but with at most one at each level. In the following, when clear from context, we 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 ℓ_{max} and ℓ_{min} be respectively the levels of the root 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, $\ell_{min} < \ell \leq \ell_{max}$, $C_\ell \subseteq C_{\ell-1}$, 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, $\ell_{min} \leq \ell \leq \ell_{max}$, and all distinct $p_i, p_j \in C_\ell$, it holds that, $d(p_i, p_j) > b^\ell$.
3. **Covering**: For all levels ℓ, $\ell_{min} \leq \ell < \ell_{max}$ and all $p_i \in C_\ell$, there exists a $p_j \in C_{\ell+1}$, such that, $d(p_i, p_j) \leq b^{\ell+1}$ and the node associated with p_j is the parent of the node associated with p_i.

An example is shown in Figures 3 and 4.

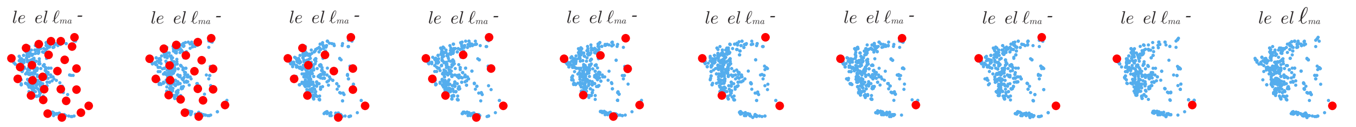


Fig. 3: An example of the top 10 levels of a cover tree for a set of items in the 2-dimensional Euclidean space. Bold points represent the items (i.e., nodes) at each level, moving from lower levels to the root level, as we move from left to right.

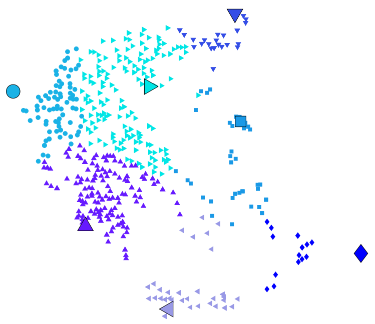


Fig. 4: The $(\ell_{max} - 5)^{\text{th}}$ level of the cover tree of Figure 3. The items (i.e., nodes) of the level are drawn with large bold symbols, while the items of lower levels covered by these nodes are drawn with the same symbol (and color) as their ancestor.

Algorithm 2 Batch Cover Tree Construction.

Input: A set of items P, a base b.
Output: A cover tree T of base b for P.

```
1: $\ell \leftarrow \lfloor \log_b (\min_{p,q \in P} d(p,q)) \rfloor$
2: $T.C_\ell \leftarrow \emptyset$
3: for all $p \in P$ do
4:     $T.C_\ell \leftarrow T.C_\ell \cup \{p\}$
5: end for
6: while $|T.C_\ell| > 1$ do
7:     $T.C_{\ell+1} \leftarrow \emptyset$
8:     $candidates \leftarrow T.C_\ell$
9:     $p^*, q^* \leftarrow \text{argmax}_{p,q \in candidates}\, d(p,q)$
10:    $T.C_{\ell+1} \leftarrow T.C_{\ell+1} \cup \{p^*, q^*\}$
11:    $candidates \leftarrow candidates \backslash \{p^*, q^*\}$
12:    while $candidates \neq \emptyset$ do
13:        $candidates \leftarrow candidates \backslash \{p : \exists q \in T.C_{\ell+1} \text{ with } d(p,q) \leq b^{\ell+1}\}$
14:        $p^* \leftarrow \text{argmax}_{p \in candidates}\, d(p, T.C_{\ell+1})$
15:        $T.C_{\ell+1} \leftarrow T.C_{\ell+1} \cup \{p^*\}$
16:        $candidates \leftarrow candidates \backslash \{p^*\}$
17:    end while
18:    for all $p \in T.C_\ell$ do
19:        $q^* \leftarrow \text{argmin}_{q \in T.C_{\ell+1}}\, d(p, T.C_{\ell+1})$
20:        make $q$ parent of $p$
21:    end for
22:    $T.C_\ell \leftarrow T.C_{\ell+1}$
23:    $\ell \leftarrow \ell + 1$
24: end while
25: return $T$
```

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.

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, we consider an incremental construction of a CT appropriate for dynamic environments.

3.2 Batch Construction

Given a set P of items, we build an appropriate CT for P using a bottom-up approach as depicted in Algorithm 2. First, we construct the lowest level that includes all items in P (lines 1-5). Then, given a level ℓ to build the next level $\ell + 1$, we select items from level ℓ whose distance is larger than $b^{\ell+1}$ (so that the separation invariant is maintained), as long as such items exist (lines 6-17). To construct a CT whose items at each level are as far apart from each other as possible, we follow a greedy approach in selecting which items from C_ℓ to include in $C_{\ell+1}$. Specifically, we start by selecting the two items in C_ℓ that are the farthest apart from each other (line 9) and continue by selecting the item that has the largest minimum distance from the items already selected (lines 12-17).

The remaining items at C_ℓ are assigned a parent node from $C_{\ell+1}$ so that the covering invariant holds (lines 18-21). To reduce the overlap among the areas covered by sibling nodes, we assign each node to its closest candidate parent (line 19). 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. We call the tree constructed using this procedure, *Batch Cover Tree* (*BCT*).

We shall prove that the set of items C_ℓ at each level ℓ of the BCT correspond to the result of applying the MaxMin greedy heuristic (Algorithm 1) on P, for $k = |C_\ell|$. Our proof uses the following observation. Let $S^{GR}(P, k)$ denote the result of applying the MaxMin greedy heuristic on P for k.

Observation 1: For any $k > 2$, $S^{GR}(P, k + 1) \supset S^{GR}(P, k)$.

Theorem 1: Let P be a set of items and T be a BCT for P. For all levels ℓ, $\ell_{min} \leq \ell < \ell_{max}$, of T, it holds:

$$C_\ell = S^{GR}(P, |C_\ell|).$$

Proof: We shall prove the theorem by induction on the level ℓ. The theorem holds trivially for ℓ equal to the lowest level of the tree, since this level includes all items in P. Assume that it holds for level ℓ. We shall show that it also holds for level $\ell + 1$.

Consider the construction of level $\ell + 1$. From the induction step, it holds that, $C_\ell = S^{GR}(P, |C_\ell|)$. Let p be the *first* item in C_ℓ such that p is the best candidate, i.e., has the maximum minimum distance from the items already selected, but p cannot be moved to $C_{\ell+1}$ because p is covered by an item already selected to be in $C_{\ell+1}$. Let C', $C' \subset C_\ell$, be the set of items already selected to be included in $C_{\ell+1}$. This means that, it holds: $\min_{q' \in C'} d(p, q') \geq \min_{q' \in C'} d(p', q')$, for all $p' \in C_\ell \backslash C'$ (1) and, also, $\exists\ q \in C'$ such that $d(p, q) \leq b^{\ell+1}$ (2). From (1) and (2), we get that for all $p' \in C_\ell \backslash C'$, $\exists\ q \in C'$ such that $d(p', q) \leq b^{\ell+1}$, that is, all remaining items are already covered by items in C'.

Thus, p is the last item that is considered for inclusion in $C_{\ell+1}$, since all other remaining items in C_ℓ are already covered. Therefore, to construct $C_{\ell+1}$, the items from C_ℓ to be included in level $\ell+1$ are considered in the same order as in the greedy heuristic, until one item that violates the separation criterion (it is covered by the selected items) is encountered. When this happens the selection stops. By the induction step and Observation 1, this concludes the proof. □

Note that, we have made an implicit assumption that no ties occur when selecting items. In the absence of ties, both the greedy heuristic and the BCT construction algorithm select items deterministically. We can raise this assumption, by considering that if ties exist, these are resolved in a specific order that may vary depending on the nature of the items, for instance, by selecting the most recent among the items.

Regarding the complexity of Algorithm 2, computational steps are shared among levels. Each level $C_{\ell+1}$ is a subset of C_ℓ and, more specifically, it consists of the items of C_ℓ in the order in which they were inserted into C_ℓ up to the first item whose minimum distance from the already selected items of C_ℓ is smaller than $b^{\ell+1}$. Therefore, it suffices to perform these computational steps only once (at the lowest level) and just maintain the order in which each item was selected from the lowest level for inclusion in the next level. This gives us an $O(n^2)$ complexity.

As a final remark, another way to view the BCT is as caching the results of the greedy heuristic for all k and indexing them for efficient retrieval.

3.3 Dynamic Construction

In dynamic environments, it is not efficient to re-construct a BCT whenever an item is inserted or deleted. Thus, we construct a cover tree *incrementally* as new items arrive and old ones expire. We refer to such trees as *Incremental Cover Trees* (*ICTs*).

Incremental Insertion. To insert a new item p into a CT, we use the recursive insert procedure shown in Algorithm 3. It is based on the insertion algorithm in [8] and subsequent corrections in [22] that we have extended to work for any $b > 1$. `Insert` is called recursively starting from the root level, until a level is found at which p is separated from all other items (lines 2-4). Each time, `Insert` is called only with the nodes that cover p (line 5). When the first level such that p is separated from all other items is located, a node that covers p is selected as its parent (lines 8-9). To select a node as a parent for p, we use a *nearest parent* heuristic (as in the batch construction) and assign p to its closest candidate parent. The

Algorithm 3 Insert(p, $T.Q_\ell$, ℓ)

Input: An item p, a set of nodes $T.Q_\ell$ of a cover tree T at level ℓ.

```
1: $C \leftarrow \{children(q) : q \in T.Q_\ell\}$
2: if $d(p, C) > b^\ell$ then
3:     return true
4: else
5:     $T.Q_{\ell-1} \leftarrow \{q \in C : d(p, q) \leq \frac{b^\ell}{b-1}\}$
6:     $flag \leftarrow Insert(p, T.Q_{\ell-1}, \ell - 1)$
7:     if $flag$ and $d(p, T.Q_\ell) \leq b^\ell$ then
8:         $q^* \leftarrow \mathrm{argmin}_{q \in T.Q_\ell, d(p,q) \leq b^\ell} d(p, q)$
9:         make $p$ a child of $q^*$
10:        return false
11:    else
12:        return flag
13:    end if
14: end if
```

complexity of the algorithm depends on how many nodes of each level cover p.

Next, we prove the correctness of the insertion algorithm.

Algorithm 4 Delete($p, \{T.Q_\ell, T.Q_{\ell+1}, \ldots, T.Q_{\ell_{max}}\}, \ell$)

Input: An item p, sets of nodes $\{T.Q_\ell, T.Q_{\ell+1}, \ldots, T.Q_{\ell_{max}}\}$ of a cover tree T, a level ℓ.

```
1: $C \leftarrow \{children(q) : q \in T.Q_\ell\}$
2: $T.Q_{\ell-1} \leftarrow \{q \in C : d(p, q) \leq \frac{b^\ell}{b-1}\}$
3: $Delete(p, \{T.Q_{\ell-1}, T.Q_\ell, \ldots, T.Q_{\ell_{max}}\}, \ell - 1)$
4: if $d(p, C) = 0$ then
5:     delete $p$ from level $\ell - 1$ and from $children(parent(p))$
6:     for $q \in children(p)$ in greedy order do
7:         $\ell' \leftarrow \ell - 1$
8:         while $d(q, T.Q_{\ell'}) > b^{\ell'}$ do
9:             add $q$ into level $\ell'$
10:            $T.Q_{\ell'} \leftarrow T.Q_{\ell'} \cup \{p\}$
11:            $\ell' \leftarrow \ell' + 1$
12:        end while
13:        $q^* \leftarrow \mathrm{argmin}_{q' \in T.Q_{\ell'}} d(p', q)$
14:        make $q$ a child of $q^*$
15:    end for
16: end if
```

Theorem 2: Let T be a cover tree for a set P and p be an item, $p \notin P$. If p can be inserted at an existing level of T, then calling `Insert` (Algorithm 3) with input p and the root level $C_{\ell_{max}}$ of T returns a cover tree for $P \cup \{p\}$.

Proof: Since p can be inserted at an existing level, there is always a (sufficiently low) level of the tree where the condition of line 2 holds for the first time. Let $\ell - 1$ be this level. Since $\ell - 1$ is the highest level where this condition holds, it must hold that $d(p, T.Q_\ell) \leq b^\ell$. Therefore, the second condition of line 7 holds and we can always find a parent for the new node, thus maintaining the covering invariant. Whenever a new node is inserted at some level, it is also inserted at all lower levels as a child of itself, thus the nesting invariant is maintained. It remains to prove the separation invariant. We shall prove it for level $\ell - 1$. The proof proceeds similarly for lower levels. Consider some other item q in $C_{\ell-1}$. If $q \in C$, then $d(p, q) > b^{\ell-1}$. If not, then there is a higher level $\ell' > \ell$ where some ancestor of q, say q' was eliminated by line 5, i.e., $d(p, q') > \frac{b^{\ell'+1}}{b-1}$. Using the triangle inequality, we have that $d(p, q) \geq d(p, q') - d(q, q') \geq d(p, q') - \sum_{j=\ell}^{\ell'} b^j = d(p, q') - \frac{b^{\ell'+1} - b^\ell}{b-1} > \frac{b^{\ell'+1}}{b-1} + \frac{b^\ell - b^{\ell'+1}}{b-1} = \frac{b^\ell}{b-1} > \frac{(b-1)b^{\ell-1}}{b-1} = b^{\ell-1}$. □

For clarity of presentation, we have made the assumption that the new item p can be inserted at an existing level of the tree. In the general case, when the new item p must be inserted at a level lower than ℓ_{min}, we keep copying nodes of $C_{\ell_{min}}$ to a new level $C_{\ell_{min}-1}$, until p is separated from all other items in the new level. Similarly, when p has a distance from the root node larger than $b^{\ell_{max}}$, we promote both the root node and p to a new higher level $\ell_{max}+1$ and repeat this process until one of the two nodes can cover the other. Note that, since the explicit representation of the tree is stored, duplication of levels is only virtual and is performed very efficiently.

Incremental Deletion. Similar to insertion, to delete an item, starting from the root, `Delete` (Algorithm 4) is called until the item p to be deleted is located, keeping note of the candidate nodes at each level that may have p as a descendant. When p is located, it is deleted from the tree. In addition, all of its children are reassigned to some other candidate parent.

Algorithm 4 includes two heuristics for improving the quality of the resulting CT. One is the usual *nearest parent* heuristic shown in line 13: we assign each child of p to the closest among its candidate parents. The other heuristic refers to the order in which the children of p are examined in line 6. We examine them in a greedy manner starting from the one farthest apart from the items at level ℓ' and continue to process them in decreasing order of their distance to the items currently in ℓ'.

Theorem 3: Let T be a cover tree for a set P and p be an item, $p \in P$. If $p \notin C_{\ell_{max}}$ of T, calling `Delete` (Algorithm 4) with input p and the root level $C_{\ell_{max}}$ of T returns a cover tree for $P \setminus \{p\}$.

Proof: The item p is deleted from all levels that include it, thus the nesting invariant is maintained. For each child q of p, we move up the tree, until a parent for q is located, inserting q in all intermediate levels ℓ' to ensure that the nesting invariant is not violated. Such a parent is guaranteed to be found (at least at the level of the root). Adding q under its new parent does not violate the separation invariant in any of the intermediate levels since $d(q,q') > b^{\ell'}$, for all q' in $T.Q_{\ell'}$. The covering constraint also holds for the parent of q. □

For ease of presentation, we assumed that $p \notin C_{\ell_{max}}$. Otherwise, we need to select a new root. Note that, it is possible that none of the children of the old root covers all of its siblings. In this case, we promote those siblings that continue to be separated from each other in a new (higher) level $\ell_{max}+1$ and continue to do so until we end up with a level having a single node.

4 Diverse Set Computation

In this section, we present algorithms that use the cover tree to solve the k-diversity problem. The Level algorithms exploit the separation property, i.e., the higher the tree level, the farthest apart its nodes. We also present an efficient implementation of the Greedy Heuristic (Algorithm 1) that exploits the covering property to prune the search space.

4.1 The Level Family of Algorithms

We consider first the intuitive algorithm of selecting k items from the highest possible level of a cover tree, that is, from level ℓ_k, such that, $|C_{\ell_k+1}| < k$ and $|C_{\ell_k}| \geq k$ (depicted in Algorithm 5). Locating this level can be implemented efficiently, e.g., by using a hash table to store the size of each level. After locating ℓ_k, the complexity of the algorithm is $O(k)$, since a random subset of C_{ℓ_k} is selected.

Algorithm 5 Level-Basic Algorithm.

Input: A cover tree T, an integer k.
Output: A set S with k diverse items in T.

```
1: $\ell_k \leftarrow \ell_{max}$
2: while $|T.C_{\ell_k}| < k$ do
3:     $\ell_k \leftarrow \ell_k - 1$
4: end while
5: $S \leftarrow$ any subset of size $k$ of $T.C_{\ell_k}$
6: return $S$
```

The following theorem characterizes the solution attained by the *Level-Basic* algorithm with respect to the optimal solution.

Theorem 4 (Approximation Bound): Let P be a set of items, $d^{OPT}(P,k)$ be the minimum distance of the optimal diverse set for the MaxMin problem for $k \geq 2$ and $d^{CT}(P,k)$ be the minimum distance of the diverse set computed by the Level-Basic algorithm. Then:

$$d^{CT}(P,k) \geq \alpha\ d^{OPT}(P,k), \text{ where } \alpha = \frac{b-1}{2b^2}.$$

Proof: Let $S^{OPT}(P,k)$ be an optimal set of k diverse items. To prove Theorem 4, we shall bound the level where the least common ancestor (LCA) of any pair of items p_1, $p_2 \in S^{OPT}(P,k)$ appears in the cover tree. Assume that the LCA of any two items p_1, p_2 in the optimal solution appears for the first time at level m. That is, m is the lowest (furthest from the root) level that such an LCA appears.

Let us now compute a bound on m. Assume that the LCA of any two items p_1, $p_2 \in S^{OPT}(P,k)$ appears at level m. Let p be this ancestor. From the triangle inequality, $d(p_1,p) + d(p_2,p) \geq d(p_1,p_2)$. Since p_1, $p_2 \in S^{OPT}(P,k)$, it holds that, $d(p_1,p_2) \geq d^{OPT}(P,k)$. Thus:

$$d(p_1,p) + d(p_2,p) \geq d^{OPT}(P,k). \quad (1)$$

From the covering invariant of the cover tree, it holds that, $d(p_1,p) \leq \sum_{j=-\infty}^{m} b^j \leq \frac{b^{m+1}}{b-1}$. Similarly, $d(p_2,p) \leq \frac{b^{m+1}}{b-1}$. Substituting in (1), we get that $2\ \frac{b^{m+1}}{b-1} \geq d^{OPT}(P,k)$. Solving for m, we have $m \geq \log_b\left(\frac{b-1}{2} d^{OPT}(P,k)\right) - 1$.

Since m is the first level that the LCA of any two items in the optimal solution appears, from the covering property, it holds that at level $m-1$, there are at least k items, i.e., the distinct ancestors of the k items in the optimal solution. Thus, there are at least k items at level

$$m - 1 = \log_b\left(\frac{b-1}{2} d^{OPT}(P,k)\right) - 2. \quad (2)$$

This means that the cover tree algorithm will select items from this or a higher level. From the separation invariant of the cover tree, we have $d^{CT}(P,k) \geq b^{m-1}$. Using (2), we get that $d^{CT}(P,k) \geq b^{\log_b\left(\frac{b-1}{2} d^{OPT}(P,k)\right)-2} \Rightarrow d^{CT}(P,k) \geq \frac{b-1}{2} d^{OPT}(P,k)\ b^{-2}$, which proves the theorem. □

We also consider algorithms that, instead of selecting any k items from level ℓ_k, select these items greedily. The first algorithm, called *Level-Greedy*, performs a greedy selection among all items at level ℓ_k. This requires $k|C_{\ell_k}|$ distance computations. The second algorithm, called *Level-Inherit*, (Algorithm 6), initializes the solution with all items in C_{ℓ_k+1} and selects the remaining $k - |C_{\ell_k+1}|$ items from C_{ℓ_k} in a greedy manner. Thus, it requires $(k - |C_{\ell_k+1}|)|C_{\ell_k}|$ distance computations.

Algorithm 6 Level-Inherit Algorithm.

Input: A cover tree T, an integer k.
Output: A set S with k diverse items in T.

```
1: $\ell_k \leftarrow \ell_{max}$
2: while $|T.C_{\ell_k}| < k$ do
3:     $\ell_k \leftarrow \ell_k - 1$
4: end while
5: $S \leftarrow T.C_{\ell_k+1}$
6: $candidates \leftarrow T.C_{\ell_k} \backslash T.C_{\ell_k+1}$
7: while $|S| < k$ do
8:     $p^* \leftarrow \text{argmax}_{p \in candidates}\, d(p, S)$
9:     $S \leftarrow S \cup \{p^*\}$
10:    $candidates \leftarrow candidates \backslash \{p^*\}$
11: end while
12: return $S$
```

Clearly, the bound of Theorem 4 holds for the solution of Level-Greedy. It also holds for the solution of Level-Inherit, since due to nesting, an item that appears at some level of the tree also appears at all levels below it, thus, all items selected by Level-Inherit belong to C_{ℓ_k}. In general, these two algorithms are expected to produce more diverse sets than the estimated general bound. In particular, for Batch Cover Trees (BCTs), we can prove a better approximation. Specifically, it follows from Theorem 1, that the application of Level-Greedy and Level-Inherit algorithms on a BCT produces the same solution with the greedy heuristic.

Corollary 1: Let P be a set of items, $k \geq 2$, $d^{GR}(P,k)$ be the minimum distance of the diverse set computed by the greedy heuristic and $d^{BCT}(P,k)$ be the minimum distance of the diverse set computed by Level-Basic or Level-Inherit when applied on a BCT for P. It holds that $d^{BCT}(P,k) = d^{GR}(P,k) \geq {}^1/_2\ d^{OPT}(P,k)$.

4.2 Greedy Heuristic using CTs

Next, we present algorithms that use the cover tree to prune the search space of the greedy heuristic.

The algorithms proceed as follows. We initialize the diverse set S by selecting either the root or the two furthest apart leaves of the tree. This corresponds to initializing the greedy heuristic with either a random or the two most distant items. Then, we proceed in rounds. At each round, we select one item by descending the tree seeking for the item p with the maximum distance, $d(p,S)$, from the current set S. Specifically, at each of the $k-1$ (or $k-2$) rounds, we start descending the tree from the highest level C_ℓ that contains items that are not already in S. We locate the item p of C_ℓ with the largest $d(p,S)$ and use it to prune its siblings. Then, we consider as candidates the children of all non-pruned nodes of C_ℓ and repeat the process for $C_{\ell-1}$. In the end, the best candidate from the leaf level is added to S and we proceed to the next round. This process is shown in Algorithm 7.

Pruning is based on the following observation. Suppose that at some point we consider for inclusion in S an item p in C_ℓ. Let $d(p,S)$ be the distance of p from S and q be any sibling of p. Then, the best candidate in the subtree of q is at distance at most: $\sum_{j=\ell_{min}+1}^{\ell} b^j = \frac{b^{\ell+1} - b^{\ell_{min}+1}}{b-1}$.
from q. Therefore, we can safely prune nodes according to the following CT pruning rule:

Algorithm 7 Greedy-CT Algorithm.

Input: A cover tree T, an integer k.
Output: A set S with the k most diverse items in T.

```
1: $S \leftarrow \{T.root\}$
2: while $|S| < k$ do
3:     $Q \leftarrow children(T.root)$
4:     while $Q \neq \emptyset$ do
5:         $p^* \leftarrow \arg\max_{p \in Q} d(p, S)$
6:         $Q' \leftarrow children(p^*)$
7:         for all $p \in Q \backslash \{p^*\}$ do
8:             if $q$ is not pruned by the pruning rule then
9:                 $Q' \leftarrow Q' \cup children(q)$
10:            end if
11:        end for
12:        $Q \leftarrow Q'$
13:    end while
14:    $S \leftarrow S \cup \{p^*\}$
15: end while
16: return $S$
```

CT PRUNING RULE: Let p and q be two nodes at level ℓ in a CT. If $d(p,S) \geq d(q,S) + \frac{b^{\ell+1} - b^{\ell_{min}+1}}{b-1}$, we can prune the subtree rooted at q.

The CT pruning rule is pessimistic, in the sense that it assumes that each node may have a child located as far as possible from it. A more efficient pruning rule can be used at the trade-off of maintaining some extra information. Specifically, at each node p in the tree, we maintain the distance of p from the node in its subtree that is the furthest apart from p. We call this distance the *distance weight* of p denoted by $w_d(p)$. We call the tree that is annotated with such weights a *Weighted Cover Tree* (*WCT*). Then, we can use Algorithm 7 along with the following pruning rule:

WCT PRUNING RULE: Let p and q be two nodes at level ℓ in a WCT. If $d(p,S) \geq d(q,S) + w_d(q)$, we can prune the subtree rooted at q.

4.3 Other Issues

Constrained Continuous *k*-Diversity. The two requirements of constrained continuous k-diversity (Definition 1) can be easily enforced using cover trees. For the durability requirement, items that are selected as diverse are marked as such and remain part of the diverse set, until they expire. Let z be the number of such items. In this case, our algorithms just select $k - z$ additional items from the tree. For the freshness requirement, non-diverse items that are older than the newest item in the current diverse set are marked as "invalid" in the CT and are not considered further as candidates for inclusion.

Adjusting *k*. The CT can be used to provide results for multiple queries with different k. Thus, each user can individually tune the amount of diverse items she wishes to receive. Furthermore, the CT supports a "zooming" type of functionality. Assume that a user selects a specific value for k. After receiving the k most diverse items, she can request a larger number of closer to each other items by choosing a larger k ("zoom-in"), or a smaller number of farther apart items by choosing a smaller k ("zoom-out"). We can exploit the nesting invariant to achieve continuity in the following sense. Let S be the set of the k most diverse items and let ℓ be the highest level of the CT at which all items of S appear.

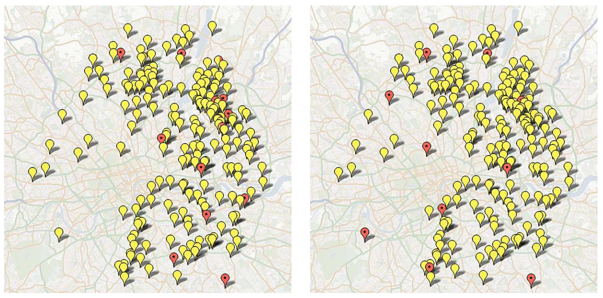

(a) Relevance. (b) Relevance and Diversity.

Fig. 5: Selecting $k = 10$ out of $n = 200$ apartments in London based (a) solely on relevance (i.e., price) and (b) incorporating diversity (i.e., geographical distance). Selected items are marked with a darker (red) color.

For $k' > k$, we would like the set S' with the k' most diverse items to be such that $S' \supseteq S$. To achieve this, we select items from level ℓ or lower, since the items in S appear at all levels $m \leq \ell$. Analogously, for $k' < k$, to construct the set S' with the k' most diverse items such that $S' \subseteq S$, we may select those items of S that appear at levels higher than ℓ.

5 Diversity and Relevance

In many cases, the items in the result of a query are ranked, most often based on their relevance to the user query. In this case, diversification also addresses the over-specialization problem, i.e., retrieving results that are very similar to each other. An example is shown in Figure 5 using our apartments dataset, where relevance is defined based on price, i.e., the cheaper the apartment the more relevant, and diversity is based on geographical location. Using only relevance, a user is presented with apartments mostly from east London, while with diversity, some relatively cheap apartments from other regions in London are also selected.

The MaxMin *k*-Diversity problem with relevance. In general, the relevance score of an item is application dependent. Without loss of generality, we assume a relevance function $r : P \rightarrow \mathbb{R}^+$ that assigns a relevance score to each item, where a higher value indicates that the item is more relevant to a particular query or user. A natural bi-criteria objective seeks to maximize both the relevance and the diversity of the selected subset. In particular, the MaxMin k-Diversity With Relevance Problem for a positive integer k, $k \leq n$, is the problem of selecting a subset S^* of P such that:

$$S^* = \operatorname{argmax}_{\substack{S \subseteq P \\ |S|=k}} f_r(S, d, r)$$

with

$$f_r(S, d, r) = \min_{p_i \in S} r(p_i) + \lambda \min_{\substack{p_i, p_j \in S \\ p_i \neq p_j}} d(p_i, p_j)$$

where $\lambda > 0$ is a parameter that tunes the importance of diversification.

A combined relevance-diversity (d_r) approach. It was shown in [20] that the MaxMin k-Diversity With Relevance Problem is equivalent with the MaxMin k-Diversity Problem if we replace the distance function d with the function d_r:

$$d_r(\lambda, p_i, p_j) = \frac{1}{2}\left(r(p_i) + r(p_j)\right) + \lambda d(p_i, p_j).$$

If we define $d_r(\lambda, p_i, p_j) = 0$, for $p_i = p_j$, it is to easy to see that d_r is a metric, if d is a metric.

To incorporate relevance, we can now build the CTs using distance d_r instead of d. It is straightforward to see that all algorithms and related bounds advanced for the diversity-only case directly apply to the combined relevance-diversity case.

Supporting a varying $\boldsymbol{\lambda}$. A drawback of the combined approach is that we need to maintain a different CT for each different value of λ. We would like to be able to adjust λ dynamically without having to reconstruct the trees. To this end, we consider building CTs based solely on distance d and enhancing our algorithms for selecting diverse sets so as to incorporate relevance in the selection.

Let C_{ℓ_k} be the highest level with at least k nodes. The enhanced Level-Basic algorithm selects the k most relevant items of C_{ℓ_k}, while the Level-Greedy algorithm performs a greedy selection among the corresponding items using the combined distance d_r, instead of d.

We also introduce a new level algorithm, called *Level-Hybrid*, whose goal is to allow nodes with large relevance scores that appear in low levels of the CT to enter the diverse set. *Level-Hybrid* uses an extended CT. In this extended CT, for each internal node p, we maintain a pointer to the node that has the largest relevance score among all nodes in the subtree rooted at p. Let $best(p)$ be this node. Level-Hybrid (Algorithm 8) performs a greedy selection among the k nodes from level C_{ℓ_k} whose descendants have the best relevance scores and these k descendants. Level-Hybrid performs $k \cdot 2k = 2k^2$ distance computations.

Algorithm 8 Level-Hybrid Algorithm.

Input: A cover tree T, an integer k, a real number λ.
Output: A set S with the k most diverse items in T.

```
1: $\ell_k \leftarrow \ell_{max}$
2: while $|T.C_{\ell_k}| < k$ do
3:     $\ell_k \leftarrow \ell_k - 1$
4: end while
5: $C \leftarrow$ the $k$ nodes $p$ in $T.C_{\ell_k}$ with the most relevant $best(p)$
6: $candidates \leftarrow \emptyset$
7: for all $p \in C$ do
8:     $candidates \leftarrow candidates \cup \{p, best(p)\}$
9: end for
10: $p^* \leftarrow \operatorname{argmax}_{p \in candidates} r(p)$
11: $S \leftarrow S \cup \{p^*\}$
12: $candidates \leftarrow candidates \setminus \{p^*\}$
13: while $|S| < k$ do
14:     $p^* \leftarrow \operatorname{argmax}_{p \in candidates} dr(\lambda, p, S)$
15:     $S \leftarrow S \cup \{p^*\}$
16:     $candidates \leftarrow candidates \setminus \{p^*\}$
17: end while
18: return $S$
```

In the CT implementation of the greedy heuristic, subtrees are pruned based on both diversity and relevance. To this end, we maintain at each internal node p, the largest relevance value, $w_r(p)$, called *relevance weight*, of any node in the subtree of p. The best possible pruning is achieved, if we also use the distance weight. Using both weights, we have the following pruning rule.

WCT Pruning Rule With d_r: Let p and q be two nodes at level ℓ in a WCT. If $d_r(p, S) \geq d_r(q, S) + \frac{1}{2}\left(r(q) + w_r(q)\right) + \lambda w_d(q)$, we can prune the subtree rooted at q.

TABLE 1: Characteristics of the datasets.

Dataset	Cardinality	Dimensions	Distance	Relevance scores
Uniform	10,000	2	Euclidean	Uniform/Clustered
Clustered	10,000	2	Euclidean	Uniform/Clustered
Cities	5,922	2	Euclidean	Clustered
Nestoria	1,000	8	Haversine	Price-based
Faces	300	256	Cosine	Uniform
Flickr	1,000	-	Jaccard	Uniform

Clearly, we could maintain only the relevance weight, in which case the distance is bounded using the CT pruning rule.

Maximal Marginal Relevance (MMR) Another popular approach for combining relevance and diversity is Maximal Marginal Relevance (MMR) (e.g., [12], [19]). MMR constructs a relevant and diverse subset S in a greedy fashion, by starting with either a random or the most relevant item and adding at each round the item p_i with the maximum contribution, i.e., the item p_i with the maximum quantity:

$$mr(\lambda, p_i, S) = \lambda r(p_i) + (1-\lambda)\min_{p_j \in S} d(p_i, p_j)$$

where $\lambda \in [0,1]$ is a parameter that tunes the relative importance of each of the two factors.

All the presented algorithms are directly applicable to MMR by using mr instead of d_r. For example, we now have the following pruning rule.

WCT PRUNING RULE WITH MMR: Let p and q be two nodes at level ℓ in a WCT. If $d_r(p,S) \geq \lambda w_r(q) + (1-\lambda)$ $(d(q,S) + w_d(q))$, we can prune the subtree rooted at q.

6 EVALUATION

In this section, we experimentally evaluate the performance of cover trees for dynamically computing diverse sets.

Datasets. We use a variety of datasets, both real and synthetic. Our *synthetic datasets* consist of two-dimensional points in the Euclidean space, where each dimension takes values in [0, 1]. Items are either uniformly distributed or form clusters of different sizes. We assign relevance scores to items either uniformly or in a "clustered" manner around specific target items, so that items that are closer to the target items get larger relevance scores than items further away. Clustered assignment is used to model the common case where we get high relevance scores around specific items that correspond to different interpretations of the query. Thus, we get four combinations: (i) uniform spatial distribution with uniform relevance scores ("*Uniform-Uniform*"), (ii) uniform spatial distribution with clustered relevance scores centered around uniformly distributed target items ("*Uniform-Clustered*"), (iii) clustered spatial distribution with uniform relevance scores ("*Clustered-Uniform*") and (iv) clustered spatial distribution with clustered relevance scores around the centers of the spatial clusters ("*Clustered-Clustered*").

We also employ *four real datasets*. "*Nestoria*" consists of information about 1,000 apartments for sale in the London area retrieved from [5]. We relate relevance with price and consider cheaper apartments as more relevant, while similarity is measured based on geographic proximity (Haversine distance). "*Cities*" is a collection of geographical points representing 5,922 cities and villages in Greece [4]. We assign relevance

TABLE 2: Parameter values.

Parameter	Range		Default	
	Synthetic	Real	Synthetic	Real
Base (b)	1.2-2.2		1.6	
Diversification factor (λ)	0.0-1.0		0.2	
Dataset size (n)	1-10,000	300-5,922	4,000	–
Size of diverse set (k)	100-300	10-100	150	50
Window size (w)	1,000	100	(no window)	
Window jump step (h)	100-900	10-90	(no window)	

scores in a clustered manner to model the fact that some specific areas may be more interesting than others. "*Faces*" consists of 256 features extracted from each of 300 human face images with the eigenfaces method [1] and uniformly distributed relevance scores. Finally, for "*Flickr*", we used data from [3] which consists of tags assigned by users to photographs uploaded to the Flickr photo service [2]. Table 1 summarizes our datasets, while Table 2 our parameters.

Our datasets capture result sets with different data characteristics. Concerning spatial distribution, for example, "*Uniform-Uniform*" contains items that cover all the available space, while "*Cities*" (due to the geography of Greece, which includes a large number of islands) provides us with both dense and sparse areas of items (Figure 3). "*Faces*" contains many distinct small dense areas, while "*Flickr*" is generally a very sparse dataset.

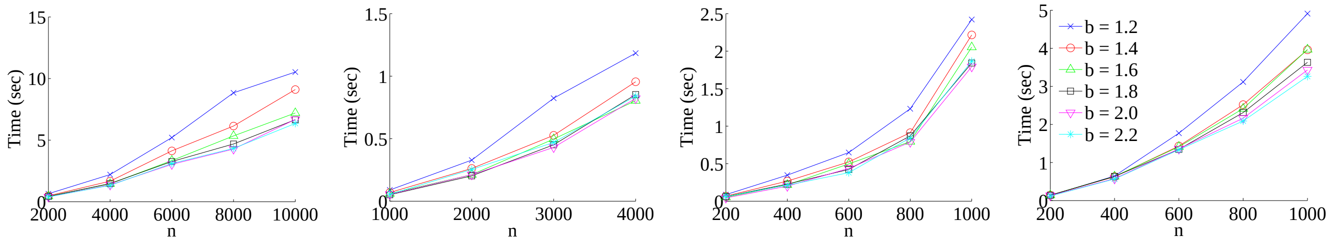
Setup. All methods are implemented in Java using JDK 1.6. Our experiments were executed on an Intel Core i3-2100 3.1GHz PC with 3GB of RAM.

6.1 Building and Maintaining Cover Trees

First, we evaluate the cost of building cover trees. Figure 6 shows the real-time cost of building an ICT by incrementally inserting items. This cost depends on b, since smaller values of b lead to new items being inserted in lower tree levels, thus increasing the cost of individual insertions. The cost also depends on the distance metric used, since some distance computations are more expensive. For example, inserting 1,000 items of the "*Flickr*" dataset, using the Jaccard distance, takes up to 5 seconds, while inserting the same number of items takes less than 0.1 seconds for our Euclidean datasets. The results are similar for the omitted datasets.

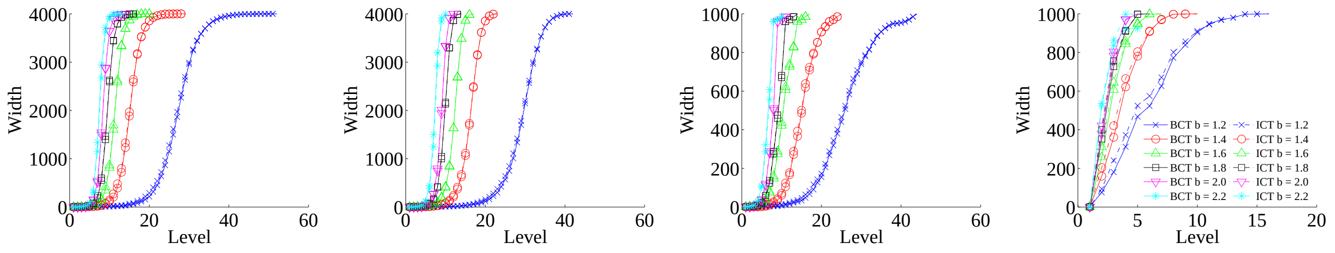
The cost of building a BCT can be divided into (i) the cost of selecting items from the leaf level to build the first non-leaf level and (ii) the cost of assigning nodes to suitable parents. Table 3 shows these costs for the uniform dataset. The cost of step (i) is the same as the cost of executing the greedy heuristic for $k = n$ and is independent of b or the dataset distribution. The cost of step (i) dominates that of step (ii) and this is why the total building cost for BCTs does not differ significantly with b or with spatial distribution. Building BCTs is orders of magnitude more expensive than building the corresponding ICTs for the same datasets.

Figure 7 depicts the size of ICTs and BCTs for different values of b ($n = 4,000$ for our synthetic datasets). The x-axis corresponds to the tree level, starting from 0 which denotes the root level, while the y-axis corresponds to the width (i.e., number of nodes) of the corresponding level. Smaller values of b lead to taller and narrower trees. Further, although ICTs



(a) Uniform. (b) Cities. (c) Nestoria. (d) Flickr.

Fig. 6: ICT building cost. The y-axis corresponds to the total time to incrementally insert all n items in the tree.



(a) Uniform. (b) Cities. (c) Nestoria. (d) Flickr.

Fig. 7: Tree sizes of the constructed ICTs and BCTs (full implicit representation). The drawn lines in each figure correspond to smaller b values as we move from left to right.

TABLE 3: BCT building cost (sec).

Uniform				
n	**step (i)**	**step (ii) *b*=1.2**	**step (ii) *b*=1.6**	**step (ii) *b*=2.0**
1,000	11.181	0.048	0.044	0.043
2,000	107.103	0.211	0.209	0.203
3,000	416.660	0.498	0.416	0.398
4,000	899.799	0.812	0.503	0.490

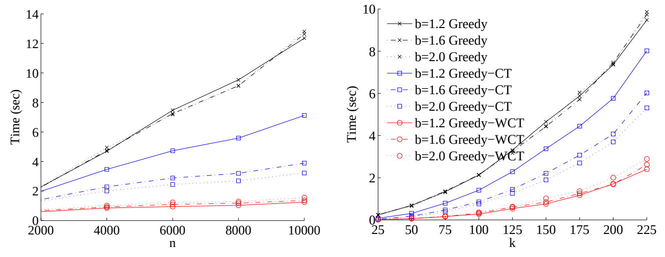
are constructed incrementally, the resulting trees have almost identical structure with the corresponding BCTs. In general, the height of the tree depends on the minimum and maximum pairwise distances in the dataset, while the width of the levels depends on the spatial distribution of the data. Therefore, for example, levels get narrower faster as we move up the tree for "*Cities*" rather than for '*Uniform*", even though their height is roughly the same, since "*Cities*" is a more clustered dataset. Similarly, a wide tree is constructed for "*Flickr*", due to sparsity.

In terms of maintenance, a single insertion in an ICT costs 1 msec for trees up to 5,000 and up to 1.3 msec for trees with 10,000 items. The cost of deletions is higher because, after a node is removed, its children have to be re-assigned to new parents. For all datasets and b values, a single deletion requires less than 3 msec for trees up to 5,000 and less than 7 msec for trees with 10,000 items. We also measured the cost of maintaining weights in the case of WCTs which may require some extra bookkeeping to update weights. For all datasets and b values, 4-6 additional nodes where accessed per insertion on average. The effect on execution time is negligible.

6.2 Computing Diverse Subsets

We next evaluate the performance of the various algorithms introduced in this paper in terms of the quality of the retrieved diverse sets and the computational cost.

Diversity Algorithms. We first measure the cost savings when applying the CT PRUNING RULE ("Greedy-CT") or WCT PRUNING RULE ("Greedy-WCT") on an ICT vs. executing our cover tree based implementation of the greedy heuristic



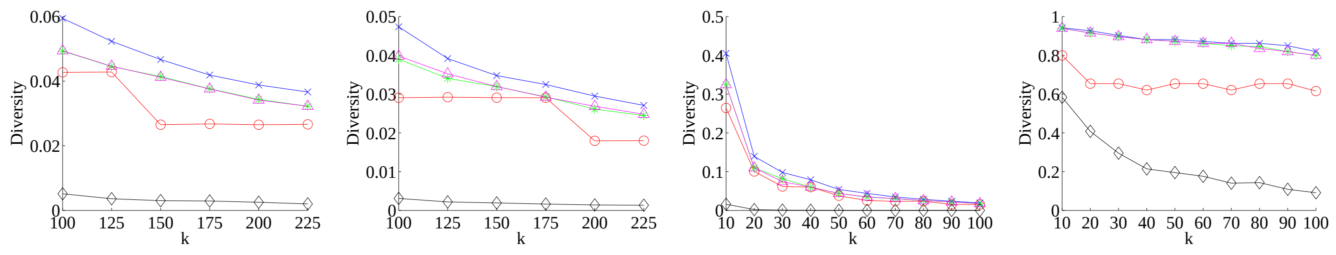
(a) Uniform (k = 150). (b) Uniform (n = 4000).

Fig. 8: Pruning for the diversity-only case.

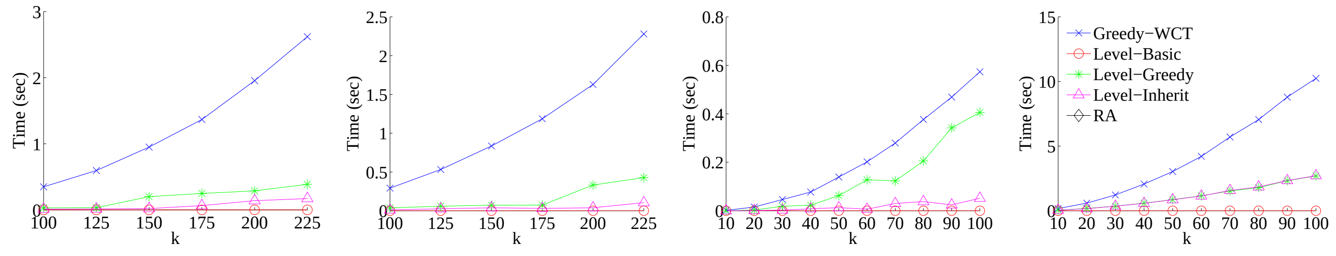
("Greedy"). We use our uniform dataset to see how pruning improves the cost of Greedy with n and k and different values of b (Figure 8). Clearly, Greedy-WCT is more effective, since the actual distance to the furthest descendant of each node is used for pruning. In general, pruning works better for non uniform datasets, since each selection of a diverse item results in pruning a largest number of items around it.

Next, we experimentally compare the performance of the greedy heuristic, using the Greedy-WCT implementation, and our Level algorithms, i.e., Level-Basic, Level-Greedy and Level-Inherit. Figure 9 depicts the achieved diversity and corresponding cost when varying k. For comparison, we also report the diversity attained by randomly selecting k of the n items (RA). Clearly, the larger the k, the less diverse the selected subset. The comparative performance of all algorithms is the same for all types of datasets. Specifically, for all datasets, Greedy-WCT achieves the best diversity at the highest cost, Level-Basic achieves the worst diversity at the lowest cost, while the other two Level algorithms lie in-between. Level-Inherit achieves similar diversity with Level-Greedy but is faster.

The Level algorithms select items from the appropriate tree level. Thus, their performance depends on the tree. Recall that, clustered datasets result in trees whose levels get narrower faster as we move up the tree. Level-Greedy and Level-Inherit perform a greedy selection among the items in the appropriate



(a) Uniform. (b) Cities. (c) Faces. (d) Flickr.



(e) Uniform. (f) Cities. (g) Faces. (h) Flickr.

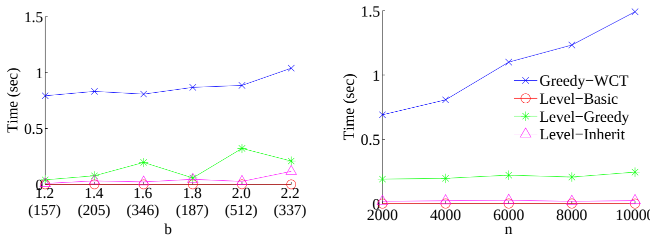
Fig. 9: Diversity and cost for the diversity-only case with varying k.

level, thus the wider the level, the worst the complexity and the better the diversity achieved. This also explains why their cost increases in "steps" as k increases, since we gradually select items from lower (and wider) levels. Level-Basic just selects any k items, thus the cost does not increase with k, while the achieved diversity decreases more rapidly as k increases, since items are selected randomly instead of greedily from wider levels.

Figure 10(a) shows how the cost varies with b. Smaller b values may increase the building cost of the tree (Figure 6) but also lead to faster diverse set computation, especially for Greedy-WCT. The cost of Level-Greedy and Level-Inherit depends on the size of the level from which items are selected. Figure 10(b) shows how the cost scales along n. All Level algorithms scale very efficiently with n, since they depend only on the size of the corresponding tree level.

Diversity with Relevance Algorithms. Let us first evaluate pruning (Figure 11). In the following, we report results for MMR (similar results are attained for d_r). We consider two rules for pruning: using only relevance weights (denoted "Greedy-CT") and using both relevance and distance weights (denoted "Greedy-WCT"). Again, using distance weights improves pruning especially for small values of λ (i.e., emphasis on diversity). Pruning is more effective for clustered relevance scores, since in this case, there are large subtrees with no relevant items that are pruned early. For the same reason, pruning generally performs better for very large values of λ. Finally, pruning is less effective for "*Flickr*" whose trees are shorter due to its sparsity.

We next compare Greedy-WCT with the Level algorithms. We also consider a CT variation, called *Priority Cover Tree (PCT)* introduced in [9] for computing priority medoids. A PCT is a CT which in addition to the three invariants of a CT, satisfies a fourth one that requires each node of the tree to have relevance score larger than or equal to the scores of all nodes in its subtree. To construct a PCT so that the fourth invariant is satisfied, items need to be inserted in descending order of relevance. In general, PCTs cannot be built incrementally. To illustrate, we present a simple example that shows that the



(a) Uniform. (b) Uniform.

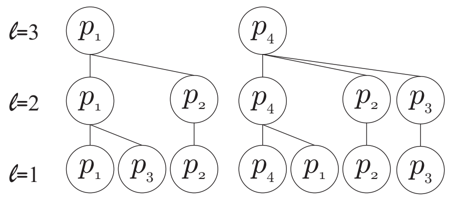
Fig. 10: Cost for the diversity-only case when varying (a) b (the numbers in parentheses are the sizes of the highest level with at least k items) and (b) n.

Distances

	p_1	p_2	p_3	p_4
p_1	0	5.0	3.5	2.5
p_2	5.0	0	5.3	7.5
p_3	3.5	5.3	0	5.0
p_4	2.5	7.5	5.0	0

Relevance

p_1	0.8
p_2	0.5
p_3	0.1
p_4	0.9

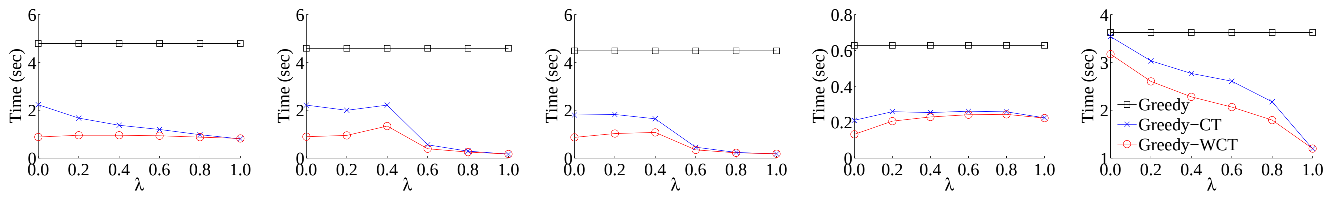


(a) Items. (b) Before insertion. (c) After insertion.

Fig. 12: The arrival of p_4 changes the relations among all nodes of the PCT.

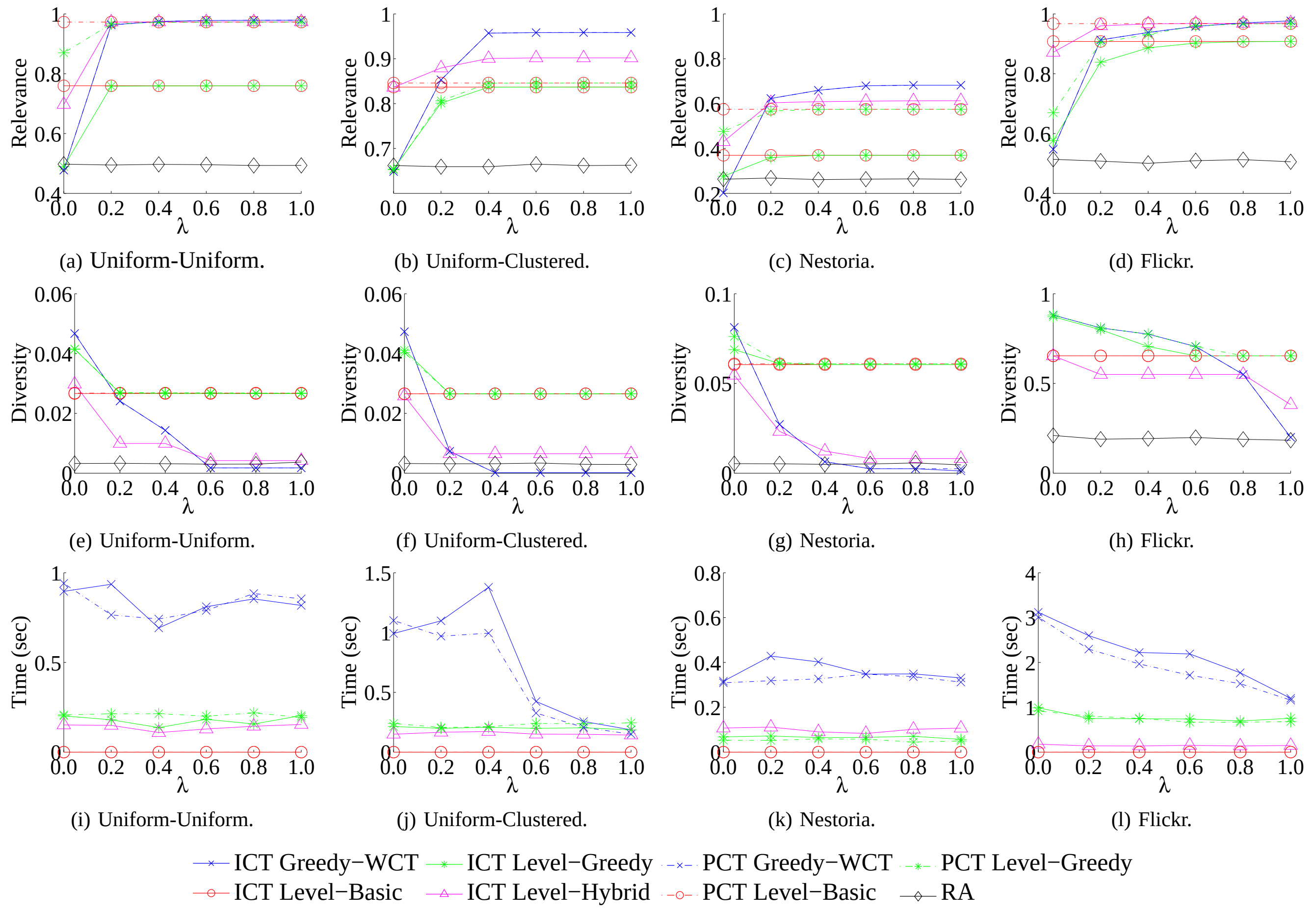
arrival of a single item may change the relations among all nodes in a PCT. Consider the PCT of Figure 12(b) with $b = 2.0$ and the relevance scores and distances of Figure 12(a). This PCT is unique for p_1, p_2, p_3, since p_1 must appear at the top due to its relevance and p_3 cannot be covered by p_2 at $\ell = 2$. Assume that p_4 arrives. Since p_4 has the largest relevance score, it must appear at the top of the tree. p_1 is not separated from p_4 at levels $\ell = 3$ and $\ell = 2$, therefore, it cannot appear there. p_2 and p_3 are separated from p_4 at $\ell = 2$ and are placed at this level. The resulting PCT is shown in Figure 12(c). Notice that all pre-existing nodes p_1, p_2, p_3 now have different parent and children nodes than before the arrival of p_4, which means that the tree is in effect re-built from scratch.

Figure 13 shows the relevance, diversity and cost of the various algorithms when varying λ. We report results for the faster greedy heuristic, i.e., Greedy-WCT and the three Level algorithms (namely, Level-Basic, Level-Greedy and Level-Hybrid) applied on an ICT and a PCT for two synthetic and



(a) Uniform-Uniform. (b) Uniform-Clustered. (c) Cities. (d) Faces. (e) Flickr.

Fig. 11: Pruning for the diversity with relevance (MMR) case with varying λ.



(a) Uniform-Uniform. (b) Uniform-Clustered. (c) Nestoria. (d) Flickr.
(e) Uniform-Uniform. (f) Uniform-Clustered. (g) Nestoria. (h) Flickr.
(i) Uniform-Uniform. (j) Uniform-Clustered. (k) Nestoria. (l) Flickr.

Fig. 13: Relevance (top row), diversity (middle row) and cost (bottom row) for the MMR case with varying λ.

two real datasets. Note that due to the fourth invariant of the PCT, $best(p) = p$ for every node p of a PCT, thus Level-Hybrid is the same with Level-Basic for PCTs and it is not depicted.

In terms of diversity and relevance, for all datasets, Level-Hybrid is the one closer to the greedy heuristic (which provides a good approximation of the optimal solution). Level-Hybrid achieves such results with much smaller cost. Among the Level algorithms, Level-Basic is clearly the fastest. Level-Hybrid performs a greedy selection among $2k$ items, while Level-Greedy performs a greedy selection among $|C_{\ell_k}|$ items, where ℓ_k is the highest level with at least k items. Therefore, the relative cost of Level-Greedy when compared with Level-Hybrid depends on the size of the level with regards to k. For example, for "*Flickr*", which has much wider levels than the other datasets, Level-Hybrid has lower cost than Level-Greedy. Note also that the cost of the level algorithms does not depend on λ. The quality and cost of the PCT solutions does not differ substantially from those of the ICT. Only pruning is slightly more efficient, since larger relevance scores appear at high levels. Due to space limitations, we omit the results for the rest of our datasets. "*Clustered-Uniform*" and "*Clustered-Clustered*" behave similarly to "*Uniform-Uniform*" and "*Uniform-Clustered*" respectively, while "*Cities*" has similar behavior to "*Uniform-Clustered*" and "*Faces*" to "*Clustered-Uniform*".

Continuous *k*-Diversity. We next focus on streaming arrivals of items and on how the application of our continuity requirements affects the retrieved solutions. We show results for "*Nestoria*", where we use the actual apartment upload time as the time in which items enter the stream. We also use the "*Clustered-Uniform*' dataset which has the most different distribution. For "*Clustered-Uniform*', the items that enter the stream are selected in a random manner.

Figure 14 reports results for the UNCONSTRAINED and the CONSTRAINED k-DIVERSITY PROBLEM. We vary the jump step h of the window and fix the other parameters

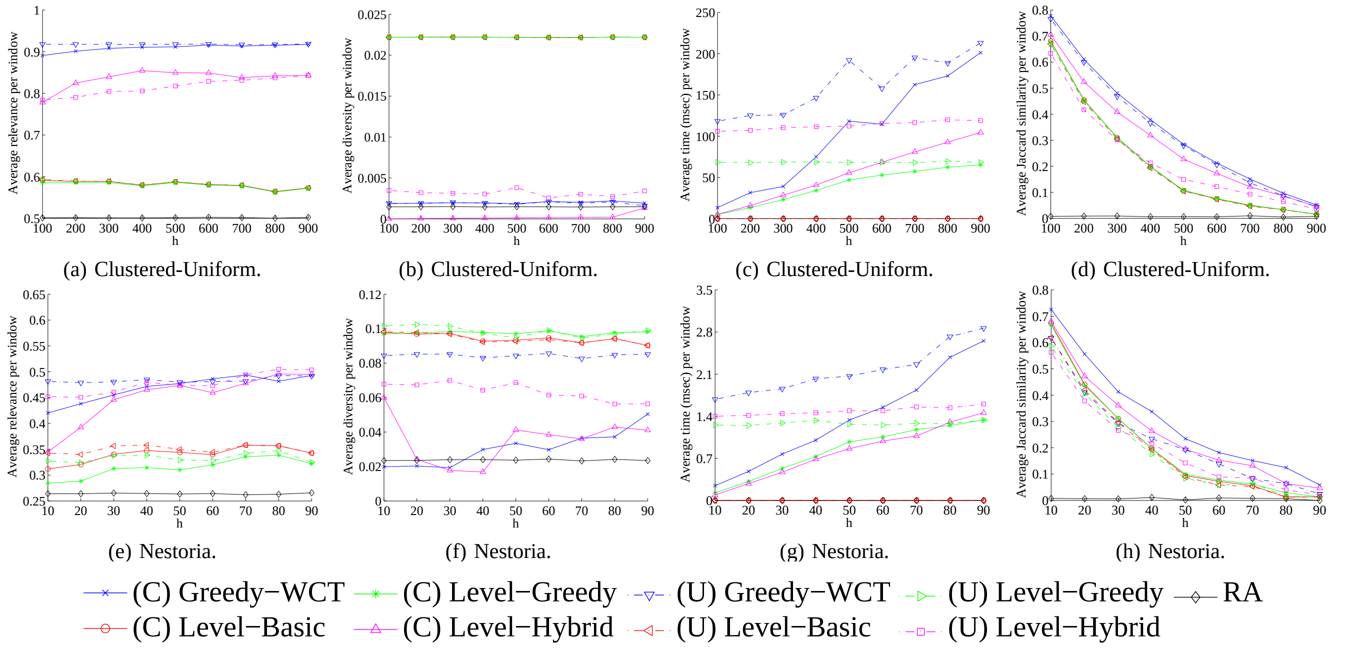


Fig. 14: Relevance ((a)/(e)), diversity ((b)/(f)), cost ((c)/(g)) and Jaccard similarity ((d)/(h)) for the Unconstrained (U) and Constrained (C) MMR case in a streaming setting when varying h ($k = 150$ (resp. $k = 15$), $w = 1000$ (resp. $w = 100$) for the synthetic (resp. real) dataset, $b = 1.6$, $\lambda = 0.2$).

to study the behavior of the algorithms as the number of valid diverse items from the previous window changes. We report average values over all windows as the window slides along the stream of items. In most cases, the constrained variations achieve similar relevance and diversity with the unconstrained alternatives. For all algorithms performing greedy computations, the constrained variations are executed faster, since the diverse subset of each window is initialized with the valid items of the diverse subset from the previous window, and thus, fewer computations are required. Level-Basic is unaffected, since it does not involve any greedy steps. Besides cost savings, another important aspect of the constrained variations is the higher sense of continuity between subsequent diverse sets seen by the users. To quantify this, we use the Jaccard similarity between the acquired diverse sets. The Jaccard similarity of two sets of items S_1, S_2 is defined as:

$$Jaccard(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

The higher the Jaccard similarity of two sets, the more common items the two sets share. In Figure 14, we see that the constrained variations exhibit higher Jaccard similarity.

7 Related Work

Due to the NP-hardness of the k-Diversity Problem, many different algorithms have been proposed for its solution (e.g., [18], [20], [29], [30]). In this paper, we have proposed an approach based on cover trees to address the dynamic case, where the data items change over time.

Static data. Most approaches rely either on greedy or interchange heuristics [18], [14]. Greedy heuristics construct the diverse subset incrementally by selecting at each step the item that is the furthest apart from the items already selected. Interchange heuristics start from a random solution and try to improve its diversity by swapping items in the solution with items outside it. It has been shown that the greedy heuristic for the MaxMin variation produces a performance guarantee of two which is shown to be the best that can be achieved by any polynomial algorithm [27]. A thorough comparison along with novel variations can be found in [29].

Another line of research considers selecting the k diverse items using top-k threshold algorithms. Such approaches assume the availability of sorted access methods. In contrast, in this paper, we propose such sorted access methods based on cover trees. The approach in [19] offers an implementation of MMR in low-dimensional vector spaces assuming the availability of both relevance-based and distance-based sorted access methods. A number of variations of sorted and random accesses are also employed in [7] to retrieve a top-k list of relevant and diverse results. The focus is on scheduling the order of the various accesses for cost efficiency.

Finally, other approaches define diversification in terms of covering different categories or interpretations of ambiguous web queries. Such approaches assume that there exists a related taxonomy of both queries and documents [6], or query logs, and a priori knowledge of the distribution of the underlying possible specializations of queries [11].

Indices and structured data. There are a couple of approaches that consider indexing to assist diversification. Most such works consider structured data. Relational data are considered in [28]. Attributes are *totally ordered* by importance in terms of diversity, so that two tuples that differ in a highly important attribute are considered highly diverse, even if they share common values in other less important attributes. This diversity measure allows the exploitation of a Dewey encoding of tuples that enables a tree structure which is later exploited to select the k most diverse tuples. Contrary to our approach, the proposed method is limited to this specific diversity measure. A spatial index is exploited in [21] to locate those relevant nearest neighbors of an item that are the most distant to each other. Our work is different since our goal is not to locate the nearest diverse neighbors of a single object but rather to locate a relevant and diverse subset of all available items. A different problem for structured data is considered in [24], that of selecting a limited number of features that can maximally differentiate the available items.

Cover trees are employed in [23] for solving the k-medoids problem. A variation of cover trees, called Priority Cover Trees (PCTs), were employed in [9] for computing priority medoids, i.e., medoids having a high relevance factor. Besides solving a different problem, this approach cannot be employed in dynamic environments, since all available items must be known in advance for building PCTs.

Continuous data. The related literature focusing on continuous data is considerably more limited. None of the existing proposals considers an index-based approach.

In our previous work [13], we evaluated various heuristics in case of continuous data, and a greedy heuristic that enforces durability was shown to outperform the other methods. A method based on interchange heuristics is proposed in [25]. Upon the arrival of a new item p, all possible interchanges between p and the items in the current solution are performed and p replaces an item in the solution, if this replacement increases diversity. A similar technique was also proposed in [16]. However, with these methods, old items do not expire, and a new item may enter the solution only upon its arrival. The MaxSum diversification problem is studied in [10], in the setting of streaming data and monotone submodular diversification functions. A $1/2$-approximation greedy algorithm is proposed which is faster than the usual greedy heuristic. Dynamic updates are also considered in the sense that when the underlying set of available items changes, interchanges are attempted to improve the computed solution. Finally, the online version of the diversity problem is considered in [26], that is, selecting a diverse subset without knowing the complete set of items

8 Summary

Recently, result diversification has attracted considerable attention. However, most current research addresses the static version of the problem. In this paper, we have studied the diversification problem in a dynamic setting where the items to be diversified change over time. We have proposed an index-based approach that allows the incremental evaluation of the diversified sets to reflect item updates. Our solution is based on cover trees. We have provided theoretical and experimental results regarding the quality of our solution.

Acknowledgments

The work of the first author has been co-financed by the European Union (ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the NSRF-Research Funding Program: Heracleitus II and of the second author by the European Union (ERDF) and Greek national funds through the Operational Program “Thessaly-Mainland Greece and Epirus 2007-2013” of the NSRF 2007-2013.

References

[1] Faces dataset. http://www.informedia.cs.cmu.edu.
[2] Flickr. http://www.flickr.com.
[3] Flickr dataset. http://www.tagora-project.eu.
[4] Greek cities dataset. http://www.rtreeportal.org.
[5] Nestoria. http://www.nestoria.co.uk.
[6] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM*, 2009.
[7] A. Angel and N. Koudas. Efficient diversity-aware search. In *SIGMOD Conference*, 2011.
[8] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, 2006.
[9] R. Boim, T. Milo, and S. Novgorodov. Diversification and refinement in collaborative filtering recommender. In *CIKM*, 2011.
[10] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *PODS*, 2012.
[11] G. Capannini, F. M. Nardini, R. Perego, and F. Silvestri. Efficient diversification of web search results. *PVLDB*, 4(7):451–459, 2011.
[12] J. G. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, 1998.
[13] M. Drosou and E. Pitoura. Diversity over continuous data. *IEEE Data Eng. Bull.*, 32(4):49–56, 2009.
[14] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Record*, 39(1):41–47, 2010.
[15] M. Drosou and E. Pitoura. Dynamic diversification of continuous data. In *EDBT*, 2012.
[16] M. Drosou, K. Stefanidis, and E. Pitoura. Preference-aware publish/subscribe delivery with diversity. In *DEBS*, 2009.
[17] E. Erkut. The discrete *p*-dispersion problem. *European Journal of Operational Research*, 46(1), 1990.
[18] E. Erkut, Y. Ülküsal, and O. Yeniçerioglu. A comparison of *p*-dispersion heuristics. *Computers & OR*, 21(10), 1994.
[19] P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Top-k bounded diversification. In *SIGMOD Conference*, 2012.
[20] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, 2009.
[21] J. R. Haritsa. The kndn problem: A quest for unity in diversity. *IEEE Data Eng. Bull.*, 32(4):15–22, 2009.
[22] T. Kollar. Fast Nearest Neighbors. http://nicksgroup.csail.mit.edu/TK/Technical_Reports/covertrees.pdf.
[23] B. Liu and H. V. Jagadish. Using trees to depict a forest. *PVLDB*, 2(1):133–144, 2009.
[24] Z. Liu and Y. Chen. Differentiating search results on structured data. *ACM Trans. Database Syst.*, 37(1):4, 2012.
[25] E. Minack, W. Siberski, and W. Nejdl. Incremental diversification for very large sets: a streaming-based approach. In *SIGIR*, 2011.
[26] D. Panigrahi, A. D. Sarma, G. Aggarwal, and A. Tomkins. Online selection of diverse results. In *WSDM*, 2012.
[27] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Facility dispersion problems: Heuristics and special cases. In *WADS*, 1991.
[28] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. Amer-Yahia. Efficient computation of diverse query results. In *ICDE*, 2008.
[29] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. T. Jr., and V. J. Tsotras. On query result diversification. In *ICDE*, 2011.
[30] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, 2009.
[31] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, 2005.

Marina Drosou is a PhD candidate at the Computer Science Department of the University of Ioannina in Greece. She received her MSc and BSc degrees from the same institution in 2008 and 2006 respectively. She is a member of the Distributed Data Management group since 2006 and a recipient of an “HERACLETUS II” scholarship. Her research interests include personalization via ranking based on diversity and recommendation systems.

Evaggelia Pitoura is a Professor in the Computer Science Department of the University of Ioannina in Greece where she also leads the Distributed Data Management group. She received her BSc from the University of Patras in Greece and her MSc and PhD from Purdue University. Her research interests are in the area of distributed data management with a recent focus on ranking based on preferences, diversity and relevance.