Top-k Representative Queries with Binary Constraints

Arijit Khan Vishwakarma Singh

ETH Zurich
arijit.khan@inf.ethz.ch

Apple USA
vsingh014@gmail.com

ABSTRACT

Given a collection of binary constraints that categorize whether a data object is relevant or not, we consider the problem of online retrieval of the top-k objects that best represent all other relevant objects in the underlying dataset. Such top-k representative queries naturally arise in a wide range of complex data analytic applications including advertisement, search, and recommendation. In this paper, we aim at identifying the top-k representative objects that are high-scoring, satisfy diverse subsets of given binary constraints, as well as representative of various other relevant objects in the dataset. We formulate our problem with the well-established notion of the top-k representative skylines, and we show that the problem is $\mathbf{NP}$-hard. Hence, we design efficient techniques to solve our problem with theoretical performance guarantees. As a side-product of our algorithm, we also improve the asymptotic time-complexity of skyline computation to log-linear time in the number of data points when all dimensions except one are binary in nature. Our empirical results attest that the proposed method efficiently finds high-quality top-k representative objects, while our technique is one order of magnitude faster than state-of-the-art methods for finding the top-k skylines with binary constraints.

1. INTRODUCTION

Over a database, the users often search for objects that satisfy user-defined multiple *binary* constraints — each of these constraints can evaluate to either true or false for every specific object. In particular, given a set D of objects, a scoring function $f(d)$ for all objects $d \in D$, a collection of binary constraints $p_1(d), p_2(d), \ldots, p_r(d)$, and a small positive integer k, we consider the problem of online retrieval of the top-k most representative objects that are high-scoring, yet satisfy diverse subsets of given binary constraints, as well as representative of various other relevant objects in the dataset. Such top-k representative queries over objects with one non-binary and many binary attributes naturally arise in many complex data analytic applications [18, 25], including information retrieval, search, advertisement, and recommendation.

EXAMPLE 1. *A real-estate agent maintains a database of available houses along with several attributes, such as price, number of rooms, number of windows, floor type, outdoor space, neighborhood quality, and distance from various points of interests. Given such a database of available houses, a customer might not always be interested in the actual value of an attribute, rather the preferences of a customer often arrive in the form of several binary constraints [25], e.g., distance from a school S less than 5 miles. Let us assume that some customer requested for the following important preferences: (1) distance from a school S less than 5 miles, (2) distance from a popular nightclub N less than 5 miles, (3) distance from the airport A less than 10 miles, (4) quiet neighborhood, (5) garden-space more than 50 sq. meter, (6) mountain and lake view, and (7) marble floor. Usually, there is a limit k on the number of houses that the agent can show in order to convince the customer to buy a house. Then, what are the top-k houses that the agent must show to this customer?*

In the above example, we say that a house is *relevant* to the customer if it satisfies at least one of the aforementioned constraints. Nevertheless, finding the top-k houses with the lowest prices (the price can be modeled as the scoring function $f()$ in our problem setting) and satisfying all, or as many as possible, given constraints is not a good idea from the agent's perspective — it could result in only a few expensive houses; while, in reality, the customer might not be equally interested in satisfying all the above-mentioned constraints. Rather, it will be more promising to the agent if she reports the top-k houses that are relatively low-cost, satisfy diverse subsets of the aforementioned important constraints, and yet representative of a large number of other relevant houses in the database.

In this paper, we formulate the problem of selecting the top-k representative data-points with the well-established notion of dominance or "pareto-optimality" [2]. We model the scoring function $f()$ and each binary constraint p_i as a separate dimension. An object r_0 dominates another object r if the value of r is no larger than that of r_0 in each dimension, and the value of r is smaller than that of r_0 in at least one dimension. Finally, we retrieve the top-k objects that collectively dominate the maximum number of other relevant objects in the dataset.

Here, we must emphasize that we adopt a similar criteria as finding the top-k representative skylines which was earlier proposed in [16]. Unfortunately, the problem of finding the top-k most representative skylines in an online manner is a non-trivial problem. The dynamic nature of our query makes any kind of pre-computation very difficult. Indeed, the asymptotic complexity of the best-known algorithm for skyline set computation is $\mathcal{O}(n(\log n)^{r-1})$ due to [2], where n is the number of data points and $(r+1)$ is the dimensionality of each data point in our problem setting. Clearly, one can apply existing approaches, e.g., [16] to solve our problem by

SSDBM '15, June 29 - July 01, 2015, La Jolla, CA, USA

http://dx.doi.org/10.1145/2791347.2791367.

considering each binary constraint as a separate dimension. However, such approaches are not optimized for the top-k representative queries with binary constraints that we study in this work. Our technical contribution lies in the fact that we design efficient algorithms to leverage the binary constraints present in our queries. While we provide the same approximation guarantee as in [16], *the presence of binary constraints in our problem setting allows for reasoning on the lattice (discussed in Section 5), and this mechanism is not straight-forward to be adapted to the case of general dimensions as in [16]*. Based on experiments over four diverse categories of real-world and synthetic datasets, we find that *our proposed techniques for finding the top-k representatives are* 10 *times faster as well as up to* 2 *times less memory consuming than state-of-the-art method* [16], which was designed for retrieving the top-k representative skyline points in the metric space.

Contributions: We make four important contributions.

(1) We formulate and investigate the novel problem of answering the top-k representative queries with binary constraints in an online manner.

(2) We improve the asymptotic complexity of skyline computation to log-linear time in the number of data points, when all dimensions except one are binary in nature [1].

(3) We propose efficient techniques to find the dominance number of an object as well as a set of objects; and thus, we can quickly identify the top-k representative objects in an online manner. *Our algorithms are designed to utilize the fact that the constraints in our queries are binary in nature.* Therefore, our proposed schemes are 10 times faster as well as consume up to 2 times less memory than state-of-the-art method [16], which was designed for retrieving the top-k skyline points in the metric space.

(4) As tested on four diverse categories of real-world and synthetic datasets, our techniques are able to select high-quality top-k answers in an efficient manner. We also show that the top-k objects retrieved by our method indeed represent a large number of other relevant objects in the dataset, even for very small values of k, e.g., $k = 5$.

2. RELATED WORK

We categorize related work as follows.

Skyline Queries. The skyline operator was originated from the maximal vector problem in computational geometry [3,14]. Börzsönyi *et al.* [2] first introduced the skyline operator for large databases, and also proposed an SQL syntax. Skyline queries for dynamic preferences over nominal attributes were considered in [28]. Constrained skyline queries were proposed in [21], and thereafter, [6, 29] considered constrained skyline queries under distributed and stream settings, respectively. Skyline queries over objects with one non-binary and many binary attributes were previously considered in [18], and the authors demonstrated the advantage of lattice-based skyline computation techniques which is similar to ours. However, their query semantics are different, as they do not consider the top-k answers to their queries. Endres et. al. further proposed "preference queries", which are constrained skyline queries in the presence of hard and soft constraints [8, 9].

For high dimensional data, the skyline set usually becomes quite large [3, 11]; and hence, less informative. Therefore, various techniques have been proposed to rank the skyline points, such as the representative skylines [16,26] and regret-minimizing skylines [19]. Particularly, we adopted the skyline ranking scheme proposed in [16] — albeit, their techniques were developed for the metric space; and hence, not optimized for answering top-k representative queries with binary constraints. The presence of binary constraints in our problem setting allows for reasoning on the lattice, and this mechanism is not straight-forward to be adapted to the case of general dimensions as in [16].

Top-k Representative Queries. A large number of models exist that support diversity in the top-k answer set, e.g., [1,4,7,17,23,27], among many others. A few works [13, 15, 24] maximize diversity as well as representativeness of the top-k results. Nevertheless, they consider representativeness in the context of object-similarity; whereas we use the notion of dominance or "pareto-optimality" as representativeness, which is more suitable for our queries. Thus, our top-k representative queries are more similar to the top-k representative skylines as proposed earlier in [16].

Constrained Optimization Queries. [30] designed efficient indexing methods to find an exact solution of the k-constrained optimization query. [22] studied the problem of multi-objective optimization, in which solutions to a combinatorial optimization problem were evaluated with respect to several cost criteria, and the authors investigated the notion of ϵ-Pareto solution. A syntactically close work to ours is the OPAC query proposed in [12]. The authors considered ϵ-Pareto answers to optimization queries under parametric aggregation constraints over multiple database tuples. However, their optimization function and constraints are aggregated over multiple tuples. On the other hand, in our query semantics, the scoring function as well as the query constraints are applied over individual database tuples.

3. PROBLEM DEFINITION

Let us denote by D the set of all data points. We next start with a few definitions.

DEFINITION 1. [Satiated Constraints by a Data Point] *Given a set of binary query constraints $P = \{p_1, p_2, \ldots, p_r\}$, the satiated constraints $P(d)$ by some data point $d \in D$ are simply the subset of constraints satisfied by d. Formally,*

$$P(d) = \{p_i \in P : p_i(d) \text{ is true}\} \quad (1)$$

We say that a data point $d \in D$ is *relevant*, if it satisfies at least one of the given query constraints, i.e., $P(d) \neq \phi$. We shall use the terms "data points" and "relevant data points" interchangeably, i.e., given the query constraints, we shall consider only the relevant data points in our algorithm.

DEFINITION 2. [Dominance] *Given two data points $d, d' \in D$, a scoring function f, and a set of constraints P, we say that d dominates d' if one of the following holds true: (1) $f(d) > f(d')$ and $P(d) \supseteq P(d')$, or (2) $f(d) = f(d')$ and $P(d) \supset P(d')$.*

The dominance relation satisfies the following properties.

PROPERTY 1. *The dominance relation is transitive. It is not reflexive, neither symmetric, nor anti-symmetric.*

Now, we formally define skyline points in a database with respect to the scoring function f and the set of binary constraints P.

DEFINITION 3. [Skyline Point] *A data point $d \in D$ is called a skyline point if it is not dominated by any other points in D.*

[1]It was shown in [2] why an $\mathcal{O}(n \log n)$ algorithm is not sufficient enough to find the skyline set when the dimensionality of each data point is more than 2. In the worst case, the asymptotic complexity of the best-known algorithm for skyline set computation is $\mathcal{O}(n(\log n)^{r-1})$.

Table 1: *Run-through Example. The query contains three constraints:* p_1, p_2, p_3.

Database Points	$f(d)$	Satiated Constraints
d_1	15	p_1
d_2	12	p_2
d_3	11	p_3
d_4	9	p_1, p_2
d_5	6	p_1
d_6	5	p_3
d_7	4	p_1, p_2
d_8	2	p_2

From the above definitions, we note that each skyline point corresponds to the highest-scoring (decided by $f()$) point for some subset of query constraints. For simplicity of description, we hereinafter assume that no two data points have the same $f()$ score. We shall discuss later how our techniques can be adapted to consider scenarios when multiple data points have the same f() score. Next, in order to identify the top-k most representative data points, we introduce a ranking criteria using the notion of *dominance number* [16, 21], which is formally defined below.

DEFINITION 4. [Dominance Number of a Point] *The dominance number $N(d)$ of a data point $d \in D$ is defined as the number of data points in D that are dominated by d.*

$$N(d) = |\{d' : d' \in D,\ d \text{ dominates } d'\}| \quad (2)$$

DEFINITION 5. [Dominance Number of a Set] *Given a set of points $D_1 \subseteq D$, the dominance number of the set D_1 is defined as the number of data points in D that are dominated by at least one of the points in D_1.*

$$N(D_1) = |\{d' : d' \in D,\ \exists d(d \in D_1,\ d \text{ dominates } d')\}| \quad (3)$$

EXAMPLE 2. *In Table 1, we show 8 data points, along with their $f()$ scores and satiated constraint sets. Based on our definition, the skyline points are d_1, d_2, d_3, and d_4, as they are not dominated by any other points in the database. The data points dominated by the skyline points d_1, d_2, d_3, and d_4 are: $\{d_5\}$, $\{d_8\}$, $\{d_6\}$, and $\{d_5, d_7, d_8\}$, respectively. We further observe that $\{d_1, d_4\}$ together dominate 3 data points: $\{d_5, d_7, d_8\}$.*

Problem Formulation. We are now ready to formally define our problem, KREP, abbreviated for k-representative.

PROBLEM 1. *[KREP] Given the dataset D, a query $\langle f, P\rangle$, and a positive integer k, find the set of k data points, such that the total number of points dominated by at least one of them is maximized.*

Note that in our problem, we do not explicitly seek for an answer set consisting only of skyline points. However, if the total number of skyline points in D with respect to some query $\langle f, P\rangle$ is at least k, it is easy to verify (Theorem 1) that the top-k representative points defined by the KREP problem will indeed be the top-k set of skyline points (in terms of the dominance number of the set). Therefore, we shall design an algorithm that identifies the top-k skyline set.

THEOREM 1. *If the total number of skyline points in D with respect to some query $\langle f, P\rangle$ is at least k, then the top-k results will be* all *skyline points in one optimal solution of our problem.*

PROOF. Let, if possible, there be a non-skyline point d' in the top-k result set as defined in the problem KREP. There can be two distinct cases: (1) at least one of the skyline points, say d, which dominates d', is in the top-k set. In that case, if we remove d' from the top-k set, that will not reduce the dominance number of the set with the remaining $(k-1)$ points. Now, we can add a new skyline point in the set, since the total number of skyline points in the dataset is at least k. Otherwise, (2) all the skyline points, that dominate d', are not in the top-k set. Then, it is advantageous to replace d' by any skyline point d that dominates d', since d has higher dominance number than d'. This completes the proof. □

Our problem is, however, nontrivial. The following theorem shows that the KREP problem is intractable.

THEOREM 2. *The* KREP *problem is* **NP***-hard.*

PROOF. The proof directly follows by reduction from the **NP**-complete set cover problem [5], defined by a collection of subsets $\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_m$ of a ground set $\mathbb{U} = \{u_1, u_2, \ldots, u_t\}$. We would like to know whether there exist k of the subsets whose union is equal to $\mathbb{U}$. Now, we show that this can be viewed as a special instance of the KREP problem. Each set $\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_m$ could be considered as a skyline point and each element in the ground set can be considered as a non-skyline point. If $\mathbb{S}_i$ covers u_j in the set cover problem, we assume that the skyline point $\mathbb{S}_i$ dominates the non-skyline point u_j. Hence, the set cover problem is equivalent to deciding iff there is a set of k skyline points, with (set) dominance number greater than or equal to t. Hence the theorem. □

4. OVERVIEW OF SOLUTION

In this section, we provide a brief overview of our solution technique. Since the KREP problem is **NP**-hard (Theorem 2), we resort to an efficient approximation algorithm with theoretical performance guarantee. Fortunately, Theorem 3 ensures that the objective function of KREP is submodular [16], and it monotonically increases with k.

THEOREM 3. *The objective function of* KREP *(i.e., the number of points dominated by the top-k representative set) is submodular, and it monotonically increases with k.*

PROOF. A function $g()$ is submodular if it satisfies the following property: $g(\mathbb{A} \cup \{x\}) - g(\mathbb{A}) \geq g(\mathbb{B} \cup \{x\}) - g(\mathbb{B})$, for all elements x and all pairs of sets $\mathbb{A} \subseteq \mathbb{B}$, i.e., the marginal gain from adding an element to a set $\mathbb{A}$ is at least as high as the marginal gain from adding the same element to a superset of $\mathbb{A}$. In our problem setting, let $\mathbb{A}$ denote a skyline set and $g(\mathbb{A})$ be the total number of points dominated by at least one skyline point in $\mathbb{A}$. Clearly, the marginal gain from adding a skyline point to set $\mathbb{A}$ will be at least as high as the marginal gain from adding the same skyline point to a superset of $\mathbb{A}$. Therefore, the objective function of KREP is submodular. Besides, the total number of points dominated by at least one skyline point in $\mathbb{A}$ monotonically increases with the size of $\mathbb{A}$. □

Due to submodularity, we apply an iterative hill-climbing method (Algorithm 1) to derive an approximate solution within a factor of $(1 - 1/e) \approx 0.63$ of the optimum solution [20]. More specifically, given already selected top $(j-1)$ skyline points, where $1 \leq j < k$, we select the j-th skyline point such that the number of points dominated by the j-th skyline, but not by any of the previous $(j-1)$ skylines, is maximized.

While the above-mentioned hill-climbing method forms the core of our solution technique, one still needs to efficiently compute the

Algorithm 1 Finding Top-k Representative Objects

Require: Database points D, query $\langle f, P \rangle$, integer k
Ensure: Top-k representative set S of size k
1: find the skyline set M
2: $S = \phi$
3: **for** $i = 1$ **to** k **do**
4: $d^* = \arg\max_{d \in D \setminus S} N(S \bigcup \{d\})$ /* N dominance number */
5: $S = S \bigcup \{d^*\}$
6: **end for**
7: output S

following two dominance numbers — (1) the dominance number of any skyline point, and (2) the dominance number of any skyline set. Our main technical contribution lies in designing efficient algorithms to solve the two aforementioned problems by leveraging the binary constraints present in our queries. We describe our algorithm to find the dominance numbers of individual skyline points in Section 5, while our method to compute the dominance number of a skyline set is given in Section 6.

5. FINDING DOMINANCE NUMBER OF A SKYLINE POINT

In this section, we shall design efficient techniques to find all skyline points and their individual dominance numbers (line 1 of Algorithm 1). The first step of our method is to build main memory-based effective data structures by performing a single scan over the database. Therefore, we first describe our data structures which will be useful in our algorithms.

5.1 Main Memory Data Structures

Query Lattice: The query lattice, denoted by $\mathcal{Q} = (V, E, L)$, is a directed acyclic graph. It is formed based on the given query constraints set P. If P contains a total of r constraints, the lattice $\mathcal{Q}$ consists of $(2^r - 1)$ nodes. Each lattice node $v \in V$ corresponds to a subset of the query constraints set, except the null subset. We exclude the null subset as we consider only the relevant data points in our algorithm. The label set of each lattice node v is denoted as $L(v)$, which is a subset of P.

There is an edge from lattice node v to v', if $L(v') \subseteq L(v)$, and $|L(v)| - |L(v')| = 1$. v' is called a *successor* of v, denoted as $v' \in succ(v)$, if there is a directed path from v to v' in $\mathcal{Q}$. Alternatively, v is called a *predecessor* of v'. The node v itself is not included in successor or predecessor of v.

EXAMPLE 3. *In Figure 1, we present the query lattice corresponding to our run-through example in Table 1. With each lattice node, we also show two additional values — equivalence count (E) and subset count (S), which will be defined shortly.*

Mapping Relation: We define a mapping relation $\Psi : D \rightarrow V$, such that $\Psi(d) = v$ if $P(d) = L(v)$. One may note that Ψ is a many-to-one mapping.

Count Values: With each lattice node $v \in V$, we associate two count values: equivalence count and subset count. The *equivalence count* $E(v)$ is the number of data points in the input dataset D whose satiated constraints set is same as $L(v)$. The *subset count* $S(v)$, on the other hand, is the number of data points in D whose satiated constraints set is a subset of $L(v)$, except the null subset.

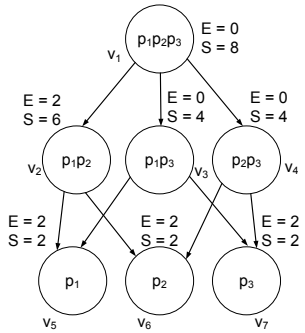


Figure 1: Query Lattice and Associated Counts for Run-through Example in Table 1

Formally,

$$E(v) = |\{d : \ d \in D, \ P(d) = L(v)\}| \tag{4}$$
$$S(v) = |\{d : \ d \in D, \ P(d) \subseteq L(v), \ P(d) \neq \phi\}|$$

The subset count $S(v)$ is essentially the number of data points in D that are dominated by (or equivalent to) some point $d \in D$, with $\Psi(d) = v$, based on the satiated constraints set. Hence, to determine the dominance number $N(d)$ of a skyline point d with $\Psi(d) = v$, we subtract from $S(v)$ the number of points that are not dominated by d in terms of $f()$ score. For simplicity of description, we assume that no two points have the same $f()$ scores. We shall discuss at the end of Section 5.2 how our techniques can be adapted to consider scenarios when multiple data points have the same $f()$ score. Finally, we compute the dominance number $N(d)$ by using our algorithm, which will also be described in Section 5.2.

Skyline Bit: With each lattice node v, we attach a variable *skyline bit* $B(v)$ that can take values either 0 or 1. The skyline bits satisfy the following during execution of our algorithm. A data point $d \in D$, such that $\Psi(d) = v$, and $B(v) = 1$, cannot be a skyline point. All skyline bits are initialized to 0 in the beginning of our algorithm.

Priority Queue: Given the scoring function f, we maintain a priority queue $\mathcal{L}$, which stores all data point ids in descending order of their $f()$ scores.

Space Complexity. The storage complexity due to our main memory data structures is $\Theta(n + 2^r)$ — the first part is due to the mapping relation and the priority queue; whereas the second component is due to the query lattice, and associated count values and skyline bit with each lattice node. Here, n is the cardinality of the dataset and r is the number of binary constraints in the query.

5.2 Algorithm Description

Initialization: Our first step consists of a *one-time sequential scanning* of the dataset D. For each data point $d \in D$, we evaluate $f(d)$, and verify how many of the user-defined constraints $\{p_1, p_2, \ldots, p_r\}$ are satisfied by d. While scanning the dataset, we perform two additional tasks. **(1)** We insert the data points $d \in D$ into a priority queue $\mathcal{L}$ that stores its elements in descending order of their $f(d)$ scores. **(2)** We build the query lattice $\mathcal{Q}$. At the end of the sequential scanning, we also compute the count values $\langle E, S \rangle$ for each lattice node. All the skyline bits $B(v)$ are initialized to 0.

Skyline Finding: Next, we access the data points from the top of the priority queue $\mathcal{L}$. Thus, we process all points $d \in D$ in descending order of their $f(d)$ scores. Based on our invariant condition on the skyline bit B, a point d is a skyline point only if $B(v) = 0$,

where $v = \Psi(d)$. If we find a point d as a skyline point, we compute its dominance number $N(d)$, which will be discussed shortly.

If d is a skyline point, we also set the following skyline bits: $B(v) = 1$; and $B(v') = 1$ for all $v' \in succ(v)$. This setting of skyline bits ensures that, if we process some data point d' at a later stage, where $\Psi(d') = v'$ and $B(v') = 1$, then d' cannot be a skyline point.

Compute Dominance Number: We recall that the subset count $S(v)$ is the number of points in D that are dominated by (or equivalent to) some data point $d \in D$, $\Psi(d) = v$, based on the satiated constraints set. Hence, to determine the dominance number $N(d)$ of a skyline point d, with $\Psi(d) = v$, we subtract from $S(v)$ the number of points that are not dominated by d in terms of its score $f()$. We keep track of dominance on $f()$ scores using a *traversal variable*, $T(v)$, associated with each lattice node v.

DEFINITION 6. [Traversal Variable] *Assume we are currently processing some data point d with score $f(d)$, and $\Psi(d) = v$. Then, the traversal variable $T(v)$ for lattice node v stores the number of points $d' \in D$, such that $\Psi(d') = v$ and $f(d') > f(d)$.*

The traversal variable of a lattice node v is incremented by 1 after we finish processing of the data point d such that $\Psi(d) = v$. Now, to compute the dominance number of a skyline point d with $\Psi(d) = v$, we traverse each successor node v' of v in the query lattice. Then, we aggregate $T(v)$ with $T(v')$ for all such v'. Finally, the dominance number of the skyline point d, where $\Psi(d) = v$, is given by:

$$N(d) = S(v) - [1 + T(v) + \sum_{v' \in succ(v)} T(v')] \quad (5)$$

One may verify that the expression $[1+T(v)+\sum_{v' \in succ(v)} T(v')]$ in the above equation denotes the number of data points that are not dominated by d in terms of its score $f()$, but dominated by (or equivalent to) d in terms of satiated constraints. Note that we include 1 in the expression $[1 + T(v) + \sum_{v' \in succ(v)} T(v')]$ to reflect the fact that a data point is not dominated by itself. Therefore, subtracting this expression from $S(v)$, which denotes the number of data points that are dominated by (or equivalent to) d only in terms of satiated constraints, provides the actual dominance number $N(d)$ of that data point d. A complete description to find all the skyline points, along with their dominance numbers, is given in Algorithm 2.

EXAMPLE 4. *We provide an example of dominance number computation using our run-through example in Table 1. We access the data points in descending order of their $f()$ scores. Assume, we have already processed data points d_1, d_2, d_3. Note that $\Psi(d_1) = v_5$, $\Psi(d_2) = v_6$, $\Psi(d_3) = v_7$. Therefore, the traversal variable for each lattice node will be as follows: $T(v_1) = T(v_2) = T(v_3) = T(v_4) = 0$, and $T(v_5) = T(v_6) = T(v_7) = 1$. The skyline bits, on the other hand, will be as follows: $B(v_1) = B(v_2) = B(v_3) = B(v_4) = 0$, and $B(v_5) = B(v_6) = B(v_7) = 1$. Next, we shall process the data point d_4. Since $\Psi(d_4) = v_2$ and the corresponding skyline bit $B(v_2)$=0, d_4 is a skyline point. To compute its dominance number, we consider all the nodes of the sub-lattice rooted at v_2 and find their traversal variables. More specifically, the dominance number of d_4 is given as: $S(v_2) - [1 + T(v_2) + T(v_5) + T(v_6)] = 3$. Finally, we increment the traversal variable of v_2 by 1, that is, $T(v_2)$ becomes 1. We also set the skyline bits of v_2 and that of all its descendents as 1.*

Early Termination: While Algorithm 2 terminates after processing all the data points (see line 4, Algorithm 2), we further propose an early termination criteria for our method. Each time we find a skyline point, we also verify whether one of the following conditions is true for every lattice node v: (i) its skyline bit $B(v) = 1$ or, (ii) its equivalence count $E(v) = 0$. If one of the aforementioned conditions is true for all lattice nodes, then the remaining points in the priority queue $\mathcal{L}$, which are not processed yet, cannot be skyline points. Hence, we terminate our algorithm.

Algorithm 2 Finding All Skyline Points and Their Dominance Numbers

Require: Database points D, query $\langle f, P \rangle$, integer k
Ensure: Skyline set M
1: compute query lattice $\mathcal{Q}$, equivalence counts E, and subset counts S
2: set traversal variable $T(v) \leftarrow 0$, skyline bit $B(v) \leftarrow 0$ for each lattice node
3: $\mathcal{L} \leftarrow$ data items in descending order of $f()$ score
4: **while** $\mathcal{L}$ not empty **do**
5: $d \leftarrow$ remove top-item from $\mathcal{L}$
6: $v \leftarrow \Psi(d)$
7: **if** $B(v) \neq 1$ **then**
8: insert d in skyline set M /* d is a skyline point */
9: dominance no. $N(d) = S(v) - [1 + T(v) + \sum_{v' \in succ(v)} T(v')]$
10: $B(v) = 1$ /* set skyline bit of v */
11: **for all** $v' \in succ(v)$ **do**
12: $B(v') = 1$ /* set skyline bits of all successors of v */
13: **end for**
14: **end if**
15: $T(v) \leftarrow T(v) + 1$ /* update traversal variable T(v) */
16: **end while**
17: output M

Multiple Data Points with Same $f()$ Scores: Our skyline set finding algorithm (Algorithm 2) can be adapted if there are multiple data points with the same $f()$ score. In such cases, data points with the same $f()$ scores are processed in batches (inside lines 5 to 15 in Algorithm 2). In particular, let us assume that $d_1, d_2, \ldots, d_k$ are the data points retrieved from the top of $\mathcal{L}$ and having the same $f()$ score. We further sort $d_1, d_2, \ldots, d_k$ in descending order of their satiated constraint set sizes $|P(d)|$, and process them in that order. If multiple data points have the same $f()$ score, they satisfy exactly the same set of constraints, and if the corresponding skyline bit in the query lattice is also set as 0, then all of them are reported as skyline points. Otherwise, a data point — whose satiated constraint set is a superset of the satiated constraint set of another data point — would be processed earlier, and the later would no longer be considered as a skyline point based on our algorithm. Equation 5, which computes the dominance number of a skyline point (line 9, Algorithm 2), is modified as follows:

$$N(d) = S(v) - [k' + T(v) + \sum_{v' \in succ(v)} T(v')] \quad (6)$$

Here, $k' \leq k$ is the number of data points in the current batch that map to the same lattice node v. Finally, we update the corresponding traversal variables (line 15 of Algorithm 2) at the end of processing the current batch of data points $d_{i_1}, d_{i_2}, \ldots, d_{i_k}$. Specially, we increment the traversal variable $T(v)$ by k', where k', as specified earlier, is the number of data points in the current batch that also map to the lattice node v.

5.3 Time Complexity

Let us denote the number of data points in D as n. Also, we assume that the number of query constraints is r, and the number of skyline points is m.

Sorting Data Points: The sorting of all data points in the priority queue based on their $f()$ scores requires $\mathcal{O}(n \log n)$ time.

Query Lattice Construction: The complexity of formulating the query lattice is $\mathcal{O}(2^r)$. One also needs to compute the equivalence count $E(v)$ and the subset count $S(v)$ for each lattice node v. In order to compute all the equivalence counts, we identify for each data point $d \in D$, its corresponding lattice node: $v = \Psi(d)$. This requires $\mathcal{O}(nr)$ time. Now, for computing the subset count of each lattice node v, we aggregate the equivalence count of that node and that of all its descendants. Therefore, the computation of all subset counts requires another $\mathcal{O}(2^{2r})$ time. Hence, the overall time complexity to compute the query lattice and associated data structures is $\mathcal{O}(nr + 4^r)$.

Finding Skylines and Dominance Numbers: We access data points from the top of priority queue $\mathcal{L}$ in descending order of their $f()$ scores. For each data point, we verify if it is a skyline point. This can be done by first looking at the mapping of that point to its corresponding lattice node, and then checking the skyline bit for that lattice node. This requires total $\mathcal{O}(nr)$ time for all n data points.

If a point d, with $\Psi(d) = v$, is evaluated to be a skyline, we need to traverse all successors of v in the query lattice, in order to determine the dominance number of d. Finally, each time we find a skyline point, we also verify all lattice nodes for the early termination criteria. All these processes require $\mathcal{O}(m2^r)$ time for total m skyline points. We note that $m \leq 2^r$, and usually, $m << n$.

Therefore, the overall time complexity to identify all the skyline points and their dominance numbers is $\mathcal{O}(n \log n + nr + 4^r)$. Thus, *we improve the asymptotic complexity of skyline computation to log-linear time in the number of data points, when all dimensions except one are binary in nature*. In reality, our method is even more efficient due to the two following reasons: (1) we effectively prune all the non-skyline points using skyline bits, and (2) we achieve early termination by verifying the skyline bits and equivalence count values.

6. FINDING DOMINANCE NUMBER OF A SKYLINE SET

In order to solve the KREP problem (Problem 1), we note that it is not sufficient to find the dominance numbers of individual skyline points. We also require to compute the dominance number of a skyline set. More specifically, since we use the iterative hill-climbing technique (Algorithm 1) to solve the KREP problem, it is necessary to find the *residual dominance* number of a skyline point with respect to a skyline set, which is defined below.

DEFINITION 7. [Residual Dominance] *Given a skyline set $\mathbb{S}$, the residual dominance $R(s, \mathbb{S})$ of a skyline point $s \notin \mathbb{S}$ is defined as the number of data points dominated by s, but not by any data point in $\mathbb{S}$.*

Clearly, the problem that we need to solve at this point is: how do we compute the residual dominance values $R(s, \mathbb{S})$ at each iteration of our iterative hill-climbing algorithm? If the intersection between the satiated constraint set of s and that of all previously selected skyline points in $\mathbb{S}$ is empty, then the residual dominance number of s is same as its individual dominance number. However, complexity arises when the intersection is non-empty. In such cases, we need to consider all the nodes of the sub-lattice rooted at v, where $v = \Psi(s)$. Now, for each lattice node $v' \in succ(v)$, we not only count how many data points d, such that $\psi(d) = v'$, are dominated by s based on $f()$ scores; additionally, we also need to count how many of such data points are not dominated by any of the previously selected skyline points from $\mathbb{S}$. In order to evaluate both these criteria effectively, we perform some modifications in our earlier data structure, which are addressed below.

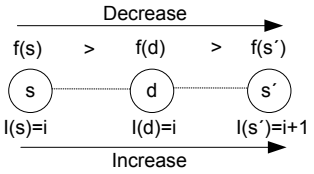


Figure 2: Interpretation of Traversal List: s and s' are skylines with time stamps i and $i+1$, respectively. d is a data point also with time stamp i. Then, $f(s) > f(d) > f(s')$.

6.1 Modification in Data Structure

We construct the query lattice $\mathcal{Q}$ as before. However, there are some updates in our data structure.

SkyCount Variable: The skycount variable J is initialized as 0. It is incremented by 1 when we find a skyline point while processing data points from the priority queue $\mathcal{L}$. We recall that $\mathcal{L}$ stores data points in descending order of their $f()$ scores.

Time Stamp: We dynamically associate a time stamp $I(d)$ with each data point $d \in D$. While processing a data point d from the priority queue $\mathcal{L}$, d's time stamp $I(d)$ is assigned to the current value of the skycount variable.

Table 2: *Time Stamps for Data Points*

Database Points	Time Stamp $I(d)$
d_1	1
d_2	2
d_3	3
d_4	4
d_5	4
d_6	4
d_7	4
d_8	4

Traversal List: We dynamically allocate a traversal list $\mathfrak{T}(v)$ with each lattice node v. The i-th entry of the traversal list $\mathfrak{T}_i(v)$ counts the number of data points $d \in D$, which are mapped to v, and which has its time stamp $I(d) = i$. Formally,

$$\mathfrak{T}_i(v) = |\{d \in D : \ \psi(d) = v, \ I(d) = i\}| \tag{7}$$

The traversal lists are updated incrementally while we process data points from the priority queue $\mathcal{L}$.

Note that $\mathfrak{T}_i(v)$ denotes the number of data points d, such that $\Psi(d) = v$ and $f(s) > f(d) > f(s')$, where s and s' are skylines with time stamps $I(s) = i$ and $I(s') = i + 1$, respectively. For a pictorial interpretation, see Figure 2. In other words, data points corresponding to $\mathfrak{T}_i(v)$ are not dominated by any skyline point with time stamp higher than i. However, they could be dominated by some skyline point with time stamp lower than or equal to i. We utilize this property to compute the residual dominance numbers in our algorithm discussed in Section 6.2.

EXAMPLE 5. *We show the time stamp value for each data point in Table 2, while the traversal lists for all lattice nodes are given in Table 3.*

Space Complexity: We now report the additional space complexity incurred by our modified data structures. The size of a traversal

Table 3: *Traversal Lists for Lattice Nodes*

Lattice nodes	Traversal Lists			
	$i=1$	$i=2$	$i=3$	$i=4$
v_1	0	0	0	0
v_2	0	0	0	2
v_3	0	0	0	0
v_4	0	0	0	0
v_5	1	0	0	1
v_6	0	1	0	1
v_7	0	0	1	1

list associated with some lattice node is $\Theta(m)$, where m is the number of skyline points. Since $m \leq 2^r$, the overall space complexity due to all traversal lists is $\mathcal{O}(4^r)$. In addition, the time stamp values incur an additional $\Theta(n)$ space complexity. Therefore, the overall space consumed by our main-memory data structure in order to find the top-k representative points is $\mathcal{O}(n + 4^r)$.

6.2 Finding Residual Dominance

In this section, we shall describe our algorithm to compute the residual dominance number. Assume, $\mathbb{S} = \{s_1, s_2, \ldots, s_{j-1}\}$ have already been selected as the top $(j-1)$ skyline points in our iterative hill-climbing algorithm (Algorithm 1). Now, given another skyline point s, we shall compute the residual dominance number $R(s, \mathbb{S})$ as follows. Specifically, we consider all the nodes of the sub-lattice rooted at v, where $v = \Psi(s)$, and aggregate their contributions in $R(s, \mathbb{S})$. Let us consider a lattice node $v' \in succ(v)$. Observe that $\sum_{i=I(s)}^{m} \mathfrak{T}_i(v')$ is the number of points $d \in D$, such that $\Psi(d) = v'$, and s dominates d. Here, m is the maximum value of the SkyCount variable, which is equivalent to the total number of skyline points. Recall that, for simplicity of description, we assume no two data points have the same $f()$ score. Now, there can be two distinct cases.

- If $L(v') \not\subset P(s_t)$, for all $s_t \in \mathbb{S}$, then s_t does not dominate d, for all $s_t \in \mathbb{S}$, where $\Psi(d) = v'$. Thus, $\sum_{i=I(s)}^{m} \mathfrak{T}_i(v')$ gives the number of points d, with $\Psi(d) = v'$, and s dominates d, but s_t does not dominate d, for all $s_t \in \mathbb{S}$.

- Otherwise, we first identify all $s_t \in \mathbb{S}$ for which $P(s_t) \supseteq L(v')$. Now, we need to verify if s dominates d and none of these s_t dominates d, for some data point d with $\psi(d) = v'$. As both s and s_t dominate d in terms of satiated query constraints, we need to consider their $f()$ scores. Thus, we find I_{min} — the minimum time stamp of all $s_t \in \mathbb{S}$ for which $P(s_t) \supseteq L(v')$. Again, two distinct cases may occur as shown in Figure 3.

  - **Case (a).** If the time stamp of s is greater than I_{min}, it implies that $f(s)$ is smaller than $f(s_t)$ for some $s_t \in \mathbb{S}$. Hence, for all points d with $\Psi(d) = v'$, if s dominates d, it follows that there exists some $s_t \in \mathbb{S}$ such that s_t also dominates d. Therefore, in such cases, v' does not contribute anything in $R(s, \mathbb{S})$.

  - **Case (b).** On the other hand, if the time stamp of s is lower than I_{min}, then $\sum_{i=I(s)}^{I_{min}-1} \mathfrak{T}_i(v')$ gives the number of points d with $\Psi(d) = v'$, such that s dominates d, but s_t does not dominate d, for all $s_t \in \mathbb{S}$.

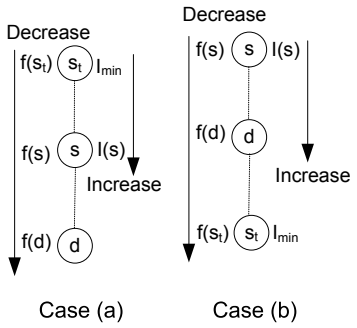


Figure 3: Computation of Residual Dominance:s_t already selected in the top-k skyline set, s a candidate for the top-k skylines in the current round, d a data point such that $\psi(d)$ is a descendent of $\psi(s)$ in the query lattice.

6.3 Time Complexity

The time complexity to construct the priority queue $\mathcal{L}$, query lattice $\mathcal{Q}$, and then finding all the m skyline points along with their dominance numbers is $\mathcal{O}(n \log n + nr + 4^r)$.

Next, to identify the top-k skyline set, we require k iterations of the hill-climbing method. As an initialization phase of the iterative hill-climbing, we also need to sort the skyline points based on their dominance numbers, which has complexity $\mathcal{O}(m \log m)$. At the j-th iteration of hill-climbing method (Algorithm 1), $1 \leq j \leq k$, we might need to consider all the remaining $(m - j)$ skyline points to identify the top j-th skyline point s_j. Now, computing the residual dominance for one skyline point at the j-th iteration requires $\mathcal{O}(m2^r)$ time. Therefore, our hill-climbing method with total k iterations have time complexity $\mathcal{O}(m^2 k 2^r)$.

Considering the initial data-structure-construction time and since $m \leq 2^r$, the overall time complexity of our method is $\mathcal{O}(n \log n + nr + k8^r)$. We note that for a relatively small number of query constraints r, the time complexity of our algorithm is dominated by $\mathcal{O}(n \log n + nr)$. In such settings, our query processing time increases log-linearly with the number of data points, and it increases linearly with the number of constraints.

7. EXPERIMENTAL RESULTS

We present experimental results to demonstrate (1) efficiency (Sec. 7.2), (2) effectiveness (Sec. 7.3), and (2) scalability (Sec. 7.4) of our techniques over three synthetic datasets. We also demonstrate case studies using one real-world dataset (Sec. 7.5).

7.1 Experimental Setup

Data Sets: We used three synthetic and one real-world datasets as summarized below.

Synthetic Datasets. We generated these synthetic datasets by using the generator obtained from [2]. In the *Independent* dataset, all attribute values are generated independently from a pre-defined range with a uniform distribution. The *Correlated* database represents an environment in which points that have higher values in one dimension also have higher values in the other dimensions. In the *anti-correlated* dataset, points which have higher values in one dimension have lower values in one or all of the other dimensions. The cardinality of each of our synthetic datasets is 1M, the dimensionality of each data point is 15. The range of each attribute lies between (1,100).

Car Dataset. We obtained the *car* dataset from [10]. The database

contains information about 598 different models of cars — each model has a certain number of reviews (range: $11 \sim 540$) and 10 various ratings (range: $0 \sim 10$) corresponding to fuel, interior, exterior, build, performance, comfort, reliability, fun, and overall-rating.

Query Selection and Parameter Setting: For our synthetic datasets, we design our queries as follows. Each data point d is 15-dimensional, and let us denote by $D_i(d)$ the value of the i-th dimension of a data point d. We design our top-k representative queries with binary constraints as given in Equation 8. We also vary the number of query constraints r from 2 to 14. Let us denote by p_i the i-th constraint, which is: $D_i(d) \geq 95$.

$$\begin{aligned} &\max_{d \in D} \quad D_{15}(d) \\ &\text{subject to} \quad D_i(d) \geq 95, \quad \forall i \in (1, r) \end{aligned} \tag{8}$$

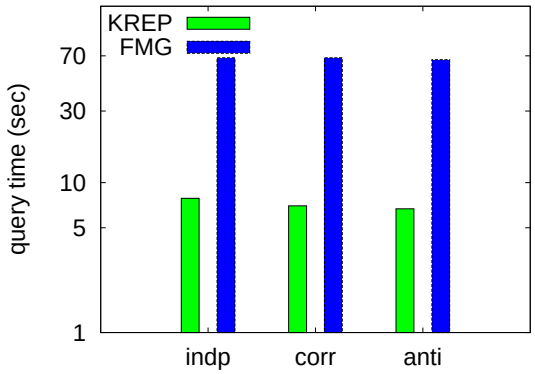
Comparing Methods: We compare the efficiency of our KREP framework with that of FMG [16], which is state-of-the-art method for finding the top-k representative skylines. FMG [16] applies the similar notion of dominance-number-based ranking of a skyline set. However, as the method was designed for the metric space, it is not optimized towards our query semantics, where the constraints are binary in nature. We compare our efficiency results and memory usage with that of FMG. The authors of [16] kindly provided us the executables (compiled using gcc). Since both our approach and FMG have the same approximation guarantee (see Theorem 3) in terms of identifying the top-k representative skyline nodes, we only present the effectiveness results of our method.

We implemented our codes using C++. Each experimental result was averaged over 20 runs. All experiments were run using a single core in 128GB, 2.4GHz Xeon server.

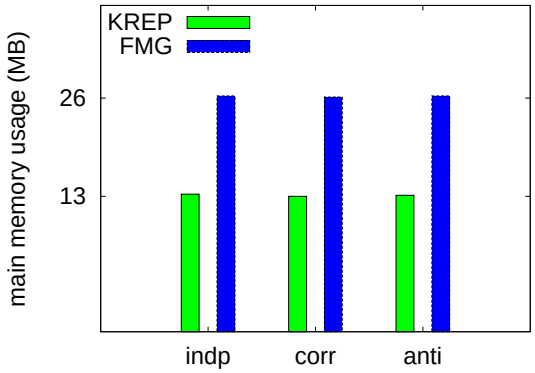
7.2 Efficiency

Comparison with FMG: In these experiments, we compare the query processing times of KREP and FMG. It is worthwhile to mention that both these methods identify the top-k skyline set with the highest dominance number. Nevertheless, FMG was designed for the metric space; and hence, it cannot exploit the fact the our constraints are binary in nature. For our comparison, we use the executables provided by the authors [16]. Since their executables can support data points with maximum dimensionality only 5, we restrict the number of query constraints as 4 in these experiments. Figure 4(a) shows that our method, KREP is one order of magnitude faster than FMG over the three synthetic datasets. In Figure 4(b), we compare the main memory usage of KREP and FMG. We find that KREP outperforms FMG in terms of memory usage in all our experiments. Particularly, for our synthetic datasets, the memory usage of KREP is almost half of the memory used by FMG: 13.2MB for KREP vs. 26MB for FMG. Our results attest that the proposed framework is more efficient than state-of-the-art FMG technique — both in terms of running time as well as in memory usage.

Varying Number of Top-k Skyline Points: In this section, we show the effect of varying top-k values on our query processing time. As our query processing time is dominated by the initialization phase, that is, sorting of the data points according to their optimization scores and building of the query DAG based on satiated constraints sets, the effect of varying top-k values is minimal over the entire query processing time. Therefore, to realize the variation of running time with respect to different top-k values, we



(a) Query Processing Time



(b) Main Memory Usage

Figure 4: Comparison between KREP and FMG: #Constraints = 4, TopK = 10

Table 4: Query Processing Time vs. Top-k, # Constraints = 9

Datasets	Initialization (sec)	Hill-Climbing (sec)			
		topk=5	topk=10	topk=20	topk=50
indp	1.083	0.3661	0.3853	0.3953	0.4007
corr	0.766	0.0407	0.0410	0.0410	0.0410
anti	0.967	0.1954	0.1972	0.1978	0.1994

differentiate in Table 4 the initialization phase from the top-k skyline finding step, which corresponds to the iterative hill-climbing algorithm. For these experiments, we set the number of query constraints as 9, and the top-k values varied from 5 to 50. We observe that the running time for finding the top-k representative data points increases almost linearly with increasing values of k.

7.3 Effectiveness

We measure the effectiveness of our top-k representative points based on how many of the relevant data points that they dominate. We recall that a data point is relevant if it satisfies at least one of the query constraints. We define a metric, *representativeness* as the percentage of relevant data points that are dominated by our top-k representative points. We present the representativeness of our top-k data points, corresponding to different values of k, in Tables 5 and 6.

Table 5 shows the representativeness of our top-k answers when we set the number of query constraints as 9. It can be observed

Table 5: Representative Power of Top-k Skylines, # Constraints = 9. Representativeness is measured as the percentage of relevant data points dominated by our top-k representative points.

Datasets	# Relevant Points	# Skylines	% Representativeness		
			topk=3	topk=5	topk=10
indp	369 478	211	93.07%	97.53%	98.79%
corr	298	8	48.99%	69.80%	100.00%
anti	254 716	120	84.58%	97.64%	99.47%

Table 6: Representative Power of Top-k Skylines, # Constraints = 14. Representativeness is measured as the percentage of relevant data points dominated by our top-k representative points.

Datasets	# Relevant Points	# Skylines	% Representativeness		
			topk=3	topk=5	topk=10
indp	512 623	1 262	82.35%	87.27%	94.88%
corr	435	14	26.44%	41.84%	76.09%
anti	363 807	595	72.72%	86.85%	92.64%

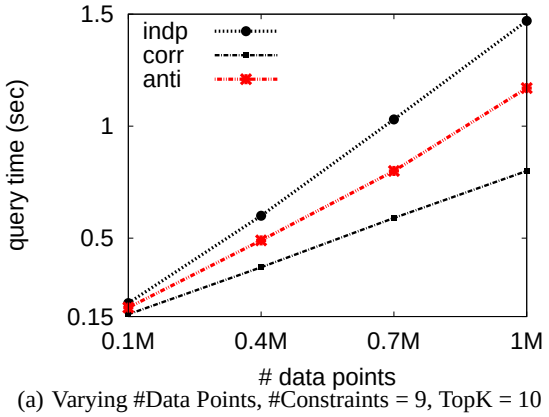
that there are more than 250K relevant data points for both *independent* and *anti-correlated* datasets, while the number of skyline points is also higher than 100 for each of these data points. However, only the top-5 points can represent more than 97% of all relevant data points. These results attest the high effectiveness of our KREP framework, even for a small value of k. We note that the total number of relevant points and skyline points for the *correlated* dataset are only 298 and 8, respectively. This is because of our query setting in Equation 8 and also due to the characteristic of the *correlated* dataset, that is, points that are good in one dimension are good in the other dimensions as well.

Table 6 shows the representativeness of our top-k answers with 14 query constraints. We find that the representativeness percentage of our top-k answers decreases as we increase the number of query constraints. This in indeed expected since both the number of relevant points and the number of skyline points increase when we have more query constraints. Nevertheless, our top-10 data points are still representative of more than 90% of the relevant data points for both *indpendent* and *anti-correlated* datasets, and more than 75% for the *correlated* dataset. These results attest the high usefulness and information content of our top-k representative answers, even for small values of k, such as $k = 5$, or 10.

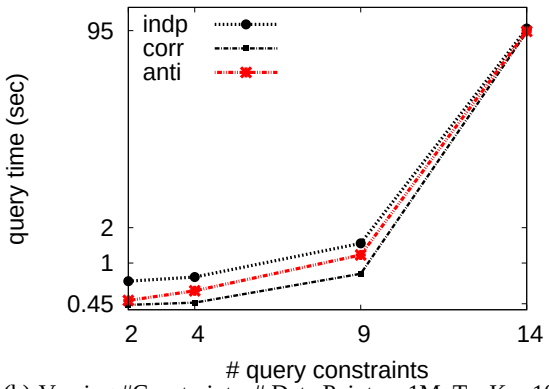
7.4 Scalability

Varying Number of Data Points: We analyze the scalability of our method with varying number of data points in Figure 5(a). We fix the number of query constraints as 9, while varying the number of data points from 0.1M to 1M. It can be observed that our query processing time increases almost linearly with respect to the number of data points. These results illustrate the high scalability of our algorithm for finding the top-k representative data-points.

Varying Number of Query Constraints: We study the scalability of our method with varying number of query constraints in Figure 5(b). Note that the Y-axis is logarithmic in this figure. Our results show that the query processing time increases almost linearly with the number of query constraints, till the number of query constraints is less than 10. However, the query processing time increases exponentially with more than 10 query constraints. This observation can be explained by our time-complexity analysis in Section 6.3. We recall that the overall time-complexity of our algorithm is $\mathcal{O}(n \log n + nr + k8^r)$, where n denotes the number of nodes, r the number of query constraints, and k denotes the top-k value. When r is small, the time-complexity is dominated by the terms: $(n \log n)$ and (nr); and therefore the running time increases almost linearly with the number of query constraints. However, when we have more than a certain number of query constraints, the query processing time increases exponentially with r, which is due to the term $(k8^r)$ in our complexity result.



(a) Varying #Data Points, #Constraints = 9, TopK = 10



(b) Varying #Constraints, # Data Points = 1M, TopK = 10

Figure 5: Scalability over Synthetic Datasets

7.5 Case Studies on Real-World Dataset

For the *car* dataset, we design our query as follows. We use the number of reviews per car-models as the scoring function — recall that the maximum number of reviews for any car model is 540. On the other hand, ratings corresponding to 10 different attributes are utilised in our query constraints. More specifically, each rating has a value in (0,10); and we say that a car-model is relevant if at least one of the ratings for that car-model is greater than or equal to 9. Out of 598 car-models, we find that 500 of them are relevant ones, while there are also 14 skyline car-models based on the aforementioned query setting. We present the top-5 car-models obtained by our method in Table 7. We find that our top-5 answers are representative of 98% of the relevant car-models, while we require only 0.09 sec to retrieve these top-5 results by using our algorithm. One may note that each of our top-5 representative car-models satisfies a diverse subsets of the query constraints, while they also have a relatively large number of reviews.

Summary: We summarize our experimental results below. **(1)** Due to a large number of relevant points, it is important to design an ef-

Table 7: *Case Study: Top-5 Representative Car-Models in* Car *Dataset*

Car-Model	# Reviews	Satiated Constraints
Saturn Aura	216	interior, exterior, build, performance, comfort, reliability, fun, overall-rating
Volkswagen Eos	106	fuel, interior, exterior, build, performance, comfort, reliability, fun, overall-rating
Toyota Tundra	234	exterior, performance, comfort, reliability, fun
Toyota Camry Hybrid	223	fuel, interior, exterior, build, comfort, reliability, fun, overall-rating
Honda Fit	358	interior, exterior, build, reliability, fun, overall-rating

ficient and effective ranking scheme and report only the top-k most representative points. **(2)** Our top-k result set is representative of a large number of relevant data points; and therefore, provides interesting insights about the underlying dataset with respect to the given query constraints. **(3)** Our proposed algorithms are 10 times faster and consume up to 2 times less memory than state-of-the-art top-k representative skyline finding technique [16]. **(4)** Our methods are very scalable with respect to the number of data points and the number of query constraints for a reasonably lower number of query constraints, which is often the case in real-life scenarios.

8. CONCLUSIONS

In this paper, we formulated and investigated the novel problem of answering the top-k representative queries with binary constraints. Our proposed KREP framework finds the top-k data points that are representative of the maximum possible number of available options with respect to the given constraints. For identifying such top-k representatives, we have designed efficient and scalable algorithms which utilize the fact that the constraints present in our queries are binary in nature. Based on detailed empirical evaluation over various real-world and synthetic datasets, we find that KREP is not only one order of magnitude faster than state-of-the-art top-k skyline-finding algorithms, it also produces highly-informative results as well as provides interesting insights about the underlying data with respect to the given query constraints. As a side-product of our algorithm, we also improve the asymptotic complexity of skyline computation to log-linear time in the number of data points, when all dimensions except one are binary in nature. In future work, we shall consider our framework for finding the top-k representative points in the presence of both binary constraints as well as multiple non-binary scoring functions.

9. REFERENCES

[1] A. Angel and N. Koudas. Efficient Diversity-aware Search. In *SIGMOD*, 2011.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *ICDE*, 2001.

[3] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the Average Number of Maxima in a Set of Vectors and Applications. *J. ACM*, 25(4):536–543, 1978.

[4] G. Capannini, F. M. Nardini, R. Perego, and F. Silvestri. Efficient Diversification of Web Search Results. In *VLDB*, 2011.

[5] S. Cook. The Complexity of Theorem-proving Procedures. In *STOC*, 1971.

[6] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou. Parallel Distributed Processing of Constrained Skyline Queries by Filtering. In *ICDE*, 2008.

[7] M. Drosou and E. Pitoura. DisC Diversity: Result Diversification Based on Dissimilarity and Coverage. In *VLDB*, 2012.

[8] M. Endres and W. Kiessling. Optimization of Preference Queries with Multiple Constraints. In *PersDB*, 2008.

[9] M. Endres and W. Kiessling. Semi-Skyline Optimization of Constrained Skyline Queries. In *ADC*, 2011.

[10] K. Ganesan, C. Zhai, and J. Han. Opinosis: A Graph-Based Approach to Abstractive Summarization of Highly Redundant Opinions. In *Computational Linguistics*, 2010.

[11] P. Godfrey, R. Shipley, and J. Gryz. Maximal Vector Computation in Large Data Sets. In *VLDB*, 2005.

[12] S. Guha, D. Gunopulos, N. Koudas, D. Srivastava, and M. Vlachos. Efficient Approximation Of Optimization Queries Under Parametric Aggregation Constraints. In *VLDB*, 2003.

[13] M. Hua, J. Pei, A. W. C. Fu, X. Lin, and H.-F. Leung. Efficiently Answering Top-k Typicality Queries on Large Databases. In *VLDB*, 2007.

[14] H. T. Kung, F. Luccio, and F. P. Preparata. On Finding the Maxima of a Set of Vectors. *J. ACM*, 22(4):469–476, 1975.

[15] R.-H. Li and J. X. Yu. Scalable Diversified Ranking on Large Graphs. In *ICDM*, 2011.

[16] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting Stars: The k Most Representative Skyline Operator. In *ICDE*, 2007.

[17] Q. Mei, J. Guo, and D. Radev. DivRank: The Interplay of Prestige and Diversity in Information Networks. In *KDD*, 2010.

[18] M. Morse, J. Patel, and H. V. Jagadish. Efficient Skyline Computation over Low-Cardinality Domains. In *VLDB*, 2007.

[19] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu. Regret Minimizing Representative Databases. In *VLDB*, 2010.

[20] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An Analysis of Approximations for Maximizing Submodular Set Functions. *Mathematical Programming*, 14(1):265–294, 1978.

[21] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An Optional and Progressive Algorithm for Skyline Queries. In *SIGMOD*, 2003.

[22] C. H. Papadimitriou and M. Yannakakis. On the Approximability of Trade-offs and Optimal Access of Web Sources. In *FOCS*, 2000.

[23] L. Qin, J. X. Yu, and L. Chang. Diversifying Top-K Results. In *PVLDB*, 2012.

[24] S. Ranu, M. X. Hoang, and A. Singh. Answering Top-k Representative Queries on Graph Databases. In *SIGMOD*, 2014.

[25] A. D. Sarma, A. Lall, D. Nanongkai, R. J. Lipton, and J. Xu. Representative Skylines using Threshold-based Preference Distributions. In *ICDE*, 2011.

[26] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based Representative Skyline. In *ICDE*, 2009.

[27] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. A. Yahia. Efficient Computation of Diverse Query Results. In *ICDE*, 2008.

[28] R. C.-W. Wong, J. Pei, A. W.-C. Fu, and K. Wang. Online Skyline Analysis with Dynamic Preferences on Nominal Attributes. *TKDE*, 21(1):35–49, 2009.

[29] L. Zhang, Y. Jia, and P. Zou. A Grid Index Based Method for Continuous Constrained Skyline Query over Data Stream. In *APWeb/WAIM Workshops*, 2009.

[30] Z. Zhang, S. w. Hwang, , K. C.-C. Chang, M. Wang, C. A. Lang, and Y. c. Chang. Boolean + Ranking: Querying a Database by K-constrained Optimization. In *SIGMOD*, 2006.