Data Science and Engineering (2021) 6:411–433
https://doi.org/10.1007/s41019-021-00173-1

# Efficient Indexing of Top-k Entities in Systems of Engagement with Extensions for Geo-tagged Entities

**Anirban Mondal[1] · Ayaan Kakkar[2] · Nilesh Padhariya[2] · Mukesh Mohania[2]**

Received: 3 June 2021 / Revised: 3 September 2021 / Accepted: 23 September 2021 / Published online: 11 October 2021


**Abstract**
Next-generation enterprise management systems are beginning to be developed based on the Systems of Engagement (SOE) model. We visualize an SOE as a set of entities. Each entity is modeled by a single parent document with *dynamic* embedded links (i.e., child documents) that contain multi-modal information about the entity from various networks. Since entities in an SOE are generally queried using keywords, our goal is to *efficiently* retrieve the top-*k* entities related to a given keyword-based query by considering the relevance scores of both their parent and child documents. Furthermore, we extend the afore-mentioned problem to incorporate the case where the entities are geo-tagged. The main contributions of this work are three-fold. First, it proposes an efficient bitmap-based approach for quickly identifying the candidate set of entities, whose parent documents contain all queried keywords. A variant of this approach is also proposed to reduce memory consumption by exploiting skews in keyword popularity. Second, it proposes the two-tier HI-tree index, which uses both hashing and inverted indexes, for efficient document relevance score lookups. Third, it proposes an R-tree-based approach to extend the afore-mentioned approaches for the case where the entities are geo-tagged. Fourth, it performs comprehensive experiments with both real and synthetic datasets to demonstrate that our proposed schemes are indeed effective in providing good top-*k* result recall performance within acceptable query response times.

**Keywords** Indexing · Top-*k* entity retrieval · Systems of engagement · Geo-tagged entities · R-tree

## 1 Introduction

Nowadays, enterprises have fully realized the value of data that they have about their customers in Customer Relationship Management (CRM) systems and transactional systems. To gain competitive advantage by knowing more about their customers, enterprises try to incorporate the social data of their customers. This has led to the emergence of next-generation CRM systems that are built upon the paradigm of *"Systems of Engagement" (SOEs)*. Thus, we are witnessing a paradigm shift from the traditional "Systems of Records" (SORs) toward the SOEs [3].

While SORs typically allow only passive one-way transactional communication between the enterprise management system and the stakeholders, SOEs enable two-way communication, thereby allowing stakeholders to engage and collaborate with each other [2]. SOEs also incorporate social orientation by combining multi-modal information from different kinds of networks (e.g., social networks and business community networks), thereby establishing a better *context* for delivering greater agility and flexibility [1]. Interestingly, the shift toward a model of engagement, as exemplified by SOEs, is also consistent with the increasingly important role that social networks, such as Facebook and Twitter, play in our lives today. Moreover, the work in [29] has motivated the fact that the strategies of companies need to be reviewed in light of this age of information technology and digital transformation. It further made the case that building systems of engagement is very important for

✉ Anirban Mondal
anirban.mondal@ashoka.edu.in

Ayaan Kakkar
ayaan18028@iiitd.ac.in

Nilesh Padhariya
nilesh.iitd@gmail.com

Mukesh Mohania
mukesh@iiitd.ac.in

[1] Ashoka University, Plot No. 2, Rajiv Gandhi Education City, National Capital Region P.O.Rai, Sonepat, Haryana 131029, India

[2] IIIT Delhi, Delhi, India

different industries to achieve service innovation capabilities. Furthermore, the work in [18] studied how management consultants can be facilitated toward navigating competing systems of engagement w.r.t. their daily routine operations and their consultancy services for clients.

Although we have motivated SOEs using CRM applications, SOEs also have significant commercial applications in important areas such as human resource management and supply chain management.

### 1.1 Problem Statement with an Example

We visualize an SOE as comprising a set of entities e.g., customers. Each entity is modeled by a single parent document and possibly multiple child documents. Here, the links embedded in a given parent document constitute its child documents. These links refer to the interactions of the entity on various systems.

Each parent document contains both structured data as well as unstructured data about the entity. Examples of structured data include information associated with a customer such as name, date of birth, city of residence, nationality, customer status (e.g., platinum, gold or silver), customer's annual income range, email address and phone number. The unstructured data pertaining to a customer can include unstructured information about the customer such as products purchased by the customer, her hobbies and preferences and so on. With regard to the problem context, suppose that the parent document pertaining to a user Alice contains structured data such as her date of birth, location, phone number and address. Suppose the unstructured data (in the parent document) contain information about the list of items that she has purchased. Let the queried keyword(s) be related to chargers for phones. These queried keywords would essentially occur in the unstructured data (in the parent document) i.e., in the list of products purchased by Alice. Since entities in an SOE are generally queried by means of *keywords*, we index both the structured and the unstructured data as a set of keywords.

Our goal is to *efficiently* retrieve the top-$k$ entities pertaining to a given keyword-based query [7, 11, 19, 20, 22, 24] by considering the relevance scores of both their parent and child documents. Notably, the parent documents of a given entity remain relatively static, while the child documents may be *dynamic* (e.g., a new blog comment by the entity). Hence, we need to consider the dynamically changing relevance scores of the child documents as part of computing the top-$k$ entities. For the sake of convenience, let us henceforth refer to the set of parent documents in which all the queried keywords occur together as the **candidate set**. (Thus, our query model uses 'AND' semantics.)

Figure 1 illustrates an SOE-oriented CRM application, where the entities are customers such as Alice, Bob and Chuck. Observe how the child documents of Alice comprise her interactions on the email system, phone-call system, social network pages and product review pages. SOE can encompass different types of data models (e.g., emails, phone-calls, social-pages, etc.) for each entity, which can be integrated by adopting data integration approaches [6]. We consider multi-modal data (e.g., speech data of phone-calls) that can be transcribed into text.

Figure 2 depicts an illustrative example for parent and child documents for an SOE-oriented CRM application. Suppose Alice has recently purchased a Samsung Galaxy note 2 phone and made comments on various systems about her phone charger problem. The company may wish to find its top-$k$ customers who complained about phone charger problems. Here, top-$k$ can be defined based on the frequency of occurrence of the keywords associated with the charger problem across various interaction networks of the customers.

Furthermore, observe that until this point, we have discussed the case where the entities are *not* geo-tagged i.e., the entities are not associated with any specific point in space. However, in certain scenarios, the entities could be geo-tagged. In particular, the parent document of a given entity could be associated with a geographical location i.e., a point in space, while the child documents need not necessarily be geo-tagged. For example, the parent document of an entity could have a location attribute at any specific spatial granularity e.g., home address, city, state or country. On the other hand, the child documents essentially constitute user interactions across different networks, hence the location of any child document (even if the child document was geo-tagged) does not provide any particularly useful information from the perspective of our application scenario of identifying the top-$k$ customers encountering mobile phone charger problems.

Referring to the afore-mentioned application scenario, the company may wish to find its top-$k$ customers, who complained about phone charger problems, *from North America*. Observe how the same query for identifying the top-$k$ customers could concern any spatial granularity such as country, state, city and so on. Notice how the ability to perform such analysis about customer complaints at *different spatial granularities* facilitates the company toward effective business decision-making by providing them with a better understanding of the problems faced by their customers in a given geographic region. In case of many multi-national companies, customers face different types of problems in different regions of the world. In practice, this can happen typically due to supply chain issues. For example, suppose the mobile phone chargers for the region of North America are being manufactured in factory X, while the chargers for the Asian region are being manufactured in factory Y. If there are a significant number of customer complaints from

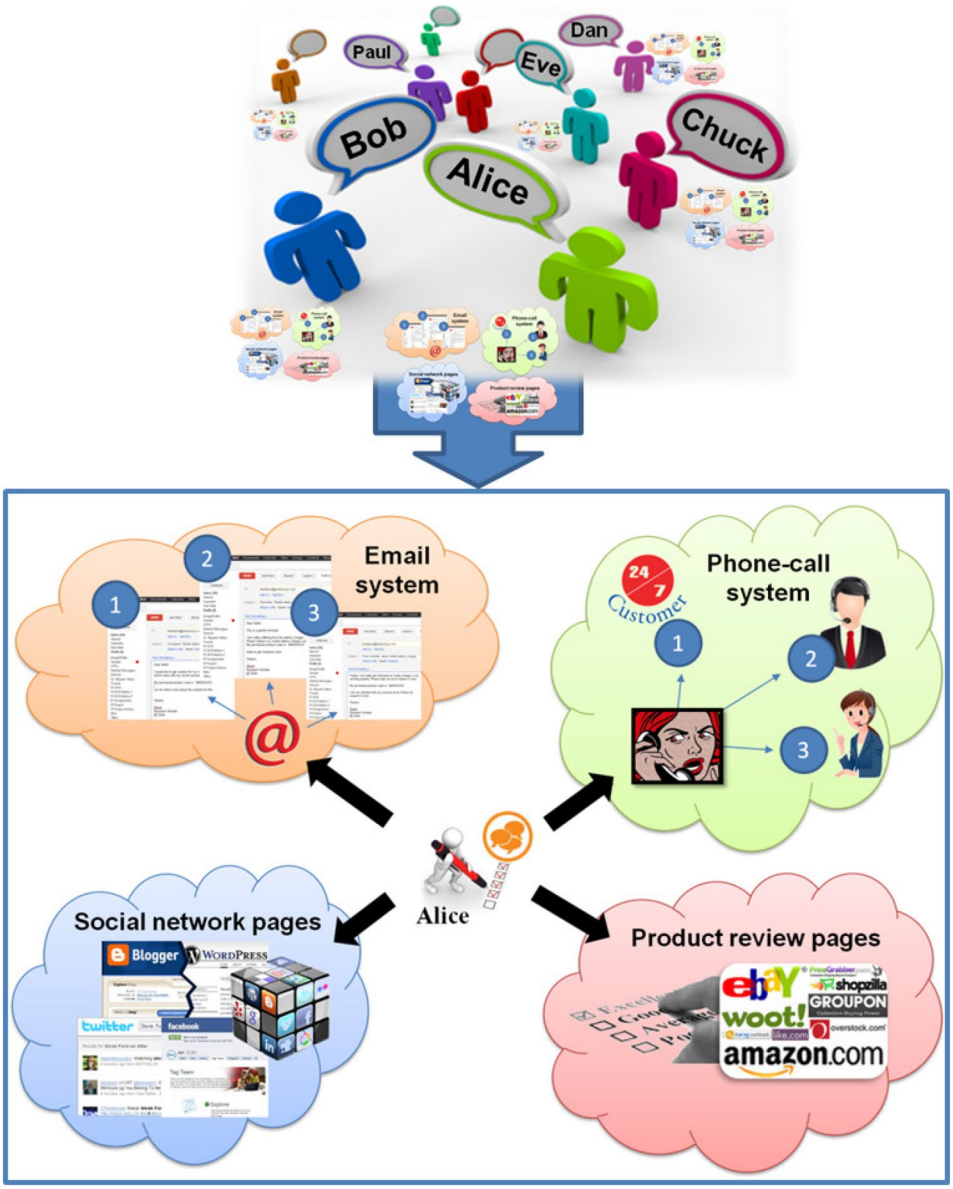



**Fig. 1** Sample SOE for a CRM application

the North American region, but little or no customer complaints from the Asian region, it may indicate that factory X has a faulty manufacturing process. Observe how this can also play an important role in identifying the source of the problems associated with customer complaints.

## 1.2 Related Work and Differentiation

The problem addressed in this paper (i.e., retrieval of the top-*k* entities in an SOE) fundamentally differs from those of existing works as follows. First, while keyword search [7, 11, 19, 20, 22, 24] has been studied for relational databases, and graph-based approaches for keyword search over relational databases have also been proposed [7, 11, 20], the *semantics* of their definition of an entity differ considerably from that of SOEs. All these works consider every individual document in the corpus as a separate entity. In contrast, in an SOE, each document is *not* considered as a separate entity i.e., there is an *implicit semantic nested structure* in the document corpus such that each entity comprises a parent document and a set of child documents (embedded links in the parent document). Thus, existing works cannot be used to perform efficient keyword search in an SOE since they fail to capture the semantic nested structure of SOE entities. Similarly, works on keyword search in text cube models for multidimensional text database analysis [13, 30, 31] also cannot be used for top-*k* entity retrieval in an SOE.

Second, keyword search approaches using link analysis (e.g., the PageRank method [25] and the hubs & authority approach [21]) exploit the link structure of the document

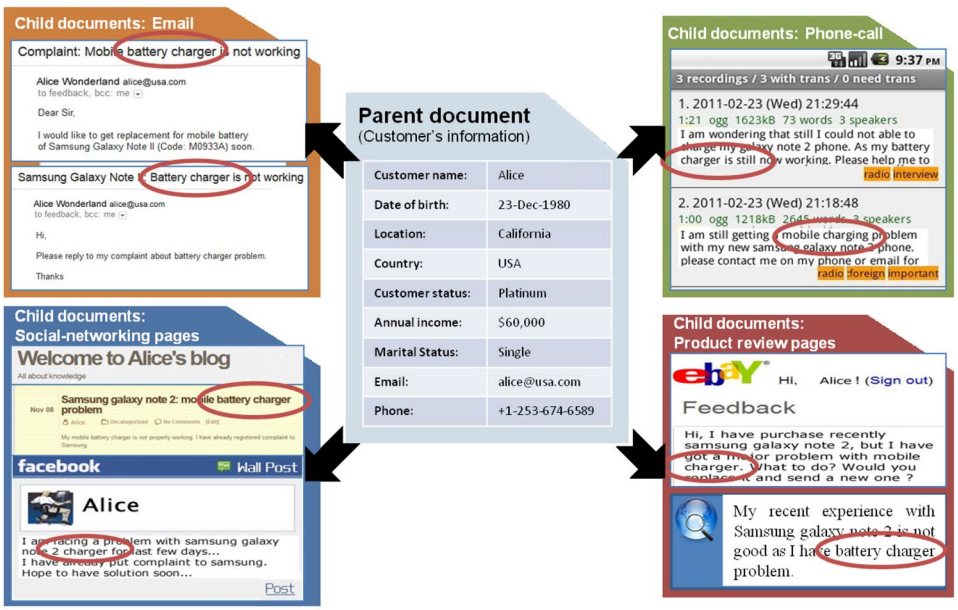



**Fig. 2** Illustrative example for parent and child documents in an SOE-oriented CRM application

corpus for ranking the documents [12, 15, 23]. For example, given a graph of a set of linked documents (e.g., webpages), they would examine the number and quality of links to each document to determine its relevance score before ranking the documents. However, they cannot be used for retrieving the top-$k$ entities in an SOE because they do not address keyword indexing for a document corpus, where each parent document can have multiple *dynamic* embedded links. In essence, they too fail to address the semantic nested structure of SOE entities. Thus, intuitively, they would return results that differ significantly from those of our proposed schemes. Furthermore, for computing the top-$k$ result, all keyword-search approaches compute a relevance score for each document in the corpus and then rank them. In contrast, our proposed schemes *efficiently prune* the search space by first determining the candidate set of entities (parent documents); then relevance scores are computed only for parent documents in the candidate set, thereby significantly reducing computation costs.

Incidentally, inverted lists [9] focus on retrieving all the documents in which a *single* given keyword occurs. An intuitive brute-force approach for top-$k$ query processing in an SOE could be to use an inverted list for identifying the list of parent documents in which each queried keyword occurs. Thus, for $m$ queried keywords, there would be $m$ such document lists. Hence, for identifying the candidate set, we would need to exhaustively intersect between these $m$ document lists (as in multi-way join processing). Then we would need to sort the candidate set based on relevance scores to derive the top-$k$ result. However, this approach is prohibitively expensive and non-scalable.

Regarding spatial indexing, the R-tree [16] is one of the popular multi-dimensional indexes. Variants of the R-tree include the R+-tree [28] and the R*-tree [10]. R-tree-based spatio-temporal indexes include the time-parameterized R-tree (TPR-tree) [27], the Multi-version 3D R-tree (MV3R-tree) [33] as well as the Spatio-Temporal R-tree (STR-tree) and the Trajectory-Bundle tree (TB-tree) [26]. The work in [34] has proposed an R-tree-based spatial index with support for keyword search queries. Moreover, the work in [8] proposed a class of R*-tree indexes to facilitate spatial-visual search of street images that are geo-tagged. In particular, this new class of R*-tree indexes aim at organizing images based on both spatial as well as visual properties. Furthermore, the work in [32] discussed an R-tree-based scheme for trajectory data protection in order to address privacy issues in location-based services. A good survey on spatial (and spatio-temporal) indexing can be found in [5]. However, none of these works consider the use of R-tree-based approaches toward efficiently identifying top-$k$ geo-tagged entities in SOEs at different spatial granularities.

### 1.3 Our Contributions

To address the above limitations, this work proposes two top-$k$ query processing schemes. Both of these schemes use the following three steps. (a) Determine the candidate set (b) Lookup the relevance scores of the documents in the candidate set w.r.t. the queried keywords (c) Sort the candidate set based on document relevance scores to obtain the top-$k$ result. The two schemes are similar in that both of them use bitmaps for efficiently determining the candidate set. They differ in that while one scheme uses a two-dimensional array-based approach for document relevance score lookups (the two dimensions being document identifiers and keyword relevance scores, as discussed later in Sect. 3.3), the other

scheme uses our proposed index (designated as the HI-tree). Thus, we shall henceforth designate them as **Bitmap-based Scheme with Array (BSA)** and **Bitmap-based Scheme with HI-tree index (BSH)**, respectively.

BSA is a viable option for environments with large memory space e.g., Cloud environments. In contrast, BSH is more suitable for environments, where space constraints necessitate a more space-efficient approach toward document relevance score lookups. To reduce the memory consumption of BSA, we propose a variant, which keeps both the bitmap arrays (i.e., the candidate set identification bitmap and the document relevance score lookup array) of only the popular keywords in memory, while storing the bitmap arrays of the relatively less popular keywords in the disk. We designate this variant as the **Bitmap-based Scheme with Array with reduction in Memory (BSAM)**.

Furthermore, we propose an R-tree-based extension of our proposed schemes for geo-tagged entities, as we shall see in Sect. 5. In essence, we assume that each entity is associated with a single location $L$ in space, and $L$ is the location of the parent document of the entity. We use an R-tree to index the location of the parent document of each entity in space. In the R-tree-based extension scheme for geo-tagged entities, we first obtain the candidate set $CS1$ of parent documents (along with their relevance scores) by using our proposed BSA, BSAM and BSH schemes. Then, given a spatial query window, we can quickly retrieve the parent documents that exist within the query window; let us denote this retrieved set of parent documents as $CS2$. Finally, we perform an intersection of the sets $CS1$ and $CS2$, and then sort in descending order on the basis of the relevance scores of the parent documents for finding the top-$k$ parent documents (entities) that are relevant to the spatial window query.

The main contributions of this work are four-fold:

1. It proposes an efficient bitmap-based approach for quickly identifying the candidate set of entities, whose parent documents contain all queried keywords. A variant of this approach is also proposed to reduce memory consumption by exploiting skews in keyword popularity.
2. It proposes the two-tier HI-tree index, which uses both hashing and inverted indexes, for efficient document relevance score lookups.
3. It proposes an R-tree-based approach to extend the aforementioned approaches for the case where the entities are geo-tagged.
4. It performs detailed experiments with both real and synthetic datasets to show that our proposed schemes are indeed effective in providing good top-$k$ result recall performance within acceptable query response times.

Ontologies and concept hierarchies [17] can also be used in conjunction with this work for query expansion purposes.

Notably, a preliminary version of our paper has appeared in [4]. In this paper, we extend our work in [4] in the following ways. First, in contrast with our work in [4], in this work, we additionally consider the case of geo-tagged entities. Moreover, we use an R-tree-based approach for *efficiently* determining the top-$k$ entities in an SOE for a given queried spatial window (region). Second, we have conducted a detailed performance evaluation for the case of geo-tagged entities, and the performance results are presented in Section 6.2.

The remainder of this paper is organized as follows. Section 2 presents the context of the problem, while Sect. 3 discusses the bitmap-based top-$k$ query processing schemes. Section 4 presents the HI-tree index. Section 5 discusses an R-tree-based extension of our proposed approaches for geo-tagged entities. Section 6 reports the performance study. Finally, we conclude in Sect. 7.

## 2 Context of the Problem

This section discusses the context of the problem. Recall that each entity has a single parent document with possibly multiple child documents. Let $D$ denote the corpus of $N$ parent documents {D1, D2, ...,D$N$} (one parent document per entity) such that each document $Di$ contains $p_i$ embedded links $\{L_1, L_2, \ldots, L_{p_i}\}$. Each link points to a single document. Thus, each embedded link is a child document. Observe that the parent documents remain relatively static over time e.g., a customer's date of birth or email address generally remains the same over time. However, the child documents may be *dynamic* in that new text (e.g., a new review comment, a new blog or a new email) may be inserted. The number of child documents corresponding to different parent documents may vary. Furthermore, the number of child documents corresponding to a given parent document can vary over time due to the addition or deletion of links.

Top-$k$ queries are of the form $\{Q_{id}, k, (a_1, a_2, \ldots, a_n)\}$, where $Q_{id}$ is the unique identifier of a given query, $k$ is the number of (top-$k$) entities that need to be retrieved, and $\{a_1, a_2, \ldots, a_n\}$ are queried keywords. Given a top-$k$ query $Q$, a parent document is considered to be in the candidate set if it contains at least one instance of each of the queried keywords in $Q$. (Note that if instances of any of the queried keywords occur in the child documents without occurring in the parent document, the parent document would not be in the candidate set.) Thus, the query model uses 'AND' semantics. However, the proposed schemes can be cost-effectively extended to apply to queries with 'OR' semantics as well by replacing the bitwise-AND operation with the bitwise-OR operation.

The relevance score $S_{Di}(a_j)$ of a given parent document $Di$ w.r.t. keyword $a_j$ is computed as the weighted average of two components (a) the frequency $f_{Di}(a_j)$ of $a_j$ in $Di$ and (b) the frequency $f_{Cw}(a_j)$ of $a_j$ in $Di$'s child documents. Notably, $S_{Di}(a_j) = 0$ if $a_j$ does not occur in the parent document (i.e., when $f_{Di}(a_j) = 0$), regardless of $a_j$'s frequency in child documents. To clarify this point, let us refer to the application scenario discussed previously. Suppose, the parent document pertaining to a user Alice contains details such as her date of birth, location, phone number, address and the list of items that she has purchased. Let the queried keyword(s) be related to chargers for phones. Now if she has not purchased any phone (as indicated from the information in the parent document), she would most likely not be complaining about charging issues for her phone. Hence, the queried keywords must appear in the parent document to be considered as part of the candidate set. Thus, Eq. 1 below for computing $S_{Di}(a_j)$ is only applicable when the parent document has at least one occurrence of $a_j$.

$$S_{Di}(a_j) = [\, w_1 \times f_{Di}(a_j) \,] + [\, w_2 \times \sum_{w=1}^{p_i} f_{Cw}(a_j) \,] \quad (1)$$

where $w_1$ and $w_2$ are weight coefficients such that $w_1 + w_2 = 1$. Here, $p_i$ is the number of child documents of $Di$. Thus, the relevance score of $Di$ w.r.t. a set of queried keywords is computed as $\sum_{j=1}^{n} S_{Di}(a_j)$, where $n$ is the number of queried keywords. Although this work has defined document relevance scores based on keyword frequencies, other methods of determining document relevance scores [19, 22, 24] can also be used in conjunction with our proposed schemes.

The following two special cases may arise in Eq. 1:

- $w_1 = 0$ : It implies that only the child documents contribute to the document relevance scores. In this work, since the effect of the parent documents must be considered, $w_1$ must always be greater than 0.
- $w_2 = 0$ : It implies that the child documents are not considered toward the relevance score computation. To our knowledge, this is the case for existing approaches.

The values of both $w_1$ and $w_2$ must be set greater than 0 because this work considers the effect of both parent and child documents on the relevance score. Thus, $0 < w_1, w_2 < 1$. Notably, the values of $w_1$ and $w_2$ depend upon the relative weights that should be allocated to the parent and child documents based on the application. Thus, there can be different ways of computing them. We leave the determination of appropriate values for $w_1$ and $w_2$ for various application scenarios to future work. As such, this work focuses on efficient top-$k$ query processing, given the values of $w_1$ and $w_2$. In this work, we set $w_1 = w_2 = 0.5$ (determined experimentally).

### 2.1 Example for Relevance Score Computation

Figure 3 depicts an illustrative example for the computation of the relevance scores of parent documents. In Fig. 3a, the parent and the child documents are D1 to D4 and C1 to C6, respectively. The child documents corresponding to D1, D2, D3 and D4 are {C1, C2, C3}, {C3, C5}, {C1, C5} and {C1, C2, C4, C6}, respectively. Consider a top-$k$ query with two keywords $a_1$ and $a_2$. Figure 3b presents the computation of relevance scores for D1 to D4 w.r.t. $a_1$ and $a_2$ in two ways i.e., with and without considering the effect of occurrences of the queried keywords in the child documents.

When the effect of keyword occurrences in the child documents is not considered, the relevance score $S_{Di}(a_j)$ is computed as the frequency $f(a_j)$ of $Di$. For example, $S_{D1}(a_1)$ equals 6 because $f(a_1)$=6 for D1. Similarly, $S_{D1}(a_2)$ equals 10. Hence, the relevance score of D1 w.r.t. keywords $a_1$ and $a_2$ is (10+6)=16. On the other hand, when the effect of keyword occurrences in the child documents is taken into account, suppose $w_1$=$w_2$=0.5, then $S_{D1}(a_1)$=[0.5×6]+ [0.5×(5+7+4)] (i.e., 11.0) because $f(a_1)$=6 for D1, and $f(a_1)$= {5, 7, 4} for D1's child documents {C1, C2, C3}, respectively. Similarly, we compute $S_{D1}(a_2)$ as 14.5. Hence, the relevance score of D1 w.r.t. keywords $a_1$ and $a_2$ is (11 + 14.5) = 25.5. Based on the relevance scores of the parent documents (see Fig. 3b), observe how the top-$k$ parent document result may differ when document relevance scores are computed with and without considering the effect of child documents.

Recall that as discussed in Sect. 1, we also address the case where the entities (more specifically, the parent document corresponding to each entity) are geo-tagged. We defer the discussion of the problem statement (as well as the solution) for the case of geo-tagged entities to Sect. 5.

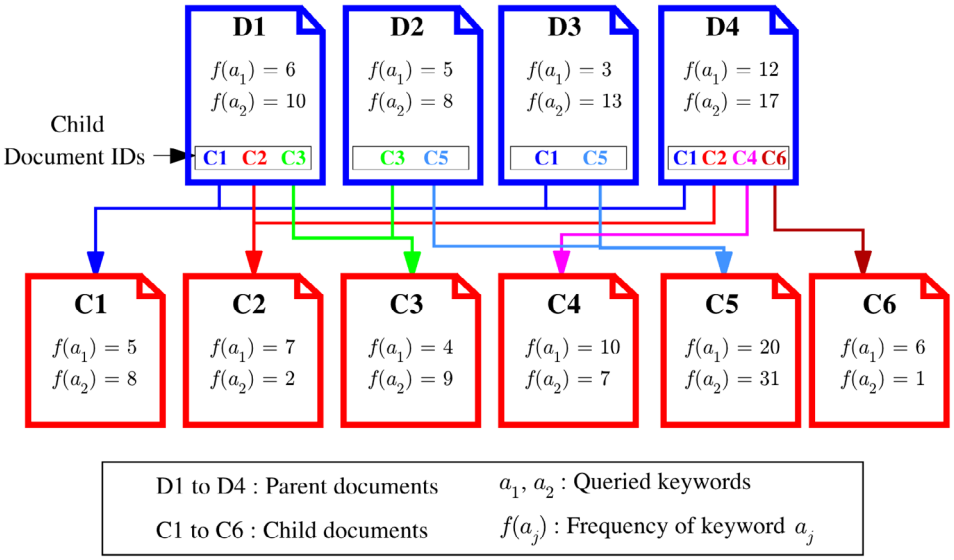
The next section discusses the *efficient* determination of the candidate set by considering the effect of both parent and child documents on the relevance score.

## 3 Top-k Query Processing Schemes

This section discusses the BSA and BSH top-$k$ query processing schemes. Both of these schemes use bitmaps for efficiently determining the candidate set. For relevance score lookups, BSA uses an array-based approach, while BSH uses the HI-tree index, which we present in Sect. 4. Now let us discuss a variant of the brute-force approach to obtain some insights concerning top-$k$ query processing.

### 3.1 A Variant of the Brute-Force Approach

Recall that a major drawback of the brute-force approach (see Sect. 1.2) is that it incurs prohibitively high processing cost in performing the exhaustive multi-way join




**(a)** Document corpus with embedded links

| Doc-ID | Without considering the effect of child documents | | | Considering the effect of child documents | | |
|---|---|---|---|---|---|---|
| | $S_{Di}(a_1)$ | $S_{Di}(a_2)$ | $S_{Di}(a_1) + S_{Di}(a_2)$ | $S_{Di}(a_1)$ | $S_{Di}(a_2)$ | $S_{Di}(a_1) + S_{Di}(a_2)$ |
| D1 | 6 | 10 | 16 | 11.0 | 14.5 | 25.5 |
| D2 | 5 | 8 | 13 | 14.5 | 24.0 | 38.5 |
| D3 | 3 | 13 | 16 | 14.0 | 26.0 | 40.0 |
| D4 | 12 | 17 | 29 | 20.0 | 17.5 | 37.5 |

**(b)** Computation of document relevance scores

**Fig. 3** Example for relevance score computation

processing step for determining the candidate set. Thus, it would not scale well when the number of documents is large. For example, consider a corpus of 1 million documents and a top-$k$ query with four keywords. Suppose each queried keyword occurs in any 10% of the documents in the corpus, but all the queried keywords need not necessarily occur together in the same documents. Here, the brute-force approach would need to intersect four document lists each containing 100,000 documents for determining the candidate set, thereby making the processing prohibitively expensive.

To address the scalability issue, we adapt a variant of Fagin's algorithm [14] as follows: (a) Sort each keyword's document list in descending order of keyword frequencies. (b) Consider only upto a fixed percentage of each sorted document list for the multi-way join processing step, thereby reducing the processing cost. Since the number of documents would generally vary across the document lists, this becomes effectively equivalent to making a '*zig-zag cut*' across the document lists. Hence, we designate this variant approach as the **Zig-zag Cut Scheme (ZCS)**. In ZCS, the number $k'_j$ of documents to be considered for the multi-way join processing step for the $j^{th}$ document list is computed as $P_D \times |L_j|$, where $P_D$ is the percentage factor, and $L_j$ is the document list corresponding to keyword $a_j$. We experimentally determine $P_D$ in Sect. 6.1.1.

For keeping track of the relevance scores of documents w.r.t. a given keyword, ZCS uses an inverted list structure, which is essentially similar to that of Tier 2 of the HI-tree, which we shall describe in detail in Sect. 4.2. In particular, as we shall see later, this structure stores the keyword frequencies of the parent and child documents for recomputing the relevance scores when updates occur.

Although ZCS reduces processing costs, it may degrade the *recall* performance. For example, given queried keywords $\{a_1, a_2\}$, suppose the frequencies of $a_1$ and $a_2$ in document D1 are 1000 and 2, respectively. Here, ZCS may fail to identify D1 as being in the candidate set. This is because $a_2$'s low frequency of occurrence in D1 may place D1 toward the end of the sorted document list of $a_2$, thereby in effect eliminating D1 from being considered for the multi-way join processing step. Now, we present an example for ZCS.

### 3.1.1 Illustrative Example for ZCS

Figure 4 depicts an example for ZCS using a corpus of documents D1 to D12. Consider a top-$k$ query with the keywords $\{a_1, a_2, a_3\}$ with their corresponding document lists $\{L_1, L_2, L_3\}$. In Fig. 4, $S$ represents the relevance

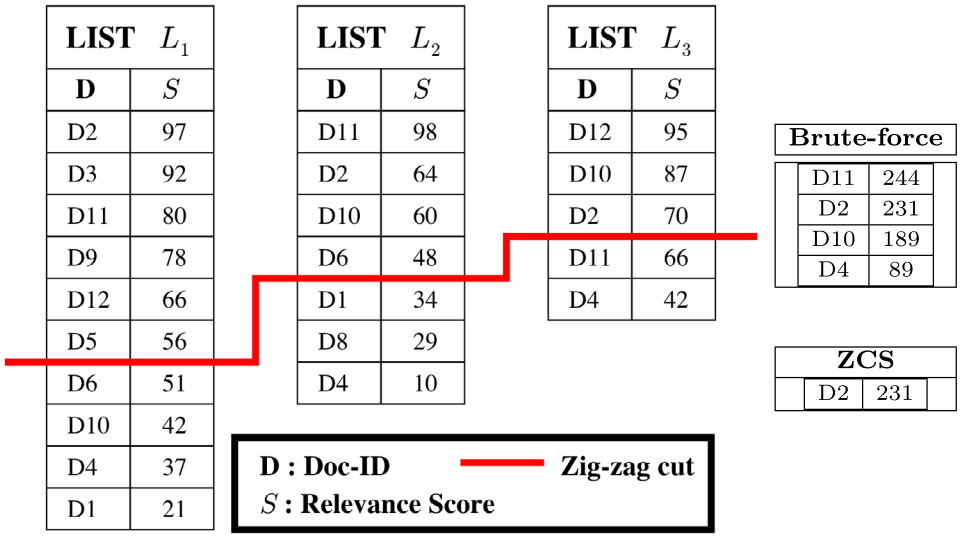



**Fig. 4** Illustrative example for ZCS (Zig-zag cut at $P_D = 60\%$) for top-4 query

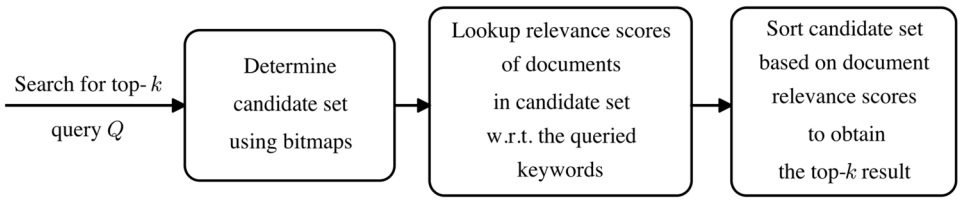



**Fig. 5** Big picture of top-$k$ query processing

score of keyword $a_j$ in document $Di$. For example, in list $L_1$, document D2 has a relevance score of 97 w.r.t. keyword $a_1$.

Observe that the candidate set is {D2, D4, D10, D11} because all the three queried keywords occur together only in these documents. Thus, as Fig. 4 shows, the brute-force approach *correctly* generates the top-$k$ result as top-4 result as {D11, D2, D10, D4} after computing document relevance scores w.r.t. all the queried keywords. (For simplicity, we compute the relevance score of a document w.r.t. all the queried keywords by summing up the relevance scores of each keyword across these documents.) In contrast, for ZCS, since $P_D$=60%, only 60% of the documents in each list are considered toward the top-$k$ query processing. Consequently, under ZCS, only D2 appears in the top-4 result set. Thus, in this example, ZCS has a recall of only 25% because it could provide only one (i.e., D2) out of the four correct results (i.e., D11, D2, D10, D4). Observe how ZCS trades off recall performance for lower processing costs.

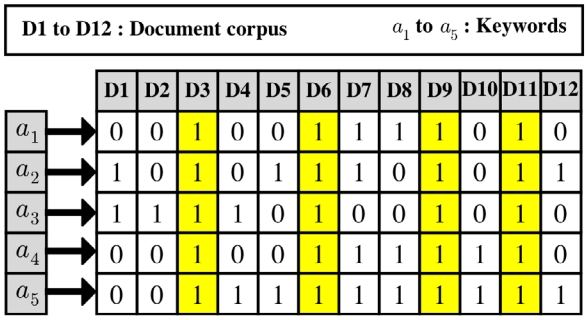
Based on the discussions in this section, there is a clear motivation for efficiently identifying the candidate set with high recall performance without incurring high processing cost. The big picture of our proposed three-step top-$k$ query processing approach is presented in Fig. 5. The subsequent sections describe these steps.

## 3.2 Determining the Candidate Set Using Bitmaps

Each keyword is associated with a bitmap, where each bit corresponds to a *parent document* in the corpus. A bit is set to 1 if the keyword occurs in the parent document, otherwise it is set to 0. The length of a keyword's bitmap is equal to the total number of parent documents in the corpus. Hence, if there are one million parent documents in the corpus, each keyword would be associated with a bitmap comprising one million bits. By performing a bitwise-AND operation on the bitmaps of the queried keywords, we identify the candidate set as comprising those parent documents, whose corresponding positions in the result bitmap are 1.

Suppose there are $N$ parent documents and $M$ keywords in the system. Thus, the total memory $Mem_{arr}$ consumed by the bitmap array equals $M \times N$ bits. When $N = 1$ million and $M = 40{,}000$, $Mem_{arr} = 40$ Gbits or approximately 5 GB.

Figure 6 shows an example for determining the candidate set using bitmaps. Figure 6a shows a document corpus comprising parent documents D1 to D12. Suppose the document corpus has five keywords $a_1$ to $a_5$. In Fig. 6a, the bitmap for keyword $a_1$ is 001001111010, thereby indicating that $a_1$ occurs in documents {D3, D6, D7, D8, D9, D11}. Similarly, the bitmap for $a_2$ is 101011101011, which suggests that $a_2$ exists in documents {D1, D3, D5, D6, D7, D9, D11, D12}.




| | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | D10 | D11 | D12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $a_1$ | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| $a_2$ | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| $a_3$ | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| $a_4$ | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| $a_5$ | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**(a)** Document corpus

| Queried keywords | bitwise-AND result | Candidate set |
|---|---|---|
| $a_2$, $a_3$ | 101001001010 | { D1, D3, D6, D9, D11 } |
| $a_4$, $a_5$ | 001001111110 | { D3, D6, D7, D8, D9, D10, D11 } |
| $a_1$, $a_2$, $a_3$ | 001001001010 | { D3, D6, D9, D11 } |
| $a_1$, $a_2$, $a_3$, $a_4$, $a_5$ | 001001001010 | { D3, D6, D9, D11 } |

**(b)** Candidate sets

**Fig. 6** Computation of candidate set using bitwise-AND operation

Figure 6b shows the computation of the candidate set for different sets of queried keywords using the bitwise-AND operation. Suppose the queried keywords are $a_2$ and $a_3$. Since the bitmaps for $a_2$ and $a_3$ are, respectively, 101011101011 and 111101001010, the bitwise-AND result is 101001001010. Hence, the candidate set comprises documents {D1, D3, D6, D9, D11}. Similarly, when the queried keywords are $a_1$ to $a_5$, the bitwise-AND result would be 001001001010. Thus, the candidate set comprises {D3, D6, D9, D11}. Observe how the bitmaps facilitate in quickly determining the candidate set without incurring prohibitively high costs.

Updates to the bitmaps can be reasonably expected to occur infrequently due to two reasons. First, the bitmaps relate *only* to the parent documents and are not impacted by updates occurring in the child documents. Recall that in our application scenarios, parent documents remain relatively static over time. Second, updates to the bitmaps are necessitated only when a given update contains a new keyword (that did not previously exist in the parent document) or all the occurrences of a given keyword are removed from a parent document. Thus, changes in the frequencies of keywords, which are already existing in the parent documents, do not necessitate bitmap updates.

In case of any updates to the parent documents, the bitmaps are modified as follows. If the update contains a keyword, which the parent document did not previously contain, the corresponding position in the bitmap is updated from 0 to 1. In Fig. 6, if a text update to document D5 contains the keyword $a_1$, the bit corresponding to D5 in the bitmap of $a_1$ would be updated to 1. Conversely, if an update results in the deletion of all the occurrences of a given keyword from a parent document, the corresponding position in the bitmap is updated from 1 to 0. In Fig. 6, if all the occurrences of $a_1$ were to be removed from document D3, the bit corresponding to D3 in the bitmap of $a_1$ would become 0.

Having determined the candidate set, we now need to look up the relevance scores for each document in the candidate set. Next, we describe how BSA performs this lookup.

### 3.3 BSA: A Bitmap Array-Based Technique for Looking Up Document Relevance Scores

Now we shall discuss the BSA scheme. For performing efficient lookups of document relevance scores, BSA uses a two-dimensional array structure, designated as *Arr*. An array entry *Arr(i,j)* indicates the relevance score of a given *parent document Di* w.r.t. keyword $a_j$. This relevance score is computed using Eq. 1 (see Sect. 2) by using the weighted keyword frequencies in both document *Di* and its child documents. In case of updates to the child documents, keyword frequencies may change, thereby necessitating *periodic* recomputation of the relevance score entries in *Arr(i,j)*. Thus, the relevance scores are updated in *Arr periodically* (i.e., not in real-time when the updates actually occur).

For recomputing a given relevance score, we store the keyword frequencies of the parent document and the *total* keyword frequencies of *all* its child documents in an auxiliary array data structure. Notably, the individual keyword frequencies for the child documents need not be stored because when child documents get updated, we can incrementally add or subtract from the total keyword frequency for the child documents. Such relevance score computations are lightweight even if the child documents get updated frequently, thereby implying that *Arr* is an update-efficient structure.

Observe how BSA enables efficient relevance score lookups in *O(1)* time for each document in the candidate set w.r.t. each queried keyword. Then for each candidate set document, it sums up the corresponding relevance scores for all the queried keywords to compute the document's relevance score. Finally, it sorts the documents in descending order of their relevance scores to obtain the top-*k* result.

Figure 7 depicts an illustrative example of BSA for the scenario presented in Fig. 6. In Fig. 7, the relevance score of a given parent document w.r.t. a given keyword $a_j$ is indicated as $S(a_j)$. Suppose the queried keywords are $a_1$ to $a_5$. Hence, based on Fig. 6b, the candidate set is {D3, D6, D9, D11}. Figure 7 illustrates how the relevance scores of each keyword for each of these four documents are summarized before being summed up to obtain the document relevance scores. For example, the retrieved relevance scores of $a_1$ to $a_5$ for D3 are {24, 16, 27, 39, 25}, respectively. Summing up these relevance scores, the relevance score of D3 w.r.t. the keywords $a_1$ to $a_5$ is 131. Finally, after sorting the documents, the top-3 documents are {D6, D11, D3}.

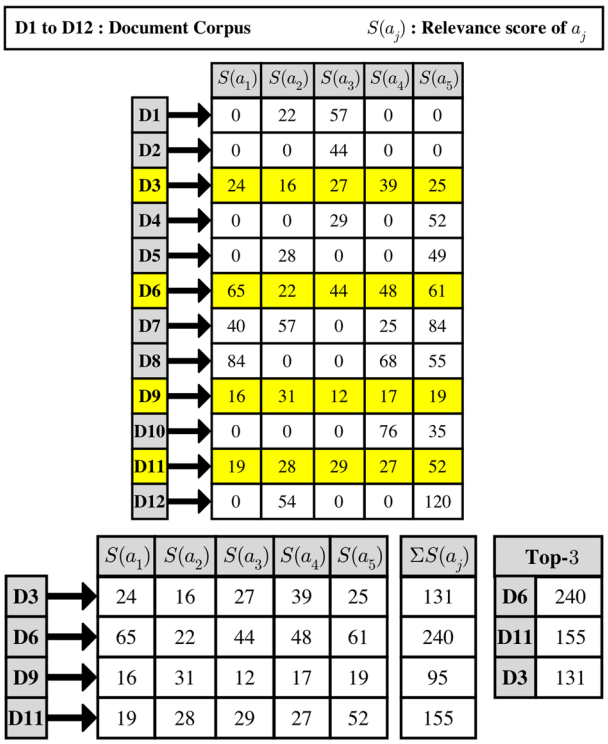



**Fig. 7** Illustrative example for BSA

Observe that although BSA enables quick keyword frequency lookups, it suffers from the drawback of consuming large amounts of memory space. In practice, a small percentage of popular keywords typically occur in a relatively large number of documents (i.e., high skew). Thus, the two-dimensional array used by BSA is likely to be sparse, thereby resulting in inefficient use of main memory space. Given $N$ parent documents and $M$ keywords, BSA consumes $MN$ bits for storing the bitmap array. Additionally, it consumes $4MN$ bytes for storing the document score lookup array. Hence, its total memory consumption is $4.125MN$ (i.e., $(1/8)MN + 4MN$) bytes. For example, when $M = 1$ million and $N = 40000$, its total memory consumption equals 165 GB. Thus, although BSA can be suitable for Cloud environments, which have large memory, we need to reduce its memory consumption for space-constrained environments.

This motivates us to propose the **BSAM approach**, which is a memory-efficient variant of the BSA approach. BSAM keeps both the bitmap arrays (i.e., the candidate set identification bitmap and the document relevance score lookup array) of only the popular query keywords in memory, while storing both the bitmap arrays of the relatively less popular keywords in the disk. When queries involve a mix of popular and unpopular keywords, BSAM would be able to do the processing of the popular keywords in memory, and it would need to access the disk for the less popular keywords. However, given that the vast majority of queries generally pertain to popular keywords, we do not expect BSAM to perform significantly worse than BSA.

Given a dictionary of 40000 keywords, suppose there are 2000 popular keywords. Given 1 million parent documents, the memory consumption of BSAM would be $(4.125 \times 2000 \times 10^6) = 8.25$ GB. Observe the significant reduction in memory consumption caused by BSAM as compared to that of BSA. Thus, when huge amounts of memory are not available, we recommend the use of BSAM. The next section presents another approach for memory-constrained environments, and this approach uses our proposed HI-tree index.

## 4 HI-tree: A Dynamic Index for Computing Relevance Scores

This section discusses the dynamic HI-tree index, which enables the efficient lookup of document relevance scores for facilitating top-$k$ query processing. The HI-tree is a two-tier index, which uses a combination of hashing and inverted indexes. In Tier 1 of the HI-tree, dictionary keywords are mapped into hash buckets, and the hash buckets are organized in the form of a $B^+$-tree to facilitate speedy access to any given hash bucket. Tier 2 comprises inverted indexes for facilitating keyword search within these hash buckets.

Notably, the HI-tree indexes only the *parent documents* because the top-$k$ result set, which is returned by our proposed schemes, comprises only the parent documents. However, as we shall see shortly, the HI-tree keeps track of keyword frequencies in the child documents to enable it to efficiently recompute document relevance scores in case of updates to the child documents.

### 4.1 Tier 1: Hash Buckets

In Tier 1 of the HI-tree, hashing is performed based on the notion of *pivots* on any given keyword $a_j$ for mapping dictionary keywords $a_1$ to $a_N$ into hash buckets $B_1$ to $B_m$ with the condition $m \lll N$. The number $P$ of pivots determines the number of hash buckets. When $P = 0$, only the first and the last characters in the keyword are considered for creating the hash buckets. When $P = 1$, the middle character is additionally considered[1]. In effect, the middle character partitions the keyword into two parts. When $P = 3$, the middle characters of these two partitions are additionally considered. Thus, a keyword is recursively partitioned depending upon the number $P$ of pivots to determine the hash buckets.

[1] For keywords comprising an even number of characters, we consider the left character among the two middle characters.

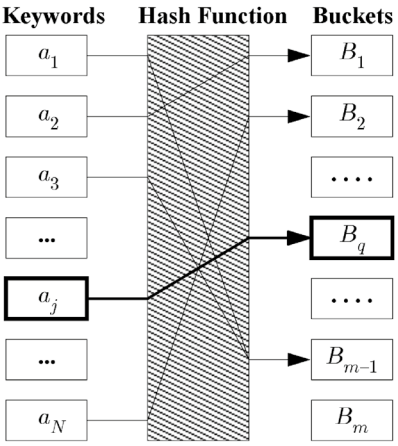



**Fig. 8** Mapping keywords into hash buckets

| Keyword | P=0 | P=1 | P=3 |
|---|---|---|---|
| procrastinate | pe | pse | posne |
| ameliorate | ae | aie | aeiae |
| circumlocution | cn | cln | ccltn |
| enjoy | ey | ejy | enjoy |
| only | oy | ony | ony |
| net | nt | net | net |
| equanimity | ey | eny | euniy |
| penultimate | pe | pte | pntae |

Figure 8 depicts Tier 1 of the HI-tree with an illustrative example. When $P = 0$, the keywords 'procrastinate' and 'ameliorate' get mapped to the hash buckets 'pe' and 'ae', respectively, because only the first and last characters of each keyword are considered. However, when $P = 1$, the same keywords get mapped to hash buckets 'pse' and 'aie', respectively, because the middle characters in these keywords are 's' and 'i', respectively. (For the even-length keyword 'ameliorate', we select the left character among the two middle characters 'i' and 'o'.) Thus, these two keywords are now partitioned around 's' and 'i', respectively. When $P = 3$, the same keywords get mapped to hash buckets 'posne' and 'amiae', respectively, because the middle characters {'o', 'n'} (for 'procrastinate') and {'m','a'} (for 'ameliorate') in the resulting recursive partitions additionally get considered.

Observe that different keywords may be mapped to the same hash bucket. In our example, when $P = 0$, the keywords 'procrastinate' and 'penultimate' both map to the same hash bucket 'pe' because their first and last characters are the same. Furthermore, a given keyword may be mapped to the same hash bucket for different values of $P$ e.g., 'net' gets mapped to the same hash bucket 'net' for $P = 1$ and $P = 3$ due to its short keyword-length.

Incidentally, the number $N_B$ of hash buckets increases dramatically as the value of $P$ increases. For example, when $P = 0$, $N_B = (26\times26) = 676$. However, when $P = 1$, $N_B = (26\times26\times26) = 17576$. Similarly, for $P = 3$ and $P = 5$, $N_B = 26^5$ and $26^9$ , respectively. The trade-off here is that lower values of $P$ result in a lower number of hash buckets, hence the relevant bucket can be found relatively quickly. But the search for a keyword within the relevant hash bucket consumes more time due to the relatively larger number of keywords in the bucket. On the other hand, higher values of $P$ result in a much larger number of hash buckets, hence finding the relevant hash bucket incurs more time. However, since most of the resulting buckets would be sparsely populated by keywords, the keyword-search time within a hash bucket is likely to decrease significantly.

Our preliminary experiments showed that $P = 1$ is a reasonable value to effectively balance the search for a hash bucket and the search for a keyword within that hash bucket.

Figure 9 illustrates how search is conducted for finding a given bucket. For quickly locating the hash bucket, which is relevant to a given keyword-search query, the hash buckets are organized in the form of a $B^+$-tree. The ordering of the hash buckets in the $B^+$-tree is based on the dictionary order of the words, which we had assigned (as identifiers) to these buckets. For example, the bucket 'aie' occurs to the left of the bucket 'eny' in the $B^+$-tree because the word 'aie' appears before the word 'eny' in dictionary order.

## 4.2 Tier 2: Searching for a Keyword Within a Given Hash Bucket

In Tier 2 of the HI-tree, the keywords within a given hash bucket are organized in the form of a $B^+$-tree structure, whose nodes are augmented with linked lists. The keyword ordering in the $B^+$-tree is based on the dictionary word order. Each node of the Tier 2 index contains entries of the form (*kword*, *LL*), where *kword* is a given dictionary keyword, and *LL* is a linked list. Each node of *LL* contains entries of the form (*doc_id*, *S*, $f_D$, $f_c$), where *doc_id* refers to the unique

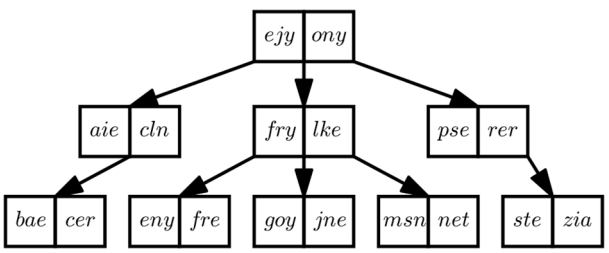



**Fig. 9** Ex: hash tree to find relevant bucket

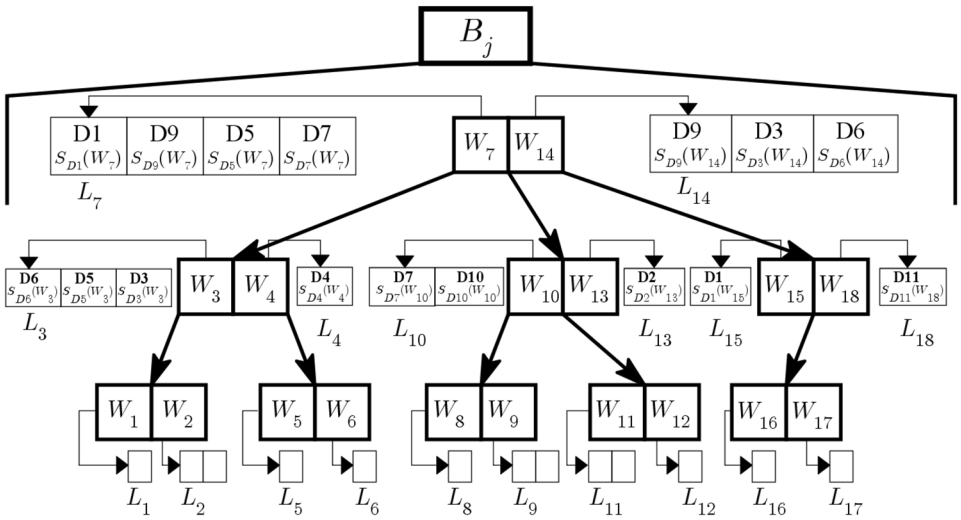



**Fig. 10** Ex: search within a given bucket *Bj*

identifier of the document containing *kword*, and *S* is the relevance score of the document w.r.t. *kword*. Here, $f_D$ and $f_c$ represent the frequencies of *kword* for the parent document and its child documents, respectively.

The values of $f_D$ and $f_c$ are required for recomputing the value of the relevance score *S* from Eq. 1 (Sect. 2), when updates to the child documents occur. We do not store the individual keyword frequencies for the child documents because when child documents get updated, we can incrementally add or subtract from the total keyword frequency for the child documents. Thus, the HI-tree is update-efficient as relevance score recomputations are lightweight even when the child documents get frequently updated.

Figure 10 shows an illustrative example of how the data are organized within a given bucket at Tier 2 of the HI-tree index. Here, $W_1$ to $W_{18}$ represent the keywords, while $L_1$ to $L_{18}$ are the respective document lists corresponding to these keywords. For simplicity, we do not show $f_D$ and $f_C$ in Fig. 10. Observe that $W_1$ would occur before $W_7$ in dictionary order because the keyword ordering in the B$^+$-tree is based on the dictionary order.

Updates to the child documents are *periodically* incorporated into the HI-tree as follows. In case of new text being inserted into any child document of a given parent document *Di*, the frequency of each keyword of the new text is first recorded. Then, for each keyword $a_j$, the HI-tree is searched. If $a_j$ exists in the HI-tree, the search proceeds through the linked list attached to $a_j$. In case *Di* occurs in this linked list, the value of $f_C$ in that linked list node is updated by adding the frequency of $a_j$ in the new text to the existing value of $f_C$. Otherwise, no action needs to be taken because the update did not impact any parent document. Recall our discussion on our application scenario concerning phone chargers as well as the discussion on the computation of relevance scores. Based on these discussions, a parent document needs to have at least one occurrence of the queried keyword to be considered for the top-*k* query result, irrespective of the frequency of the keyword in the child documents. Similarly, in case $a_j$ does not exist in the HI-tree, no action needs to be taken because there is no impact on any parent document.

Recall that in our application scenarios, parent documents remain relatively static over time. However, in case new text is added to any parent document, the above update procedure for child documents is applicable with some modifications. Suppose new text is inserted into a parent document *Di*. If *Di* does not exist in the linked list corresponding to a given keyword $a_j$ (which occurs in the new text), an additional node is added to the linked list and populated with the values of $f_D$ and $f_C$ from which the relevance score S is computed for *Di*. (For determining the value of $f_C$, all the child documents of *Di* need to be parsed to obtain the frequency of $a_j$ in the child documents.) Furthermore, if $a_j$ does not exist in the HI-tree, it is inserted in the HI-tree based on dictionary word order, and its corresponding linked list is populated as discussed above. Due to space constraints, we do not discuss how text deletions are handled by the HI-tree. However, deletions follow similar intuition as that of insertions in the HI-tree.

## 5 R-tree-Based Extension of Our Proposed Approaches for Geo-Tagged Entities

In this section, we shall discuss an R-tree-based extension of our proposed schemes to address the case for geo-tagged entities.

### 5.1 Context and Problem Statement

Recall our application scenario in which a given company may wish to find its top-*k* customers, who complained about

phone charger problems, *from North America*. Moreover, recall that we model each entity by a single parent document and possibly multiple child documents, which are essentially embedded links in the given parent document. As discussed previously in Sect. 1, in case of geo-tagged entities, we assume that the parent document of a given entity is associated with a geographical location i.e., a point in space, while the child documents need not necessarily be geo-tagged. In particular, the location attribute of the parent document of an entity could be at any specific spatial granularity e.g., home address, city, state or country. Moreover, a given parent document is associated with only a *single* point in space. Observe that it is possible for multiple parent documents to be geo-tagged to the same point in space. As we shall see shortly, our proposed R-tree-based scheme works effectively in such cases as well. Notably, even if some of the child documents are geo-tagged, this work does not exploit the geo-tagging of the child documents. This is because the location information of the child documents does not add any significant value to our application scenarios, as discussed previously in Sect. 1.

Given the above context, we extend our problem statement of Sect. 2 as follows. Consider $D$ as the corpus of $N$ parent documents {D1, D2, ...,D$N$} (one parent document per entity) such that each document $Di$ contains $p_i$ embedded links $\{L_1, L_2, \ldots, L_{p_i}\}$. Each link points to a single document (child document). Given a spatial query window and a set of query keywords, our problem is to *efficiently* retrieve the top-$k$ entities (parent documents) that occur within the spatial query window. Here, the notion of top-$k$ is based on the frequency of occurrence of the queried keywords, as discussed earlier in the paper. Similar to the discussions in Sect. 2, our query model for geo-tagged entities also uses 'AND' semantics.

## 5.2 R-tree-Based Approach for Geo-Tagged Entities

Given the location in space (i.e., the ($x$, $y$) coordinates) of the parent document of each entity, we use an R-tree [16] to index the location of the parent document of each entity in space. Notably, we use the standard R-tree creation algorithm in [16] to build the R-tree. The non-leaf nodes of the R-tree are of the form (*ptr*, *mbr*), where *ptr* is a pointer to the child node in the R-tree, and *mbr* is the MBR (Minimum Bounding Rectangle), which covers all of the MBRs in that child node. The leaf nodes of the R-tree contain pointers to the database of the actual parent documents that occur within the MBR of that leaf node.

Figure 11 depicts an illustrative example of the respective locations of the parent documents in space. Each dot/point in Fig. 11 represents the location of a parent document. Figure 12 depicts the structure of the R-tree corresponding to the distribution of parent documents in space shown in Fig. 11. Observe how the universal space is divided into three MBRs, namely R1, R2 and R3. Observe how R1 is further sub-divided into MBRs A, B and C; R2 is further sub-divided into D, E and F, and R3 is further sub-divided into G, H and I.

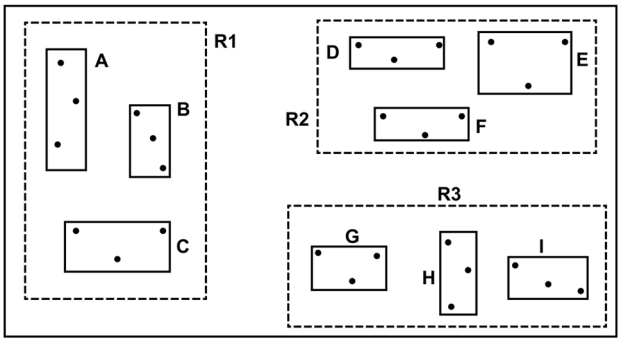



**Fig. 11** An illustrative example for the distribution of parent documents in space

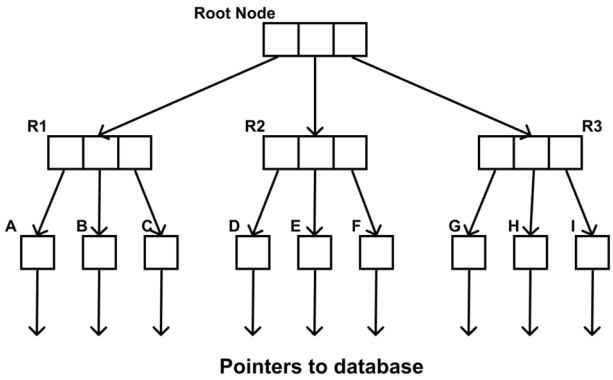



**Fig. 12** Illustrative example for the R-tree index structure

In our R-tree-based extension scheme for geo-tagged entities, we first obtain the candidate set $CS1$ of parent documents (along with their relevance scores) by using our proposed BSA, BSAM and BSH schemes. Then, given a spatial query window, we can quickly retrieve the parent documents that exist within the query window; let us denote this retrieved set of parent documents as $CS2$. (We shall discuss the details of how the set $CS2$ is obtained in the subsequent paragraph.) Finally, we perform an intersection of the sets $CS1$ and $CS2$ to obtain set $CS3$. Then we sort the parent documents in $CS3$ in descending order on the basis of the relevance scores of the parent documents for finding the top-$k$ parent documents (entities) that are relevant to the spatial window query. Figure 13 depicts a schematic diagram of the process for obtaining the top-$k$ geo-tagged entities.

Now let us discuss the details of how the set $CS2$ is obtained. Once the R-tree has been built, we use the R-tree search algorithm [16] to retrieve the parent documents

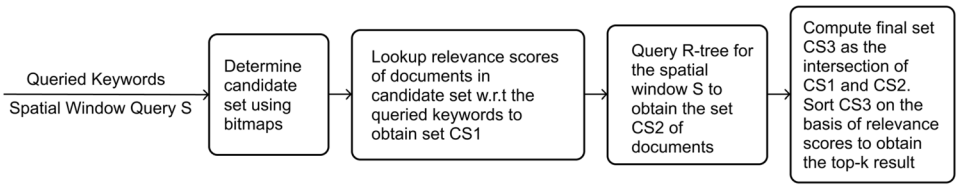



**Fig. 13** Big picture of top-$k$ spatial query processing for geo-tagged entities

that occur within the input spatial query window. Observe that the input spatial query window $S$ can be *at any spatial granularity*, and the query window (rectangle) can be of any size (area). Given $S$, the R-tree search algorithm follows a top-down traversal of the nodes starting at the root node of the R-tree. In particular, the R-tree search algorithm works by using the following recursive approach. If a node is not a leaf node, it checks if $S$ intersects with the corresponding MBR of the node. If so, then for every node element, whose MBR intersects $S$, we recursively call the search function for the sub-tree that is rooted in that node element. On the other hand, if the node is a leaf node, then for each node element, which is contained within $S$, we follow the pointers to the database of the actual parent documents, and then we add the corresponding parent documents to the set $CS2$.

Notably, we shall henceforth refer to the R-tree-based spatial extensions of our proposed BSH, BSA and BSAM schemes as **S_ BSH**, **S_BSA** and **S_BSAM,** respectively.

## 6 Performance Evaluation

This section reports the performance evaluation using both real and synthetic datasets. We divide our performance study into two parts: (a) the case where the entities are *not* geo-tagged and (b) the case where the entities are geo-tagged.

### 6.1 Performance Study for the Case of Non-Geo-Tagged Entities

The synthetic dataset had 1 million parent documents. For each parent document, the number of child documents was randomly selected to be between 8 and 12. The experiments use a set of 40,000 unique dictionary-keywords (maximum length of 25 characters). Each of parent and child documents have 1000 unique keywords, their frequencies randomly varying from 5 to 40.

In practice, a few popular keywords receive a large percentage of the queries. Hence, we classify the 40,000 keywords into 2000 popular keywords, the rest being regarded as unpopular keywords. Thus, for BSAM, the bitmap arrays of only these 2000 popular keywords were kept in memory, while the rest were stored on the disk. Here, we consider 2000 popular keywords as an example, although the actual number of popular keywords could be significantly less depending upon the application.

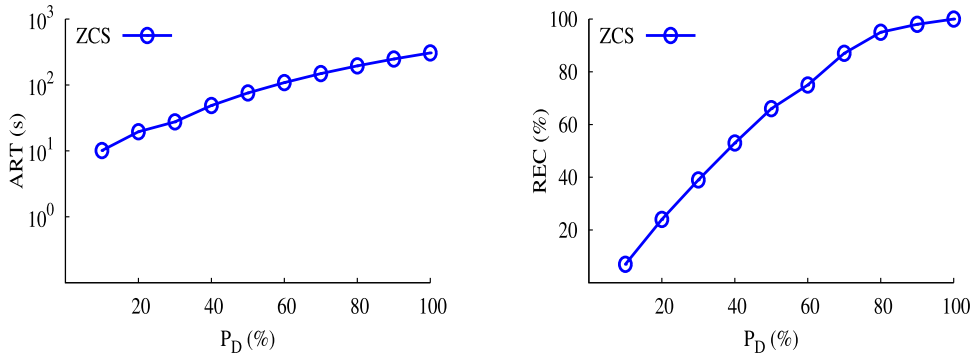
Table 1 summarizes the parameters used in the performance study.

The experiments consider 10 different keyword classes. The number of keywords in each keyword class is determined based on a Zipf distribution with a zipf factor of 0.7 (i.e., high skew). The 40,000 unique dictionary-keywords are randomly assigned to any *one* of these keyword classes. Furthermore, 40,000 keywords are distributed across the total number of documents with a zipf factor $ZF_D$ of 0.7 (i.e., high skew). Here, we consider 10 different document buckets, in which the keywords are randomly assigned from any *one* of the 10 keyword classes. Moreover, the number of queries corresponding to each keyword class is determined based on a Zipf distribution with a zipf factor $ZF_Q$ over 10 query buckets. Consistent with real-world scenarios, we set the value of $ZF_Q$ to 0.7 (i.e., high skew) to ensure that a relatively small percentage of keywords receive a disproportionately large number of queries.

We could not obtain an SOE-related real dataset for our motivating CRM application scenario because such SOEs are still mostly beginning to be deployed. Hence, for our experiments, we used real data from the virtualtourist.com travel website for modeling parent and child documents. The real dataset used in the experiments comprised a total of 100,000 webpages (as the parent documents) from the virtualtourist.com travel website. Each of these web pages had 10 embedded links, each link pointing to the homepage of a specific hotel. (Each such homepage contains details such as hotel news, prices and user review comments). Thus, the documents pointed to by each of these 10 links were downloaded to serve as the child documents for the real dataset.

**Table 1** Parameters of the performance study

| Parameter | Default | Variations |
|---|---|---|
| Percentage of documents: $P_D$ | 60 | 20, 40, 80, 100 |
| No. of documents: $N_D$ ($10^5$) | 10 | 2, 4, 6, 8 |
| No. of queried keywords: $N_{QK}$ | 5 | 2, 3, 4, 6 |
| $k$ (of top-$k$ query) | 100 | |
| Zipf factor for Queries: $ZF_Q$ | 0.7 | |
| Zipf factor for Data: $ZF_D$ | 0.7 | 0.1, 0.3, 0.5, 0.9 |




**(a)** ART **(b)** REC

**Fig. 14** Determining percentage factor for ZCS

The performance metric is **average response time (ART)** of queries. $ART = \frac{1}{N_Q} \sum_{q=1}^{N_Q} t_r$, where $t_r$ is the query response time, and $N_Q$ is the total number of queries. Additionally, to quantify the top-$k$ result accuracy, **query recall percentage (REC)** is used as a performance metric. REC quantifies the percentage of correct top-$k$ documents that a given scheme was able to retrieve. Notably, REC is computed as the average query recall percentage over *all* the queries. Thus, $REC = \frac{1}{N_Q} \sum_{q=1}^{N_Q} [|Tc \cap Tr| / |Tc|]$, where $Tc$ represents the document set comprising the correct top-$k$ query result, while $Tr$ is the document set comprising the top-$k$ query result retrieved by a given scheme. For example, if the correct top-5 result were D1 to D5, and the top-$k$ result retrieved by a given scheme was {D1, D2, D3, D8, D10}, REC would be 60% because only three out of the five correct top-5 documents were retrieved. Furthermore, we also show the memory consumption for each of the proposed approaches in our experiments.

Our experiments were run on a machine with Intel Core-2-Duo T6600 2.2GHz processor with 64-bit OS. It had only 16 GB RAM. Recall that more than 160 GB of RAM is needed for running BSA completely in memory because BSA consumes huge amounts of memory due to the bitmap array for document relevance score lookups. Hence, we implemented BSA by keeping the relevance score lookup array in the disk and loaded it into the memory in 8 GB chunks, thereby increasing BSA's processing time. We ran each experiment 10 times and averaged the results. The confidence intervals for our experiments ranged from 92–95%.

As reference, we use the ZCS scheme (see Sect. 3.1). Recall that ZCS requires as input a percentage factor $P_D$, which addresses the trade-off between recall performance and query response times. Now, we find $P_D$ experimentally.

#### 6.1.1 Determining the Percentage Factor for ZCS

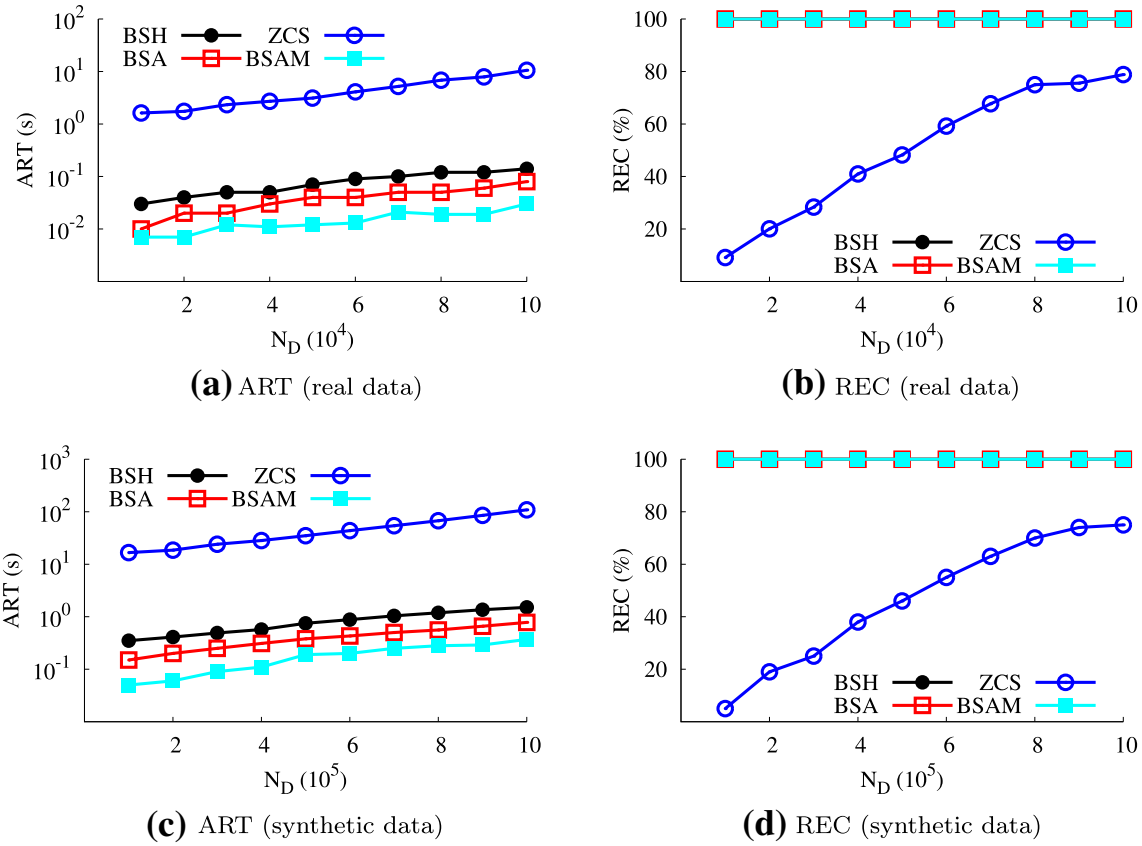
Figure 14 depicts the results of our experiment for determining the percentage factor $P_D$ of documents that are considered for the multi-way join processing step in case of ZCS.

As $P_D$ is increased, a larger percentage of the documents in the corpus are considered for the multi-way join processing step, thereby improving REC. However, with increase in $P_D$, ART also keeps increasing significantly due to costly multi-way join computations across a larger number of documents. The results in Fig. 14 show that beyond $P_D = 80\%$, REC exhibits a saturation effect. This occurs because at this point, REC is already high (i.e., 80%) due to considering a large percentage of the documents. Hence, further increasing the number of documents to be processed does not significantly improve REC. However, ART keeps increasing beyond $P_D = 80\%$ due to the computational costs of processing a larger number of documents.

Observe the trade-off between REC and ART i.e., increased recall can be obtained at higher processing cost. The results of this experiment suggest that at $P_D = 60\%$, a reasonably good value of REC (i.e., 75%) can be obtained without incurring prohibitively high ART. Hence, we set $P_D = 60\%$ for all the experiments involving ZCS.

#### 6.1.2 Effect of Varying the Number of Documents

Figure 15a and b depict the effect of varying the number $N_D$ of documents for the real dataset. As $N_D$ increases, ART increases for all the schemes due to the increased computational costs of processing a larger number of documents. However, ART increases at a much slower rate for the proposed BSH, BSA and BSAM schemes than for




**(a)** ART (real data)

**(b)** REC (real data)

**(c)** ART (synthetic data)

**(d)** REC (synthetic data)

| Scheme | real data | synthetic data |
|---|---|---|
| **BSH** | 0.81 GB | 5.22 GB |
| **BSA** | 8.56 GB | 13.38 GB |
| **BSAM** | 0.87 GB | 8.25 GB |
| **ZCS** | 0.22 GB | 0.43 GB |

**(e)** Memory consumption

**Fig. 15** Effect of varying the number of documents

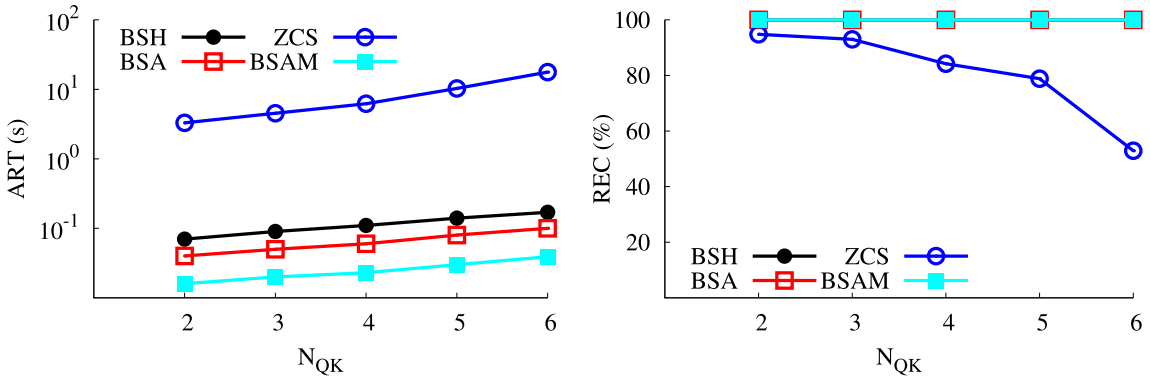
ZCS. This occurs because these schemes compute the candidate set efficiently by using the bitwise-AND operation, while ZCS uses costly multi-way join processing.

BSAM incurs lower ART than BSA because it reduces memory space consumption by exploiting skews in keyword popularity. Since most of the queries concern popular keywords, BSAM is able to avoid disk accesses for most of the queries, thereby saving on query processing times. On the other hand, BSA incurs significant processing times due to loading 8 GB chunks from disk into memory, thereby increasing ART. However, BSA incurs lower ART than BSH because its array-based structure allows it to look up document relevance scores quicker than BSH, which requires HI-tree index traversal for document relevance score lookups.

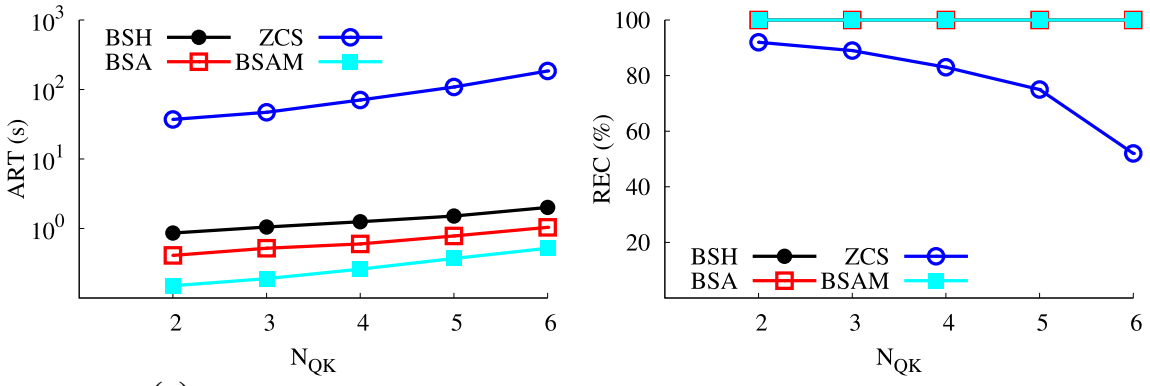
As $N_D$ increases, REC remains constant at its highest possible value (i.e., 100%) for BSH, BSA and BSAM. This occurs because the bitwise-AND operation used by these schemes is able to efficiently generate the candidate set with 100% recall. For ZCS, REC increases due to the involvement of an increased number of documents toward the top-$k$ processing.

The results indicate that both BSH and BSA indeed scale well w.r.t. the number of documents. To further investigate the scalability of our proposed schemes, we varied the number of documents with the synthetic dataset comprising 1 million documents. Figure 15c and d depict the results of this experiment. Observe that the results for real and synthetic datasets exhibit similar trend. ART varies across these results due to differences in dataset size.

Figure 15e shows the memory consumption for each scheme. Recall that the real dataset comprises 100,000 parent documents and their corresponding child documents. In




**(a)** ART (real data) **(b)** REC (real data)




**(c)** ART (synthetic data) **(d)** REC (synthetic data)

| Scheme | real data | synthetic data |
|---|---|---|
| **BSH** | 0.85 GB | 5.41 GB |
| **BSA** | 8.69 GB | 13.52 GB |
| **BSAM** | 0.91 GB | 8.44 GB |
| **ZCS** | 0.27 GB | 0.61 GB |

**(e)** Memory consumption

**Fig. 16** Effect of varying the number of queried keywords

case of the real dataset, observe that BSH and ZCS consume significantly less memory than that of BSA. This occurs because BSH and ZCS mostly use the disk for storage, while BSA is an in-memory approach. BSH incurs more memory consumption than ZCS because it finds the candidate set using bitmaps that reside in memory. Moreover, BSAM consumes less memory than BSA because it keeps only the bitmap arrays of the 2000 most popular keywords in memory.

For the synthetic dataset, observe the dramatic increase in memory consumption for BSH from 0.81 GB (real dataset) to 5.22 GB (synthetic dataset). This increase occurs because the bitmaps maintained in the memory by BSH (for candidate set selection) increase in size as the number of parent documents in the corpus increases. The increase in the memory consumption of BSA from 8.56 GB (real dataset) to 13.38 GB (synthetic dataset) can be explained in a similar manner i.e., this increase occurs because the synthetic dataset is significantly larger than the real dataset. The increase in memory consumption for BSAM across the real and synthetic dataset can also be explained by the relative dataset size differences between these two datasets. Finally, observe that the memory consumption of ZCS is not significantly impacted by increase in dataset size primarily because ZCS primarily uses the disk for storage.

### 6.1.3 Effect of Varying the Number of Queried Keywords

Figure 16a and b depict the effect of varying the number $N_{QK}$ of queried keywords for the real dataset.

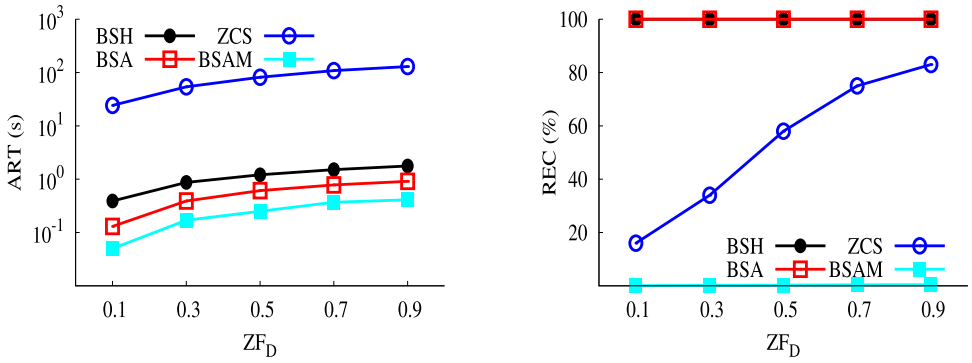



**Fig. 17** Effect of skew in keywords distribution

**(a)** ART (synthetic data) **(b)** REC (synthetic data)

As $N_{QK}$ increases, ART increases for BSH, BSA and BSAM because the bitwise-AND operation is performed on an increased number of bitmaps. (Recall that each keyword corresponds to one bitmap.) REC remains constant at 100% for these schemes due to the recall accuracy of the bitwise-AND operation, as explained for the results in Fig. 15.

With increase in $N_{QK}$, ART increases more for ZCS than the proposed schemes because join operations need to be performed across a larger number of document lists (one document list/keyword). For example, when $N_{QK} = 3$, it implies a three-way join, while for $N_{QK} = 5$, a more costly five-way join is required. As $N_{QK}$ increases, ZCS exhibits lower values of REC partly because performing the multi-way join step across only 60% of the documents per document list degrades the recall performance and partly due to the reasons explained for the results in Fig. 15. Figure 16c and d depict the results of this experiment for the synthetic data. Notably, the results for real and synthetic datasets exhibit similar trend. Figure 16e shows the memory consumption for each scheme. The explanation for the results in Fig. 16e is essentially similar to that of Fig. 15e.

#### 6.1.4 Effect of Skew in Keywords Distribution

This experiment examines the effect of skew in the distribution of the keywords across the documents. Recall that higher values of $ZF_D$ imply that a small percentage of the keywords (i.e., the 'popular' keywords) occur in a large percentage of the documents. Figure 17 depicts the results.

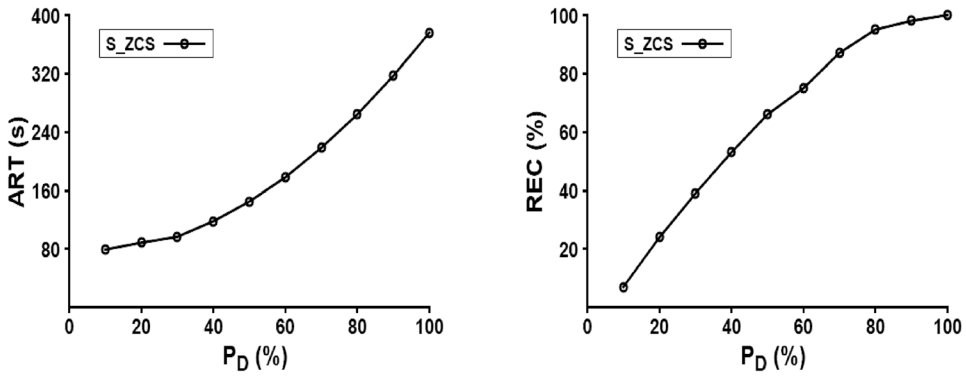
For this experiment, queries were also highly skewed toward a small number of these 'popular' keywords (i.e., using the default value of $ZF_Q = 0.7$). Hence, as $ZF_D$ increases, ART increases for all the schemes due to increase in the size of the document lists associated with each of the 'popular' queried keywords, thereby necessitating more documents to be processed. Moreover, as $ZF_D$ increases, REC increases for ZCS primarily due to the involvement of a larger number of documents toward the top-$k$ query processing.

### 6.2 Performance Study for the Case of Geo-Tagged Entities

The experimental setup in case of our performance study on geo-tagged entities involved the same synthetic and real datasets, which had been created for the performance study in Sect. 6.1. Recall that Sect. 6.1 also provides a detailed description concerning the generation of the synthetic dataset. Moreover, the parameters of the performance study (and their associated values) remain the same as in the case of the performance study in Sect. 6.1. Furthermore, we use the same performance metrics i.e., **average response time (ART)** of queries and **query recall percentage (REC)**, as defined in Sect. 6.1.

In this section, the key difference from the experimental setup of Sect. 6.1 is that we assigned $(x, y)$ coordinates to each parent document for both the real dataset and the synthetic dataset as follows. For assigning the $(x, y)$ coordinates to the parent documents, the universe was divided into an $8x$ 8 grid. Each parent document was assigned to a particular grid cell by using a Zipf distribution with a skew of 0.7 (i.e., high skew). Given a parent document, which has been assigned to a grid cell, its $(x, y)$ coordinate was randomly chosen to fall within the bounding region of the grid cell.

Once the positioning of the parent documents was completed, an R-tree was built to index the locations of the parent documents. In the implementation of the R-tree, each MBR was represented using two $(x, y)$ coordinates. We used a branching factor of 128. We selected the disk page size to be 16 KB, and each R-tree node is assumed to fit in a disk page. Hence, for our experiments, one disk I/O is counted as one disk page access or equivalently one R-tree node access. Notably, in the performance results, we present ART (average response time) of queries as a metric; ART includes the effect of disk I/Os and the query processing times for in-memory processing. Due to the disk-resident nature of the R-tree, ART is dominated by the effect of disk I/Os incurred due to the traversal of the R-tree.




**(a)** ART **(b)** REC

**Fig. 18** Determining Percentage Factor for S_ZCS

For querying in case of this approach, we introduced a new parameter, which we designated as the **query window size** ($Q_S$). This is the percentage of the overall universal space that we wish to query. The default value of $Q_S$ was chosen to be 8%. Furthermore, the position of the query window was selected as follows. We randomly selected a point in the universal space and created a rectangle of size (area) $Q_S$ with that point as the centroid of the rectangle. This rectangle is the spatial query window.

Recall that for the case of geo-spatial entities, as discussed in Sect. 5, we had designated our proposed BSH, BSA and BSAM schemes as S_BSH, S_BSA and S_BSAM, respectively. Furthermore, as reference, we use the ZCS scheme (as in the case of Sect. 6.1) in conjunction with the R-tree to find the top-$k$ results for the geo-tagged entities. In other words, the candidate set of parent documents is first retrieved by the ZCS scheme. Then we intersect these set of parent documents with the set of parent documents, which occur within the spatial query window, to obtain the top-$k$ query result. We shall henceforth refer to this scheme as the **S_ZCS** scheme. Recall that our reference ZCS scheme (see Sect. 3.1) requires as input a percentage factor $P_D$, which addresses the trade-off between recall performance and query response times. Now, we shall find $P_D$ experimentally for the S_ZCS reference scheme.

#### 6.2.1 Determining the Percentage Factor for S_ZCS

Figure 18 depicts the results of our experiment for determining the percentage factor $P_D$ of documents that are considered for the multi-way join processing step in case of S_ZCS. The trends for the results in Fig. 18 are comparable to those of Fig. 14. This occurs because the additional processing time incurred by the R-tree introduces a similar increase in ART for all the possible values of the percentage factor ($P_D$) in case of S_ZCS. The explanation for the results in Fig. 18 essentially follow those of the results in Fig. 14. In other words, as $P_D$ is increased, a larger percentage of the documents in the corpus are considered for the multi-way join processing step, thereby improving REC albeit at the cost of increased ART.

Observe the trade-off between REC and ART i.e., increased recall can be obtained at higher processing cost. The results of this experiment suggest that at $P_D = 60\%$, a reasonably good value of REC can be obtained without incurring prohibitively high ART. Hence, we set $P_D = 60\%$ for all the experiments involving S_ZCS.

#### 6.2.2 Effect of Varying the Number of Documents

The results in Fig. 19 depict the effect of varying the number $N_D$ of documents. The explanations for the results in Fig. 19 essentially follow the explanations for the results in Fig. 15. In other words, as $N_D$ increases, ART increases for all the schemes because a larger number of documents needs to be processed. However, ART increases at a much slower rate for our proposed S_BSH, S_BSA and S_BSAM schemes than for S_ZCS. This occurs because these schemes compute the candidate set efficiently by using the bitwise-AND operation, while S_ZCS uses expensive multi-way join processing. Notably, the results for REC also follow the same trend as the results in Fig. 15 due to the same reasons, as discussed for the results of Fig. 15.

Observe how ART is comparable for our proposed S_BSH, S_BSA and S_BSAM schemes at any given value of $N_D$. This is because R-tree is a disk-resident data structure, and it requires a significant amount of disk I/Os for querying. Therefore, the cost of querying the R-tree dominates the cost of querying the parent documents in all of our proposed schemes. This dominant effect of the R-tree results in the graphs of the S_BSA, S_BSAM and S_BSH schemes to coincide. However, the ART incurred by the S_ZCS scheme is still significantly higher than the ART incurred by our three proposed schemes. This is because of the multi-way join operation used in case of the S_ZCS scheme.

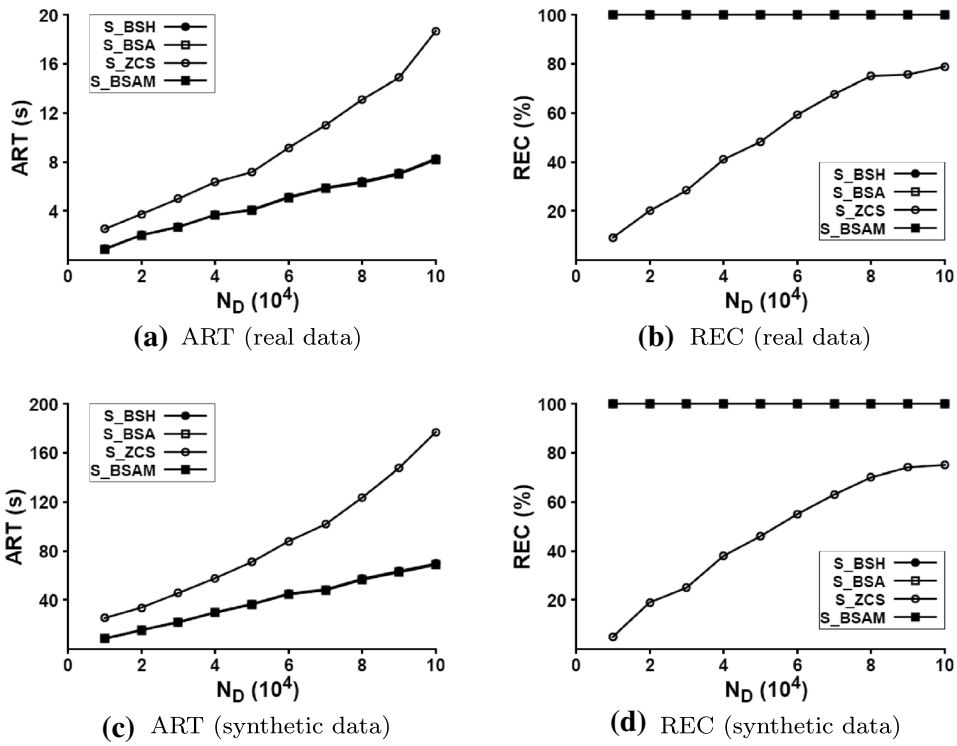



**Fig. 19** Effect of varying the number of documents

**(a)** ART (real data) **(b)** REC (real data)

**(c)** ART (synthetic data) **(d)** REC (synthetic data)

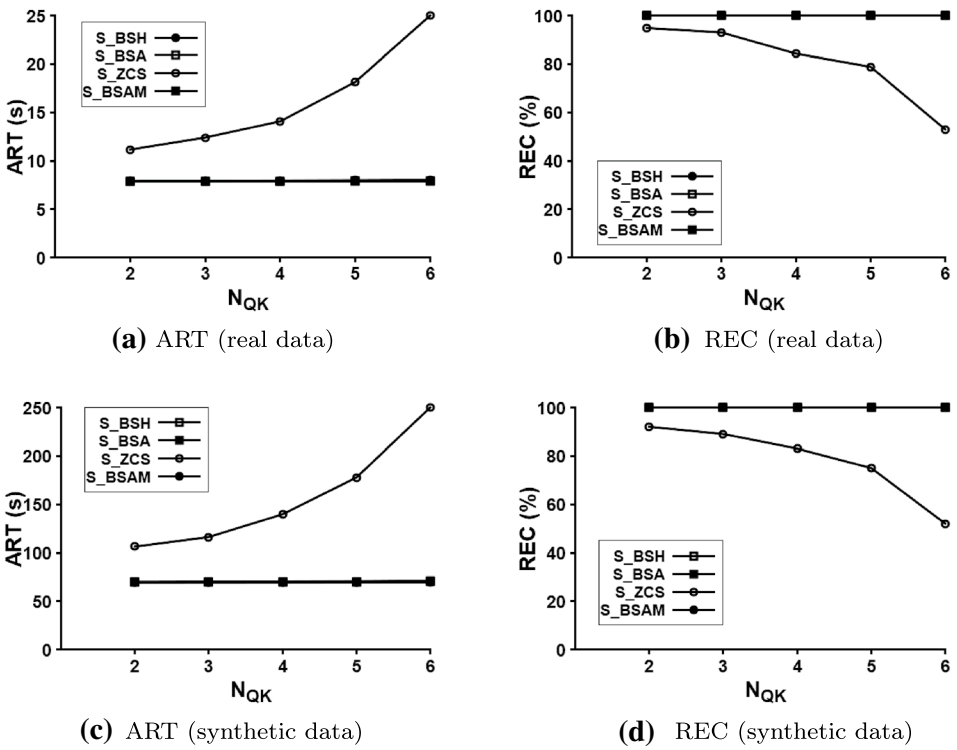


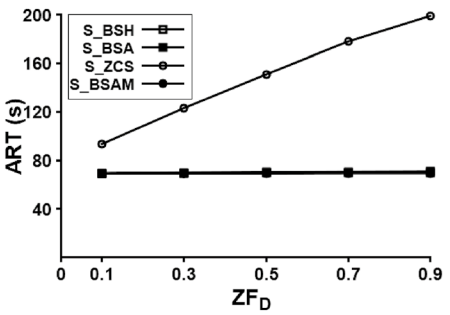

**Fig. 20** Effect of varying the number of queried keywords

**(a)** ART (real data) **(b)** REC (real data)

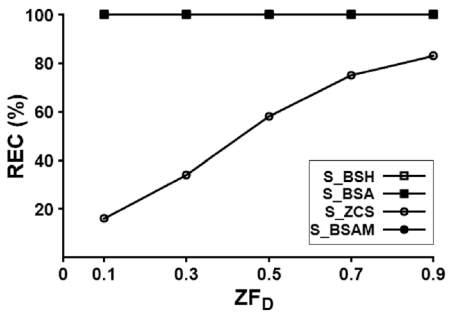
**(c)** ART (synthetic data) **(d)** REC (synthetic data)

### 6.2.3 Effect of Varying the Number of Queried Keywords

The results in Fig. 20 depict the effect of varying the number $N_{QK}$ of queried keywords. Varying the number of queried keywords has no effect on the query processing time incurred by the R-tree. Thus, the results for ART follow the same trend as the results of Fig. 16 due to the same reasons, as discussed for the results of Fig. 16. However, due to the disk-resident nature of the R-tree, the cost of querying the R-tree has a dominating effect, thereby causing the graphs

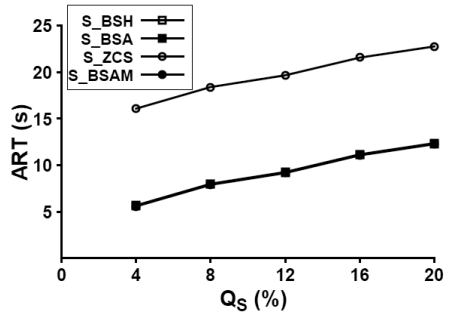



**(a)** ART (synthetic data)

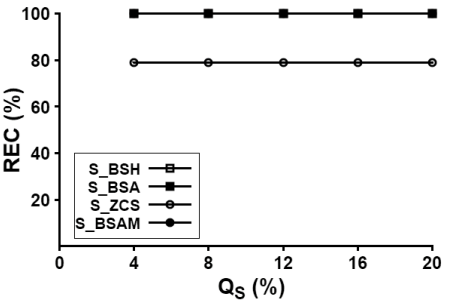



**(b)** REC (synthetic data)

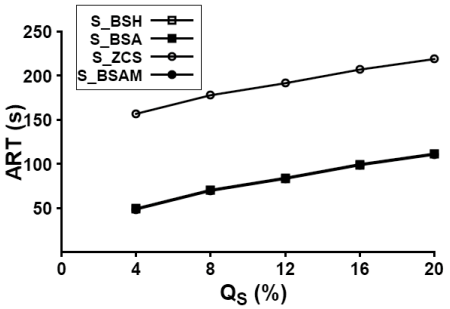
**Fig. 21** Effect of skew in keywords distribution



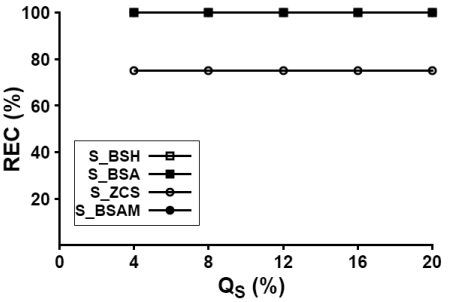

**(a)** ART (real data)




**(b)** REC (real data)




**(c)** ART (synthetic data)




**(d)** REC (synthetic data)

**Fig. 22** Effect of varying the spatial query window size

obtained for the S_BSA, S_BSAM, S_BSH schemes to coincide. The results for REC also follow the same trend as the results of Fig. 16 due to the same reasons, as explained for the results of Fig. 16.

#### 6.2.4 Effect of Skew in Keywords Distribution

The results in Fig. 21 depict the effect of skew in the distribution of the keywords across the documents. Recall that higher values of $ZF_D$ imply that a small percentage of the keywords (i.e., the 'popular' keywords) occur in a large percentage of the documents. For this experiment, queries were also highly skewed toward a small number of these 'popular' keywords (i.e., using the default value of $ZF_Q = 0.7$).

Notably, the skew in the distribution of the keywords has no effect on the spatial window query processing time of the R-tree. This is because the query processing time incurred by the R-tree only depends on the respective locations of the parent documents. Therefore, the results for ART essentially follow the same trend as the results of Fig. 17 due to the same reasons, as discussed for the results of Fig. 17. However, due to the disk-resident nature of the R-tree, the cost of querying the R-Tree has a dominating effect leading to the overlap of the graphs obtained for the S_BSA, S_BSAM and S_BSH schemes. Furthermore, observe that the results for REC also exhibit similar trends to those of the results of Fig. 17 due to the same reasons, as explained for the results of Fig. 17.

#### 6.2.5 Effect of Varying the Spatial Query Window Size

Figure 22 depicts the effect of varying the size $Q_S$ of the spatial query window. As $Q_S$ increases, the number of parent documents occurring within the query window increases. The implication is that a larger number of parent documents needs to be processed. Consequently, this leads to increase in ART with increase in $Q_S$. Notice that the graphs for our proposed S_BSA, S_BSAM, and S_BSH schemes coincide because the processing time incurred by the R-tree dominates the query processing time.

Notably, the size of the query window does not have any effect on REC. Hence, the performance of our three proposed schemes as well as the reference S_ZCS scheme remains comparable across different values of $Q_S$. The reasons for our proposed S_BSA, S_BSAM, and S_BSH schemes outperforming S_ZCS in terms of REC are essentially the same as those of the results of Fig. 15. In other words, REC remains constant at its highest possible value (i.e., 100%) for S_BSH, S_BSA and S_BSAM. This occurs because the bitwise-AND operation used by these schemes is able to efficiently generate the candidate set with 100% recall. However, for S_ZCS, REC is significantly lower because a significant percentage of parent documents do not get considered toward the top-$k$ processing.

## 7 Conclusion

This work has addressed the problem of *efficiently* retrieving the top-$k$ entities in an SOE for keyword-based queries. Furthermore, we have extended the afore-mentioned problem to address the case of geo-tagged entities. In particular, we have proposed an efficient bitmap-based approach for quickly identifying the candidate set as well as a variant for reducing memory consumption by exploiting skews in keyword popularity. We have also proposed the two-tier HI-tree index, which uses both hashing and inverted indexes, for efficient document relevance score lookups. Additionally, we have proposed an R-tree-based approach to extend the afore-mentioned approaches to address the case of geo-tagged entities. Our detailed performance evaluation with both real and synthetic datasets shows that our proposed schemes are indeed effective in providing good top-$k$ result recall performance within acceptable query response times with good scalability. In the near future, we plan to extend our work by associating temporal information with the entities in addition to spatial information. This would significantly facilitate in answering spatio-temporal queries in the context of our application scenarios.

**Acknowledgements** Not applicable.

**Author Contributions** AM: Envisioning the idea, defining the research problem and contributing to the solution of the problem, and editing the paper. AK: Contributing to the solution of the problem and doing the implementation of the paper and editing the paper. NP: Contributing to the solution of the problem and doing the implementation of the paper and editing the paper. MM: Envisioning the idea, defining the research problem and contributing to the solution of the problem. All the authors of this paper have collaborated and worked toward developing the paper and made significant contributions to the paper.

**Funding** Not applicable

**Declaration**

**Conflict of interest** The authors declare that they have no conflict of interest.

**Data availability** Yes, all related materials concerning this paper are available with us and can be provided upon request.

**Code availability** Yes, the code is available with us.



## References

1. http://blogs.hbr.org/cs/2011/10/movingfromtransactiontoeng.html
2. http://www-01.ibm.com/software/ebusiness/jstart/systemsofengagement/
3. http://www.bersin.com/blog/post/Systems-of-Engagement-vs-Systems-of-Record-About-HR-software2c-design-and-Workday.aspx
4. Mondal A, Padhariya N, Mohania MK (2020) Towards efficient retrieval of top-$k$ entities in systems of engagement. In: Huang Z, Beek W, Wang H, Zhou R, Zhang Y (eds) Web Information Systems Engineering – WISE 2020. WISE 2020. Lecture notes in computer science, vol 12343. Springer, Cham, pp 52–67
5. Abraham T, Roddick JF (1999) Survey of spatio-temporal databases. GeoInformatica 3(1):61–99
6. Agrawal R, Fuxman A, Kannan A, Shafer J, Talukdar PP (2012) Associating structured records to text documents. In: WWW, pp. 451–452
7. Agrawal S, Chaudhuri S, Das G (2002) DBXplorer: a system for keyword-based search over relational databases. In: ICDE, pp. 5–16
8. Alfarrarjeh A, Kim SH, Hegde V, Shahabi C, Xie Q, Ravada S et al (2020) A class of R*-tree indexes for spatial-visual search of geo-tagged street images. In: 2020 IEEE 36th international conference on data engineering (ICDE), pp. 1990–1993. IEEE

9. Baeza-Yates RA, Ribeiro-Neto BA (1999) Modern information retrieval. ACM Press, New York
10. Beckmann N, Kriegel HP, Schneider R, Seeger,B (1990) The R*-tree: an efficient and robust access method for points and rectangles. In: ACM SIGMOD, pp. 322–331
11. Bhalotia G, Hulgeri A, Nakhe C, Chakrabarti S, Sudarshan S (2002) Keyword searching and browsing in databases using BANKS. In: ICDE, pp. 431–440
12. Chakrabarti S, Dom B, Indyk P (1998) Enhanced hypertext categorization using hyperlinks. ACM Sigmod Record 27(2):307–318
13. Ding B, Zhao B, Lin C, Han J, Zhai C (2010) TopCells: keyword-based search of top-k aggregated documents in text cube. In: ICDE, pp. 381–384
14. Fagin R, Lotem A, Naor M (2003) Optimal aggregation algorithms for middleware. Comput Syst Sci 66(4):614–656
15. Feldman R (2002) Link analysis: current state of the art. In: KDD Tutorial
16. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: ACM SIGMOD, pp. 47–57
17. Han J, Fu Y (1994) Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In: KDD Workshop, pp. 157–168
18. Hartley J, Holti R, Carli G (2021) Management consultants navigating competing systems of engagement. In: academy of management proceedings, vol. 2021, p. 15423. Academy of Management Briarcliff Manor, NY 10510
19. Hristidis V, Gravano L, Papakonstantinou Y (2003) Efficient IR-style keyword search over relational databases. In: VLDB, pp. 850–861
20. Kimelfeld B, Sagiv Y (2006) Finding and approximating top-k answers in keyword proximity search. In: PODS, pp. 173–182
21. Kleinberg JM (1999) Authoritative sources in a hyperlinked environment. J ACM 46(5):604–632
22. Liu F, Yu C, Meng W, Chowdhury A (2006) Effective keyword search in relational databases. In: ACM SIGMOD, pp. 563–574
23. Lu Q, Getoor L (2003) Link-based classification. In: ICML, pp. 496–503
24. Luo Y, Lin X, Wang W (2007) SPARK: Top-k keyword query in relational databases. In: ACM SIGMOD, pp. 115–126
25. Page L, Brin S, Motwani R, Winograd T (1999) The page rank citation ranking: bringing order to the web. Technical report, Stanford InfoLab
26. Pfoser D, Jensen CS, Theodoridis Y et al (2000) Novel approaches to the indexing of moving object trajectories. In: VLDB, pp. 395–406
27. Šaltenis S, Jensen CS, Leutenegger ST, Lopez MA (2000) Indexing the positions of continuously moving objects. In: ACM SIGMOD, pp. 331–342
28. Sellis T, Roussopoulos N, Faloutsos C (1987) The R+-Tree: a dynamic index for multi dimensional objects. In: VLDB, pp. 507–518
29. Weiß P, Warg M, Zolnowski A (2019) Building systems of engagement to overcome the challenges of digital transformation. In: Naples Service Forum
30. Wu P, Sismanis Y, Reinwald B (2007) Towards keyword-driven analytical processing. In: ACM SIGMOD, pp. 617–628
31. Xin D, Han J, Cheng H, Li X (2006) Answering top-k queries with multi-dimensional selections: the ranking cube approach. In: VLDB, pp. 463–474
32. Yuan S, Pi D, Zhao X, Xu M (2021) Differential privacy trajectory data protection scheme based on R-tree. Expert Syst Appl 27:1152215
33. Yufei T, Papadias D (2000) MV3R-tree: a spatio-temporal access method for timestamp and interval queries. Techical report, Citeseer
34. Zhang D, Chee YM, Mondal A, Tung AK, Kitsuregawa, M (2009) Keyword search in spatial databases: towards searching by document. In: ICDE, pp. 688–699. IEEE