Efficient Computation of k Representative Regret Minimization G-Skyline Groups

Kangao Wang[1], Xixian Han[1], Xiaolong Wan[1](✉), and Yan Wang[2]

[1] Harbin Institute of Technology, Harbin, China
wka@stu.hit.edu.cn, hanxx@hit.edu.cn, wxl@hit.edu.cn
[2] Macquarie University, Sydney, Australia
yan.wang@mq.edu.au

Abstract. The G-Skyline queries identify Pareto optimal groups not g-dominated by any other group, playing a crucial role in various fields. The k representative G-Skyline queries aim to control the output size and obtain representative results, facilitating user decision-making. However, existing k representative G-Skyline queries cannot meet user requirements well, particularly lacking in quantitative representativeness and high efficiency. In this paper, we propose a novel k representative G-Skyline query, ***k*** representative ***r***egret ***m***inimization ***G***-Skyline (*kRMG*) query, designed to find k G-Skyline groups to minimize the maximum regret ratio. The *kRMG* query provides maximum regret ratio as quantitative representativeness, aiding users in assessing result quality. Then, we propose a novel algorithm, PHP to rapidly obtain *kRMG*. Specifically, PHP proposes prominent G-Skyline groups based on group vectors as small-scale candidate groups, significantly reducing the number of candidates. Additionally, PHP proposes an efficient hierarchical pruning strategy to rapidly obtain prominent G-Skyline groups, effectively eliminating numerous redundant groups. Extensive experiments on synthetic and real datasets demonstrate the efficiency and reliability of PHP.

Keywords: G-Skyline · k representative · pruning strategy.

1 Introduction

The G-Skyline queries aim to identify the optimal groups (termed as *G-Skyline groups*), which play an important role in decision-making [3]. However, the G-Skyline queries often yield massive results, hindering practical applications [7]. To manage this, k representative G-Skyline queries return k representative groups, which are applicable when user preferences are unclear or diverse [11]. For example, in enterprises, they help identify diverse and matched interviewer groups based on skills, improving recruitment efficiency [8].

Motivations and Challenges. Existing efforts [9, 11] have not fully met the user' requirements, particularly lacking in *quantitative representativeness* and *high efficiency*. G-Clustering algorithm [9], based on group distance, fails to achieve *scale invariance* [6] and lacks *quantitative representativeness*. Additionally, it does not meet *high efficiency* due to generating numerous G-Skyline groups. Similarly, TP algorithm [11] also lacks *quantitative representativeness* and does not satisfy *high efficiency* because of the numerous candidates.

Based on the above analysis, three major challenges emerge: *Challenge 1* is: *how to propose a k representative G-Skyline query that not only satisfies scale invariance and stability but also provides quantitative representativeness.* To meet *high efficiency*, it is ideal to obtain the results from small-scale candidates rather than massive groups. Therefore, *Challenge 2* is: *how to significantly narrow down the candidate groups.* Obtaining small-scale candidates requires additional time overhead, so *Challenge 3* is: *how to rapidly obtain the candidate groups.*

Our approach and Contributions. In this paper, we propose a novel ***k*** representative ***r***egret ***m***inimization ***G***-Skyline (*kRMG*) query, such that *kRMG* minimizes the maximum regret ratio. To the best of our knowledge, this is the first work to solve *kRMG* query in the literature. We measure *kRMG* by a *quantitative representativeness*, maximum regret ratio (*mrr*). The *mrr* reflects how regretful users feel about *kRMG* by comparing it with users' most expected results [2]. Benefiting from *mrr*, *kRMG* query satisfies *scale invariance* and *stability* [8]. We propose an efficient algorithm called PHP (**P**resorted index based **H**ierarchical **P**runing algorithm). In PHP, we identify the prominent G-Skyline groups as small-scale candidates and prove that *kRMG* is a subset of them, whose number is much smaller than all G-Skyline groups. Then, we propose an innovative hierarchical pruning strategy to avoid generating numerous groups.

The main contributions are summarized as follows.

- Targeting *Challenge 1*, we first propose a novel k representative G-Skyline query, ***k*** representative **r**egret **m**inimization **G**-Skyline (*kRMG*) query, which aims to find k G-Skyline groups to minimize the maximum regret ratio.
- Targeting *Challenge 2*, in our PHP algorithm, we take the prominent G-Skyline groups as small-scale candidates to obtain *kRMG* and prove that their number is notably reduced compared to the G-Skyline groups
- Targeting *Challenge 3*, in our PHP algorithm, we design a novel hierarchical pruning strategy to rapidly obtain the prominent G-Skyline groups.
- Comprehensive experiments are conducted on synthetic and real datasets, which demonstrate that our PHP algorithm is efficient and reliable.

2 Related Work

The G-Skyline queries are proposed to identify optimized groups, which are highly recognized for not relying on aggregate functions and for including adequate optimized groups [3].

To manage result size, the k representative G-Skyline queries are proposed to identify the k G-Skyline groups which can represent all G-Skyline groups [11], particularly in information retrieval [8]. Yu et al. [9] proposed G-Clustering algorithm to select k G-Skyline groups, which is inspired by k-means algorithm. All G-Skyline groups are generated and group distances between them are calculated. Here, group distance is determined by the Euclidean distance between the tuples in two groups. Finally, G-Clustering returns the k groups closest to the k centers. Zhou et al. [11] proposed TP algorithm to get k G-Skyline groups based on dominance size. The dominance size of a group is the volume defined by its tuples' attribute values and the maximum values. TP iteratively generates G-Skyline groups of size s, containing i skyline tuples ($1 \leq i \leq s$). Finally, TP outputs the k G-Skyline groups with the largest dominance sizes.

3 Problem definition

Let the first $|SC|$ attributes of M attributes serve as the specified skyline criteria SC ($1 \leq |SC| \leq M$). In the dataset D, tuple t_1 dominates tuple t_2 ($t_1 \succ t_2$) if $t_1[i] \geq t_2[i]$ and $t_1[i] > t_2[i]$ in at least one i ($1 \leq i \leq |SC|$). In addition, skyline tuples are those not dominated by any other tuple in D [1].

Definition 1. ***(G-dominance).*** *G_1 and G_2 are two different groups of size s in D. G_1 g-dominates G_2 (denoted by $G_1 \succ_g G_2$) if there are two permutations of G_1 and G_2, $G_1 = \{t_1, t_2, \ldots, t_s\}$ and $G_2 = \{t'_1, t'_2, \ldots, t'_s\}$ respectively, such that $t_i \succ t'_i$ or $t_i = t'_i$ in all i ($1 \leq i \leq s$), and $t_i \succ t'_i$ in at least one i.*

Definition 2. ***(G-Skyline).*** *G-Skyline of size s are defined as those groups containing s tuples not g-dominated by any group of that size, denoted as GS.*

Definition 3. ***(Group vector).*** *The group vector VG of the G-Skyline group $G = \{t_1, t_2, \ldots, t_s\}$ is an $|SC|$-dimensional vector, where the ith element is the sum of the ith attributes of all tuples in G, $VG[i] = \sum_{j=1}^{s} t_j[i] (1 \leq i \leq |SC|)$.*

For *kRMG*, the regret ratio reflects the user regret degree based on the utility function f, which is the mapping $f = \{f[1], f[2], ..., f[|SC|]\} : \mathbb{R}_+^{|SC|} \rightarrow \mathbb{R}_+$. The utilities of t and G are $f(t) = \sum_{i=1}^{|SC|} f[i] * t[i]$ and $f(G) = \sum_{i=1}^{|SC|} f[i] * VG[i]$. Given *kRMG* and a utility function f, the regret ratio of *kRMG* over GS is $rr_{GS}(kRMG, f) = \frac{max_{G' \in GS} f(G') - max_{G \in kRMG} f(G)}{max_{G' \in GS} f(G')}$. We emphasize linear functions LF rather than a utility function [2]. The maximum regret ratio shows the worst case for any $f \in LF$ [8].

Definition 4. ***(Maximum regret ratio, mrr).*** *Given kRMG and the linear utility functions LF, the maximum regret ratio of kRMG over all G-Skyline groups GS is $mrr_{GS}(kRMG, LF) = max_{f \in LF} rr_{GS}(kRMG, f)$.*

Definition 5. ***(k representative regret minimization G-Skyline query, kRMG query).*** *For the group size s and the integer k, kRMG query returns up to k G-Skyline groups of size s while minimizing the maximum regret ratio.*

4 PHP algorithm

4.1 Obtaining the candidate tuples

In this part, PHP obtains the candidate tuples (CT) forming *kRMG*, which is dominated by more than $s - 1$ tuples. For each candidate tuple t, its parent set $P(t)$ includes all tuples that dominate t.

D_1	PI_D	1	2	3	4	5	6	7	8	9	10
	A_1	1	3	5	8	9	1	3	4	5	2
	A_2	10	9	8	7	4	4	2	3	4	8
L_1	PI_1	1	2	3	3	4	5	5	6	7	7
	PI_D	5	4	3	9	8	2	7	10	1	6
	A_1	9	8	5	5	4	3	3	2	1	1
L_2	PI_2	1	2	3	3	4	5	5	5	6	7
	PI_D	1	2	3	10	4	5	6	9	8	7
	A_2	10	9	8	8	7	4	4	4	3	2
PL_1	PI_1	7	5	3	2	1	7	5	4	3	6
	PI_D	1	2	3	4	5	6	7	8	9	10
	A_1	1	3	5	8	9	1	3	4	5	2
PL_2	PI_2	1	2	3	4	5	5	7	6	5	3
	PI_D	1	2	3	4	5	6	7	8	9	10
	A_2	10	9	8	7	4	4	2	3	4	8
PT	MPI	1	1	2	2	3	3	3	4	5	5
	MRN	7	5	5	4	3	5	6	6	7	7
	PI_D	1	5	2	4	3	9	10	8	7	6
	A_1	1	9	3	8	5	5	2	4	3	1
	A_2	10	4	9	7	8	4	8	3	2	4

Fig. 1. The example of constructing PT.

Procedure 1 GetCT

Input: Presorted Table PT, skyline criteria SC, group size s.
Output: Candidate tuples CT.

```
1: CT is initialized to empty
2: Maximum heap MH is initialized to empty
3: for i = 1 to n do // scan on PT
4:    curr ← PT(i) // currently scanned tuple
5:    if |MH| < s or MH.max > curr.MRN
6:       remove root node and add curr to MH
7:    if |MH| = s and MH.max ≤ curr.MPI
8:       break                          ▷ Theorem 1
9:    for every tuple candi in CT do
10:      if curr ≻ candi
11:         add curr to P(candi)
12:         if |P(candi)| = s
13:            remove candi from CT
14:      if candi ≻ curr
15:         add candi to P(curr)
16:         if |P(curr)| = s
17:            break
18:   if |P(curr)| < s
19:      add curr to CT
20: return CT
```

Presorted Table (PT) is a disk-resident index used to rapidly obtain CT. PT $(MPI,\ MRN,\ PI_D,\ A_1, \ldots,\ A_M)$ has n rows and $M+3$ columns for D. Every row reflects a tuple, and A_i is the ith attribute value of each tuple $(1 \leq i \leq M)$. For each row, PI_i denotes the ranking number of representing A_i, then $MPI = min_{1 \leq i \leq M} PI_i$, $MRN = max_{1 \leq i \leq M} PI_i$, PI_D shows position of each tuple in D.

Algorithm Details. Procedure 1 illustrates the process of getting CT. First, we initialize CT as empty (line 1) and use a maximum heap MH to store up to s tuples, sorted by their MRN (line 2). We obtain CT by scanning tuples in PT (lines 3-19), with $curr$ representing the current tuple (line 4). We update MH based on the $curr$ and its maximum MRN (lines 5-6). According to Theorem 1, early termination is achieved when $|MH| = s$ and $MH.max \leq curr.MPI$ (lines 7-8). Then, to decide if $curr$ should be included in CT, we compare it with each tuple $candi$ in CT (lines 9-17). $curr$ is added into CT if $|P(curr)| < s$ after all comparisons (lines 18-19). Finally, CT is returned (line 20).

Theorem 1. *Tuple t is being scanned in PT, all candidate tuples forming kRMG have been scanned, if $MH.max \leq t.MPI$, $|MH| = s$ and $t \neq$ root tuple in MH.*

Proof. $MH.max \leq t.MPI$ and $t \neq$ root tuple in MH, so s tuples in MH dominate t. Since MPI is monotonically non-decreasing in PT, no candidates remain.

Table 1. Process of obtaining the candidate tuples.

PT Tuple	**Candidate Tuples *CT***	**Max heap *MH***
$PT(1) = (1, 7, 1, 1, 10)$	$\{\}$	$\{\}:$
$PT(2) = (1, 5, 5, 9, 4)$	$\{PT(1)\}$	$\{PT(1):7\}:7$
$PT(3) = (2, 5, 2, 3, 9)$	$\{PT(1), PT(2)\}$	$\{PT(1):7, PT(2):5\}:7$
$PT(4) = (2, 4, 4, 8, 7)$	$\{PT(1), PT(2), PT(3)\}$	$\{PT(1):7, PT(2):5, PT(3):5\}:7$
$PT(5) = (3, 3, 3, 5, 8)$	$\{PT(1), PT(2), PT(3), PT(4)\}$	$\{PT(2):5, PT(3):5, PT(4):4\}:5$
$PT(6) = (3, 5, 9, 5, 4)$	$\{PT(1), PT(2), PT(3), PT(4), PT(5)\}$	$\{PT(3):5, PT(4):4, PT(5):3\}:5$
$PT(7) = (3, 6, 10, 2, 8)$	$\{PT(1), PT(2), PT(3), PT(4), PT(5)\}$	$\{PT(2):5, PT(4):4, PT(5):3\}:5$
$PT(8) = (4, 6, 8, 4, 3)$	$\{PT(1), PT(2), PT(3), PT(4), PT(5), PT(7){:}\ P=\{3,5\}\}$	$\{PT(2):5, PT(4):4, PT(5):3\}:5$
$PT(9) = (5, 7, 7, 3, 2)$	$\{PT(1), PT(2), PT(3), PT(4), PT(5), PT(7){:}P=\{3,5\}\}$	$\{PT(2):5, PT(4):4, PT(5):3\}:5$

Example 4.1. Table 1 illustrates the process of obtaining CT when $s = 3$. First, CT and MH are supplemented with $PT(1)$, $PT(2)$, and $PT(3)$. Because no tuple dominates $PT(4)$, it is added into CT. $PT(4).MRN<$ $MH.max = 7$, thus $PT(1)$ in MH is removed, and $PT(4)$ is added into MH, updating $MH.max$ to 5. Likewise, we add $PT(5)$ and $PT(7)$ into CT while $PT(6)$ $PT(8)$ are not candidates. Based on Theorem 1, we skip $PT(9)$ and $PT(10)$ and return CT.

4.2 Obtaining the candidate G-Skyline groups

In this part, PHP addresses *Challenges 2* and *3* well by quickly obtaining small-scale candidates, i.e., the prominent G-Skyline groups (PG).

PG refers to the G-Skyline groups whose group vectors are not dominated by the group vector of any other G-Skyline group. Specifically, group vector VG dominates VG' (denoted by $VG \succ VG'$) if $\forall i$ ($1 \le i \le |SC|$), $VG[i] \ge VG'[i]$ in all i, and $VG[i] > VG'[i]$ in at least one i. Then, we prove that $kRMG$ is a k subset of PG by Theorem 2.

Theorem 2. *$\forall G \in kRMG$, there is $G \in PG$.*

Proof. $G \in kRMG$, if $G \notin PG$, at least one $G' \in PG$ exists such that $VG' \succ VG$, so $kRMG$ should contain G' instead of G. In conclusion, if $G \in kRMG$, $G \in PG$.

We propose an efficient hierarchical pruning strategy to quickly obtain PG, as shown in Theorem 3. The key idea is: based on the monotonicity, when generating new G-Skyline groups from smaller ones, we can safely prune partial groups whose group vectors are dominated.

Theorem 3. *Generating G-Skyline groups of size s from those of size $s-1$ by sorted skyline tuples $SKY = \{t_1, t_2, ..., t_m\}$, if $VG \succ VG'$, and $1 \le i < j \le m$ for the last tuples t_i and t_j in G and G' respectively, G' can be safely pruned.*

Proof. For $G' = G' \cup \{t_a\}$ ($j < a \le m$), there must be $G = G \cup \{t_a\}$ such that $VG \succ VG'$. Thus, G' can be safely pruned.

Algorithm Details. Procedure 2 illustrates how to obtain PG. SKY represents the skyline tuples in CT, sorted by attribute sums in descending order (line 1). Each tuple in SKY is considered PG of size 1 (line 2). We iteratively generate G-Skyline groups of sizes from 2 to s using SKY (lines 3-16). In each iteration, *temp*

Procedure 2 GetPG

```
Input: Candidate tuples CT and group size s.
Output: Prominent G-Skyline groups PG
1: SKY ← skyline tuples sorted by attribute sums in descending order
2: PG ← each skyline tuple as a group
3: repeat
4:   temp ← empty set to keep candidates
5:   for each group G ∈ PG do
       //SKY[a] = G[size]
6:     for i = a + 1 to |SKY| do
7:       G ← G ∪ {SKY[i]} and compute VG
8:       for each group G' ∈ temp do
9:         if VG ≻ VG'
10:          remove G'                ▷ Theorem 3
11:        if VG' ≻ VG
12:          break                    ▷ Theorem 3
13:      if VG' ⊁ VG for G' ∈ temp
14:        add G to temp
15:  PG ← temp
16: until the size of all the groups in PG is s
17: Sort the tuples in CT according to their parent sizes in descending order.
18: Initialize root node v to be null
19: PG = DepthFirstExplore(v)
20: return PG
---------------------------------------------
21: Procedure DepthFirstExplore(Node v)
22:   if |G| ≥ s
23:     Update PG if |G| = s
24:   else      // CT[a] = v or a = 0 if v is null
25:     for i = a + 1 to |CT| do
26:       v ← CT[i] and G ← G ∪ P(v) ∪ v
27:       PG = DepthFirstExplore(v)
28:   return PG
```

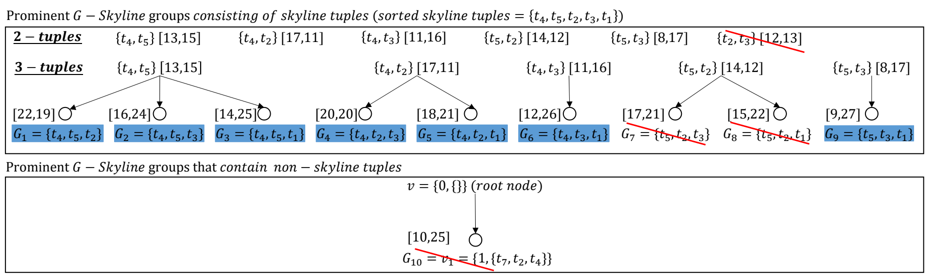


Fig. 2. The illustration of obtaining prominent G-Skyline groups.

stores current PG (line 4), updated based on the generated G-Skyline groups (lines 5-15). Specifically, for each group G, if $VG \succ VG'$, G' is removed from $temp$ based on Theorem 3 (lines 9-10); if $VG' \succ VG$, G is not PG and is pruned based on Theorem 3 (lines 11-12). G is added into $temp$ if no $VG' \succ VG$ for $G' \in temp$ (lines 13-14). Then, we generate PG containing non-skyline tuples (lines 17-19). We sort the tuples in CT in descending order by parent sizes and set root node v to null (lines 17-18). For current group G, we update it with PG when $|G| = s$ (lines 22-23); otherwise, G is expanded by tuple t' and $P(t')$ where t' follows the root node t in CT (lines 25-26). Finally, PG is returned (line 28).

Example 4.2. Figure 2 shows the process of obtaining PG of size 3 by CT, where CT is obtained in Table 1. We denote each $PT(i)$ in CT as t_i. Skyline tuples $SKY = \{t_4, t_5, t_2, t_3, t_1\}$ are sorted by their sums of VG. First, we generate 2-tuples groups by SKY. For $\{t_2, t_3\}$, its VG is dominated by that of $\{t_4, t_5\}$, and t_5 precedes t_3 in SKY, so it is pruned by Theorem 3. Next, we generate 3-tuples groups. G_1 is added into PG since VG_1 is not dominated, as well as G_2, G_3, G_4, G_5, G_6 and G_9. We then obtain PG containing non-skyline tuples, with CT sorted as $\{t_7, t_4, t_5, t_2, t_3, t_1\}$. The root node v is expanded by t_7 to get G_{10}. Since $VG_3 \succ VG_{10}$, G_{10} is not PG. Finally, $PG = \{G_1, G_2, G_3, G_4, G_5, G_6, G_9\}$.

Procedure 3 Greedy

```
Input: Prominent G-Skyline groups PG and an integer k.
Output: k representative regret minimization G-Skyline groups kRMG.
1: Add G' ∈ PG with the largest VG[1] to kRMG
2: while |kRMG|< k do
3:     temp ← 0 and G' ← null
4:     for each group G ∈ PG\kRMG do
5:         now ← mrr_{kRMG∪{G}}(kRMG, LF)
6:         if now > temp
7:             temp ← now, G' ← G
8:     if temp > 0
9:         Add G' to kRMG
10:    else
11:        break
12: return kRMG
```

Table 2. The k representative G-Skyline groups example.

PG	$VG[1]$	$VG[2]$	f(0.6, 0.4)	f(0.3, 0.7)	f(0.1, 0.9)
G_1	22	19	20.8	19.9	19.3
G_2	16	24	19.2	21.6	23.2
G_3	14	25	18.4	21.7	23.9
G_4	20	20	20	20	20
G_5	18	21	19.2	20.1	20.7
G_6	12	26	17.6	21.8	24.6
G_9	9	27	16.2	21.6	25.2

Table 3. Datasets used.

Names	Tuple numbers	Attributes
INDE	1×10^7	6
CORR	1×10^7	6
ANTI	1×10^7	6
HEPMASS	7×10^6	27

Table 4. Experimental parameters.

Parameters	Used values	Default Values
Tuple number n (10^4)	0.1∼1000	1
Skyline criteria size $\|SC\|$	2∼6	4
Group size s	2∼6	3
k	5∼25	10

4.3 Obtaining the *k* representative G-Skyline groups

In this part, PHP obtains $kRMG$ based on PG. PHP employs the greedy idea [6] to obtain the $kRMG$.

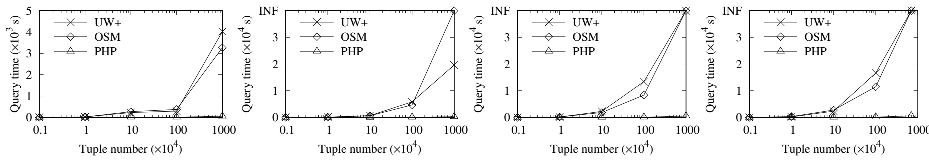
Algorithm Details. Procedure 3 illustrates the process of obtaining $kRMG$. First, we add the group with the largest $VG[1]$ into $kRMG$ (line 1). In subsequent iterations (lines 2-11), we seek the group G that maximizes the mrr (lines 4-7) and add it into $kRMG$ (lines 8-9). Here, we use linear program to compute mrr (line 5). $kRMG$ is returned when $|kRMG| = k$ (line 2) or $mrr = 0$ (lines 10-11).

Example 4.3. Let k be 2, Procedure 3 returns 2 G-Skyline groups from PG in Table 2. First, G_1 with the largest $VG[1]$ is selected. Adding G_2 into $kRMG$, the rr is $\frac{20.8-20.8}{20.8} = 0$ for $f(0.6, 0.4)$, is 0.078 for $f(0.3, 0.7)$ and is 0.168 for $f(0.1, 0.9)$ respectively, so the mrr is 0.168 for LF. Similarly, adding each group of G_3, G_4, G_5, G_6 and G_9 into $kRMG$, the mrr is 0.192, 0.035, 0.067, 0.215 and 0.234 respectively. Thus, G_9 is selected, and $kRMG = \{G_1, G_9\}$.

5 Performance evaluation

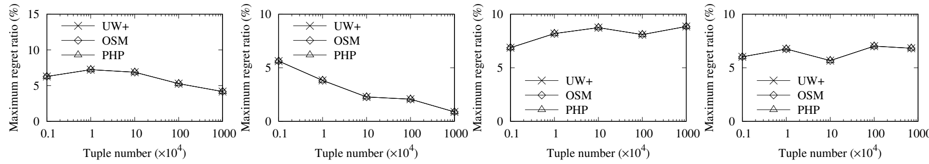
Experiments are conducted on DELL Vostro3681 with Intel Core i7-10700 and 64G RAM. We employ "java -Xmx48G" to control the memory allocation pool.

Table 3 shows the synthetic and real datasets. The synthetic datasets contain independent (INDE), correlated (CORR) and anti-correlated (ANTI) datasets, while the real dataset is HEPMASS dataset, available on archive.ics.uci.edu. Table 4 displays the experimental parameters, which are aligned with those



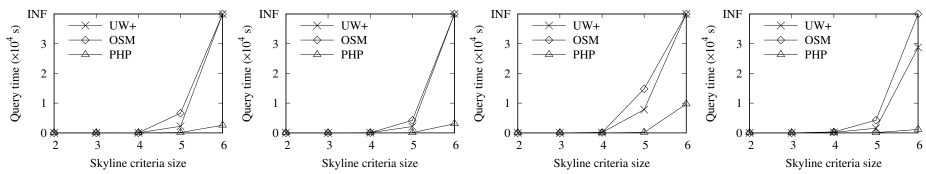
(a) CORR (b) INDE (c) ANTI (d) HEPMASS

Fig. 3. Query time for each dataset of varying tuple number n.



(a) CORR (b) INDE (c) ANTI (d) HEPMASS

Fig. 4. Maximum regret ratio for each dataset of varying tuple number n.



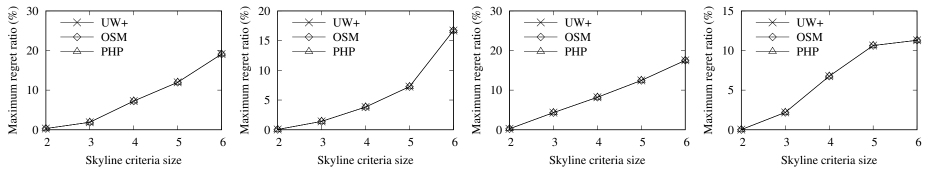
(a) CORR (b) INDE (c) ANTI (d) HEPMASS

Fig. 5. Query time for each dataset of varying skyline criteria size $|SC|$.

used in G-Skyline queries [3, 10] and k representative G-Skyline queries [9, 11]. We compare the performance of PHP, UW+ [5] and OSM [4]. We implement UW+ and OSM to obtain PG and then obtain $kRMG$ based on PG by Procedure in Section 4.3. The efficiency and reliability of PHP are evaluated by query time and maximum regret ratio. Query time over 4×10^4 seconds is shown as INF. We focus on whether the maximum regret ratio is small enough.

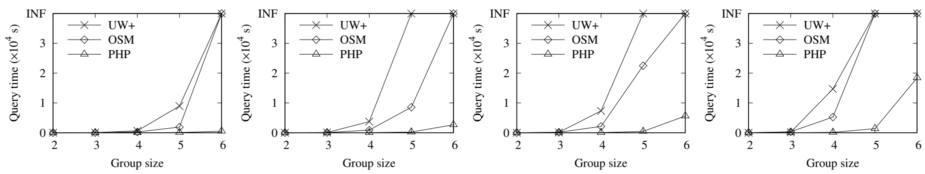
Exp 1: impact of tuple number n. Figure 3 and Figure 4 depict the query time and maximum regret ratio across varying n, respectively. As n increases, query time rises due to the more candidates. The numbers of generated groups of UW+ and OSM grow linearly, so does their query time. The query time of PHP increases significantly slower owing to its hierarchical pruning. The mmr shows no significant correlation with n since the algorithms focus on the selected tuples. The mmr consistently stays below 10%, confirming result representativeness.

Exp 2: impact of skyline criteria size $|SC|$. Figure 5 and Figure 6 depict the query time and maximum regret ratio across varying $|SC|$, respectively. As $|SC|$ increases, the numbers of generated groups of UW+ and OSM increase exponentially, leading to corresponding increases in query time. PHP achieves a speedup of 1-2 orders of magnitude, as it only materializes groups most likely to be the results. The mmr increases with $|SC|$ due to the curse of dimensionality. It becomes harder for a group vector to dominate another, significantly increasing the number of PG. However, the mmr stays below 20%, which is great for users.



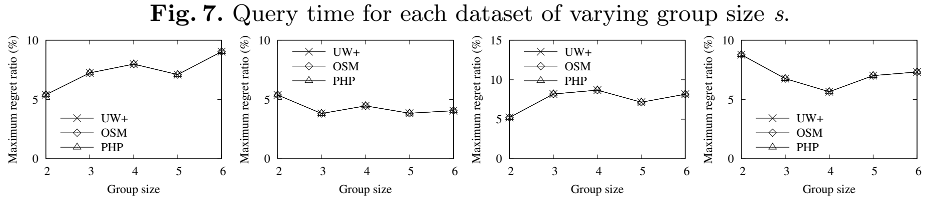
(a) CORR (b) INDE (c) ANTI (d) HEPMASS

Fig. 6. Maximum regret ratio for each dataset of skyline criteria size $|SC|$.



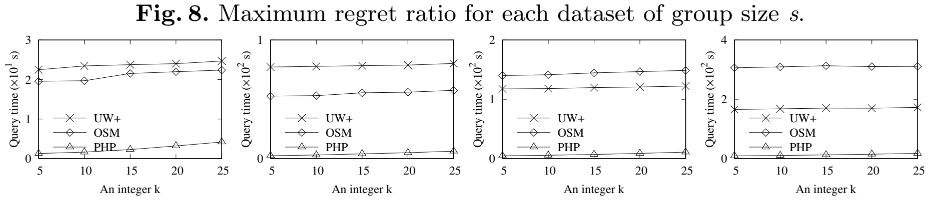
(a) CORR (b) INDE (c) ANTI (d) HEPMASS

Fig. 7. Query time for each dataset of varying group size s.



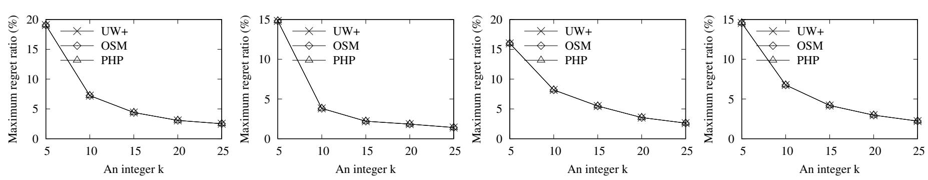
(a) CORR (b) INDE (c) ANTI (d) HEPMASS

Fig. 8. Maximum regret ratio for each dataset of group size s.



(a) CORR (b) INDE (c) ANTI (d) HEPMASS

Fig. 9. Query time for each dataset of varying the result size k.



(a) CORR (b) INDE (c) ANTI (d) HEPMASS

Fig. 10. Maximum regret ratio for each dataset of the result size k.

Exp 3: impact of group size s. Figure 7 and Figure 8 depict the query time and maximum regret ratio across varying s, respectively. Query time of all algorithms rises with s, but PHP's growth is slower due to its hierarchical pruning, achieving 1-2 orders of magnitude speedup over UW+ and OSM. The mmr is not significantly affected by s, because it does not directly impact the group vectors. The mmr remains stable and below 10%, ensuring user satisfaction.

Exp 4: impact of result size *k*. Figure 9 and Figure 10 depict the query time and maximum regret ratio across varying k, respectively. Query time is unaffected by k, as it only affects the time of obtaining $kRMG$ from PG. Additionally, PHP consistently outperforms UW+ and OSM, as it only generates groups that are most likely to be PG. The mmr decreases with k, as more groups better satisfy users. The mmr stays below 10% when $k = 10$, so the results are small-scale and high-quality.

6 Conclusion

This paper proposes a novel ***k*** representative ***r***egret ***m***inimization ***G***-Skyline ($kRMG$) query. $kRMG$ query ensures scale invariance and stability while providing maximum regret ratio as quantitative representativeness to evaluate result quality. This paper proposes an efficient algorithm, PHP, for $kRMG$ query. PHP presents prominent G-Skyline groups as small-scale candidates, which significantly narrow down the candidates. Additionally, PHP designs an efficient hierarchical pruning strategy to rapidly generate prominent G-Skyline groups, eliminating many redundant groups. Extensive experimental results show that PHP efficiently returns $kRMG$ while achieving a small maximum regret ratio.

Acknowledgments. This work was supported in part by NSFC U21A20513, 62402135, Taishan Scholars Program of Shandong Province grant tsqn202211091, Natural Science Foundation of Shandong Province grant ZR2023QF059.

References

1. Borzsony, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proc. IEEE 17th Int. Conf. Data Eng. pp. 421–430. IEEE (2001)
2. Chen, Z., Fu, A.W., Long, C., Wu, Y.: k-pleased querying. IEEE Trans. Knowl. Data Eng. **35**(4), 4003–4017 (2023)
3. Han, X., Wang, J., Li, J., Gao, H.: Efficient computation of G-Skyline groups on massive data. Inf. Sci. **587**, 300–322 (2022)
4. Li, C., Zhang, N., Hassan, N., Rajasekaran, S., Das, G.: On skyline groups. In: Proc. 21st ACM Int. Conf. Inf. Knowl. Manage. pp. 2119–2123 (2012)
5. Liu, J., Xiong, L., Pei, J., Luo, J., Zhang, H., Yu, W.: Group-Based Skyline for Pareto Optimal Groups. IEEE Trans. Knowl. Data Eng. **33**(7), 2914–2929 (2021)
6. Nanongkai, D., Sarma, A.D., Lall, A., Lipton, R.J., Xu, J.J.: Regret-minimizing representative databases. Proc. VLDB Endow. **3**(1), 1114–1124 (2010)
7. Wang, K., Han, X., Wan, X., Wang, J.: Efficient computation of top-k g-skyline groups on large-scale database. Inf. Sci. **670**, 120614 (2024)
8. Xie, M., Wong, R.C., Lall, A.: An experimental survey of regret minimization query and variants: bridging the best worlds between top-k query and skyline query. VLDB J. **29**(1), 147–175 (2020)
9. Yu, W., Liu, J., Pei, J., Xiong, L., Chen, X., Qin, Z.: Efficient contour computation of group-based skyline. IEEE Trans. Knowl. Data Eng. **32**(7), 1317–1332 (2020)
10. Yu, W., Qin, Z., Liu, J., Xiong, L., Chen, X., Zhang, H.: Fast algorithms for pareto optimal group-based skyline. In: Proc. 17th ACM Int. Conf. Inf. Knowl. Manage. pp. 417–426 (2017)
11. Zhou, X., Li, K., Yang, Z., Gao, Y., Li, K.: Efficient approaches to k representative g-skyline queries. ACM Trans. Knowl. Disco. Data (TKDD) **14**(5), 1–27 (2020)