Indexing for Keyword Search with Structured Constraints

Shangqi Lu
sqlu@cse.cuhk.edu.hk
Chinese University of Hong Kong
Hong Kong, China

Yufei Tao
taoyf@cse.cuhk.edu.hk
Chinese University of Hong Kong
Hong Kong, China

ABSTRACT

Keyword search, which finds the documents containing all the keywords supplied by a user, has proved to be an effective approach for querying *non-structured* information that does not conform to any pre-set schemas. In the last two decades, a vast amount of research — especially in the field of "spatial keyword search" — has been carried out to design indexes for answering queries that integrate keyword search with *structured* predicates imposed on pre-determined attributes (common examples of such predicates are range conditions, linear constraints, prioritization by distance, etc.). Although the past investigation has led to a plethora of empirical solutions, little progress has been made in theory. In fact, for most problems in the literature, the state of the art is still the naive method that answers a query purely based on the keyword conditions or the structured predicates. In this paper, we remedy the issue by developing new indexes with strong theoretical guarantees for a suite of problems with heavy importance in real applications. Many of our indexes are near-optimal, subject to widely-accepted conjectures.

CCS CONCEPTS

• **Theory of computation → Data structures design and analysis**.

KEYWORDS

Keyword Search, Non-Structured Data, Indexing, Lower Bounds

ACM Reference Format:
Shangqi Lu and Yufei Tao. 2023. Indexing for Keyword Search with Structured Constraints. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3584372.3588663

1 INTRODUCTION

By tradition, database systems support *structured* retrieval by requiring query predicates to conform to pre-set schemas. Valuable knowledge, however, exists in *non-structured* formats. An important example is "free-form" text data, which are common both in schema-free systems such as NoSQL databases and schema-oriented systems such as relational databases (e.g., a `remark` column for capturing miscellaneous details that do not fit in any specifically-purposed attributes). Keyword search, popularized by Internet search engines, has proven to be effective in querying non-structured information. The mechanism works by having a user supply a small number of keywords and then instructing the system to fetch the data objects whose text fields contain all the keywords simultaneously. This search functionality serves as a powerful addition to conventional query languages (e.g., SQL) and is widely supported by modern database systems nowadays.

Keyword search is often integrated with structured retrieval to enrich expressive power significantly. Consider, for example, a relation with schema `Hotel(price, rating, Doc)`. Here, each tuple describes a hotel's nightly price and guest rating; moreover, the tuple's `Doc` column contains textual tags describing the hotel's special features. A "pure" keyword query would demand that a hotel's `Doc` should contain, for instance, the words '`pool`', '`free-parking`', and '`pet-friendly`'. In a more realistic scenario, the same query may include additional conditions concerning the other (structured) attributes, about which two representative examples are:

- **C1** `price` $\in$ [\$100, \$200] and `rating` ≥ 8;
- **C2** $c_1 \cdot$ `price` $+ c_2 \cdot (10 -$ `rating`$) \leq c_3$, where c_1, c_2, and c_3 are appropriate constants (here we assume that `rating` is from 0 to 10).

The distinctive nature of the two examples is notable. **C1** places a *separate* constraint on each attribute, whereas **C2** is a *joint* constraint involving multiple attributes.

Queries like the above can be answered by two naive approaches:

- (Structured only) Retrieve all the data objects satisfying the structured condition (i.e., **C1** or **C2**) and then eliminate those whose documents do not contain all the keywords.
- (Keywords only) Retrieve all the objects whose documents include all the keywords and then eliminate those that do not satisfy the remaining conditions.

The common drawback of both approaches is that they may need to examine a huge number of "candidate objects" (i.e., those passing *either* the structured *or* keywords condition), even though very few objects are reported eventually. In particular, their query time can be asymptotically the same as reading all the data, even if no objects are reported at all. The drawback has motivated a vast amount of research in the last two decades, aiming to design innovative techniques that can "fuse" the processing of structured and keyword predicates (a review will appear in Section 2). The past investigation has produced numerous indexes that perform well on "real data". Nonetheless, surprisingly little progress has been achieved in theory such that the two naive solutions are still the state of the art till this day.

We believe that there is a need to provide solid theoretical guidance on this topic, especially given the significance of non-structured

PODS '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 979-8-4007-0127-6/23/06…$15.00
https://doi.org/10.1145/3584372.3588663

data today. To that end, this work develops new indexes with attractive guarantees for keyword search coupled with a variety of structured constraints. Also presented are hardness results suggesting that many of our indexes can no longer be substantially improved, subject to widely-accepted conjectures. The rest of the section offers an overview of our results, which are summarized in Table 1.

1.1 New Indexing Results

In all the problems to be studied, the input dataset is a set D of elements — called *objects* — such that each object $e \in D$ is associated with a non-empty *document*, denoted as $e.\mathtt{Doc}$, formulated as a set of integers. Given integers $\mathtt{w}_1, \mathtt{w}_2, ...\mathtt{w}_k$ (for some $k \geq 1$) — called *keywords* — we define

$$D(\mathtt{w}_1, ..., \mathtt{w}_k) := \{e \in D \mid \mathtt{w}_i \in e.\mathtt{Doc} \text{ for all } i \in [1, k]\} \quad (1)$$

namely, the set of objects whose documents contain all the keywords $\mathtt{w}_1, ..., \mathtt{w}_k$. The *input size* is

$$N := \sum_{e \in D} |e.\mathtt{Doc}| \quad (2)$$

i.e., the total size of all the objects' documents (in all our settings, the input data require $\Theta(N)$ words to store).

The objective is to create an index on D to answer queries efficiently. Concrete problems differ in the details of objects and queries, as introduced below. The computational model assumed is the standard RAM model where each word has at least $\log_2 N$ bits.

Range Reporting with Keywords and Relevant Problems. Let us start with a fundamental problem that enjoys a prominent position in the literature.

> *Orthogonal Range Reporting with Keywords* (ORP-KW). D is a set of points in $\mathbb{R}^d$, where $\mathbb{R}$ is the real domain and $d \geq 1$ is a constant. Fix an integer $k \geq 2$. Given a d-rectangle[1] q and keywords $\mathtt{w}_1, ..., \mathtt{w}_k$, a query returns $q \cap D(\mathtt{w}_1, ..., \mathtt{w}_k)$, where $D(\mathtt{w}_1, ..., \mathtt{w}_k)$ is given in (1). In other words, the query returns every point of D that falls in q and contains the k keywords in their documents.

The query in our introductory example with condition **C1** illustrates an ORP-KW query. We will prove:

THEOREM 1. *For ORP-KW with $d \leq 2$, there is an index of $O(N)$ space that answers a query in $O(N^{1-1/k} \cdot (1 + \text{OUT}^{1/k}))$ time, where N is the input size, k is the number of query keywords, and* OUT *is the number of points reported.*

Evidence will be presented later that the above query time is unlikely to allow significant improvement. We further show that there only needs to be an $O(\log \log N)$ blow up in space whenever the dimensionality d increases by one:

THEOREM 2. *For ORP-KW in d-dimensional space where $d \geq 3$, there is an index of $O(N \cdot (\log \log N)^{d-2})$ space that answers a query in $O(N^{1-1/k} \cdot (1 + \text{OUT}^{1/k}))$ time, where N, k, and* OUT *have the same meanings as in Theorem 1.*

[1] A d-rectangle is a rectangle of the form $[x_1, y_1] \times [x_2, y_2] \times ... \times [x_d, y_d]$.

By virtue of ORP-KW's fundamental nature, Theorems 1 and 2 have implications on many problems that have been considered in the literature. Below we discuss two notable implications.

> *Rectangle Reporting with Keywords* (RR-KW). D is a set of d-rectangles in $\mathbb{R}^d$, where $d \geq 1$ is a constant. Fix an integer $k \geq 2$. Given a d-rectangle q and keywords $\mathtt{w}_1, ..., \mathtt{w}_k$, a query returns all the rectangles in $D(\mathtt{w}_1, ..., \mathtt{w}_k)$ intersecting with q.

For $d = 1$, RR-KW is at the center of keyword search on temporal documents [7] (i.e., each document has a time interval indicating its "lifespan"). For $d \geq 2$, the problem is the essence of keyword search on geographic entities whose regions are modeled as "minimum bounding rectangles" [34]. Appendix F shows that Theorems 1 and 2 yield almost immediately:

COROLLARY 3. *For RR-KW, there is an index of $O(N \cdot (\log \log N)^{2d-2})$ space that answers a query in $O(N^{1-1/k} \cdot (1 + \text{OUT}^{1/k}))$ time, where N is the input size, k is the number of query keywords, and* OUT *is the number of rectangles reported.*

> *L_∞-Nearest Neighbor with Keywords* (L_∞NN-KW). D is a set of points in $\mathbb{R}^d$, where $d \geq 1$ is a constant. Fix an integer $k \geq 2$. Given a point q in $\mathbb{R}^d$, an integer $t \geq 1$, and keywords $\mathtt{w}_1, ..., \mathtt{w}_k$, a query returns t points in $D(\mathtt{w}_1, ..., \mathtt{w}_k)$ that are closest to q under the L_∞ distance[2].

In Appendix F, we use Theorems 1 and 2 to prove:

COROLLARY 4. *For L_∞NN-KW, there is an index of $O(N \cdot (\log \log N)^{d-2})$ space that answers a query in $O(N^{1-1/k} \cdot t^{1/k} \cdot \log N)$ time, where N is the input size, k is the number of query keywords, and t is the number of points reported.*

The problem represents one form of similarity search with keywords [48] (e.g., find the hotel nearest to an address, among all the hotels whose features include '`pool`', '`free-parking`', and '`pet-friendly`'). Corollary 4 can also be interpreted as an approximation result under L_2 distance (Euclidean distance) because the L_∞ distance between any two points is a constant-factor approximation of their L_2 distance. An exact index under L_2 distance (with slightly worse guarantees) will be presented shortly.

Linear Conjunction with Keywords and Relevant Problems. Let us continue with another fundamental problem.

> *Linear Conjunction with Keywords* (LC-KW). D is a set of points in $\mathbb{R}^d$, where $d \geq 2$ is a constant. A *linear constraint* q has the form $\sum_{i=1}^{d} c_i \cdot x[i] \leq c_{d+1}$, where $c_1, c_2, ..., c_{d+1}$ are real coefficients; a point $p := (p[1], ..., p[d])$ satisfies the constraint if $\sum_{i=1}^{d} c_i \cdot p[i] \leq c_{d+1}$. Fix an integer $k \geq 2$. Given $s := O(1)$ linear constraints and keywords $\mathtt{w}_1, ..., \mathtt{w}_k$, a query returns every point $p \in D(\mathtt{w}_1, ..., \mathtt{w}_k)$ satisfying the s constraints.

[2] The L_∞ distance of $p := (p[1], p[2], ..., p[d])$ and $q := (q[1], q[2], ..., q[d])$ is $\max_{i=1}^{d} |p[i] - q[i]|$.

problem		space	query	remark
orthogonal range reporting with keywords	$d \le 2$	$O(N)$	$O(N^{1-1/k} \cdot (1 + \text{OUT}^{1/k}))$	opt†
	$d \ge 3$	$O(N(\log\log N)^{d-2})$		opt†
	$d \le k$	$O(N)$	$O(N^{1-1/k} \cdot (\log N + \text{OUT}^{1/k}))$	opt†
rectangle reporting with keywords		$O(N(\log\log N)^{2d-2})$	$O(N^{1-1/k} \cdot (1 + \text{OUT}^{1/k}))$	opt†
L_∞-nearest neighbor with keywords		$O(N(\log\log N)^{d-2})$	$O(N^{1-1/k} \cdot t^{1/k} \cdot \log N)$	opt‡
linear conjunction with keywords	$d \le k$	$O(N)$	$O(N^{1-1/k} \cdot (\log N + \text{OUT}^{1/k}))$	opt†
	$d > k$		$O(N^{1-1/d} + N^{1-1/k} \cdot \text{OUT}^{1/k})$	
spherical range reporting with keywords	$d \le k-1$	$O(N)$	$O(N^{1-1/k} + N^{1-1/k} \cdot \text{OUT}^{1/k})$	opt†
	$d > k-1$		$O(N^{1-1/(d+1)} + N^{1-1/k} \cdot \text{OUT}^{1/k})$	
L_2-nearest neighbor with keywords	$d \le k-1$	$O(N)$	$O(\log N \cdot N^{1-1/k} \cdot (\log N + t^{1/k}))$	opt‡
	$d > k$		$O(\log N \cdot (N^{1-1/(d+1)} + N^{1-1/k} \cdot t^{1/k}))$	

†Optimality up to a sub-polynomial factor in the following sense: no indexes of $O(N \text{ polylog } N)$ space can have query time $O(N^{1-\frac{1}{k}-\epsilon} + N^{1-\frac{1}{k}} \cdot \text{OUT}^{\frac{1}{k}})$ or $O(N^{1-\frac{1}{k}} + N^{1-\frac{1}{k}} \cdot \text{OUT}^{\frac{1}{k}-\epsilon} + \text{OUT})$, regardless of the constant $\epsilon > 0$, subject to the strong set-intersection and strong k-set-disjointness conjectures.

‡No indexes of $O(N \text{ polylog } N)$ space can have query time $O(N^{1-\frac{1}{k}} \cdot t^{\frac{1}{k}-\epsilon} + t)$, regardless of the constant $\epsilon > 0$, subject to the aforementioned conjectures.

Table 1: Summary of our results

The query in our introductory example with condition **C2** illustrates an LC-KW query with one linear constraint. We will prove in Appendix D:

THEOREM 5. *For LC-KW with $d \le k$, there is an index of $O(N)$ space that answers a query in $O(N^{1-1/k} \cdot (\log N + \text{OUT}^{1/k}))$ time, where N is the input size, k is the number of query keywords, and* OUT *is the number of points reported. For LC-KW with $d > k$, there is an index of $O(N)$ space that answers a query in $O(N^{1-1/d} + N^{1-1/k} \cdot \text{OUT}^{1/k})$ time.*

LC-KW stands at the core of many problems. One example is ORP-KW — our first fundamental problem — noticing that a d-rectangle can be regarded as the conjunction of $2d = O(1)$ linear constraints. When $d \le k$, Theorem 5 improves the space of Theorem 2 to $O(N)$ while ensuring nearly the same query time.[3] Next, we discuss two other notable implications of Theorem 5.

Spherical Range Reporting with Keywords (SRP-KW). D is a set of points in $\mathbb{R}^d$, where $\mathbb{R}$ is the real domain and $d \ge 1$ is a constant. Fix an integer $k \ge 2$. Given a sphere q in $\mathbb{R}^d$ and keywords $\mathtt{w}_1, ..., \mathtt{w}_k$, a query returns $q \cap D(\mathtt{w}_1, ..., \mathtt{w}_k)$, where $D(\mathtt{w}_1, ..., \mathtt{w}_k)$ is given in (1).

The problem is also known as "boolean range query with keywords" in the literature [22] (a query example: find all hotels that are within 1km of a given address and contain keywords $\mathtt{w}_1, ..., \mathtt{w}_k$). In Appendix F, we will see that Theorem 5 yields almost immediately:

COROLLARY 6. *For SRP-KW with $d \le k-1$, there is an index of $O(N)$ space that answers a query in $O(N^{1-1/k} \cdot (\log N + \text{OUT}^{1/k}))$ time, where N is the input size, k is the number of query keywords, and* OUT *is the number of points reported. For SRP-KW with $d > k-1$, there is an index of $O(N)$ space that answers a query in $O(N^{1-1/(d+1)} + N^{1-1/k} \cdot \text{OUT}^{1/k})$ time.*

[3]Similar improvement can also be obtained on the RR-KW and L_∞NN-KW problems, which the reader will figure out after going through the relevant proofs.

L_2-Nearest Neighbor with Keywords (L_2NN-KW). D is a set of points in $\mathbb{N}^d$, where $\mathbb{N}$ is the set of $O(\log N)$-bits integers and $d \ge 2$ is a constant. Fix an integer $k \ge 2$. Given a point q in $\mathbb{N}^d$, an integer $t \ge 1$, and keywords $\mathtt{w}_1, ..., \mathtt{w}_k$, a query returns t points in $D(\mathtt{w}_1, ..., \mathtt{w}_k)$ that are closest to q under the L_2 distance.

In Appendix F, we will show that Corollary 6 leads to

COROLLARY 7. *For L_2NN-KW with $d \le k-1$, there is an index of $O(N)$ space that answers a query in $O(\log N \cdot N^{1-1/k} \cdot (\log N + t^{1/k}))$ time, where N is the input size, k is the number of query keywords, and t is the number of points reported. For L_2NN-KW with $d > k-1$, there is an index of $O(N)$ space that answers a query in $O(\log N \cdot (N^{1-1/(d+1)} + N^{1-1/k} \cdot t^{1/k}))$ time.*

1.2 Tightness of Our Results

The reader must have noticed that the query time of our indexes in Section 1.1 often contains a term $O(N^{1-1/k} \cdot (1 + \text{OUT}^{1/k}))$. Next, we will provide justification on the term. The main source of hardness comes from keyword search, which is in fact the following problem in disguise.

k-Set Intersection (k-SI). The input data consists of $m \ge 2$ sets $S_1, S_2, ..., S_m$ of integers. A *reporting query* picks $k := O(1)$ distinct integers $\mathtt{w}_1, \mathtt{w}_2, ..., \mathtt{w}_k \in [1, m]$ and returns $\bigcap_{i=1}^{k} S_{\mathtt{w}_i}$. An *emptiness query* is similar except that it reports only whether $\bigcap_{i=1}^{k} S_{\mathtt{w}_i}$ is empty.

A "pure" keyword search query — namely, one that computes $D(\mathtt{w}_1, ..., \mathtt{w}_k)$ as given in (1) — is equivalent to a k-SI reporting query. This is because one can, for each keyword $\mathtt{w}$, create a set $S_{\mathtt{w}}$ which contains the id of every object $e \in D$ such that $\mathtt{w} \in e.\texttt{DOC}$; thus, $D(\mathtt{w}_1, ..., \mathtt{w}_k)$ equals exactly $\bigcap_{i=1}^{k} S_{\mathtt{w}_i}$ (this is the well-known "inverted index" idea). Conversely, given an instance of k-SI, one can create a keyword search instance by treating each set id $i \in [1, m]$ as a keyword and creating $D := \bigcup_{i=1}^{m} S_i$ where each element $e \in D$

has a document $e.\texttt{Doc} := \{i \mid e \in S_i\}$. A reporting query with set ids $\texttt{w}_1, ..., \texttt{w}_k$ has the same result as $D(\texttt{w}_1, ..., \texttt{w}_k)$.

There exist two conjectures on k-SI, both originating from [31].

- *Strong set-intersection conjecture:* for any constant $\delta \in (0, 1]$, an index answering a k-SI reporting query in $O(N^{1-\delta} + \text{OUT})$ time — where $N := \sum_{i=1}^{m} |S_i|$ and OUT is the size of the reported intersection — must use $\Omega(N^{1+\delta}/\text{polylog}\, N)$ space.
- *Strong k-set-disjointness conjecture:* for any constant $\delta \in (0, 1 - 1/k]$, an index answering a k-SI reporting query in $O(N^{1-\frac{1}{k}-\delta})$ time must use $\Omega(N^{1+k\delta}/\text{polylog}\, N)$ space.

The above conjectures have been widely used [6, 10, 14, 15, 25, 32] to study fine-grained computational complexities.

Appendix E will prove the next hardness result on k-SI reporting.

LEMMA 8. *Consider k-SI reporting queries. Suppose that there is an index of $O(N \text{ polylog } N)$ space with query time*

$$O\left(N^{1-1/k} + N^{1-\frac{1}{k}} \cdot \text{OUT}^{\frac{1}{k}-\epsilon} + \text{OUT}\right) \tag{3}$$

for some constant $\epsilon > 0$. Then, the same index also achieves query time $O(N^{1-\delta} + \text{OUT})$, where $\delta = \min\{\frac{1}{k}, \frac{\epsilon}{1-(1/k)+\epsilon}\}$. This defies the strong set-intersection conjecture.

The lemma allows us to better appreciate a k-SI reporting index that occupies $O(N \text{ polylog } N)$ space and guarantees $O(N^{1-1/k} \cdot (1 + \text{OUT}^{1/k}))$ query time. Let us re-write the query complexity into the following equivalent form:

$$O\left(N^{1-1/k} + N^{1-1/k} \cdot \text{OUT}^{1/k} + \text{OUT}\right). \tag{4}$$

Every term in the complexity is tight up to a sub-polynomial factor. First, if the bound could be lowered to $O(N^{1-\frac{1}{k}-\epsilon} + N^{1-1/k} \cdot \text{OUT}^{1/k} + \text{OUT})$ for some constant $\epsilon > 0$, it would have to terminate in $O(N^{1-\frac{1}{k}-\epsilon})$ time when $\text{OUT} = 0$. This, in turn, would imply that the index could be used to answer a k-SI emptiness query in $O(N^{1-\frac{1}{k}-\epsilon})$ time[4], thus breaking the strong k-set-disjointness conjecture. Lemma 8 rules out the possibility of improving the second additive term $N^{1-1/k} \cdot \text{OUT}^{1/k}$ of (4) even by a factor polynomial in OUT (let alone in N), subject to the strong set-intersection conjecture. Finally, the third additive term OUT in (4) is clearly necessary.

The above discussion indicates that, for the ORP-KW, RR-KW, and LC-KW problems introduced in Section 1.1, if an index uses $O(N \text{ polylog } N)$ space, the query time in (4) is already the best we can hope for (up to a sub-polynomial factor), subject to the two aforementioned conjectures. This is because all those problems *generalize* k-SI reporting. Consider, for example, ORP-KW. Given an instance of k-SI with sets $S_1, ..., S_m$, we can create an ORP-KW instance as follows. First (as explained before), obtain a "pure" keyword search instance where $D := \bigcup_{i=1}^{m} S_i$ and $e.\texttt{Doc} := \{i \mid e \in S_i\}$ for each $e \in D$. Then, map each object $e \in D$ to an arbitrary point in $\mathbb{R}^d$. Given a k-SI reporting query with set ids $\texttt{w}_1, ..., \texttt{w}_k$, issue an ORP-KW query with keywords $\texttt{w}_1, ..., \texttt{w}_k$ and a search rectangle $q := \mathbb{R}^d$. The two queries return the same result. It thus follows that none of Theorem 1, Theorem 2, and Corollary 3 is likely to admit significant improvement. The same is true for Theorem 5 and Corollary 6 when $d \leq k$ and $d + 1 \leq k$, respectively. With more effort, similar (conditional) tightness can also be proved for Corollaries 4 and 7, as discussed in Appendix G.

We close the section with a remark that our hardness discussion also explains why it makes sense to focus on queries that issue a small number k of keywords. As k increases, the complexity in (4) continuously approaches $O(N)$, thus losing all the advantages over the naive solution that simply reads the input data in whole.

2 PREVIOUS WORK

We will first review the relevant results in theory before attending to the related work from the system community.

Let us start with k-SI reporting, which as mentioned is identical to pure keyword search. By resorting to (perfect) hashing, one can build an $O(N)$-space index to answer a query in $O(N)$ time. Improving the query time (but retaining the linear space complexity) has drawn considerable interests. The existing research can be divided into two lines. The first one aims to achieve query time of the form $o(N) + O(\text{OUT})$, where OUT is the output size (i.e., the intersection size). Bille et al. [11] obtained query time $O(N(\log \textit{wlen})^2/\textit{wlen} + \text{OUT})$ in expectation where *wlen* is the word length (i.e., the number of bits in a word). Eppstein et al. [27] improved the bound to $O(N(\log \textit{wlen})/\textit{wlen} + \text{OUT})$, but assuming $k = 2$. Goodrich [33] removed the assumption and attained the same query time for any constant k. In the common scenario where $\textit{wlen} = \Theta(\log N)$, the query time becomes $O(N(\log\log N)/\log N + \text{OUT})$ expected, faster than the naive solution by an $O(\log N/\log\log N)$ factor for $\text{OUT} = O(N(\log\log N)/\log N)$.

The second line of works aims at greater improvement over the naive method when OUT is small. Cohen and Porat [23] developed an index of space $O(N)$ that answers a query with $k = 2$ in $O(\sqrt{N}(1 + \sqrt{\text{OUT}}))$ time. This complexity, interestingly, beats $O(N)$ whenever possible. Specifically, if $\text{OUT} = \Omega(N)$, the complexity is $O(N)$ but, in that case, any algorithm must spend $\Omega(N)$ time outputting the result anyway. On the other hand, $O(\sqrt{N} \cdot (1 + \sqrt{\text{OUT}}))$ is $o(N)$ as long as $\text{OUT} = o(N)$. More importantly, when OUT is small, e.g., $\text{OUT} \leq N^{1-\epsilon}$, the complexity improves $O(N)$ by a polynomial factor. In [38], Kopelowitz et al. explained how to achieve a smooth tradeoff between space and query time, which captured the result of [23] as a special case.

Also focusing on $k = 2$, Afshani and Nielsen [2] investigated the same tradeoff from the lower bound perspective in the pointer-machine model. They proved that, for any constant $\epsilon \in [0, 1]$, achieving $O(N^{1-\epsilon} + \text{OUT})$ query time demands $\Omega(N^{1+\epsilon-o(1)})$ space, effectively confirming the strong set-intersection conjecture on pointer machines. Assuming $\text{OUT} \geq 1$, they also proved that any structure with $O((N \cdot \text{OUT})^{1/2-\epsilon} + \text{OUT})$ query time must use $\Omega(N^{1+\delta-o(1)})$ space where $\delta > 0$ is a constant dependent on ϵ.[5] For additional hardness results (less relevant to our work), the reader may refer to [32] for alternative space-vs-query-time tradeoffs when D and

[4] Simply run a reporting query. If it does not terminate within $O(N^{1-(1/k)-\epsilon})$ time, the intersection must be non-empty and we can terminate the query manually.

[5] It is worth mentioning that the structures of [23, 38] require random accesses beyond the pointer-machine model.

N satisfy extra requirements, and to [31, 39, 40, 46] for tradeoffs between an index's preprocessing time and query time.

For every problem in Section 1.1, by ignoring keywords completely — i.e., replacing $D(\mathtt{w}_1, ..., \mathtt{w}_k)$ with D in every problem definition — one obtains an indexing problem concerning only geometry. All those (geometry) problems are classical in computation geometry and have been well understood; see [1, 3, 13, 16, 44, 45, 47] and the references therein. The pertinent results, however, shed little light on the characteristics of our problems because, as discussed, the hardness in our context stems from k-SI reporting.

We are aware of no previous work that has established non-trivial guarantees for any of the problems in Section 1.1. The only exception is ORP-KW with $d = 1$, for which Goodrich [33] presented an $O(N)$-size index and $O(N(\log\log N)/\log N + \text{OUT})$ expected query time, assuming $wlen = O(\log N)$. We, instead, aim at matching the query time $O(N^{1-1/k} \cdot (1 + \text{OUT}^{1/k}))$ of pure keyword search, even in the presence of structured constraints.

In the system community, the idea of supporting keyword search in relational databases was introduced at the beginning of the millennium; see the pioneering works [4, 9, 30, 36, 43]. The set of problems in Section 1.1 have been extensively studied — see representative works [5, 7, 12, 17–22, 28, 34, 35, 37, 41, 42, 48–53] and the references therein — under the topic of "spatial/temporal keyword search". We will not delve into those works further because the indexes therein, although shown to be empirically efficient, do not have interesting theoretical guarantees.

3 INDEX TRANSFORMATION FRAMEWORK

As mentioned, if we ignore the keyword components, the problems defined in Section 1.1 degenerate into conventional geometry problems, for which effective indexes are known. It would be nice if all those (pure geometry) indexes could be automatically transformed to new indexes that can support keyword predicates as well. While this may sound over-ambitious, a primary technical contribution of this paper is a generic framework for achieving the purpose on a class of geometry indexes.

This section will illustrate the framework by adapting the 2D kd-tree to settle the ORP-KW problem with the guarantees in Theorem 1. The adaptation contains all the key elements in our framework and yet involves only light technical complication. After grasping the rationale behind, the reader will then be ready to apply the framework to more sophisticated structures, such as the partition tree, which we discuss in Appendix D in our endeavor to prove Theorem 5.

We summarize the proposed framework into four main steps.

3.1 Step 1: Identifying a Space-Partitioning Index

Let us review the kd-tree to a level sufficient for our discussion. Denote by P a set of N points in $\mathbb{R}^2$. A kd-tree on P is a binary tree $\mathcal{T}$ where there are N leaves and each internal node has two child nodes. Every leaf stores a distinct point of P. Given a node u, we use P_u to represent the set of points of P stored in the subtree of u.

The kd-tree is *space-partitioning* because

- every node u in $\mathcal{T}$ is associated with a 2-rectangle Δ_u as its *cell*, which covers all the points in P_u;
- the cell of the root is the entire $\mathbb{R}^2$;
- for every internal node u, the cells of its child nodes are interior disjoint and have Δ_u as their union.

Given a node u, we use $level(u)$ to denote its *level*.[6] The relationship $|P_u| = O(N/2^{level(u)})$ holds for all the nodes u in $\mathcal{T}$. If u is an internal node with child nodes v_1 and v_2, the cells of v_1 and v_2 obey the following rules.

- If $level(u)$ is even, Δ_u is split into two rectangles by a vertical line ℓ. Those rectangles, which touch only at boundary[7] and are interior disjoint, are taken as Δ_{v_1} and Δ_{v2}, respectively.
- If $level(u)$ is odd, the split is analogous except that ℓ is a horizontal line.

3.2 Step 2: Conversion under General Position

Next, we transform the kd-tree into an ORP-KW index. Recall that the input is a set D of points (a.k.a., objects) in $\mathbb{R}^2$, where every object $e \in D$ has a non-empty set $e.\mathtt{Doc}$ of integer keywords. Let $W := |\bigcup_{e\in D} e.\mathtt{Doc}|$ (the total number of distinct keywords in all documents). W.l.o.g., each keyword is treated as an integer in $[1, W]$. As before, let k be the number of keywords in an ORP-KW query.

We will first assume D in *general position*, which here means that no two objects in D have the same x- or y-coordinate. The assumption's removal will be the last step of our framework.

Verbose Set. We are ready to construct a kd-tree $\mathcal{T}$. However, it is imperative to build $\mathcal{T}$ on a *verbose* version of D, rather than on D itself. Specifically, for each $e \in D$, we add $|e.\mathtt{Doc}|$ copies of e to an initially-empty set P. Those copies are regarded as distinct points in P. Henceforth, we will reserve symbol "e" for points in D (which will be consistently called "objects") and symbol "p" for points in P (the term "point" will now be reserved for the elements of P). The kd-tree $\mathcal{T}$ is constructed on P, which has size N.

Active and Pivot Sets. For each node u in $\mathcal{T}$, we will introduce two notions: *active set* D_u^{act} and *pivot set* D_u^{pvt}, which satisfy $D_u^{pvt} \subseteq D_u^{act} \subseteq D$. As will be clear later, D_u^{act} contains all the objects stored in the subtree of u, among which those in D_u^{pvt} are stored at u.

We define the two notions in an inductive manner. If u is the root of $\mathcal{T}$, then $D_u^{act} := D$. Inductively, consider u as an internal node whose D_u^{act} has been properly defined, but not yet for D_u^{pvt}. Let v_1 and v_2 be the child nodes of u. Then:

- D_u^{pvt} is the set of objects in D_u^{act} falling on the boundary of Δ_{v_1} or Δ_{v_2}. As D is in general position, the pivot set of u has only a constant number of objects.[8]
- For each $i \in [1, 2]$, $D_{v_i}^{act}$ is the set of objects in D_u^{act} that fall in the interior of Δ_{v_i}.

[6] In this paper, we follow the convention that the root of a tree is at level 0 and the level of a child is greater than that of the parent by one.

[7] In general, the "boundary" and "interior" of a polyhedron Δ in $\mathbb{R}^d$ are defined as follows. Let p be a point covered by Δ. If any ball centered at p with a positive radius contains a point outside Δ, p is on the *boundary* of Δ; otherwise, p is in the *interior*.

[8] Each of Δ_{v_1} or Δ_{v_2} has 4 facets, each of which can contain only one object.

Phrased differently, we "push down" an object $e \in D_u^{act}$ into the active set of v_1 if e falls in the interior of Δ_{v_1} (similarly for v_2). After the push-downs, the objects still in D_u^{act} constitute the pivot set of u.

The above inductive definition is completed by defining $D_z^{pvt} := D_z^{act}$ for every leaf z of $\mathcal{T}$.

Large and Small Keywords at a Node. Recall from (1) that, for each keyword $\mathtt{w} \in [1, W]$, $D(\mathtt{w})$ is the set of objects that contain $\mathtt{w}$ in their documents. In the same fashion, define for each $\mathtt{w} \in [1, W]$ and each node u in $\mathcal{T}$:

$$D_u^{act}(\mathtt{w}) := \{e \in D_u^{act} \mid \mathtt{w} \in e.\mathtt{Doc}\}. \tag{5}$$

For each node u in $\mathcal{T}$, let

$$N_u := \sum\nolimits_{e \in D_u^{act}} |e.\mathtt{Doc}|. \tag{6}$$

Observe that $N_u \leq |P_u| = O(N/2^{level(u)})$ because, for every object $e \in D_u^{act}$, all its copies in P are stored in the subtree of u.

For each $\mathtt{w} \in [1, W]$ and each node u in $\mathcal{T}$, we classify $\mathtt{w}$ as

- *large at* u if $|D_u^{act}(\mathtt{w})| \geq {N_u}^{1-1/k}$;
- *small at* u, otherwise.

As $\sum_{\mathtt{w}=1}^{W} D_u^{act}(\mathtt{w}) = N_u$, at most ${N_u}^{1/k}$ large keywords exist at u.

Structure. We associate each node u of $\mathcal{T}$ with a secondary structure T_u. The first purpose of T_u is to store the pivot set D_u^{pvt} of u. If u is a leaf, T_u has no other functionality. Otherwise, T_u should also support two operations in constant time:

- given a keyword $\mathtt{w}$, return whether $\mathtt{w}$ is large at u;
- given (i) k distinct keywords $\mathtt{w}_1, ..., \mathtt{w}_k$ that are large at u, and (ii) a child node v of u, return if $\bigcap_{i=1}^{k} D_v^{act}(\mathtt{w}_i)$ is empty.

Since at most ${N_u}^{1/k}$ keywords are large at u, it is rudimentary to implement T_u with $O({N_u}^{1/k})$ words plus $O(N_u)$ bits (specifically, create a hash table on the large keywords and a k-dimensional bit array where each cell indicates whether $\bigcap_{i=1}^{k} D_v^{act}(\mathtt{w}_i)$ is empty for a distinct combination of large $\mathtt{w}_1, ..., \mathtt{w}_k$).

Finally, to complete the structure, we choose some $D_u^{act}(\mathtt{w})$ — where u ranges over all the nodes in $\mathcal{T}$ and $\mathtt{w}$ ranges over $[1, W]$ — to store explicitly, in which case we say that $D_u^{act}(\mathtt{w})$ is *materialized*. Specifically, $D_u^{act}(\mathtt{w})$ is materialized if

- $\mathtt{w}$ is small at u;
- $\mathtt{w}$ is large at all the proper ancestors of u.

If either condition is violated, $D_u^{act}(\mathtt{w})$ remains conceptual and is never stored. We prove in Appendix B that the overall space consumption is $O(N)$ words.

3.3 Step 3: Bounding the Crossing Sensitivity

We now explain how to answer an ORP-KW query. Our discussion will bring out a notion — we call "crossing sensitivity" — which measures how "friendly" the underlying geometry index (here, the kd-tree) is to the transformation framework. Analyzing the crossing selectivity is typically the most non-trivial part of the framework.

Algorithm. Given a query with search rectangle q and keywords $\mathtt{w}_1, ..., \mathtt{w}_k$, we start by visiting the root of $\mathcal{T}$.

In general, to "visit" a node u, we read every object e in the node's pivot set, check whether e is covered by q and whether $e.\mathtt{Doc}$ contains[9] all of $\mathtt{w}_1, ..., \mathtt{w}_k$, and report e upon a "yes" answer to both questions. If u is a leaf, we are done.

The processing continues if u is internal. In that case, we use T_u to find out, in $O(1)$ time, if $\mathtt{w}_1, ..., \mathtt{w}_k$ are all large at u. If so, it may be necessary to visit the child nodes of u. Specifically, for each child v, we visit it only if two conditions are met:

- $\bigcap_{i=1}^{k} D_v^{act}(\mathtt{w}_i) \neq \emptyset$, and
- q has a non-empty intersection with cell Δ_v.

Checking the conditions takes $O(1)$ time (use T_u for the first condition).

It remains to discuss what happens if at least one of $\mathtt{w}_1, ..., \mathtt{w}_k$ is small at u; w.l.o.g., suppose that $\mathtt{w}_1$ is small. No proper descendant of u will be visited. Moreover, $D_u^{act}(\mathtt{w}_1)$ must have been materialized.[10] We read every object $e \in D_u^{act}(\mathtt{w}_1)$, check if $e \in q$ and $e.\mathtt{Doc}$ has all the other $k-1$ keywords, and report e upon "yes". The cost spent on u is $O({N_u}^{1-1/k})$ because $D_u^{act}(\mathtt{w}_1)$ has size at most ${N_u}^{1-1/k}$.

We prove in Appendix B that the algorithm correctly outputs all the objects in $q \cap D(\mathtt{w}_1, ..., \mathtt{w}_k)$.

Analysis. The set of nodes visited by the query algorithm forms a tree $\mathcal{T}_{qry}$. As can be easily verified, we pay constant time at every internal node of $\mathcal{T}_{qry}$, whereas we pay $O({N_z}^{1-1/k})$ time at every leaf node z of $\mathcal{T}_{qry}$ (note: z may not be a leaf of $\mathcal{T}$). Furthermore, every node u in $\mathcal{T}_{qry}$ must have a cell Δ_u intersecting with q.

We classify each node u of $\mathcal{T}_{qry}$ into two types:

- *covered*, if the cell Δ_u is (fully) covered by q;
- *crossing*, otherwise.

By resorting to Friedgut's inequality [29], we prove in Appendix B:

LEMMA 9. *The total cost spent on the covered nodes is* $O(N^{1-1/k}(1 + \mathrm{OUT}^{1/k}))$.

The challenging part is to bound the cost on the crossing nodes. To do so, let $\mathcal{T}_{cross}$ be the tree obtained from $\mathcal{T}_{qry}$ by deleting all the covered nodes. We define the *crossing sensitivity* of q as

$$\sum_{\text{internal } u \text{ of } \mathcal{T}_{cross}} 1 + \sum_{\text{leaf } z \text{ of } \mathcal{T}_{cross}} {N_z}^{1-1/k} \tag{7}$$

which is (asymptotically) the total cost on the crossing nodes of $\mathcal{T}_{qry}$.

The above serves as the template of query analysis for our framework. Lemma 9 often holds almost directly (with minor changes in the proof) for many choices of the underlying geometry index. What is specific to each individual choice is crossing sensitivity. Proving a small sensitivity must utilize the geometry index's properties, as we show for the kd-tree next.

Crossing Sensitivity of the kd-Tree. We will prove that any 2-rectangle has crossing sensitivity of $O(N^{1-1/k})$, which, together

[9] This check can be done in $O(k) = O(1)$ time, assuming a perfect hash table on $e.\mathtt{Doc}$. The hash tables of all objects use $O(N)$ space.

[10] Otherwise, u has a proper ancestor at which $\mathtt{w}_1$ is small, but in that scenario our algorithm would not have descended to u.

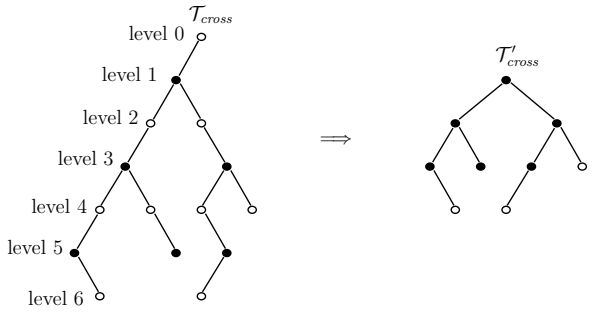

Figure 1: Compacting $\mathcal{T}_{cross}$ for bounding (8)

with Lemma 9, yields a bound of $O(N^{1-1/k}(1+\text{OUT}^{1/k}))$ on the ORP-KW query time.

We first deal with the special case where q is a vertical line. By standard kd-tree analysis [24], $\mathcal{T}_{cross}$ has $O(\sqrt{N})$ nodes[11]. Hence, $\sum_{\text{internal } u \text{ of } \mathcal{T}_{cross}} 1 = O(\sqrt{N}) = O(N^{1-1/k})$ as $k \geq 2$. Moreover, as any node u in the kd-tree satisfies $N_u = O(N/2^{level(u)})$, we have

$$\begin{aligned}
\sum_{\text{leaf } z \text{ of } \mathcal{T}_{cross}} N_z^{1-1/k} &= \sum_{\text{leaf } z \text{ of } \mathcal{T}_{cross}} O\left(\frac{N}{2^{level(z)}}\right)^{1-\frac{1}{k}} \\
&= O(N^{1-\frac{1}{k}}) \sum_{\text{leaf } z \text{ of } \mathcal{T}_{cross}} \left(\frac{1}{2^{level(z)}}\right)^{1-\frac{1}{k}} \\
(\text{as } k \geq 2) \quad &= O(N^{1-\frac{1}{k}}) \sum_{\text{leaf } z \text{ of } \mathcal{T}_{cross}} \sqrt{\frac{1}{2^{level(z)}}}. \qquad (8)
\end{aligned}$$

To proceed, we must resort to the properties of the kd-tree. Recall that, if a node u is at an even level, its cell is split by a vertical line. Let us first consider that q is different from all those split lines. Crucially, in $\mathcal{T}_{cross}$, every internal node of an even level has only one child![12] Motivated by this, we "compact" $\mathcal{T}_{cross}$ by doing the following for each even-level internal node u in $\mathcal{T}_{cross}$: delete u and make the parent of u the new parent of the only child of u; see Figure 1 for an illustration. Let $\mathcal{T}'_{cross}$ be the tree after the compaction. For each leaf z of $\mathcal{T}_{cross}$, its new level in $\mathcal{T}'_{cross}$ — denoted as $level'(z)$ — equals $\lfloor level(z)/2 \rfloor$. We have:

$$\sum_{\text{leaf } z \text{ of } \mathcal{T}_{cross}} \sqrt{\frac{1}{2^{level(z)}}} \leq \sum_{\text{leaf } z \text{ of } \mathcal{T}'_{cross}} 2^{-level'(z)} \leq 1.$$

where the second inequality used the fact [23] that, in any binary tree, $\sum_{\text{leaf } z} 1/2^{\text{level of } z \text{ in the tree}} \leq 1$.

We now drop the assumption that q differs from all nodes' split lines. Without the assumption, some even-level nodes in $\mathcal{T}_{cross}$ can have two children. In Appendix B, we argue that there cannot be too many such nodes and the above argument can be extended to prove:

LEMMA 10. *Expression* (8) *is* $O(N^{1-1/k})$ *for any vertical line* q.

We thus have proved that any vertical line has crossing sensitivity $O(N^{1-1/k})$. An analogous argument shows that the crossing sensitivity of any horizontal line is also $O(N^{1-1/k})$. It is thus clear that the crossing sensitivity is still $O(N^{1-1/k})$ for any 2-rectangle q (i.e., at most the total crossing sensitivity of the four lines passing through the four boundary edges of q, respectively).

[11] Any axis-parallel line intersects the cells of $O(\sqrt{N})$ nodes in a kd-tree on N points.
[12] Let u be such a node. Δ_u is split by a vertical line ℓ into two rectangles, one of which must be disjoint with q.

3.4 Step 4: Removing General Position

So far we have assumed that no two objects in D share the same x- or y-coordinate. The assumption can be easily eliminated by converting coordinates into the "rank space". To that end, sort the objects on each dimension and break ties by favoring the object with a smaller id. It is standard to convert a 2-rectangle of the original space into a 2-rectangle of the rank space in $O(\log N)$ time without affecting the query result. We now complete the whole proof of Theorem 1.

3.5 Remarks

Our solution was inspired an index of Cohen and Porat [23] for the 2-SI problem defined in Section 1.2 (i.e., $k = 2$ and no geometry predicates). They also classified keywords as "large" or "small" and stored hash tables and bit arrays. However, all the ideas about integration with the kd-tree are new. Several of our techniques are particularly crucial. The first one is to define the active and pivot sets by distinguishing objects falling in the cell interior or on the cell boundaries. The second is the separation between covered and crossing nodes and then the introduction of "crossing sensitivity". The third is the analysis of the kd-tree's crossing sensitivity. The last one is the four-step framework itself, whose power will be showcased again in Appendix D on the LC-KW problem.

We close the section with a note that, although the kd-tree also works for $d \geq 3$, its conversion to ORP-KW will suffer from a query time $O(N^{1-1/\max\{k,d\}} + N^{1-1/k}\text{OUT}^{1/k}))$. We will not delve into this further because nearly the same performance can be obtained through LC-KW (Theorem 5). The next section will improve the query time dramatically to $O(N^{1-1/k}(1+\text{OUT}^{1/k}))$ with a different technique.

4 A DIMENSION REDUCTION TECHNIQUE UNDER KEYWORDS

This section is dedicated to Theorem 2 for ORP-KW, in particular, why it suffices to pay only an extra $O(\log\log N)$ factor in space every time the dimensionality increases by 1. We will prove:

LEMMA 11. *For ORP-KW, if we can build an index of* $O(N(\log\log N)^{\lambda-2})$ *space and* $O(N^{1-1/k}(1+\text{OUT}^{1/k}))$ *query time for* $d = \lambda \geq 2$*, then we can build an index of* $O(N(\log\log N)^{\lambda-1})$ *space and* $O(N^{1-1/k}(1+\text{OUT}^{1/k}))$ *query time for* $d = \lambda + 1$.

Combining the lemma with Theorem 1 yields Theorem 2. The rest of the section serves as a proof of the lemma. Our discussion will henceforth focus on ORP-KW in $\mathbb{R}^{\lambda+1}$. For simplicity, we will call dimension 1 of $\mathbb{R}^{\lambda+1}$ the "x-dimension" and a coordinate on this dimension an "x-coordinate". Recall that the input D is a set of points (a.k.a. objects) in $\mathbb{R}^{\lambda+1}$, where each object $e \in D$ is associated with a non-empty set $e.\texttt{Doc}$ of integers (a.k.a. keywords).

Consider any subset $D' \subseteq D$. We define

$$weight(D') := \sum_{e \in D'} |e.\texttt{Doc}|. \qquad (9)$$

Note that $weight(D') \geq |D'|$ always holds. Given also an integer $f \geq 2$, we define an *f-balanced cut* of D' as a tuple $(D_1, D_2, ..., D_f, e_1^*, e_2^*, ..., e_{f-1}^*)$ where

- for each $i \in [1, f]$, D_i is a (possibly empty) subset of D', and for each $i \in [1, f-1]$, e_i^* is either an object in D' or null;
- $D_1, D_2, ..., D_f, \{e_1^*\}, \{e_2^*\}, ..., \{e_{f-1}^*\}$ are mutually disjoint and have D' as the union;
- for any $1 \leq i < j \leq f$, every object in D_i has a smaller x-coordinate than all objects in D_j;
- $weight(D_i) \leq weight(D')/f$ for all $i \in [1, f]$, where the $weight(.)$ function is given in (9).

Such a cut always exists regardless of D' and f.[13]

Structure. We will build a tree $\mathcal{T}$ where each node u is associated with an active set D_u^{act} and a pivot set D_u^{pvt} satisfying $D_u^{pvt} \subseteq D_u^{act} \subseteq D$. What is special about $\mathcal{T}$ is that the node fanouts (i.e., the number of children of a node) are set in an unusual manner. To kick off an inductive definition, set $D_u^{act} := D$ if u is the root of $\mathcal{T}$.

In general, suppose that u is a node of $\mathcal{T}$ whose D_u^{act} has been defined, but not yet for D_u^{pvt}. Denote by $level(u)$ the level of u (level 0 for root and level increases by 1 with each child descent). Define

$$f_u := 2 \cdot 2^{k^{level(u)}} \tag{10}$$

where k is the number of keywords issued by an ORP-KW query. Take an arbitrary f_u-balanced cut $(D_1, ..., D_{f_u}, e_1^*, ..., e_{f_u-1}^*)$ of D_u. The pivot set of u can now be finalized as $D_u^{pvt} := \{e_1^*, ..., e_{f_u-1}^*\}$. If $D_1, ..., D_{f_u}$ are all empty, we make u a leaf of $\mathcal{T}$. Otherwise, for every non-empty D_i ($i \in [1, f_u]$), create a child v for u whose active set is $D_v^{act} := D_i$. This completes the definition of $\mathcal{T}$.

For each node u in $\mathcal{T}$, we build a secondary structure T_u to answer queries of the form: given k keywords $\mathtt{w}_1, ..., \mathtt{w}_k$ and a $(\lambda+1)$-rectangle q whose x-projection is $(-\infty, \infty)$, report the objects $e \in D_u^{act}$ such that e (which is a point) is covered by q, and $e.\mathtt{Doc}$ contains all of $\mathtt{w}_1, ..., \mathtt{w}_k$. Note that T_u is merely an ORP-KW index of $d = \lambda$ (because we can ignore the x-dimension), which is already available according to the statement of Lemma 11. To complete the whole index, we store the pivot set of each node in $\mathcal{T}$ explicitly.

Query. To explain our query algorithm, for each node u in $\mathcal{T}$, define $\sigma(u)$ as the tightest interval covering the x-coordinates of all the points in D_u^{act}. Also, given a $(\lambda+1)$-rectangle q, let $q[i]$ represent the interval that is the projection of q on dimension $i \in [1, \lambda+1]$; note that $q[1]$ is the interval for the x-dimension.

Now, consider a query with $(\lambda+1)$-rectangle q and keywords $\mathtt{w}_1, ..., \mathtt{w}_k$. To answer it, we visit every node u of $\mathcal{T}$ such that

- $\sigma(u) \cap q[1] \neq \emptyset$, and
- no proper ancestor v of u has $\sigma(v)$ contained in $q[1]$.

[13] We describe a simple algorithm here. Sort the objects of D' by x-coordinate. Scan the objects in the sorted order and pack as many into D_1 as possible subject to $weight(D_1) \leq weight(D')/f$. Then, set e_1^* to the next object in the sorted list. Continue the scan and pack as many into D_2 as possible subject to $weight(D_2) \leq weight(D')/f$, and set e_2^* to the next object. Repeat until exhausting the list.

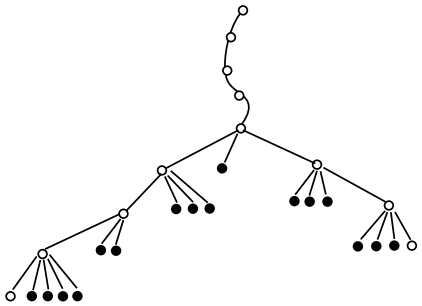

Figure 2: Nodes in $\mathcal{T}_{qry}$: black for type 1 and white for type 2

Denote by $\mathcal{T}_{qry}$ the tree induced by the nodes visited. Divide the nodes u of $\mathcal{T}_{qry}$ into:

- (type 1) $\sigma(u)$ is contained in $q[1]$;
- (type 2) the rest.

Each level of $\mathcal{T}$ can have up to two nodes of type-2; see Figure 2 for an illustration. At each type-1 node u, we use T_u to report the objects $e \in D_u^{act}$ satisfying (i) $e.\mathtt{Doc}$ has all of $\mathtt{w}_1, ..., \mathtt{w}_k$ and (ii) e falls in the $(\lambda+1)$-rectangle $(-\infty, \infty) \times q[2] \times ... \times q[\lambda+1]$. At each type-2 node u, we pay $O(|D_u^{pvt}|) = O(f_u)$ time to examine all the objects in its pivot set and report those satisfying the query predicate.

Analysis. Let us start with three technical propositions, whose proofs can be found in Appendix C.

PROPOSITION 1. *$\mathcal{T}$ has $O(\log\log N)$ levels.*

PROPOSITION 2. *For any node u of $\mathcal{T}$, $weight(D_u^{act}) \cdot f_u^{\frac{1}{k-1}} = O(N/2^{level(u)})$.*

PROPOSITION 3. *For any node u of $\mathcal{T}$, $f_u = O(N^{1-1/k})$.*

To understand the space complexity of our index, first note that $\mathcal{T}$ itself uses $O(N)$ space, and all the pivot sets of the nodes in $\mathcal{T}$ take up $O(N)$ space in total[14]. Next, we discuss the secondary structures T_u of the nodes u in $\mathcal{T}$. Each T_u — which, as mentioned, is a λ-dimensional index on D_u^{act} — occupies $O(weight(D_u^{act}) \cdot (\log\log N)^{\lambda-2})$ space (as is a given condition in Lemma 11). If we sum up the term $weight(D_u^{act})$ for all the nodes u at the same level of $\mathcal{T}$, we get at most N because every object in D belongs to the active set of at most one node at this level. Hence, all the secondary structures of the nodes at each level of $\mathcal{T}$ occupy $O(N \cdot (\log\log N)^{\lambda-2})$ space in total. As $\mathcal{T}$ has $O(\log\log N)$ levels (Proposition 1), the overall space usage is $O(N \cdot (\log\log N)^{\lambda-1})$.

The subsequent discussion focuses on query time. We analyze the cost spent on type-1 and -2 nodes separately. Recall that each level of $\mathcal{T}$ has at most two type-2 nodes, and we pay $O(f_u)$ time for every such node u. By Propositions 1 and 3, it is easy to bound the cost of all type-2 nodes as $O(N^{1-1/k} \log\log N)$. However, we can eliminate the $O(\log\log N)$ factor by observing that f_u, given in (10), increases quickly with $level(u)$. Hence, the total cost spent on all the type-2 nodes is asymptotically dominated by the term f_u of the *deepest* type-2 node, which is $O(N^{1-1/k})$ (Proposition 3).

[14] Every object of D appears in exactly one pivot set.

It remains to bound the cost on type-1 nodes. Recall that, on every such node u, we issue a λ-dimensional ORP-KW query on its secondary structure T_u. If this query returns OUT_u objects, by the condition given in Lemma 11, its cost is $O(weight(D_u^{act})^{1-1/k}(1 + \text{OUT}_u^{1/k}))$. Therefore, the total cost spent on all the type-1 nodes is asymptotically:

$$\sum_{\text{type-1 } u} \left(weight(D_u^{act})^{1-1/k} + weight(D_u^{act})^{1-1/k} \cdot \text{OUT}_u^{1/k} \right). \quad (11)$$

Observe that all the type-1 nodes u have disjoint x-ranges $\sigma(u)$. Hence, every object in D is in the active set of at most one type-1 node, which gives us $\sum_{\text{type-1 } u} weight(D_u^{act}) \leq N$. Moreover, we obviously have $\sum_{\text{type-1 } u} \text{OUT}_u \leq \text{OUT}$. Thus, Friedgut's inequality (see Appendix A) immediately yields:

$$\sum_{\text{type-1 } u} weight(D_u^{act})^{1-1/k} \cdot \text{OUT}_u^{1/k} \leq N^{1-1/k} \cdot \text{OUT}^{1/k}. \quad (12)$$

In the argument below, we will prove $\sum_{\text{type-1 } u} weight(D_u^{act})^{1-1/k} = O(N^{1-1/k})$, which will complete the whole proof of Lemma 11 together with (11) and (12).

Every type-1 node's parent is of type 2. On the other hand, for every type-2 node v, we have

$$\begin{aligned} & \sum_{\text{type-1 child } u \text{ of } v} weight(D_u^{act})^{1-1/k} \\ \leq & \sum_{\text{type-1 child } u \text{ of } v} \left(\frac{weight(D_v^{act})}{f_v} \right)^{1-1/k} \\ \leq & f_v \cdot \left(\frac{weight(D_v^{act})}{f_v} \right)^{1-\frac{1}{k}} = \left(weight(D_v^{act}) \cdot f_v^{\frac{1}{k-1}} \right)^{1-\frac{1}{k}} \\ = & O\left(\left(\frac{N}{2^{level(v)}} \right)^{1-1/k} \right) \end{aligned}$$

where the last step used Proposition 2. Therefore:

$$\begin{aligned} & \sum_{\text{type-1 } u} weight(D_u^{act})^{1-1/k} \\ \leq & \sum_{\text{type-2 } v} \sum_{\text{type-1 child } u \text{ of } v} weight(D_u^{act})^{1-1/k} \\ = & \sum_{\text{type-2 } v} O\left(\left(\frac{N}{2^{level(v)}} \right)^{1-1/k} \right) \end{aligned}$$

which is $O(N^{1-1/k})$ because at most two type-2 nodes exist at each level of $\mathcal{T}$, and $(\frac{N}{2^{level(v)}})^{1-1/k}$ decreases geometrically with $level(v)$.

ACKNOWLEDGEMENTS

This work was supported in part by GRF projects 14207820, 14203421, and 14222822 from HKRGC.

REFERENCES

[1] Peyman Afshani and Timothy M. Chan. 2009. Optimal halfspace range reporting in three dimensions. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 180–186.

[2] Peyman Afshani and Jesper Sindahl Nielsen. 2016. Data Structure Lower Bounds for Document Indexing Problems. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*. 93:1–93:15.

[3] Pankaj K. Agarwal. 2004. Range Searching. In *Handbook of Discrete and Computational Geometry, 2nd Ed.* Chapman and Hall/CRC, 809–837.

[4] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. 2002. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *Proceedings of International Conference on Data Engineering (ICDE)*. 5–16.

[5] Abdulaziz Almaslukh and Amr Magdy. 2018. Evaluating spatial-keyword queries on streaming data. In *Proceedings of International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*. 209–218.

[6] Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. 2020. Dynamic and Internal Longest Common Substring. *Algorithmica* 82, 12 (2020), 3707–3743.

[7] Avishek Anand, Srikanta J. Bedathur, Klaus Berberich, and Ralf Schenkel. 2010. Efficient temporal keyword search over versioned text. In *Proceedings of Conference on Information and Knowledge Management (CIKM)*. 699–708.

[8] Franz Aurenhammer. 1987. A Criterion for the Affine Equivalence of Cell Complexes in R^d and Convex Polyhedra in R^{d+1}. *Discrete & Computational Geometry* 2 (1987), 49–64.

[9] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. 2002. Keyword Searching and Browsing in Databases using BANKS. In *Proceedings of International Conference on Data Engineering (ICDE)*. 431–440.

[10] Philip Bille, Inge Li Gortz, Max Rishoj Pedersen, and Teresa Anna Steiner. 2021. Gapped Indexing for Consecutive Occurrences. In *Proceedings of Annual Symposium on Combinatorial Pattern Matching (CPM)*. 10:1–10:19.

[11] Philip Bille, Anna Pagh, and Rasmus Pagh. 2007. Fast Evaluation of Union-Intersection Expressions. In *International Symposium on Algorithms and Computation (ISAAC)*. 739–750.

[12] Ariel Cary, Ouri Wolfson, and Naphtali Rishe. 2010. Efficient and Scalable Method for Processing Top-k Spatial Boolean Queries. In *Proceedings of Scientific and Statistical Database Management (SSDBM)*. 87–95.

[13] Timothy M. Chan. 2012. Optimal Partition Trees. *Discrete & Computational Geometry* 47, 4 (2012), 661–690.

[14] Timothy M. Chan, Saladi Rahul, and Jie Xue. 2020. Range closest-pair search in higher dimensions. *Comput. Geom.* 91 (2020), 101669.

[15] Panagiotis Charalampopoulos, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. 2021. An Almost Optimal Edit Distance Oracle. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*. 48:1–48:20.

[16] Bernard Chazelle. 1988. A Functional Approach to Data Structures and Its Use in Multidimensional Searching. *SIAM Journal of Computing* 17, 3 (1988), 427–462.

[17] Gang Chen, Jingwen Zhao, Yunjun Gao, Lei Chen, and Rui Chen. 2018. Time-Aware Boolean Spatial Keyword Queries (Extended Abstract). In *Proceedings of International Conference on Data Engineering (ICDE)*. 1781–1782.

[18] Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. 2013. Spatial Keyword Query Processing: An Experimental Evaluation. *Proceedings of the VLDB Endowment (PVLDB)* 6, 3 (2013), 217–228.

[19] Lisi Chen, Yan Cui, Gao Cong, and Xin Cao. 2014. SOPS: A System for Efficient Processing of Spatial-Keyword Publish/Subscribe. *Proceedings of the VLDB Endowment (PVLDB)* 7, 13 (2014), 1601–1604.

[20] Lisi Chen, Shuo Shang, Chengcheng Yang, and Jing Li. 2020. Spatial keyword search: a survey. *GeoInformatica* 24, 1 (2020), 85–106.

[21] Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. 2006. Efficient query processing in geographic web search engines. In *Proceedings of ACM Management of Data (SIGMOD)*. 277–288.

[22] Zhida Chen, Lisi Chen, Gao Cong, and Christian S. Jensen. 2021. Location- and keyword-based querying of geo-textual data: a survey. *The VLDB Journal* 30, 4 (2021), 603–640.

[23] Hagai Cohen and Ely Porat. 2010. Fast set intersection and two-patterns matching. *Theoretical Computer Science* 411, 40-42 (2010), 3795–3800.

[24] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. *Computational Geometry: Algorithms and Applications* (3rd ed.). Springer-Verlag.

[25] Shaleen Deep and Paraschos Koutris. 2018. Compressed Representations of Conjunctive Query Results. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*. 307–322.

[26] Herbert Edelsbrunner and Ernst P. Mucke. 1990. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.* 9, 1 (1990), 66–104.

[27] David Eppstein, Michael T. Goodrich, Michael Mitzenmacher, and Manuel R. Torres. 2017. 2-3 Cuckoo Filters for Faster Triangle Listing and Set Intersection. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*. 247–260.

[28] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. 2008. Keyword Search on Spatial Databases. In *Proceedings of International Conference on Data Engineering (ICDE)*. 656–665.

[29] Ehud Friedgut. 2004. Hypergraphs, Entropy, and Inequalities. *Am. Math. Mon.* 111, 9 (2004), 749–760.

[30] Roy Goldman, Narayanan Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. 1998. Proximity Search in Databases. In *Proceedings of Very Large Data Bases (VLDB)*. 26–37.

[31] Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. 2017. Conditional Lower Bounds for Space/Time Tradeoffs. In *Algorithms and Data Structures*

Workshop (WADS). Springer, 421–436.
[32] Isaac Goldstein, Moshe Lewenstein, and Ely Porat. 2019. On the Hardness of Set Disjointness and Set Intersection with Bounded Universe. In *International Symposium on Algorithms and Computation (ISAAC)*. 7:1–7:22.
[33] Michael T. Goodrich. 2017. Answering Spatial Multiple-Set Intersection Queries Using 2-3 Cuckoo Hash-Filters. *CoRR* abs/1708.09059 (2017).
[34] Ramaswamy Hariharan, Bijit Hore, Chen Li, and Sharad Mehrotra. 2007. Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems. In *Proceedings of Scientific and Statistical Database Management (SSDBM)*. 16.
[35] Tuan-Anh Hoang-Vu, Huy T. Vo, and Juliana Freire. 2016. A Unified Index for Spatio-Temporal Keyword Queries. In *Proceedings of Conference on Information and Knowledge Management (CIKM)*. 135–144.
[36] Vagelis Hristidis and Yannis Papakonstantinou. 2002. DISCOVER: Keyword Search in Relational Databases. In *Proceedings of Very Large Data Bases (VLDB)*. 670–681.
[37] Wenyu Huo and Vassilis J. Tsotras. 2012. A Comparison of Top-k Temporal Keyword Querying over Versioned Text Collections. In *Proceedings of International Conference on Database and Expert Systems Applications (DEXA)*. 360–374.
[38] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. 2015. Dynamic Set Intersection. In *Algorithms and Data Structures Workshop (WADS)*. 470–481.
[39] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. 2016. Higher Lower Bounds from the 3SUM Conjecture. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1272–1287.
[40] Tsvi Kopelowitz and Virginia Vassilevska Williams. 2020. Towards Optimal Set-Disjointness and Set-Intersection Data Structures. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, Vol. 168. 74:1–74:16.
[41] Taesung Lee, Jin-Woo Park, Sanghoon Lee, Seung-won Hwang, Sameh Elnikety, and Yuxiong He. 2015. Processing and Optimizing Main Memory Spatial-Keyword Queries. *Proceedings of the VLDB Endowment (PVLDB)* 9, 3 (2015), 132–143.
[42] Zhisheng Li, Ken C. K. Lee, Baihua Zheng, Wang-Chien Lee, Dik Lun Lee, and Xufa Wang. 2011. IR-Tree: An Efficient Index for Geographic Document Search. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 23, 4 (2011), 585–599.
[43] Ute Masermann and Gottfried Vossen. 2000. SISQL: Schema-Independent Database Querying (On and Off the Web). In *Proceedings of International Database Engineering and Applications Symposium (IDEAS)*. 55–64.
[44] Jiri Matousek. 1992. Efficient Partition Trees. *Discrete & Computational Geometry* 8 (1992), 315–334.
[45] Jiri Matousek. 1992. Reporting Points in Halfspaces. *Computational Geometry* 2 (1992), 169–186.
[46] Mihai Patrascu. 2010. Towards polynomial lower bounds for dynamic problems. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*. 603–610.
[47] Saladi Rahul and Yufei Tao. 2016. Efficient Top-k Indexing via General Reductions. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*. 277–288.
[48] Yufei Tao and Cheng Sheng. 2014. Fast Nearest Neighbor Search with Keywords. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 26, 4 (2014), 878–888.
[49] Subodh Vaid, Christopher B. Jones, Hideo Joho, and Mark Sanderson. 2005. Spatio-textual Indexing for Geographical Search on the Web. In *Proceedings of Symposium on Advances in Spatial and Temporal Databases (SSTD)*, Vol. 3633. 218–235.
[50] Dingming Wu, Gao Cong, and Christian S. Jensen. 2012. A framework for efficient spatial web object retrieval. *The VLDB Journal* 21, 6 (2012), 797–822.
[51] Dingming Wu, Man Lung Yiu, Gao Cong, and Christian S. Jensen. 2012. Joint Top-K Spatial Keyword Query Processing. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 24, 10 (2012), 1889–1903.
[52] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. 2016. Inverted Linear Quadtree: Efficient Top K Spatial Keyword Search. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 28, 7 (2016), 1706–1721.
[53] Dongxiang Zhang, Chee-Yong Chan, and Kian-Lee Tan. 2014. Processing spatial keyword query as a top-k aggregation query. In *International Conference on Research and Development in Information Retrieval (SIGIR)*. 355–364.

APPENDIX

A FRIEDGUT'S INEQUALITY

Let $a_1, a_2, ..., a_n$ and $b_1, b_2, ..., b_n$ be non-negative real values where $n \geq 1$. For any real value $\phi \in [0, 1]$, it holds [29] that

$$\sum_{i=1}^{n} a_i^{\phi} b_i^{1-\phi} \quad \leq \quad \left(\sum_{i=1}^{n} a_i\right)^{\phi} \cdot \left(\sum_{i=1}^{n} b_i\right)^{1-\phi}. \tag{13}$$

B SUPPLEMENTARY PROOFS FOR SECTION 3

Space Consumption. $\mathcal{T}$ itself obviously uses $O(N)$ space. The total space of all the materialized $D_u^{act}(\mathtt{w})$ — ranging over all nodes u in $\mathcal{T}$ and keywords $\mathtt{w} \in [1, W]$ — is $O(N)$ because every object $e \in D$ can appear in at most $|e.\mathtt{Doc}|$ materialized sets. As explained in Section 3.2, the secondary structure T_u at each node u uses $O(N^{1/k})$ words plus $O(N_u)$ bits. We will prove $\sum_{\text{node } u} N^{1/k} = O(N)$ and $\sum_{\text{node } u} N_u = O(N \log N)$. The space of all the secondary structures is thus $O(N)$ words plus $O(N \log N)$ bits, which is $O(N)$ words.

In fact, $\sum_{\text{node } u} N_u = O(N \log N)$ is somewhat obvious because $\mathcal{T}$ has $O(\log N)$ levels, and the sum of N_u for all the nodes u at the same level is $O(N)$ (an object can appear in the active set of at most one node at the same level).

Now, we show $\sum_{\text{node } u} N^{1/k} = O(N)$. At each level $i \in [0, h]$ where $h := O(1) + \log_2 N$ is the maximum level of $\mathcal{T}$, there are at most 2^i nodes u in $\mathcal{T}$, each satisfying $N_u = O(N/2^i)$. Therefore

$$\begin{aligned}
\sum_{\text{node } u} N_u^{1/k} &= \sum_{i=0}^{h} \sum_{\text{node } u \text{ at level } i} O(N_u^{1/k}) \\
&= \sum_{i=0}^{h} O\left(2^i \cdot (N/2^i)^{1/k}\right) \\
&= \sum_{i=0}^{h} O\left(2^{i(1-1/k)} \cdot N^{1/k}\right) = O(N).
\end{aligned}$$

Correctness of the Query Algorithm. Consider an arbitrary object $e \in q \cap D(\mathtt{w}_1, ..., \mathtt{w}_k)$. We will prove that our algorithm manages to output e. For each $i \in [1, k]$, pinpoint a node u_i as follows. If e is in some materialized $D_v^{act}(\mathtt{w}_i)$, set u_i to the node v at which $D_v^{act}(\mathtt{w}_i)$ is defined; otherwise, set u to the node v whose pivot set contains e. All of $u_1, ..., u_k$ must be on the same root-to-leaf path of $\mathcal{T}$ (because their active sets all contain e). Assume, w.l.o.g., that u_1 is the node closest to the root among $u_1, ..., u_k$.

Let π be the path from the root of to u_1. We will show that our algorithm visits every node v on π. This is obvious if v is the root. Now consider v as a non-root node. By the way $u_1, ..., u_k$ are defined, all the keywords $\mathtt{w}_1, ..., \mathtt{w}_k$ are large at the parent of v. We must have $\bigcap_{i=1}^{k} D_v^{act}(\mathtt{w}_i) \neq \emptyset$, because at least e is in the intersection. Furthermore, the cell Δ_v of v must contain e and, hence, intersect with q. It thus follows that our algorithm visits v for sure.

We can now complete the proof by observing that the algorithm outputs e during its visit to u_1.

Proof of Lemma 9. Let us first consider that the search rectangle q does not cover the whole space $\mathbb{R}^2$. In this scenario, at every covered leaf z of $\mathcal{T}_{qry}$, our algorithm outputs at least one *distinct* object in its active set D_z^{act}. To explain, let u be the parent of z in $\mathcal{T}$.[15]. The algorithm's visit to z suggests that $\bigcap_{i=1}^{k} D_z^{act}(\mathtt{w}_i)$ must be non-empty (recall that $\mathtt{w}_1, ..., \mathtt{w}_k$ are the query keywords). Moreover, z being covered means that the cell Δ_z is contained in q. Hence, all the objects in $\bigcap_{i=1}^{k} D_z^{act}(\mathtt{w}_i)$ must be reported. None of those objects

[15]The parent exists because z cannot be the root, as the root's cell is $\mathbb{R}^2$.

can belong to the active set of another covered leaf because the active sets of all covered leaves are disjoint.

It thus follows that $\mathcal{T}_{qry}$ has at most OUT covered leaves. If OUT = 0, $\mathcal{T}_{qry}$ has no covered leaves and, hence, no covered internal nodes as well (because any leaf descendant of a covered internal node must also be covered). The lemma trivially holds in the absence of covered nodes.

When OUT > 0, $\mathcal{T}_{qry}$ having at most OUT covered leaves means that $\mathcal{T}_{qry}$ has $O(\text{OUT}\log(N/\text{OUT}))$ covered internal nodes.[16] We spend constant time on each of those internal nodes and, hence, $O(\text{OUT}\log(N/\text{OUT}))$ time in total. On the other hand, at every covered leaf z, we pay a cost of $O(N_z^{1-1/k})$. The sum of N_z for all such leaves is N because their active sets are disjoint. Thus, the total cost spent on the covered leaves is asymptotically

$$\begin{aligned}\sum_{\text{covered leaf } z \text{ of } \mathcal{T}_{qry}} N_z^{1-1/k} &= \sum_{\text{covered leaf } z \text{ of } \mathcal{T}_{qry}} N_z^{1-1/k} \cdot 1^{1/k} \\ \text{(using (13))} &\leq N^{1-1/k}\text{OUT}^{1/k}.\end{aligned}$$

We can now conclude that the total cost on the covered nodes of $\mathcal{T}_{qry}$ is $O(\text{OUT}\log(N/\text{OUT}) + N^{1-1/k}\text{OUT}^{1/k}) = O(N^{1-1/k}\text{OUT}^{1/k})$.

Finally, let us consider the case where $q = \mathbb{R}^2$. If the root of $\mathcal{T}$ is not the only node in $\mathcal{T}_{qry}$, the above analysis applies directly. Otherwise, it is easy to verify that the query time is $O(N^{1-1/k})$.

Proof of Lemma 10. Consider any vertical line q and any leaf z of $\mathcal{T}_{cross}$. Let π be the root-to-z path of $\mathcal{T}_{cross}$. At most one node on π can have (i) q as its split line and (ii) two child nodes in $\mathcal{T}_{cross}$. To explain, let u be the highest node on π satisfying conditions (i) and (ii). For every proper descendant v of u on π, q is either disjoint with Δ_v or passes a vertical boundary of Δ_v. In the former case, the split line of Δ_v obviously cannot be q. In the latter case, if q is the split line of Δ_v, then one of the rectangles produced by splitting Δ_v with q must be contained in q. Hence, at most one child of v can be crossing (the other child must be covered).

The above observation allows us to compact $\mathcal{T}_{cross}$ in a way slightly different from Section 3.3: for each even-level internal node u in $\mathcal{T}_{cross}$ *which has only one child in* $\mathcal{T}_{cross}$, delete u and make the parent of u the new parent of the only child of u. Let $\mathcal{T}'_{cross}$ be the tree after the compaction.

For each leaf z of $\mathcal{T}_{cross}$, its new level in $\mathcal{T}'_{cross}$ — denoted as $level'(z)$ — is either $\lfloor level(z)/2 \rfloor$ or $\lfloor level(z)/2 \rfloor + 1$. We have

$$\sum_{\text{leaf } z \text{ of } \mathcal{T}_{cross}} \sqrt{\frac{1}{2^{level(z)}}} \leq \sum_{\text{leaf } z \text{ of } \mathcal{T}'_{cross}} 2^{-(level'(z)-1)} \leq 2.$$

where the last inequality used the fact [23] that, in any binary tree, $\sum_{\text{leaf } z} 1/2^{\text{level of } z \text{ in the tree}} \leq 1$. Therefore, the expression in (8) evaluates to $O(N^{1-1/k})$.

[16] Consider the tree obtained by deleting all the crossing nodes in $\mathcal{T}_{qry}$. This is a binary tree with at most OUT leaves and $O(\log N)$ height. It is easy to verify that the tree can have $O(\text{OUT}\log(N/\text{OUT}))$ internal nodes.

C SUPPLEMENTARY PROOFS FOR SECTION 4

Proof of Proposition 1. Our construction, based on f-balanced cuts with the design of f in (10), guarantees for any node u of $\mathcal{T}$:

$$\begin{aligned}weight(D_u^{act}) &\leq \frac{N}{\prod_{i=0}^{level(u)-1}(2 \cdot 2^{k^i})} \\ &= \frac{N}{2^{\Theta(k^{level(u)})}}.\end{aligned} \tag{14}$$

Furthermore, $weight(D_u^{act}) \geq |D_u^{act}| \geq 1$. It thus follows that $level(u) = O(\log\log N)$.

Proof of Proposition 2. By calculating precisely the denominator on the right hand side of Inequality (14), we get:

$$\begin{aligned}weight(D_u^{act}) &\leq \frac{N}{2^{level(u)} \cdot 2^{\frac{k^{level(u)}-1}{k-1}}} \\ &= O\left(\frac{N}{2^{level(u)} \cdot 2^{\frac{k^{level(u)}}{k-1}}}\right) = O\left(\frac{N}{2^{level(u)} \cdot f_u^{1/(k-1)}}\right).\end{aligned}$$

Rearranging the terms proves the proposition.

Proof of Proposition 3. If u is an internal node of $\mathcal{T}$, we must have $weight(D_u^{act}) \geq f_u$. To see why, consider an arbitrary child node v of u. Our construction algorithm ensures $1 \leq weight(D_v^{act}) \leq weight(D_u^{act})/f_u$, which implies $weight(D_u^{act}) \geq f_u$. Using this inequality, we can derive $f_u \cdot f_u^{1/(k-1)} = O(N)$ from Proposition 2, which simplifies into $f_u = O(N^{1-1/k})$.

Now, consider u as a leaf node. If $weight(D_u^{act}) > N^{1-1/k}$, Proposition 2 indicates $f_u^{1/(k-1)} = O(N^{1/k})$, which again simplifies into $f_u = O(N^{1-1/k})$. If, on the other hand, $weight(D_u^{act}) \leq N^{1-1/k}$, then trivially $f_u \leq |D_u^{act}| \leq weight(D_u^{act}) \leq N^{1-1/k}$.

D PROOF OF THEOREM 5

This section will utilize our index transformation framework to convert the partition tree [13] to an index for answering LC-KW queries with the performance guarantees in Theorem 5.

We will deal with an alternative problem, which captures the essence of LC-KW. Let us first recall that a *d-simplex* is a polyhedron in $\mathbb{R}^d$ with $d+1$ facets. For example, a 0-simplex is a point, a 1-simplex is a line segment, a 2-simplex is a triangle, a 3-simplex is a tetrahedron, etc. The problem we will focus on is:

> *Simplex Reporting with Keywords* (SP-KW). D is a set of points in $\mathbb{R}^d$, where $d \geq 1$ is a constant. Fix an integer $k \geq 2$. Given a d-simplex q in $\mathbb{R}^d$ and keywords $w_1, ..., w_k$, a query returns $q \cap D(w_1, ..., w_k)$, where $D(w_1, ..., w_k)$ is given in (1).

We will prove:

THEOREM 12. *For SP-KW with $d \leq k$, there is an index of $O(N)$ space that answers a query in $O(N^{1-1/k} \cdot (\log N + \text{OUT}^{1/k}))$ time, where N is the input size, k is the number of query keywords, and* OUT *is the number of points reported. For SP-KW with $d > k$, there is an index of $O(N)$ space that answers a query in $O(N^{1-1/d} + N^{1-1/k} \cdot \text{OUT}^{1/k})$ time.*

The above result implies Theorem 5. To understand why, notice that, in LC-KW, the set of locations in $\mathbb{R}^d$ satisfying $s := O(1)$ linear constraints forms a polyhedron with s facets. Such a polyhedron can be partitioned into a constant number of d-simplicies (the constant depends on s and the dimensionality d). We can then find the points in D satisfying all the constraints by issuing an SP-KW query for every d-simplex. It is easy to verify from Theorem 12 that the space and query complexities are as claimed in Theorem 5.

Next, we will go through the four steps of our framework to establish Theorem 12.

D.1 Step 1: Identifying a Space-Partitioning Index

Let us first review the partition tree [13]. Denote by P a set of N points in $\mathbb{R}^d$. A partition tree on P is a tree $\mathcal{T}$ in which there are N leaves and each internal node has at most f child nodes where $f \geq 2$ is a constant. Every leaf stores a distinct point of P. Given a node u, we use P_u to represent the set of points stored in the subtree of u.

The partition tree is space-partitioning because

- every node u in $\mathcal{T}$ is associated with a d-simplex Δ_u as its *cell*, which covers all the points in P_u;
- the cell of the root is the entire $\mathbb{R}^d$;
- for every internal node u, the cells of its child nodes are interior disjoint and have Δ_u as their union.

Given a node u, we use $level(u)$ to denote its *level*. In general, the partition tree guarantees $|P_u| = O(N/f^{level(u)})$ for all nodes u. It thus follows that $\mathcal{T}$ has height $O(1) + \log_f N$.

D.2 Step 2: Conversion under General Position

Next, we transform the partition tree into an SP-KW index, assuming D in *general position*, which here means that no $d+1$ points in D fall on the same hyperplane in $\mathbb{R}^d$. Let $W := |\bigcup_{e \in D} e.\texttt{Doc}|$; w.l.o.g., each keyword is treated as an integer in $[1, W]$.

Verbose Set. As in Section 3.2, create P — the verbose version of D — by inserting $|e.\texttt{Doc}|$ copies of each point $e \in D$. Again, we will reserve symbol "e" and term "object" for the elements of D, as opposed to symbol "p" and term "point" for the elements of P. Create a partition tree $\mathcal{T}$ on P.

Active and Pivot Sets. For each node u in $\mathcal{T}$, we define its *active set* D_u^{act} and *pivot set* D_u^{pvt} by induction. If u is the root, $D_u^{act} := D$. Inductively, consider u as an internal node whose D_u^{act} has been properly defined, but not yet for D_u^{pvt}. Let $v_1, ..., v_f$ be the child nodes of u. Then:

- D_u^{pvt} is the set of objects in D_u^{act} falling on the boundary of $\Delta_{v_1}, \Delta_{v_2}, ...,$ or Δ_{v_f}. As D is in general position, the pivot set of u has only a constant number of objects.
- For each $i \in [1, f]$, $D_{v_i}^{act}$ is the set of objects in D_u^{act} that fall in the interior of Δ_{v_i}.

For every leaf z of $\mathcal{T}$, define $D_z^{pvt} := D_z^{act}$.

Large and Small Keywords at a Node. Given a keyword $\texttt{w} \in [1, W]$ and a node u in $\mathcal{T}$, define $D_u^{act}(\texttt{w}) := \{e \in D_u^{act} \mid \texttt{w} \in e.\texttt{Doc}\}$ and $N_u := \sum_{e \in D_u^{act}} |e.\texttt{Doc}| \leq |P_u| = O(N/f^{level(u)})$. We say that $\texttt{w}$ is "large" at u if $|D_u^{act}(\texttt{w})| \geq N_u^{1-1/k}$, or "small" at u, otherwise.

Structure. Each node u of $\mathcal{T}$ is associated with a secondary structure T_u, whose first purpose is to store the pivot set D_u^{pvt}. If u is a leaf, T_u has no other functionality. Otherwise, T_u should also support two operations in constant time:

- given a keyword $\texttt{w}$, return whether $\texttt{w}$ is large at u;
- given (i) k distinct keywords $\texttt{w}_1, ..., \texttt{w}_k$ that are large at u, and (ii) a child node v of u, return if $\bigcap_{i=1}^{k} D_v^{act}(\texttt{w}_i)$ is empty.

Finally, we materialize an active set $D_u^{act}(\texttt{w})$ — where u ranges over all the nodes in $\mathcal{T}$ and $\texttt{w}$ ranges over $[1, W]$ — if $\texttt{w}$ is small at u but is large at all the proper ancestors of u. The overall space is $O(N)$ words, as can be proved by adapting the space analysis of Appendix B in a straightforward manner.

D.3 Step 3: Bounding the Crossing Sensitivity

Algorithm. To answer a query with d-simplex q and keywords $\texttt{w}_1, ..., \texttt{w}_k$, we start by visiting the root of $\mathcal{T}$.

In general, to "visit" a node u, we read every object e in the node's pivot set, and report e if $e \in q$ and $e.\texttt{Doc}$ contains all of $\texttt{w}_1, ..., \texttt{w}_k$. If u is a leaf, the visit is complete. If u is internal, we continue by using T_u to decide whether $\texttt{w}_1, ..., \texttt{w}_k$ are all large at u. Suppose that this is true. Then, for each child v of u, we recursively visit v if $\bigcap_{i=1}^{k} D_v^{act}(\texttt{w}_i) \neq \emptyset$ and $q \cap \Delta_v \neq \emptyset$. Consider now the opposite where at least one of $\texttt{w}_1, ..., \texttt{w}_k$ — say, $\texttt{w}_1$ — is small at u. In this case, $D_u^{act}(\texttt{w}_1)$ must have been materialized. We read every object $e \in D_u^{act}(\texttt{w}_1)$ and report e if $e \in q$ and $e.\texttt{Doc}$ has all the k keywords.

Analysis. Let $\mathcal{T}_{qry}$ be the tree formed by the nodes visited. The algorithm spends constant time at every internal node of $\mathcal{T}_{qry}$ and $O(N_z{}^{1-1/k})$ time at every leaf node z of $\mathcal{T}_{qry}$. The cell Δ_u of every node u in $\mathcal{T}_{qry}$ must intersect with q. We classify u as *covered* if Δ_u is covered by q, or *crossing*, otherwise.

By adapting the proof of Lemma 9, the reader can verify that the total cost spent on the covered nodes is $O(N^{1-1/k}(1 + \text{OUT}^{1/k}))$. To analyze the cost on the crossing nodes, let $\mathcal{T}_{cross}$ be the tree obtained from $\mathcal{T}_{qry}$ by deleting all the covered nodes. Define the *crossing sensitivity* of q exactly as in (7).

Crossing Sensitivity of the Partition Tree. We will prove that q has crossing sensitivity $O(N^{1-1/d})$ when $k \leq d-1$, or $O(N^{1-1/k} \log N)$ when $k \geq d$. Combining these facts with the earlier discussion yields the query cost in Theorem 12.

By standard partition-tree analysis, $\mathcal{T}_{cross}$ has $O(N^{1-1/d})$ nodes[17]. Hence, the term "$\sum_{\text{internal } u \text{ of } \mathcal{T}_{cross}} 1$" in (7) is $O(N^{1-1/d})$. The subsequent discussion will prove:

$$\sum_{\text{crossing leaf } z} N_z^{1-1/k} = \begin{cases} O(N^{1-1/k} \log N) & \text{if } d \leq k \\ O(N^{1-1/d}) & \text{if } d > k \end{cases} \quad (15)$$

which validates our claim about the crossing sensitivity of q.

[17] Any d-simplex can intersect, but not fully covering, the cells of $O(N^{1-1/d})$ nodes in a partition tree on N points.

To proceed, we need a property of the partition tree established by Chan [13]. Fix an arbitrary level ℓ of $\mathcal{T}$ (recall that root is at level 0). Chan (in proving Theorem 3.2 of [13]) showed that, for any hyperplane in $\mathbb{R}^d$, the number of level-ℓ nodes whose cells intersect the hyperplane is

$$O(f^{\ell(1-\frac{1}{d})} + f^{\ell(1-\frac{1}{d-1}+\delta)} \log N) \tag{16}$$

where $\delta > 0$ is an arbitrarily small constant. As each facet of q is enclosed by a hyperplane, the number of level-ℓ nodes whose cells intersect the facet is also bounded by (16). Notice that if u is a crossing node, its cell Δ_u must intersect at least one facet of q. It thus follows that, at level ℓ of $\mathcal{T}$, the number of crossing nodes is bounded by (16).

Setting $h := O(1) + \log_f N$ to be the maximum level in $\mathcal{T}$, we can now derive

$$\begin{aligned}
&\sum_{\text{crossing leaf } z} N_z^{1-\frac{1}{k}} \\
= &\sum_{\ell=0}^{h} \sum_{\text{crossing leaf } z \text{ at level } \ell} N_z^{1-\frac{1}{k}} \\
= &\sum_{\ell=0}^{h} \sum_{\text{crossing leaf } z \text{ at level } \ell} O\left(\frac{N}{f^\ell}\right)^{1-\frac{1}{k}} \\
= &\; O\left(\sum_{\ell=0}^{h} \left(\frac{N}{f^\ell}\right)^{1-\frac{1}{k}} \left(f^{\ell(1-\frac{1}{d})} + f^{\ell(1-\frac{1}{d-1}+\delta)} \log N\right)\right) \\
&\text{(using (16))} \\
= &\; O(N^{1-\frac{1}{k}}) \cdot \left[\left(\sum_{\ell=0}^{h} f^{\ell(\frac{1}{k}-\frac{1}{d})}\right) + \left(\sum_{\ell=0}^{h} \frac{\log N}{f^{\ell(\frac{1}{d-1}-\frac{1}{k}-\delta)}}\right)\right]. \quad (17)
\end{aligned}$$

Let us first analyze $\sum_{\ell=0}^{h} f^{\ell(\frac{1}{k}-\frac{1}{d})}$. When $k > d$, the sum is dominated by the term at $\ell = 0$, which is $O(1)$. When $k = d$, the sum is $O(h) = O(\log N)$. When $k < d$, the sum is dominated by the term at $\ell = h$, which is $O(f^{h(\frac{1}{k}-\frac{1}{d})}) = O(N^{\frac{1}{k}-\frac{1}{d}})$.

Let us turn attention to $\sum_{\ell=0}^{h} (\log N)/f^{\ell(\frac{1}{d-1}-\frac{1}{k}-\delta)}$. When $k > d-1$, the sum is dominated by the term at $\ell = 0$, which is $O(\log N)$. When $k \leq d-1$, the term is dominated by the term at $\ell = h$, which is $O(\log N \cdot f^{h(\delta+\frac{1}{k}-\frac{1}{d-1})}) = O(\log N \cdot N^{\delta+\frac{1}{k}-\frac{1}{d-1}})$.

Combining the above, we can see that when $k \geq d$, (17) is bounded by $O(N^{1-1/k} \log N)$; when $k \leq d-1$, (17) is bounded by $O(N^{1-\frac{1}{k}}(N^{\frac{1}{k}-\frac{1}{d}} + \log N \cdot N^{\delta+\frac{1}{k}-\frac{1}{d-1}})) = O(N^{1-\frac{1}{k}} \cdot N^{\frac{1}{k}-\frac{1}{d}}) = O(N^{1-1/d})$. This establishes (15).

D.4 Step 4: Removing General Position

We will remove the general position assumption that D has no $d+1$ objects lying on the same hyperplane in $\mathbb{R}^d$. Before continuing, let us make a crucial observation. The index presented in Sections D.2 and D.3 — including both its algorithms and analysis — works as long as two conditions are met for each node u in the underlying partition tree $\mathcal{T}$:

(1) For every object $e \in D_u^{act}$, all the copies of e in the verbose set P are stored in the subtree of u.

(2) Its pivot set D_u^{pvt} has a constant size.

When D is not in general position, condition (2) may not hold. To understand this, consider u as an internal node with child nodes $v_1, ..., v_f$. Recall that D_u^{pvt} consists of the objects in D_u^{act} that fall on the boundaries of cells $\Delta_{v_1}, ..., \Delta_{v_f}$. The number of such objects may be arbitrarily large without the general position assumption.

This issue can be remedied using the standard *perturbation technique*. At a high level, the technique works as follows in our context. Suppose that we perturb (i.e., shift) each object in D by a small distance controlled by a parameter $\epsilon \geq 0$, such that $\epsilon = 0$ means no shifting. Denote by $D(\epsilon)$ the data input obtained by perturbing D under value ϵ (note: $D(0) = D$). The perturbation can be done in a way (e.g., [26]) to ensure that $D(\epsilon)$ be in general position as long as $0 < \epsilon < \epsilon_0$, where ϵ_0 is a sufficiently small value (which may depend on D). Let $P(\epsilon)$ be the verbose version of $D(\epsilon)$, and $\mathcal{T}(\epsilon)$ be the partition tree on $P(\epsilon)$; note that $\mathcal{T}(0)$ is just the partition tree on the verbose version P of D. It is known that, when $0 < \epsilon < \epsilon_0$, $\mathcal{T}(\epsilon)$ and $\mathcal{T}(0)$ are identical in the following sense:

- The trees $\mathcal{T}(0)$ and $\mathcal{T}(\epsilon)$ are isomorphic to each other.
- A point $p \in P$ is stored in the subtree of a node u in $\mathcal{T}(0)$ if and only if the perturbed version of p in $P(\epsilon)$ is stored in the subtree of the node in $\mathcal{T}(\epsilon)$ corresponding to u under isomorphism.
- The cell of each node in $\mathcal{T}(\epsilon)$ can be expressed as a continuous function of ϵ such that, when ϵ decreases to 0, the cell becomes the cell of the corresponding node in $\mathcal{T}$.

To remove the general position assumption, we perturb D to $D(\epsilon)$ with an infinitesimally small $\epsilon > 0$. Since $D(\epsilon)$ is in general position, $\mathcal{T}(\epsilon)$ satisfies conditions (1) and (2). Assign active and pivot sets for the nodes of $\mathcal{T}(\epsilon)$ as described in Section D.2. Now, decrease ϵ all the way to 0, which morphs $\mathcal{T}(\epsilon)$ into $\mathcal{T}(0)$. Each node in $\mathcal{T}(0)$ *retains* the same pivot and active sets as its corresponding node in $\mathcal{T}(\epsilon)$. We thus have obtained an index on D that fulfills conditions (1) and (2), thereby completing the proof of Theorem 12.

Remark. The reader may wonder how perturbation "actually" remedies the issue we were facing in the beginning. Consider again an internal node u. Let v be a child of u. There can still be many objects falling on a facet of cell Δ_v. However, if this happens, there must be another child v' of u whose cell $\Delta_{v'}$ is a degenerated simplex fully contained in *that* facet of Δ_v. Most objects on the facet have all their copies (in the verbose set P) stored in the subtree of v'. Such objects appear in the active set of v', rather than in the pivot set of u.

E PROOF OF LEMMA 8

Let us first discuss the scenario where $\text{OUT} \geq N^{(1-\frac{1}{k})/(1-\frac{1}{k}+\epsilon)}$. Re-arranging the terms gives $\text{OUT} \geq N^{1-1/k} \cdot \text{OUT}^{\frac{1}{k}-\epsilon}$. In this case, the complexity in (3) becomes $O(N^{1-1/k} + \text{OUT})$.

On the other hand, when $\text{OUT} < N^{(1-\frac{1}{k})/(1-\frac{1}{k}+\epsilon)}$, we can derive

$$\begin{aligned}
\text{OUT} &< N^{\frac{1-(1/k)}{1-(1/k)+\epsilon}} \\
\Rightarrow \text{OUT}^{(1/k)-\epsilon} &< N^{\frac{1-(1/k)}{1-(1/k)+\epsilon} \cdot ((1/k)-\epsilon)} \\
&= N^{(1-(1/k))\left(\frac{1}{1-(1/k)+\epsilon}-1\right)}
\end{aligned}$$

$$\begin{aligned} & & (\text{using } \tfrac{(1/k)-\epsilon}{1-(1/k)+\epsilon} = \tfrac{1}{1-(1/k)+\epsilon} - 1) \\ & & = N^{\frac{1-(1/k)}{1-(1/k)+\epsilon}-(1-(1/k))} \\ \Rightarrow N^{1-(1/k)} \cdot \text{OUT}^{(1/k)-\epsilon} & < N^{\frac{1-(1/k)}{1-(1/k)+\epsilon}} \\ & = N^{1-\frac{\epsilon}{1-(1/k)+\epsilon}}. \end{aligned}$$

Under the above condition, the complexity in (3) becomes $O(N^{1-1/k} + N^{1-\frac{\epsilon}{1-(1/k)+\epsilon}})$.

We now conclude that the query time is $O(N^{1-\delta} + \text{OUT})$ time where $\delta = \min\{\frac{1}{k}, \frac{\epsilon}{1-(1/k)+\epsilon}\}$ in all cases.

F PROOFS OF COROLLARIES 3, 4, 6, AND 7

Proof of Corollary 3. A d-rectangle $[a_1, b_1] \times [a_2, b_2] \times ... \times [a_d, b_d]$ intersects another d-rectangle $[x_1, y_1] \times [x_2, y_2] \times ... \times [x_d, y_d]$ if and only if the $2d$-dimensional point $(a_1, b_1, a_2, b_2, ..., a_d, b_d)$ falls in the $2d$-rectangle $(\infty, y_1] \times [x_1, \infty) \times (\infty, y_2] \times [x_2, \infty) \times ... \times (\infty, y_d] \times [x_d, \infty)$. Hence, the d-dimensional RR-KW problem can be solved by a $2d$-dimensional ORP-KW index. Corollary 3 thus follows from Theorems 1 and 2.

Proof of Corollary 4. Given a d-rectangle B in $\mathbb{R}^d$, keywords $\mathtt{w}_1, ..., \mathtt{w}_k$, and an integer $t \geq 1$, our index in Theorems 1 and 2 can be deployed to detect whether $B \cap D(\mathtt{w}_1, ..., \mathtt{w}_k)$ has size at least t in $O(N^{1-1/k} \cdot t^{1/k})$ time. For this purpose, simply run an ORP-KW query with parameters B and $\mathtt{w}_1, ..., \mathtt{w}_k$. If $\text{OUT} := |B \cap D(\mathtt{w}_1, ..., \mathtt{w}_k)|$ is less than t, the query must terminate in $O(N^{1-1/k} \cdot t^{1/k})$ time. If it does not finish after $O(N^{1-1/k} \cdot t^{1/k})$ time, we manually terminate it and declare $\text{OUT} \geq t$.

Given a point $q \in \mathbb{R}^d$ and a radius $r \geq 0$, let $B(q, r)$ — an L_∞-ball — be the set of locations in $\mathbb{R}^d$ with L_∞ distance at most r to q. Note that $B(q, r)$ is a d-rectangle. Consider an L_∞NN-KW query with parameters q (a point), t (an integer in $[1, |D|]$), and $\mathtt{w}_1, ..., \mathtt{w}_k$ (keywords). We want to find the smallest r such that $B(q, r) \cap D(\mathtt{w}_1, ..., \mathtt{w}_k)$ has size at least t. As shown below, this can be done by performing binary search in a set of $O(N)$ "candidate radius values". We will test $O(\log N)$ values of r, and each test (as mentioned) takes $O(N^{1-1/k} \cdot t^{1/k})$ time using an ORP-KW index. The total query time is therefore $O(\log N \cdot N^{1-1/k} \cdot t^{1/k})$

Formally, a *candidate radius* is defined as the coordinate difference between q and an object $e \in D$ on one of the d dimensions. In other words, each object gives d candidate radii such that the total number of candidate radii is $d|D| = O(N)$. Next, we will proceed by first making the general position assumption that all those candidate radii are distinct, and then removing the assumption in the end.

If e is the t-th closest object (under L_∞ distance) to q among all the objects in $D(\mathtt{w}_1, ..., \mathtt{w}_k)$, the L_∞ distance between e and q is one of the candidate radii. Our job is to identify the smallest candidate radius r such that $B(q, r) \cap D(\mathtt{w}_1, ..., \mathtt{w}_k)$ has at least t objects. To permit binary search, we need a method to select the i-th — for any $i \in [1, d|D|]$ — smallest value from all the $d|D|$ candidate values in $O(N^{1-1/k})$ time. This target complexity is very loose for such a rudimentary task, which is straightforward to accomplish in $O(\text{polylog}\, N)$ time by resorting to d binary search trees, each created on the coordinates of a different dimension.

Finally, the general position assumption can be removed by converting the coordinates to the rank space. We omit the standard details here.

Proof of Corollary 6. The lifting technique [8] (see also Section 11.6 of [24]) maps each point $p \in \mathbb{R}^d$ to a point $p' \in \mathbb{R}^{d+1}$ such that, for any sphere B in $\mathbb{R}^d$, p falls in B if and only if p' is covered by a $(d+1)$-simplex in $\mathbb{R}^{d+1}$ that is computed based on B (the simplex is a degenerated one, or more specifically, a halfspace in $\mathbb{R}^{d+1}$). Hence, the d-dimensional SRP-KW problem can be solved by a $(d+1)$-dimensional LC-KW index. Corollary 6 thus follows from Theorem 5.

Proof of Corollary 7. We will combine Corollary 6 with binary search in a fashion similar to how we used Theorems 1 and 2 to prove Corollary 4.

We consider only the case where $d > k - 1$ because the $d \leq k - 1$ case can be settled by modifying our argument in a straightforward manner. Given a sphere B in $\mathbb{R}^d$, keywords $\mathtt{w}_1, ..., \mathtt{w}_k$, and an integer $t \geq 1$, our index in Corollary 6 can be deployed to detect whether $B \cap D(\mathtt{w}_1, ..., \mathtt{w}_k)$ has size at least t in $O(N^{1-\frac{1}{d+1}} + N^{1-1/k} \cdot t^{1/k})$ time (following the ideas in the proof of Corollary 4).

Given a point $q \in \mathbb{N}^d$ and a radius $r \geq 0$, let $B(q, r)$ — an "L_2-ball" — be the set of locations in $\mathbb{R}^d$ with Euclidean distance at most r to q. Note that $B(q, r)$ is a sphere in $\mathbb{R}^d$. Consider an L_2NN-KW query with parameters q (a point), t (an integer in $[1, |D|]$), and $\mathtt{w}_1, ..., \mathtt{w}_k$ (keywords). We want to find the smallest r such that $B(q, r) \cap D(\mathtt{w}_1, ..., \mathtt{w}_k)$ has size at least t.

We will proceed by making the general position assumption that the objects in D have distinct Euclidean distances to q; the assumption will be removed in the end. Recall that $\mathbb{N}$ is the set of $O(\log N)$-bit integers. The distance between any two points in $\mathbb{N}^d$ has $N^{O(1)}$ possibilities, each we call a *candidate radius*. We perform binary search to identify the smallest candidate radius r such that $B(q, r) \cap D(\mathtt{w}_1, ..., \mathtt{w}_k)$ has at least t objects. As $O(\log N)$ distances need to be tested, the total query time is $O(\log N \cdot (N^{1-\frac{1}{d+1}} + N^{1-1/k} \cdot t^{1/k}))$.

Finally, the general position assumption can be removed with infinitesimal perturbation. We omit the standard details here.

G TIGHTNESS OF COROLLARIES 4 AND 7

We will discuss only Corollary 4 because a similar argument applies to Corollary 7.

Our objective is to argue that, for L_∞NN-KW, no structure of $O(N \,\text{polylog}\, N)$ space can answer a query in $O(N^{1-1/k} t^{(1/k)-\epsilon} + t)$ time, no matter how small the constant $\epsilon > 0$ is, subject to the strong set-intersection and strong k-set-disjointness conjectures. Assume, on the contrary, that such a structure exists. Next, we will use it to obtain a k-SI reporting index of $O(N \,\text{polylog}\, N)$ space whose query complexity is given in (3). As discussed in Section 1.2, such a k-SI index cannot exist, subject to the aforementioned conjectures.

Given an instance of k-SI with sets $S_1, ..., S_m$, we generate $D := \bigcup_{i=1}^m S_i$ and $e.\mathtt{Doc} := \{i \mid e \in S_i\}$ for each $e \in D$, and then map

each object $e \in D$ to an arbitrary point in $\mathbb{N}^d$. Create the provided L_∞NN-KW index on D.

Given a k-SI reporting query with set ids $w_1, ..., w_k$, we reduce it to L_∞NN-KW by executing the following steps, starting with $t := 1$.

- Issue an L_∞NN-KW query with an arbitrary point $q \in \mathbb{N}^d$, the current value of t, and keywords $w_1, ..., w_k$.
- If the L_∞NN-KW query reports less than t objects, it must have found the entire $D(w_1, ..., w_k)$. Otherwise, we double t and repeat the previous step.

The algorithm terminates with $t = \Theta(1 + \text{OUT})$, where $\text{OUT} := |D(w_1, ..., w_k)|$. The overall running time is asymptotically dominated by the worst-case cost of the last query, which is $O(N^{1-1/k} + N^{1-1/k}\text{OUT}^{(1/k)-\epsilon} + \text{OUT})$.