6-2021

On m-impact regions and standing top-k influence problems

Bo TANG

Kyriakos MOURATIDIS
Singapore Management University, kyriakos@smu.edu.sg

Mingji HAN

On m-Impact Regions and Standing Top-k Influence Problems

Bo Tang[†‡] Kyriakos Mouratidis[§] Mingji Han[†]
[†]Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation,
Department of Computer Science and Engineering, Southern Univ. of Science and Technology
[‡]Research Inst. of Trustworthy Autonomous Systems, Southern Univ. of Science and Technology
[§]School of Computing and Information Systems, Singapore Management University
tangb3@sustech.edu.cn; kyriakos@smu.edu.sg; 11711416@mail.sustech.edu.cn

ABSTRACT

In this paper, we study the *m-impact region* problem (*m*IR). In a context where users look for available products with top-k queries, *m*IR identifies the part of the product space that attracts the most user attention. Specifically, *m*IR determines the kind of attribute values that lead a (new or existing) product to the top-k result for at least a fraction of the user population. *m*IR has several applications, ranging from effective marketing to product improvement. Importantly, it also leads to (exact and efficient) solutions for standing top-k impact problems, which were previously solved heuristically only, or whose current solutions face serious scalability limitations. We experiment, among others, on data mined from *actual* user reviews for *real* products, and demonstrate the practicality and efficiency of our algorithms, both for *m*IR and for standing top-k impact problems.

CCS CONCEPTS

• **Information systems** → *Top-k retrieval in databases.*

KEYWORDS

Top-k query; Multi-dimensional datasets; Market analysis

ACM Reference Format:
Bo Tang, Kyriakos Mouratidis, Mingji Han. 2021. On m-Impact Regions and Standing Top-k Influence Problems. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3448016.3452832

1 INTRODUCTION

With the advent of e-commerce, users' preferences over products have been commonly captured by preference functions. The most prevalent model encodes a user's preferences by a *user vector* $\boldsymbol{w}$ that comprises a numeric weight per product attribute. The suitability (score) of a product $\boldsymbol{p}$ is expressed by the weighted sum of its attribute values [30]. This intuitive scoring (and thus ranking) mechanism has been shown by user studies to capture closely the way humans make multi-criteria decisions [51]. Shortlisting the k products with the highest scores is an instance of the well-studied top-k query [17, 24, 29]. Naturally, the presence of a product in the top-k result for many users has been used as a measure of its market impact [25, 57, 58, 66, 67].

In this context, we introduce the *m-impact region* problem (*m*IR). Its input includes a product set $\mathcal{P}$, a user set $\mathcal{U}$, and an integer m between 1 and $|\mathcal{U}|$. Set $\mathcal{U}$ includes a vector $\boldsymbol{w}$ and the k value for each user. We say that a product *covers* a user if it belongs to her top-k result. As output, *m*IR reports the region R in product space (the domain of product attributes) where any existing or hypothetical product covers at least m users in $\mathcal{U}$. Region R is *maximal*, i.e., any product outside of it covers fewer than m users.

Typical product data may come from comparison shopping websites for electronics (CNET.com), restaurants (Yelp.com), etc, which provide ratings of available products on a predefined set of aspects (attributes). User vectors, i.e., preference weights on these aspects, may be explicitly provided by the users [57, 74], learned via interactive polling [32, 51], inferred from their past purchases and browsing history [13, 33], or extracted from the users' product reviews on the same websites [39, 50]. As a practical example, in our evaluation we drew $\mathcal{P}$ and $\mathcal{U}$ from real hotel attributes and user reviews on TripAdvisor.com [7]. For each hotel, the site maintains 7 attributes, i.e., ratings on 7 aspects (value, cleanliness, service, etc). It also records the comments and overall score given by each user who reviewed the hotel. To extract user vectors, we employ the established method in [61]; designed specifically for the weighted sum scoring model, it estimates the relative weights placed on each aspect by a user from the text and scores in her reviews.

To exemplify *m*IR, we use a sample of the aforementioned hotel/user data as input, keeping only two attributes (value and service) for easy visualization, in Figure 1(a). Each row in the lower table indicates a user's vector $\boldsymbol{w}$ (comprising weights $w[1]$ and $w[2]$), her k value, and the respective top-k result. Assuming $m = 3$, Figure 1(b) illustrates the *m*IR output, i.e., region R, shown shaded. It is the union of cells R_1, R_2, R_3, and R_4. Every product inside R_1 covers all four users, whereas a product inside R_2, R_3, or R_4 covers three. Any product outside the shaded region covers fewer than $m = 3$ users. Observe that the *m*IR result (region R) is not convex.

The *m*IR problem has several important applications. Firstly, it helps understand preference and market dynamics. In the prism of existing product competition, it determines the "hottest" part of the market, i.e., the part that attracts most of the users' attention. In our hotel scenario, for example, a travel agency would want to identify the part of the hotel spectrum that concentrates most user interest in the current competitive landscape. That may guide their

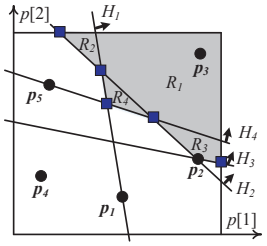
SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452832

Hotel	Value	Service
p_1	0.57	0.24
p_2	0.81	0.42
p_3	0.93	0.91
p_4	0.18	0.43
p_5	0.34	0.75

User	$w[1]$	$w[2]$	k	Top-k result
w_1	0.76	0.24	3	$\{p_3, p_2, p_1\}$
w_2	0.49	0.51	2	$\{p_3, p_2\}$
w_3	0.09	0.91	3	$\{p_3, p_5, p_2\}$
w_4	0.28	0.72	2	$\{p_3, p_5\}$

(a) Product set $\mathcal{P}$ and user set $\mathcal{U}$



(b) mIR result (shown shaded)

Figure 1: mIR example ($m = 3$)

marketing and advertising strategy, and help craft their promotional pricing tactics. Furthermore, mIR may guide the improvement (or the design) of products. For instance, a hotel's management who plan to upgrade their premises/services aspiring to appeal to at least a certain fraction of the user population, would aim for a placement in region R, computed for the respective m value. With R at hand, they can determine which aspects they need to boost more aggressively to meet their goal (e.g., focus more on improving value than service).

Importantly, in addition to its own applications, mIR leads to solutions for existing top-k impact problems, which were previously either not completely/exactly resolved, or whose available solutions face serious performance limitations. Namely, CO and IS:

Influence-based cost optimization (CO): In designing a new product $\boldsymbol{p}$, Yang et al. [67] suggest that the primary requirement for $\boldsymbol{p}$ is to be *influential*, i.e., to be top-1 for at least m users in a given user set. As secondary criterion, $\boldsymbol{p}$ should incur minimum creation cost, for some monotone and convex cost function of its attributes (as is typically the case in practice [25, 47]). That work is specific to $k = 1$. Also, its exact algorithm exhibits long running times, failing to terminate within a day in some cases. To bypass this issue, a sampling alternative is described which, however, is inevitably inexact. The CO problem, *in its general form* (i.e., for $k \geq 1$), can be solved *exactly* by processing mIR, and subsequently computing the cost-optimal $\boldsymbol{p}$ in region R, by off-the-shelf convex optimization solvers. Note that the maximality of R, as required in mIR definition, is essential for the exactness of this approach. Our mIR algorithms lead to at least one order of magnitude faster CO processing than [67] for $k = 1$, while applying for general k too.

Improvement strategies (IS): Another top-k influence problem of practical relevance is defined, but only heuristically solved, in [66]. It seeks to upgrade a product $\boldsymbol{p}$, so that its updated version $\boldsymbol{p'}$ covers the maximum possible number of users in $\mathcal{U}$, subject to a given upgrade budget. The upgrade cost is defined by a function similar to CO, applied to the improvement vector $\boldsymbol{p'} - \boldsymbol{p}$, i.e., defined over the per-attribute increments in $\boldsymbol{p}$. We show that our mIR framework can be extended to IS, and solve it exactly for the first time.

To summarize our contributions:

- We introduce mIR, a practical problem in the multi-criteria top-k context, and propose a novel methodology for its processing;
- In addition to its direct applications, we show that mIR provides exact and scalable solutions to important, standing problems on top-k influence, i.e., CO and IS, as well as to *crossbreeds of the two*, elaborated in Section 5.5; and
- We experiment, among others, with user preferences extracted from *actual* reviews on *real* products, using an established data mining approach. To our knowledge, the database literature so far has only used synthetic preferences in top-k-related evaluations.

2 RELATED WORK

The most common preference queries are the top-k and the *skyline* operators. There are many approaches for efficient top-k processing, including indexing [54, 56], batching [26], and pre-computation [17, 29, 40]; a survey can be found in [30]. The skyline operator reports those products that are not *dominated* by any other, i.e., there is no competitor in $\mathcal{P}$ with higher values in all attributes [14, 48].

The skyline literature includes a few impact definitions, based on dominance. Some studies deem a product marketable if it dominates many, but is dominated by few [27, 36, 70], and use this as a guide to design new products. Others consider the creation or upgrade of products, so that they enter the skyline at minimum or budget-constrained cost [38, 49, 60]. The *dynamic skyline* has also led to similar definitions [31, 63]. Nevertheless, skyline-based influence cannot capture the personalized semantics of the top-k model.

In spatial databases, influence is defined as the number of users who have a product among their k nearest [35]. Identifying spatially influential products has been well-explored [55, 69], and has given rise to influence maximization problems, like [62, 75]. Spatial influence is effectively determined by containment in circles (or rectangles [15, 37, 73]), and thus different from top-k influence.

Some studies consider Boolean decisions per product attribute to enhance marketability. Miah et al. [41] determine which attributes of a product to reveal in order to satisfy the most Boolean conjunctive queries from a known set. Asudeh et al. [9] determine whether to enhance a product with new features at a cost (as a Boolean decision each), while Das et al. [22] aim to attract favorable tags from online reviews. These settings are inherently different from mIR.

In the rest of this section, we focus on top-k-centered problems. Table 1 categorizes them into four types, depending on their preference input (specific user vectors versus preference regions) and their output (points versus regions). mIR falls under Type PR, since its preference input is a user set $\mathcal{U}$, and its output is a region R. Incidentally, mIR is the only problem of Type PR in the top-k context.

Type	Pref. Input	Output
PP	User vector(s)	Point(s)
PR	User vector(s)	Region(s)
RP	Pref. Region	Point(s)
RR	Pref. Region	Region(s)

Table 1: Classification of top-k influence problems

Type PP: Given a user set $\mathcal{U}$, a product set $\mathcal{P}$, and a specific product $\boldsymbol{p}$ in $\mathcal{P}$, Vlachou et al. [57, 59] compute the *reverse top-k* set of $\boldsymbol{p}$ as the set of those users who are covered by it (i.e., who hold $\boldsymbol{p}$ in their top-k result). The same authors [58] identify the products in $\mathcal{P}$ that have the largest reverse top-k sets. These studies employ algebraic bounds to prune the search space, as opposed to the necessarily geometric computation of region R in our setting.

Yang et al. [67] define CO for $k = 1$, as described in Introduction. Their exact algorithm indexes the product space with a Quad-tree, and employs cost and influence bounds to prune index nodes. When the bounds fail to disqualify a leaf, they reduce CO to the (highly

complex) *k-level* problem [12], which they solve by the algorithm in [45]. As a faster, yet inexact alternative, they describe a sampling method. The long version of [67] extends the work to batch CO queries (still for $k = 1$) [68]. Our *m*IR adaptation for CO does not rely on ad hoc partitioning of the product space into quads, nor does it build on the (particularly complex) k-level computation. Moreover, it applies to general $k \geq 1$.

IS is introduced in [66], and formulated as an integer linear programming problem. Due to the high complexity of the latter, the authors propose a heuristic, greedy algorithm. Our *m*IR methodology approaches IS geometrically, and solves it exactly for the first time.

Koh et al. [34] assume that m new products can be assembled by drawing attributes from a set of component tables, resulting in exponential possible combinations. Aiming to cover the most users, they propose approximate assembling algorithms. Gao et al. [25] consider a set of users $\mathcal{U}$ not covered by a product $\boldsymbol{p}$, and propose a sampling method to compute the smallest perturbation required in $\boldsymbol{p}$, the vectors in $\mathcal{U}$, and the k values, so that $\boldsymbol{p}$ covers them all. Given a user set $\mathcal{U}$ and a product set $\mathcal{P}$, Zhang et al. [74] identify the m users in $\mathcal{U}$ that rank a specific product the highest.

Type RP: Ciaccia and Martinenghi [20] assume that a user vector $\boldsymbol{w}$ may lie anywhere in a given *preference region* in weight space, and report all possible top-1 products. In the same setting, Mouratidis and Tang [43] compute all possible top-k results. A related work [44] identifies the best rank a specific product $\boldsymbol{p}$ may achieve for any possible $\boldsymbol{w}$. These studies work in the weight space, i.e., in the domain of $\boldsymbol{w}$. The case in *m*IR is the opposite; user vectors have discrete and known locations, while region R is in the product space (not the weight space) and to be computed (not given as input).

Studies on *regret minimization* select a subset of the products in $\mathcal{P}$, with the objective that the top product in that subset for any possible user vector will not score far worse than the top-scorer in the entire $\mathcal{P}$ [10, 46, 64, 65]. That task is very different from *m*IR.

Type RR: Specified a product $\boldsymbol{p}$ in $\mathcal{P}$, the *monochromatic reverse top-k* query identifies the region(s) in weight space where any user vector is covered by $\boldsymbol{p}$. Tang et al. [52] propose a solution for multiple dimensions, generalizing from the initial two-dimensional approaches [19, 57]. In this problem, the preference input is the entire weight space (as opposed to a specific user set $\mathcal{U}$), and the output does not include any region (or point) in the product space.

Another Type RR problem is presented in [53]. Given as input a preference region (in weight space) and a product set, it computes the region (in product space) where a new product $\boldsymbol{p}$ should be created, so that it covers *every* possible user vector in the specified preference region. The requirement that *all* (instead of m) user vectors should be covered, renders that problem fundamentally distinct from *m*IR, because it removes the combinatorial challenge of covering any m vectors from $\mathcal{U}$. Also, the continuity of the preference region (as opposed to a discrete user set in *m*IR) requires very different treatment in weight space.

3 PROBLEM FORMULATION

Consider a product set $\mathcal{P}$, where each product $\boldsymbol{p}$ has d attributes, i.e., $p[1], p[2], \cdots, p[d]$. A user's preference profile is represented by a vector $\boldsymbol{w} = (w[1], w[2], \cdots, w[d])$ and a positive integer k. The top-k result of that user is denoted by $TK(\boldsymbol{w})$, and comprises the k products with the highest scores, as defined by $S(\boldsymbol{p}, \boldsymbol{w}) = \sum_{i=1}^{d} p[i] \cdot w[i]$. Without loss of generality [17], we assume that $\sum_{i=1}^{d} w[i] = 1$, and $w[i] \geq 0$ for each $i \in [1, d]$. We consider a population of users, whose profiles are kept in a user set $\mathcal{U}$. When a product $\boldsymbol{p}$ belongs to the top-k result of a user, we say that $\boldsymbol{p}$ *covers* that user. We define the *m-impact region* problem (*m*IR) as follows.

Problem 1 (*m*IR Problem). *Given a product set $\mathcal{P}$, a user set $\mathcal{U}$, and a positive integer m, the* mIR *problem is to compute the maximal region* R *in product space, inside which any (existing or hypothetical) product* $\mathbf{p}$ *covers at least m users from $\mathcal{U}$, i.e.,* $|\{\mathbf{w}|\mathbf{w} \in \mathcal{U} \wedge \mathbf{p} \in TK(\mathbf{w})\}| \geq m$.

We stress that R must be *maximal*, meaning that a product covers at least m users if and only if it lies in R, i.e., any product outside R covers fewer than m. In the example of Figure 1, R is shown shaded.

Our objective is to design exact and efficient *m*IR algorithms. Note that responsiveness is of the essence in our target applications. For instance, a business analyst who examines preference/market dynamics in an exploratory manner may need to solve *m*IR for different m or k values, or for different subsets of attributes (decision aspects). Even for fixed parameters, she may need to consider different user categories in consecutive runs (e.g., according to net worth or gender). Practical response times are important to support this type of exploratory analysis.

In the following, we refer to a user vector $\boldsymbol{w} \in \mathcal{U}$ simply as user $\boldsymbol{w}$. Users may have different k values; in order not to complicate presentation, we use the generic notation k to apply each time to the individual users implied by context.

4 BASIC SOLUTIONS

In this section, we present two basic *m*IR algorithms, which reveal important characteristics of the problem, but also bring out the computational challenges it entails.

4.1 Naïve Algorithm

Let $S^k_{\boldsymbol{w}_i}$ be the top-k-th score for user $\boldsymbol{w}_i$ across the products in $\mathcal{P}$. In order for an (existing or hypothetical) product $\boldsymbol{p}$ to be in the top-k result for $\boldsymbol{w}_i$, it should hold that $S_{\boldsymbol{w}_i}(\boldsymbol{p}) \geq S^k_{\boldsymbol{w}_i}$. This inequality *corresponds to a halfspace in product space*, whose supporting hyperplane serves as an entry boundary to $TK(\boldsymbol{w}_i)$. We denote that halfspace as H_i, and call it the *influential halfspace* of $\boldsymbol{w}_i$. Formally,

$$\mathsf{H}_i = \{\boldsymbol{p} | S_{\boldsymbol{w}_i}(\boldsymbol{p}) \geq S^k_{\boldsymbol{w}_i}\}$$

Due to the monotonicity of the scoring function, H_i always includes the *top corner* of the product space, i.e., the hypothetical product with the maximum possible values in all d attributes (dimensions).

Given $\mathcal{P}$, $\mathcal{U}$, and m, the idea in the naïve *m*IR algorithm (NVE) is to consider every possible m-sized user set $\mathcal{U}_j$ from $\mathcal{U}$. For each of these m-sized sets to (i) generate the influential halfspace H_i for every user $\boldsymbol{w}_i$ in the set, and (ii) use a standard geometric library, such as [5], to compute the intersection of their m influential halfspaces. Let $\mathsf{R}_{\mathcal{U}_j}$ denote that intersection for the j-th user set. By definition, those and only those products in $\mathsf{R}_{\mathcal{U}_j}$ cover all users in $\mathcal{U}_j$. Following directly the formulation in Problem 1, the *m*IR result

(i.e., region R) is the union of all the $R_{\mathcal{U}_j}$ regions. Note that there are $\binom{|\mathcal{U}|}{m}$ m-sized user sets (and an equal number of $R_{\mathcal{U}_j}$ regions).

Lemma 1. *The time complexity of* NVE *is* $O(\binom{|\mathcal{U}|}{m} \cdot m^{\lfloor \frac{d}{2} \rfloor})$, *where* d *is the number of product attributes.*

Proof. It takes $O(m^{\lfloor \frac{d}{2} \rfloor})$ time to compute the intersection of m halfspaces [18]. Since there are $\binom{|\mathcal{U}|}{m}$ m-sized user sets to consider, the overall cost is $O(\binom{|\mathcal{U}|}{m} \cdot m^{\lfloor \frac{d}{2} \rfloor})$. □

The naïve algorithm is exact and technically correct, but it is clearly not practical for real-sized problems, due to its excessive computational cost. That said, it reveals a few interesting points about mIR. First, since all influential halfspaces H_i include the top corner of the product space, so does each $R_{\mathcal{U}_j}$ region. Hence, all $R_{\mathcal{U}_j}$ regions overlap with each other, which means that the mIR result (being their union) is a *single, connected region.* Another remark regards the extreme m settings. When $m = 1$, the mIR result is simply the union of $|\mathcal{U}|$ halfspaces. When $m = |\mathcal{U}|$, it is the intersection of $|\mathcal{U}|$ halfspaces. These extremes represent the easiest mIR settings. We expect the toughest settings to be when m is about half of $|\mathcal{U}|$. As a final point, observe that the individual k values of different users do not affect the algorithmic design at all, as long as the influential halfspace for each of them is set according to her personal k setting. The same holds for all subsequent mIR algorithms.

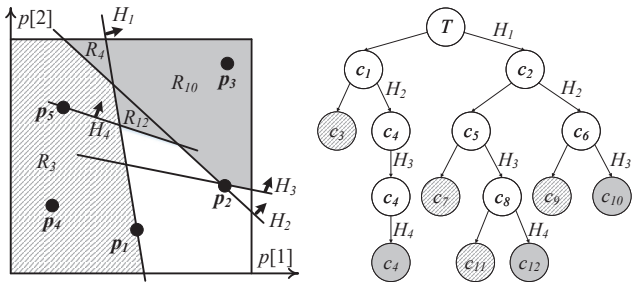
4.2 Baseline Algorithm

The naïve algorithm has more of a theoretical interest. Here we describe a more workable baseline solution (BSL). It serves as a stepping stone to present our advanced approach later.

We observe that several regions in the product space are considered multiple times by NVE. For example, in Figure 1, NVE considers $\binom{4}{3} = 4$ user sets $\mathcal{U}_j$, whose respective $R_{\mathcal{U}_j}$ regions all include cell R_1 in Figure 1(b). On the other hand, some parts of the product space are not included in any $R_{\mathcal{U}_j}$. This motivates the BSL approach, which follows a space-centric paradigm, as opposed to considering all possible m-sized user sets. With an appropriate partitioning of the product space, many of its partitions can be directly reported (as parts of the mIR result) or eliminated.

The main idea in BSL is that the influential halfspaces of all users define a partitioning of the product space, formally called a *halfspace arrangement*, and its partitions, called *cells* [12]. A cell belongs to the mIR result if and only if it is inside at least m influential halfspaces. In Figure 1, the complete arrangement includes 10 cells, out of which only R_1, R_2, R_3, and R_4 cover enough users. Instead of building the complete arrangement, BSL constructs it gradually by inserting influential halfspaces one by one. This allows the early reporting of cells that are already found to cover m users, and the direct elimination of those already certain not to reach that number.

In our running example, suppose we first insert H_1 and H_2 to the arrangement, producing four cells. Referring to Figure 2(a), among these four, cell R_3 (shown striped) can be *early eliminated*, because any product $\boldsymbol{p} \in R_3$ is already known not to cover $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$, and even if it is inside the (not yet inserted) H_3 and H_4, it still cannot meet the $m = 3$ requirement. In contrast, after the insertion of H_3, cell R_{10} already covers $m = 3$ users (i.e., $\boldsymbol{w}_1$, $\boldsymbol{w}_2$, $\boldsymbol{w}_3$) and can be *early reported* as part of the final result R. Once eliminated or reported, a cell is ignored (not split any further) by subsequent insertions. BSL terminates when all cells are either eliminated or reported.



(a) Halfspace arrangement (b) Binary tree representation

Figure 2: BSL **example**

For the maintenance of the arrangement we follow the cell-tree approach [52], although alternatives could be used, like the Quad-tree in [71]. The cell-tree is a binary tree, whose root T corresponds to the entire product space, and each leaf c to a cell in the current arrangement[1]. Inserting a halfspace, adds a level to the tree by splitting the leaves it cuts through. Cell-tree nodes are implicitly represented by the set of halfspaces that define them. That is, each node c is associated with a list $c.in$ which stores the halfspaces that include it, and a list $c.out$ which stores those that exclude it; halfspaces that cut through c are propagated down to its children (or, if c is a leaf, cause it to split). Effectively, in addition to maintaining the arrangement, that structure also allows to derive for any leaf c (through referral to its own *in* and *out* lists, as well as those of its ancestor nodes) the number of inserted halfspaces that include it, i.e., $\text{Size}(c.in)$, and of those that exclude it, i.e., $\text{Size}(c.out)$. Figure 2(b) illustrates the tree for the final arrangement constructed in Figure 2(a). The striped leaves correspond to eliminated cells, while the shaded to those reported as parts of the result R, respectively. Algorithm 1 summarizes BSL, including the recursive InsertHS() module that inserts a new halfspace to the arrangement.

Lemma 2. *The time and space complexity of* BSL *is* $O(|\mathcal{U}|^d)$.

Proof. In the worst case, BSL fails to perform any early reporting/elimination, thus being equivalent to computing the complete arrangement of $|\mathcal{U}|$ halfspaces, requiring $O(|\mathcal{U}|^d)$ time and space [8]. □

5 ADVANCED APPROACH

Although the worst case scenario expressed by Lemma 2 (i.e., no early reporting/elimination at all) is very unlikely, BSL is still impractical for large problem instances. In this section, we present our advanced approach (AA) to solve the mIR problem efficiently, while retaining the correctness and exactness of the solution. The AA methodology differs from BSL in two major ways. First, instead of processing the influential halfspaces one by one, we process them in groups, formed according to geometric properties of the problem. Second, unlike BSL where each halfspace insertion is "global" and applies to all tree leaves (arrangement cells), in AA

[1]We denote tree nodes by c_i, but we may also refer to their respective regions R_i in product space interchangeably, depending on context.

Algorithm 1 BSL($\mathcal{P}, \mathcal{U}, k, m$)

```
1: Initialize result set R ← ∅
2: Initialize root T of the binary tree
3: for each user w_i in U do
4:     InsertHS(T, H_i)                          ▷ insert influential halfspace of w_i
5:     for each leaf node c do                   ▷ i.e., for each cell in the arrangement
6:         if Size(c.in) ≥ m then
7:             R ← R ∪ c                         ▷ early reporting
8:             Ignore c onwards
9:         else if |U| − Size(c.out) < m then
10:            Ignore c onwards                  ▷ early elimination
11: Return R

12: Routine InsertHS(c, H)
13: if c ∩ H = c then                            ▷ H covers c
14:     c.in ← c.in ∪ {H}
15: else if c ∩ H = ∅ then                       ▷ H excludes c
16:     c.out ← c.out ∪ {H}
17: else                                         ▷ H cuts through c
18:     if c is leaf then                        ▷ split c into two parts
19:         Initialize left leaf c.l and right leaf c.r
20:         c_l.out ← H
21:         c_r.in ← H
22:     else                                     ▷ H cuts through c; insert H to c's children
23:         InsertHS(c_l, H)
24:         InsertHS(c_r, H)
```

we disassociate the partitioning performed at different cells, and individualize the decision of which halfspace (or group of halfspaces, to be exact) will be inserted to each of them next. As we demonstrate logically/analytically in this section, and empirically in the experiments, these design principles are instrumental in our two-fold objective to:

(1) Radically enhance the effectiveness of early reporting and early elimination; and
(2) Keep the size of the arrangement (i.e., the number of cells created) small, without compromising on pruning effectiveness.

The rest of this section is structured as follows. Section 5.1 describes group formation, and Section 5.2 the processing within each group. Section 5.3 presents the complete AA algorithm, while Section 5.4 includes further enhancements applicable to the special $d = 2$ case. Finally, Section 5.5 adapts our methodology to CO, IS, and other related problems.

5.1 Group Formation

Each user $\boldsymbol{w}_i$ corresponds to an influential halfspace H_i, thus, we refer to grouping users and grouping halfspaces interchangeably. We place into the same group G the users $\boldsymbol{w}_i$ that have the same top-k-th product. Letting $\boldsymbol{r}$ be that product, the hyperplanes that define any H_i in G are all bound to pass through $\boldsymbol{r}$.

To estimate the number of groups formed, assume that all users have the same k. The top-k-th product for any user must fall in the k-*skyband* of $\mathcal{P}$ [48], whose expected cardinality is $\eta = \frac{k \ln^{d-1} |\mathcal{P}|}{d!}$ [28]. If each of the η candidates is equally likely to be the top-k-th for a user, the probability that a candidate is the top-k-th for at least one user is $1 - (\frac{\eta-1}{\eta})^{|\mathcal{U}|}$. Hence, the expected number of groups is $\eta(1 - (\frac{\eta-1}{\eta})^{|\mathcal{U}|})$.

Next, we elaborate on the effectiveness of this grouping, i.e., how exactly it serves Objectives (1) and (2) set in the beginning of Section 5. Consider the two-dimensional example in Figure 3, where group G includes the influential halfspaces of 5 users, i.e., $\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_5$, whose top-$k$-th product is $\boldsymbol{r}$. Assume that $m = 5$ and that $\mathcal{U}$ includes $|\mathcal{U}| = 9$ users in total.

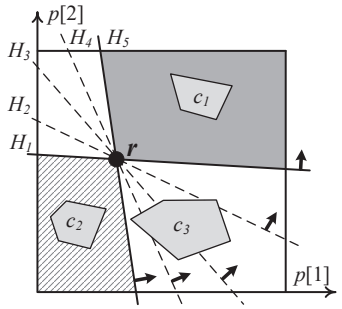


Figure 3: Effectiveness of grouping

Serving Objective (1): First, we demonstrate why considering G in a batch facilitates early reporting and early elimination. We observe in Figure 3 that any product or cell that falls in the shaded area covers every user in G, while every product in the striped one covers none. Considering G for c_1 would directly report it (since $|\mathsf{G}| \geq m$), while it would immediately eliminate c_2 (since the maximum number of users it may cover is $|\mathcal{U}| - |\mathsf{G}| = 4$, i.e., already smaller than $m = 5$). Even in the more general case where m and $\mathcal{U}$ could be much larger, considering G would aggressively increase the size of $c_1.in$ and $c_2.out$ by $|\mathsf{G}|$, thus facilitating the early reporting/elimination of those cells in subsequent processing.

The observation we made above applies to general dimensionality, and is formalized by two lemmas. The first (Lemma 3) stipulates the conditions where all users in G are covered by a product $\boldsymbol{p}$, while the second (Lemma 4) specifies the conditions which guarantee that $\boldsymbol{p}$ does not cover any of them. Effectively, they define the shaded and striped regions in Figure 3, respectively; regarding Lemma 4, note that $\overline{\mathsf{H}_i}$ denotes the complement of H_i. Central to both lemmas is the *convex hull* of G (in the weight space), i.e., the smallest convex set that encloses all the users in G. The convex hull is a standard concept in computational geometry, whose efficient computation has been well explored [11, 21].

Lemma 3. *Let* G *be a group of users who have a common top-k-th product* $\mathbf{r}$*, and* $\mathcal{V}$ *be the set of vertices (user vectors) that define the convex hull of* G*. A product* $\mathbf{p}$ *covers all users in* G *if and only if* $\forall \mathbf{v}_j \in \mathcal{V}, S_{\mathbf{v}_j}(\mathbf{p}) \geq S_{\mathbf{v}_j}(\mathbf{r})$*, i.e., if and only if* $\mathbf{p} \in \cap_{\mathbf{v}_j \in \mathcal{V}} \mathsf{H}_{\mathbf{v}_j}$.

Proof. Any user vector $\boldsymbol{w}$ inside the convex hull can be expressed as a linear combination of its defining vertices [12], i.e., there is a set of $|\mathcal{V}|$ positive values α_j that sum to 1, such that $\boldsymbol{w} = \sum_{\boldsymbol{v}_j \in \mathcal{V}} \alpha_j \boldsymbol{v}_j$. Keeping in mind that $\forall \boldsymbol{v}_j \in \mathcal{V}, S_{\boldsymbol{v}_j}(\boldsymbol{p}) \geq S_{\boldsymbol{v}_j}(\boldsymbol{r}) \Leftrightarrow \boldsymbol{v}_j \cdot \boldsymbol{p} \geq \boldsymbol{v}_j \cdot \boldsymbol{r}$, the following holds:

$$
\begin{aligned}
S_{\boldsymbol{w}}(\boldsymbol{p}) &= \Big(\sum_{\boldsymbol{v}_j \in \mathcal{V}} \alpha_j \boldsymbol{v}_j\Big) \cdot \boldsymbol{p} = \sum_{\boldsymbol{v}_j \in \mathcal{V}} (\alpha_j \boldsymbol{v}_j \cdot \boldsymbol{p}) \\
&\geq \sum_{\boldsymbol{v}_j \in \mathcal{V}} (\alpha_j \boldsymbol{v}_j \cdot \boldsymbol{r}) \text{ , since } \forall \boldsymbol{v}_j \in \mathcal{V}, \boldsymbol{v}_j \cdot \boldsymbol{p} \geq \boldsymbol{v}_j \cdot \boldsymbol{r} \\
&= \Big(\sum_{\boldsymbol{v}_j \in \mathcal{V}} \alpha_j \boldsymbol{v}_j\Big) \cdot \boldsymbol{r} = \boldsymbol{w} \cdot \boldsymbol{r} = S_{\boldsymbol{w}}(\boldsymbol{r})
\end{aligned}
$$

That is, $S_{\boldsymbol{w}}(\boldsymbol{p}) \geq S_{\boldsymbol{w}}(\boldsymbol{r})$ for any $\boldsymbol{w} \in \mathsf{G}$.

For the lemma's other direction, every defining vertex of the convex hull is also a user vector in G. Thus, if $S_{\boldsymbol{v}_j}(\boldsymbol{p}) < S_{\boldsymbol{v}_j}(\boldsymbol{r})$ for some $\boldsymbol{v}_j \in \mathcal{V}$, there is a user in G (i.e., $\boldsymbol{v}_j$) not covered by $\boldsymbol{p}$. □

Lemma 4. *Let* G *be a group of users who have a common top-k-th product* $\mathbf{r}$*, and* $\mathcal{V}$ *be the set of vertices that define the convex hull of* G*. A product* $\mathbf{p}$ *does not cover any user in* G *if and only if* $\forall \mathbf{v}_j \in \mathcal{V}, S_{\mathbf{v}_j}(\mathbf{p}) < S_{\mathbf{v}_j}(\mathbf{r})$*, i.e., if and only if* $\mathbf{p} \in \cap_{\mathbf{v}_j \in \mathcal{V}} \overline{\mathsf{H}_{\mathbf{v}_j}}$.

Proof. Same as Lemma 3, subject to replacing '$\geq$' with '$<$'. □

Note that, although the lemmas refer to points $\boldsymbol{p}$ in product space, they define one convex area each, as an intersection of specific halfspaces. Checking containment of an arrangement cell (i.e., a convex region) in either of these areas is a commonplace operation. However, in Section 5.3 we will describe an optimization which, in many cases, may apply the lemmas *without* resorting to standard containment tests, thus saving computations.

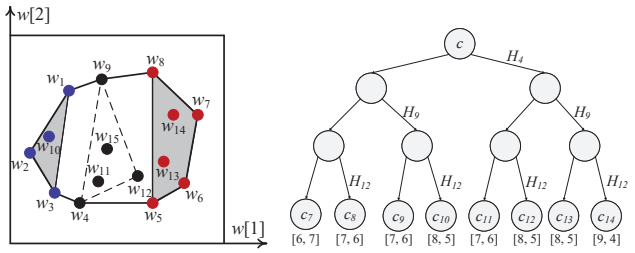
Serving Objective (2): To demonstrate that our grouping strategy also produces fewer cells in the arrangement, consider the two-dimensional example in Figure 3 again, and assume that we are about to insert the first group G of influential halfspaces to our arrangement. At that stage, the only cell in the arrangement covers the entire product space. Due to the passing through a common point $\boldsymbol{r}$, inserting G leads to exactly $2 \cdot |\mathsf{G}|$ cells (i.e., $2 \cdot 5 = 10$ in our example). That is a much smaller number than the $O(|\mathsf{G}|^2)$ cells expected in the arrangement of $|\mathsf{G}|$ general halfspaces in two dimensions [8]. This reduction in cell numbers translates directly to fewer splits, to a simpler arrangement, and to faster operations in that arrangement. The said reduction is not specific to $d = 2$ only. In any dimensionality, the *zone theorem* [23] suggests that each halfspace inserted to a general arrangement will split $O(n^{d-1})$ pre-existing cells, but if the supporting hyperplanes of all halfspaces pass through a common point, that is reduced to $O(n^{d-2})$.

Having elaborated on the usefulness of Lemmas 3 and 4, there are still some cases like cell c_3 in Figure 3 where they are inapplicable, and therefore consideration of the halfspaces within a group G is necessary. This does not mean that we resort to BSL-like halfspace insertions. Instead, major optimizations are possible in this case too, via *inner group processing* described next. In our experiments, inner group processing is responsible for a 4- to 6-fold improvement in processing time (see Section 6.4).

5.2 Inner Group Processing

When a group G is considered for a cell c and is deemed undecided by Lemmas 3 and 4, the inner group processing classifies the group's halfspaces into three categories: halfspaces that cover c are placed in set G^c, those that exclude it are placed in G^e, and those that cut through it in G^i. Sets in the first and second category are directly inserted to $c.in$ and $c.out$, while some (just some!) of those in the third category will be used to partition c. For cell c_3 in Figure 3, the three sets are $\mathsf{G}^c = \{\mathsf{H}_5\}$, $\mathsf{G}^e = \{\mathsf{H}_1\}$, and $\mathsf{G}^i = \{\mathsf{H}_2, \mathsf{H}_3, \mathsf{H}_4\}$.

While the above process is intuitive, to naïvely classify the halfspaces in G one by one takes considerable time, since it requires $|\mathsf{G}|$ containment tests for c, each involving non-trivial geometric calculations. We aim to avoid or defer such operations as much as possible. Our approach to efficiently classify the halfspaces in G relies on $\mathcal{V}$, i.e., the set of vertices (user vectors) that define the convex hull of G. Note that $\mathcal{V}$ is expected to be much smaller than G, e.g., $|\mathcal{V}|$ is only polylogarithmic in $|\mathsf{G}|$ for uniform/normal product distributions [16]. Recall also that $\mathcal{V}$ is already available,



(a) G in weight space (b) Arrangement (sub-)tree of c

Figure 4: Inner group processing example

because it was computed when cell c was considered by Lemmas 3 and 4. Classification into the three categories G^c, G^e, G^i proceeds as follows.

First, we classify the vertices in $\mathcal{V}$ into sets $\mathcal{V}^c$, $\mathcal{V}^e$, and $\mathcal{V}^i$, depending on whether their influential halfspace covers, excludes, or cuts through c. That step is done by standard containment tests, as usual. Next, we compute the convex hull of $\mathcal{V}^c$. The beauty of Lemma 3 is that it applies to every user vector that falls inside the convex hull of $\mathcal{V}^c$; these user vectors can be inserted directly to G^c, since they are guaranteed to cover c too! Similarly, we compute the convex hull of $\mathcal{V}^e$ and, by Lemma 4, classify the user vectors it includes under G^e. The remaining vectors of G are placed in G^i.

Consider Figure 4(a) and user group $\mathsf{G} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_{15}\}$ in weight space[2]. The set of vertices that define its convex hull is $\mathcal{V} = \{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_9\}$. Given a cell c, we test it for containment against $\mathsf{H}_1, \cdots, \mathsf{H}_9$ one by one. From these tests, assume that we derive $\mathcal{V}^c = \{\boldsymbol{w}_5, \boldsymbol{w}_6, \boldsymbol{w}_7, \boldsymbol{w}_8\}$ and $\mathcal{V}^e = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3\}$. Considering the respective convex hulls of $\mathcal{V}^c$ and $\mathcal{V}^e$, we may readily produce $\mathsf{G}^c = \{\boldsymbol{w}_5, \boldsymbol{w}_6, \boldsymbol{w}_7, \boldsymbol{w}_8, \boldsymbol{w}_{13}, \boldsymbol{w}_{14}\}$ and $\mathsf{G}^e = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3, \boldsymbol{w}_{10}\}$. The rest of the users form $\mathsf{G}^i = \{\boldsymbol{w}_4, \boldsymbol{w}_9, \boldsymbol{w}_{11}, \boldsymbol{w}_{12}, \boldsymbol{w}_{15}\}$.

After directly inserting G^c and G^e to lists $c.in$ and $c.out$, respectively, we consider G^i for insertion to cell c. Recall that c is a leaf in the binary tree that maintains the arrangement (see discussion on Figure 2(b)), thus insertions will lead to repetitive splits, forming a sub-tree rooted at c. The leaves of the sub-tree correspond to new cells in the arrangement. The more the inserted halfspaces, the higher the computation cost spent on splits, and the more the new levels and cells introduced in the arrangement tree. To avoid the cost of this eager insertion, instead of inserting all the halfspaces of G^i, we employ a *delayed insertion* strategy, by (i) only inserting halfspaces H_i that correspond to user vectors $\boldsymbol{w}_i$ on the convex hull of G^i, and (ii) propagating/deferring the insertion of the remaining halfspaces in G^i to the leaves of c's sub-tree. The rationale for the deferral is that when AA processes each of these new leaves, alternative groups will also be considered, which could lead to direct reporting/elimination for some of them (a situation that, in hindsight, would render the eager partitioning of c with *all* halfspaces in G^i an unnecessary waste of computations).

In Figure 4(a), the convex hull set of G^i is $\mathcal{V} = \{\boldsymbol{w}_4, \boldsymbol{w}_9, \boldsymbol{w}_{12}\}$. The influential halfspaces of the three users are inserted one by one to c, in a manner similar to BSL. Figure 4(b) illustrates the sub-tree

[2]The assumption that $\sum_{i=1}^{d} w[i] = 1$ effectively reduces the weight space dimensionality by one, since $w[d] = 1 - \sum_{i=1}^{d-1} w[i]$. Hence, to be accurate, the user set in Figure 4(a) refers to $d = 3$, but this is a detail we choose not to confuse the reader with.

of c after the insertions, indicating the size of the *in* and *out* lists under each new leaf (equivalently, new cell) created. These sizes are checked for the early reporting/elimination of the cells, as per normal. Regarding the surviving cells, we propagate to them the remaining users in G^i (i.e., $\boldsymbol{w}_{11}$ and $\boldsymbol{w}_{15}$), to be considered (together with other user groups) when these cells are chosen for processing by AA. How exactly AA chooses the next cell to process, and how the different user groups are considered for it, are among the topics clarified in the complete AA process described in the next section.

5.3 AA Algorithm

To put all pieces in perspective, we outline AA, followed by its pseudo-code in Algorithm 2. First, the user set $\mathcal{U}$ is divided into different groups $G_1, G_2, \cdots, G_g$, as explained in Section 5.1.

In each iteration, AA selects for processing one of the (not yet reported or eliminated) cells in the arrangement[3]. That is the cell c closest to being reported or eliminated, i.e., the cell with the smallest value of $min\{m - \text{Size}(c.in), |\mathcal{U}| - m - \text{Size}(c.out) + 1\}$ (the first compared quantity is the number of *covering* halfspaces required to report c, while the second is the number of *excluding* halfspaces required to eliminate it). Having chosen the cell c to process, AA considers all user groups by Lemmas 3 and 4, and updates lists $c.in$ and $c.out$ accordingly (routine Update()). If early reporting or elimination is not possible (routine Verify()), the largest group G is chosen for insertion to c. The group insertion (routine InsertGroup()) follows the process in Section 5.2, by classifying the users in G into three categories (i.e., G^e, G^c, G^i), placing G^c and G^e to $c.in$ and $c.out$, respectively, and eventually partitioning c only with halfspaces that correspond to the convex hull of G^i. Note that group insertion is specific to c and it is the only cell it affects.

Individualized cell partitioning: Importantly, the leaves created by the partitioning of c receive an updated list of user groups that remain to be inserted. We denote that individualized list as $c.\mathcal{G}$. While group insertion executes on c, we remove from $c.\mathcal{G}$ all user groups that were found (by Lemmas 3 and 4) to be entirely covered or entirely not covered by c, and update the inserted group G to only include the remaining halfspaces of G^i (e.g., in Figure 4(a), group G is updated to $\{\boldsymbol{w}_{11}, \boldsymbol{w}_{15}\}$). The updated $c.\mathcal{G}$ list is passed on to the new cells created by the partitioning of c, and will be considered when these cells are processed in a future iteration. The role of group list $c.\mathcal{G}$ is essential in *individualizing the search within (and the partitioning of) each cell*, as we set out to achieve in the beginning of Section 5.

Algorithm 2 sketches the AA process described above. Grouping in Line 2 requires an all-top-k computation. We employ the algorithm in [26] for that. To provide perspective, we note that all-top-k computation is very fast compared to subsequent geometric processing; e.g., in our default experimental setting, it accounts for less than 1% of AA's total time. On a different note, to efficiently organize the arrangement cells according to their $min\{m - \text{Size}(c.in), |\mathcal{U}| - m - \text{Size}(c.out) + 1\}$ value, we use a min-heap $\mathcal{H}$ (see Lines 4, 6, 14). In Line 43, InsertHS() refers to the regular halfspace insertion routine provided in Algorithm 1. AA terminates when the min-heap $\mathcal{H}$ becomes empty.

[3]Originally there is a single cell (which corresponds to the entire product space), but in subsequent iterations the arrangement includes many.

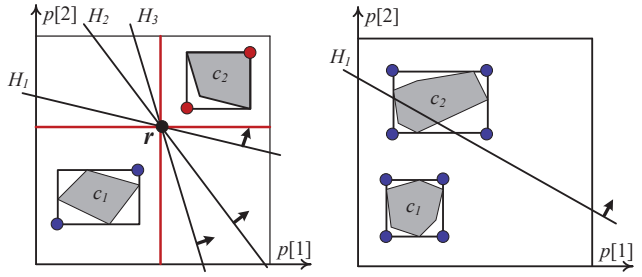
Algorithm 2 AA($\mathcal{P}, \mathcal{U}, k, m$)

```
1: Initialize result set R ← ∅
2: G ← group users in U by top-k-th product                    ▷ Section 5.1
3: Initialize root T of the binary tree with T.G ← G
4: Initialize min-heap H.push(T)
5: while H is not empty do
6:     c ← H.top(); H.pop()
7:     Update(c)
8:     if Verify(c, R) = False then
9:         G ← the largest group in c.G
10:        InsertGroup(c, G, R)
11:        for each leaf c_i in sub-tree rooted at c do
12:            c_i.G ← c.G
13:            if Verify(c_i, R) = False then
14:                H.push(c_i)
15: Return R

16: Routine Update(c)
17: for each G in c.G do
18:     V ← ConvexHull(G)
19:     if ∩_{v_j ∈ V} H_{v_j} ∩ c = c then                    ▷ Lemma 3
20:         c.in ← c.in ∪ G
21:         c.G ← c.G − {G}
22:     else if ∩_{v_j ∈ V} H̄_{v_j} ∩ c = c then               ▷ Lemma 4
23:         c.out ← c.out ∪ G
24:         c.G ← c.G − {G}

25: Routine Verify(c, R)
26: if Size(c.in) ≥ m then
27:     R ← R ∪ c; Ignore c onwards                            ▷ early reporting
28:     return True
29: else if |U| − Size(c.out) < m then
30:     Ignore c onwards                                       ▷ early elimination
31:     return True
32: else                                                       ▷ surviving cell
33:     return False

34: Routine InsertGroup(c, G, R)                               ▷ Section 5.2
35: {G^c, G^i, G^e} ← classify G
36: c.in ← c.in ∪ G^c; c.out ← c.out ∪ G^e
37: G ← G − G^e − G^c
38: if Verify(c, R) = False then
39:     V ← ConvexHull(G^i)
40:     G ← G − V
41:     Update the copy of G in c.G
42:     for each v_j in V do
43:         InsertHS(c, H_{v_j})
```

Fast geometric testing: AA needs to perform numerous checks of user groups G against cells c according to Lemmas 3 and 4 (in Lines 19 and 22, respectively). The containment tests involved take non-trivial time. To improve on that, we use a fast test first, before resorting to standard testing, in a *filter-and-refine* fashion.

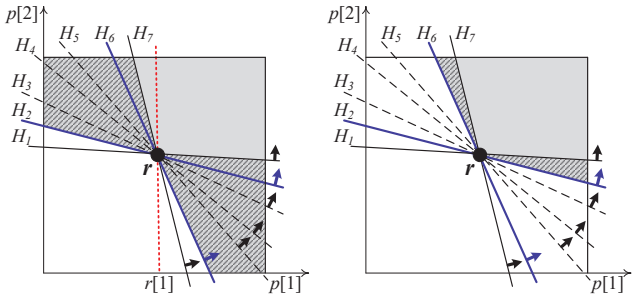
Consider cell c_2 in Figure 5(a), which is checked against group $\{H_1, H_2, H_3\}$. We may identify that Lemma 3 applies (i.e., that c_2 lies inside all halfspaces in the group) with little computational effort, if we compute the *minimum bounding box* (MBB) of c and detect that its min-corner dominates the common point $\boldsymbol{r}$ of the halfspace group. Indeed, since any product $\boldsymbol{p}$ in c dominates the min-corner of its MBB, from the transitivity of dominance, $\boldsymbol{p}$ also dominates $\boldsymbol{r}$, and thus outscores it for any user [14]. The situation is symmetric for c_1. The max-corner of its MBB is dominated by $\boldsymbol{r}$. Hence, no user in the group can be covered by any product in c_1, i.e., they can be directly placed in $c_1.out$. Clearly, when the MBB-based dominance tests are inconclusive, we resort to standard containment tests.

In the same spirit, we apply a similar optimization for Line 35. The classification of users into categories G^c, G^i, and G^e, entails multiple containment tests for a cell against halfspaces. These operations can be expedited by using the MBB of the cell, again in a filter-and-refine manner. In Figure 5(b), for example, c_1 can be



(a) Group testing w.r.t. cells (b) Classifying user groups

Figure 5: Filter-and-refine for fast testing



(a) $m = 2$ $\langle |\mathrm{G}| \geq 2 \cdot m \rangle$ (b) $m = 6$ $\langle m \leq |\mathrm{G}| < 2 \cdot m \rangle$

Figure 6: Specialized InsertGroup() **for** $d = 2$

readily detected as lying outside H_1. The fast test is inconclusive in the case of c_2, necessitating standard containment testing.

Regarding the complexity of AA, in the worst case all its optimizations may fail, thus degenerating to BSL, and sharing its analysis in Lemma 2. That scenario, however, is virtually impossible, with AA demonstrating consistent efficiency and scalability on all standard benchmarks tested in the experiments. Another important remark is that AA is parallelizable in principle, since group insertion is performed independently to each cell (Line 10 in Algorithm 2). That property paves the way for even greater scalability.

5.4 AA in Special Case of d = 2

In the special case of $d = 2$, AA can be significantly improved by optimizations specific to that dimensionality. In Figure 6(a), assume that the current cell c corresponds to the entire product space, and that we are inserting user group $\mathrm{G} = \{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_7\}$, whose top-$k$-th product is $\boldsymbol{r}$. Supposing that $m = 2$, we can visually identify $\mathrm{H}_2 \cup \mathrm{H}_6$ (i.e., the colored area, both striped and otherwise) as the mIR region for G. Moreover, in the remainder region (white area), only $\boldsymbol{w}_1$ and $\boldsymbol{w}_7$ may affect future AA iterations, rendering $\boldsymbol{w}_2, \cdots, \boldsymbol{w}_6$ irrelevant to subsequent processing, and thus safely ignored. Juxtapose this with the general AA in Section 5.3, which would unnecessarily partition the striped area by H_1 and H_7, and fail to eliminate any ineffective users. To simplify presentation, assume that $\boldsymbol{w}_i$ is the user with the i-th largest $w[1]$ value in G.

Lemma 5. *In the $d = 2$ setting, let $\mathbf{w}_i$ and $\mathbf{w}_j$ be user vectors with the same top-k-th product $\mathbf{r}$, and with $w_i[1] < w_j[1]$. In the part of product space where $p[1] \leq r[1]$, it holds that $\mathrm{H}_j \subset \mathrm{H}_i$. Conversely, in the part where $p[1] > r[1]$, it holds that $\mathrm{H}_i \subset \mathrm{H}_j$.*

Proof. The impact halfspace of $\boldsymbol{w}_i$ is defined as

$$
\begin{aligned}
& S_{\boldsymbol{w}_i}(\boldsymbol{p}) = \boldsymbol{w}_i \cdot \boldsymbol{p} \geq S^k_{\boldsymbol{w}_i} = \boldsymbol{w}_i \cdot \boldsymbol{r} \\
\Leftrightarrow & w_i[1]p[1] + w_i[2]p[2] \geq w_i[1]r[1] + w_i[2]r[2] \\
\Leftrightarrow & w_i[1]p[1] + (1 - w_i[1])p[2] \geq w_i[1]r[1] + (1 - w_i[1])r[2] \\
\Leftrightarrow & p[2] \geq \frac{w_i[1]}{1 - w_i[1]}(r[1] - p[1]) + r[2]
\end{aligned}
$$

Similarly, the impact halfspace of $\boldsymbol{w}_j$ is defined as $p[2] \geq \frac{w_j[1]}{1-w_j[1]}(r[1]-p[1])+r[2]$. Letting $\kappa_i = \frac{w_i[1]}{1-w_i[1]}$ and $\kappa_j = \frac{w_j[1]}{1-w_j[1]}$, since $0 < w_i[1] < w_j[1]$, it holds that $0 < \kappa_i < \kappa_j$.

For any $\boldsymbol{p} \in \mathrm{H}_j$ with $p[1] \leq r[1]$, it holds that $p[2] \geq \kappa_j(r[1] - p[1]) + r[2] \Rightarrow p[2] \geq \kappa_i(r[1] - p[1]) + r[2]$, thus $\boldsymbol{p} \in \mathrm{H}_i$. The case for $p[1] > r[1]$ is proven symmetrically. □

To exemplify, consider $\boldsymbol{w}_1$, $\boldsymbol{w}_2$, and their influential halfspaces $\mathrm{H}_1, \mathrm{H}_2$ in Figure 6(a). On the left of the vertical dotted line we have $\mathrm{H}_2 \subset \mathrm{H}_1$, while on the right $\mathrm{H}_1 \subset \mathrm{H}_2$.

Lemma 6. *In the $d = 2$ setting, let G be a group of user vectors (that have a common top-k-th product $\mathbf{r}$), and t be its cardinality. If $t \geq 2 \cdot m$, the* m|R *result for* G *is* $\mathrm{H}_m \cup \mathrm{H}_{t-m+1}$. *Otherwise, if $m \leq t < 2 \cdot m$, the* m|R *result for* G *is* $\mathrm{H}_m \cap \mathrm{H}_{t-m+1}$.

Proof. In the part of product space where $p[1] \leq r[1]$, Lemma 5 implies that any $\boldsymbol{p} \in \mathrm{H}_m$ is also in $\mathrm{H}_i, \forall i \in [1, m]$, i.e., it covers at least m users. Similarly, in the part where $p[1] > r[1]$, any $\boldsymbol{p} \in \mathrm{H}_{t-m+1}$ is also in $\mathrm{H}_i, \forall i \in [t-m+1, t]$. When $|\mathrm{G}| \geq 2 \cdot m$, the mIR result is $\mathrm{H}_m \cup \mathrm{H}_{t-m+1}$ (case in Figure 6(a)). When $m \leq |\mathrm{G}| < 2 \cdot m$, the mIR result for G is $\mathrm{H}_m \cap \mathrm{H}_{t-m+1}$ (case in Figure 6(b)). □

Having elaborated the first case (i.e., $t \geq 2 \cdot m$) in Figure 6(a), we demonstrate in Figure 6(b) the second case (i.e., $m \leq t < 2 \cdot m$), assuming $m = 6$. By Lemma 6, we deduce that the mIR region for G is $\mathrm{H}_2 \cap \mathrm{H}_6$ (shown colored). That saves the unnecessary consideration of $\boldsymbol{w}_1$, $\boldsymbol{w}_7$, and the partitioning of the colored area by $\mathrm{H}_1, \mathrm{H}_7$.

Benefits: Compared to group insertion in general AA, the specialized version: (i) removes the need to classify G's users into G^c, G^e, G^i; (ii) avoids the convex hull computation for G^i; (iii) reports directly a part of c to the mIR result; and (iv) when $t \geq 2 \cdot m$, eliminates from c.G users $\{\boldsymbol{w}_i | i \in [m, t - m + 1]\}$.

The only modification required in Algorithm 2 is to replace InsertGroup() with the Insert2D() routine in Algorithm 3.

5.5 Extension to Standing Top-k Problems

Extension to CO: Following directly from the definition in Problem 1, the mIR result includes the cost-optimal position $\boldsymbol{p}^*$ for the CO problem. That is, since region R includes all possible positions that cover at least m users from $\mathcal{U}$, we first execute AA to compute R, and use the latter as the constraint in a standard optimization solver. E.g., if the commonly assumed L_2 norm is used as cost function, a quadratic programming solver (e.g., [42]) could be invoked to derive the cost-optimal placement of $\boldsymbol{p}^*$ in R. Note that the current state of

Algorithm 3 Insert2D(c, G)

```
1: t ← Size(G)
2: if t ≥ 2m then                                   ▷ Lemma 6 (first case)
3:     R ← R ∪ (H_m ∪ H_{t−m+1}); c ← c − (H_m ∪ H_{t−m+1})
4:     G ← G − {w_i | i ∈ [m, t − m + 1]}
5:     Update the copy of G in c.G
6: else
7:     if m ≤ t < 2m then                           ▷ Lemma 6 (second case)
8:         l ← m; r ← t − m + 1
9:     else
10:        l ← 1; r ← t
11:    G ← G − {w_l, w_r}
12:    Update the copy of G in c.G
13:    InsertHS(c, H_1);
14:    InsertHS(c, H_t);
```

the art for CO [67] requires the cost function $f(\boldsymbol{p}^*)$ to be monotone and convex, yet our *m*IR adaptation extends to more general cost functions, subject to the (orthogonal to our work) availability of an efficient optimization solver to compute the cost-optimal $\boldsymbol{p}^*$ in R.

Extension to IS: In IS, we need to compute the new position $\boldsymbol{p}'$ that an existing product $\boldsymbol{p}$ should assume in product space, so that it covers as many users from $\mathcal{U}$ as possible, subject to the upgrade cost $f(\boldsymbol{p}' - \boldsymbol{p})$ not exceeding a given budget B. As in the original paper [66] (and also typically in practice), suppose that $f()$ is monotone and convex. Since the problem considers strictly upgrades, i.e., enhancement of attribute values, we constrain search to the part of the product space that dominates $\boldsymbol{p}$. In the inner workings of AA, we firstly modify the cell processing order to prioritize by Size($c.in$), but also we directly eliminate any encountered cells whose MBB's min-corner corresponds to an $f()$ value that exceeds B. When a cell c is chosen for processing (Line 6 in Algorithm 2), we compute the minimum value of $f()$ in c itself (not its MBB), and only process c if it is within budget. During the process, we deem a cell *finalized* if its $c.\mathcal{G}$ list becomes empty. Among finalized cells, we maintain as interim result the one with the maximum Size($c.in$) so far, and use it for early elimination of cells whose *out* lists are too large to cover more users than the interim result. The process terminates when there are no more unfinalized surviving cells.

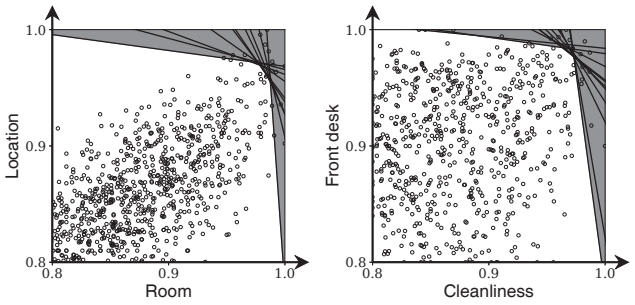
Crossbreed problems: Whether it is a new product to be created or an existing to be improved, and whether there is a specific coverage target m or a budget B, are two orthogonal matters in problem formulation. For example, we may define the *budgeted* CO problem, where we wish to create a product with maximum coverage, subject to a given budget B. Similarly, in the *thresholded* IS problem (which was defined but, again, only heuristically solved in [66]), we wish to identify the cost-optimal improvement for an existing product, so that its upgraded version covers at least m users. Our *m*IR framework extends to these variants too, with modifications very similar to those for the original IS and CO, respectively.

6 EXPERIMENTAL EVALUATION

We start this experimental section with setup description. In Section 6.1, we evaluate our *m*IR solution on realistic/real datasets. In Section 6.2, we present the main performance evaluation, examining robustness to various parameters using common benchmark datasets. In Section 6.3, we consider CO, IS, and their crossbreeds. Finally, in Section 6.4, we assess the effectiveness of optimizations within AA.

Parameter	Tested values
Number of products $\lvert\mathcal{P}\rvert$	0.1M, 0.5M, **1.0M**, 1.5M, 2.0M
Dimensionality d	2, 3, **4**, 5, 6, 7
Number of users $\lvert\mathcal{U}\rvert$	1K, 5K, **10K**, 100K, 1M
Value k	1, 5, **10**, 20, 40 ,80
Value m ($\times\lvert\mathcal{U}\rvert$)	0.001, 0.01, 0.1, 0.3, **0.5** 0.7, 0.9

Table 2: Experiment parameters



(a) Room-location product space (b) Cleanliness-front desk product space

Figure 7: TripAdvisor case study

We run all experiments on a machine with Intel Xeon Gold 5122, 3.6GHz CPU on CentOS. The algorithms are implemented in C++, and complied by GCC with Level 2 optimization. We use lp_solve [3] and qhull [6] for basic geometric operations. Table 2 summarizes the problem parameters, and their tested values. In each experiment, we vary one parameter and set the rest to their defaults (in bold). Unless otherwise specified, all users have the same k.

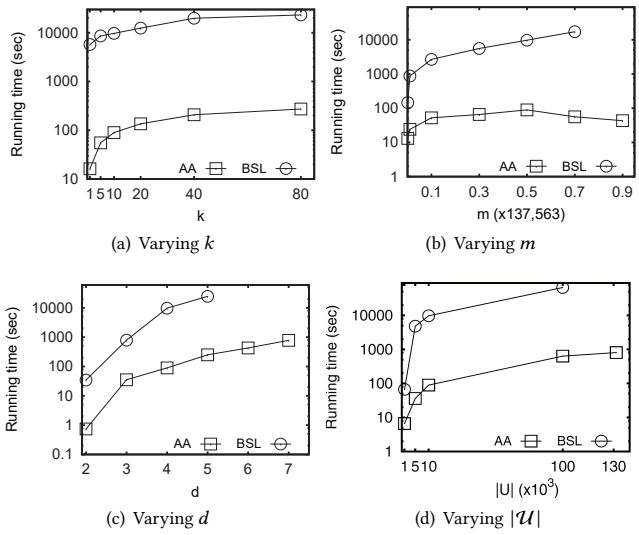
6.1 Real and Realistic Datasets

TripAdvisor data: As explained in Introduction, we use real TripAdvisor data (TA), available at [7]. These data fit naturally our setting, and help assess practicality and performance in as realistic a testbed as possible. Furthermore, they demonstrate the availability of product sets $\mathcal{P}$ in practice, and provide an example of how user preferences $\mathcal{U}$ can be realistically extracted from product reviews.

Regarding the product set, TA includes 7 ratings per hotel, on value, room, location, cleanliness, front desk, service, and business service. Regarding the user set, TA includes actual reviews of these hotels, each comprising a comment and an overall score. To extract user vectors, we employ the well-regarded method of [61], which estimates the per-aspect weights of each user based on her reviews. TA includes 137,563 user vectors in $\mathcal{U}$, and 1,850 hotels in $\mathcal{P}$.

Before any performance investigations, we visualize region R in a case study. We use $d = 2$ of the available attributes in TA each time, and the default $k = 10$ and $m = 0.5$. Figure 7(a) shows the *m*IR output (shaded) and its defining influential halfplanes, when executed in the room-location space. Similarly, Figure 7(b) processes *m*IR in the cleanliness-front desk space. The figures are truncated to the $[0.8, 1]^2$ area for clearer visualization, and they also demonstrate the hotels in that area. In the room-location space there is a stronger correlation between the attributes (compared to cleanliness-front desk), which leads to a larger R, and to fewer hotels inside that R.

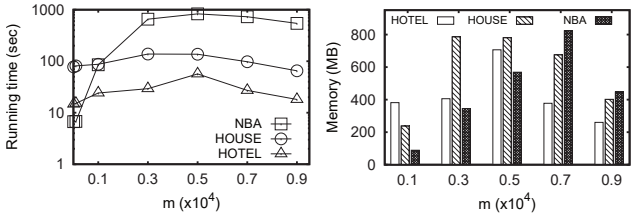
Turning to performance, Figure 8 presents the running time of AA and BSL on TA, using the default values and testing ranges for k, m, and d from Table 2. Regarding $\lvert\mathcal{U}\rvert$, we simulate its different values by taking random samples of the entire user set, from 1K up



(a) Varying k (b) Varying m

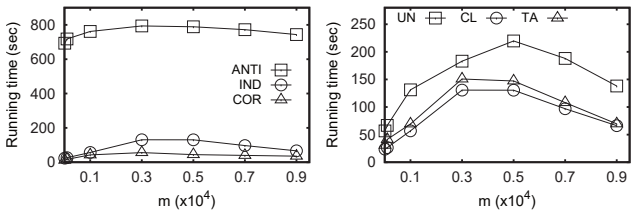
(c) Varying d (d) Varying $|\mathcal{U}|$

Figure 8: TripAdvisor product and user sets



(a) Running time (b) Memory consumption

Figure 9: Results on real product sets



(a) Different product distributions (b) Varying m

Figure 10: Effect of product distribution and m

to its full cardinality; by default, the complete $\mathcal{U}$ is used. NVE is excluded as it is too inefficient. Some measurements for BSL are missing, because we force stop it when it exceeds 10 hours. We leave the explanation of the trends for Section 6.2. Here, we evince the practicality of AA's response time in our most realistic testbed.

Additionally, we demonstrate the inefficiency of BSL, which is 2 to 3 orders of magnitude slower than AA. BSL fails to capitalize effectively on early reporting/elimination, because it processes users individually and in random order, thus inserting more half-spaces. To elaborate, although both algorithms output the same region R, they report it as the union of multiple cells. In the default setting, BSL breaks R to 3 thousand times more cells. Due to its ineffectiveness, we exclude BSL from subsequent experiments.

Real product sets: Next, we experiment on three product sets used very commonly in the top-k literature. HOTEL holds 418,843 hotel records with $d = 4$ attributes, such as stars, price, etc [1]. HOUSE contains 315,265 records of household expenditures on $d = 6$ aspects [2]. NBA includes $d = 8$ statistics for 21,960 NBA players [4]. In lack of real preferences for them, we follow standard practice (e.g., [57, 67]) to generate *Clustered* (CL) user sets. CL models the practical scenario where many users share similar preferences; user vectors form 5 Gaussian clusters of equal size, with variance 0.05^2.

Figure 9 reports the running time and memory consumption of AA on these data. The time and space requirements of AA demonstrate its practicality and general applicability. Processing in NBA is faster than HOTEL and HOUSE for small m, but slower for larger m. When m is small, the mIR problem is simpler, and NBA's smaller cardinality matters. However, for tougher problem instances, the effect of its higher dimensionality outweighs that of its cardinality.

6.2 Main Performance Evaluation

In this section, we use synthetic product sets of typical distributions in multi-criteria decisions [14], i.e., *Independent* (IND), *Correlated* (COR), and *Anti-correlated* (ANTI). As user sets, we employ CL, the user set of TA, and uniform (UN). The default product set is IND and the default user set is CL.

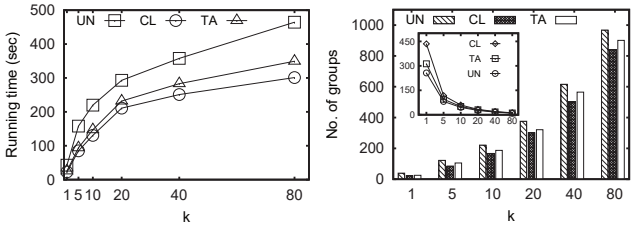
Product distribution: Figure 10(a) considers product sets of different distributions, and reports the running time of AA as a function of m (using the default CL user set). Processing is the fastest in COR, and the slowest in ANTI. In COR, a product with a large value in one attribute, tends to have large values in the others too. In ANTI, a large value in one attribute typically implies small values in the rest. This directly affects the performance of AA, and especially the effectiveness of its grouping; there are 95, 167, and 365 groups for COR, IND, and ANTI, respectively.

Next, we use IND products, and test AA for CL, TA, and UN users, varying one problem parameter each time. For TA users, we use random samples of the required cardinality in each setting.

Varying m: Figure 10(b) shows that AA takes the longest for mid-range m values, as expected. AA is more efficient in CL, because clustered users are more likely to have the same top-k-th product, thus forming fewer/larger groups, with AA benefiting the most from its group-based optimizations. In contrast, UN users are the most likely to have different top-k-th products. On the average, in CL and UN there are 167 vs. 221 groups, containing 60 vs. 45 users.

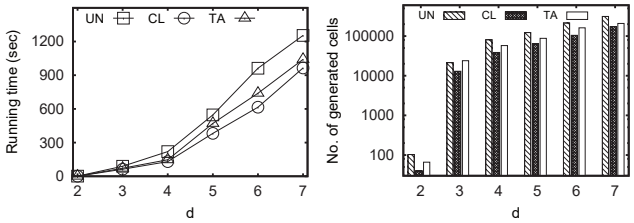
Varying k: Figure 11(a) illustrates that the running time of AA increases with k. As k grows, it becomes less likely for users to have a common top-k-th product, thus the number of groups increases which, in turn, affects performance. To verify this, in Figure 11(b) we report the average number of groups and, in the embedded chart, the average group size. The results validate our intuition and the analysis in the beginning of Section 5.1.

Varying d: Figure 12(a) shows that the running time increases with d, which is common for problems of geometric nature; as we demonstrate in Figure 12(b), the total number of cells generated in the arrangement grows quickly with d. That said, performance remains acceptable, e.g., for CL with $d = 7$ we need 964.4s. We note that score-based ranking, including the traditional top-k query, generally loses its meaning for more than 5-6 dimensions, because the scores of all competitors (products) converge to the same value [44, 72].



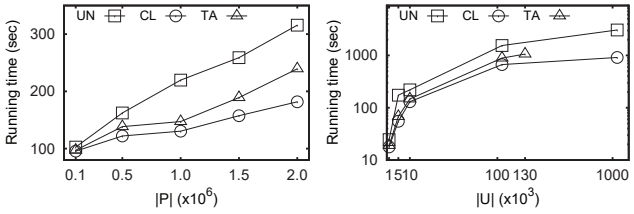
(a) Running time (b) No. of groups

Figure 11: Effect of k



(a) Running time (b) No. of generated cells

Figure 12: Effect of d



(a) Varying $|\mathcal{P}|$ (b) Varying $|\mathcal{U}|$

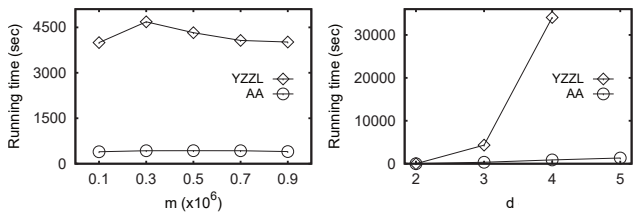
Figure 13: Effect of $|\mathcal{P}|$ and $|\mathcal{U}|$

Varying $|\mathcal{P}|$ and $|\mathcal{U}|$: Figures 13(a) and 13(b) investigate the effect of $|\mathcal{P}|$ and $|\mathcal{U}|$, respectively. AA scales well with both cardinalities. E.g., for $|\mathcal{P}|$ = 1M and 2M in CL, it takes 130.5s and 181.8s, respectively. $|\mathcal{U}|$ has a more pronounced effect on performance, because more users imply more influential halfspaces, and hence a higher computational geometric load. Note that the TA user set includes 137,563 vectors, thus its line stops earlier in Figure 13(b).

6.3 CO, IS, and Crossbreed Problems

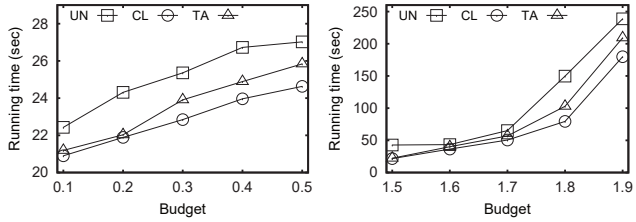
In this section, we apply AA to previous top-k influence problems.

Influence-based cost optimization (CO): We compare against the exact CO algorithm by Yang et al. [67] (denoted as YZZL from their initials), which applies specifically to $k = 1$. To be as fair as possible, we use their code and their default experimental setup. They randomly selected $d = 3$ dimensions from HOUSE to form $\mathcal{P}$, and used CL with 1M user vectors as $\mathcal{U}$. Figure 14(a) compares YZZL with the AA adaptation for CO for their tested m values. Our approach is 9.5 times to 10.9 times faster. The main reason for this difference is the high complexity of their k-level building block [45], which is a $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ module. In Figure 14(b), we repeat the experiment of [67] on dimensionality. As also noted in that work, YZZL fails to terminate within a day for $d = 5$. That is reasonable, given the exponential complexity of the k-level module in d. The adapted AA, on the other hand, scales without issues.



(a) Varying m (b) Varying d

Figure 14: Adaptation to CO problem



(a) IS (b) Budgeted CO

Figure 15: Adaptation to IS and to budgeted CO

Improvement strategies (IS): Next, we consider the IS problem. Since the only previous method [66] is heuristic, we focus on our exact mIR-based approach, and aim to demonstrate its practicality. Similarly to [25, 66], we model the upgrade cost by the L_2 norm, i.e., the Euclidean distance between $\boldsymbol{p}$ and its upgraded version $\boldsymbol{p'}$. In Figure 15(a), we choose at random the product $\boldsymbol{p}$ to be improved, and vary the available budget B. We plot the running time of AA for CL, TA, and UN user sets, on IND products in our default setting. AA produces exact IS solutions, requiring at most 27s in all cases. We deem that performance practical for real applications, especially since the past attempts on IS resorted to heuristics, due to its complexity.

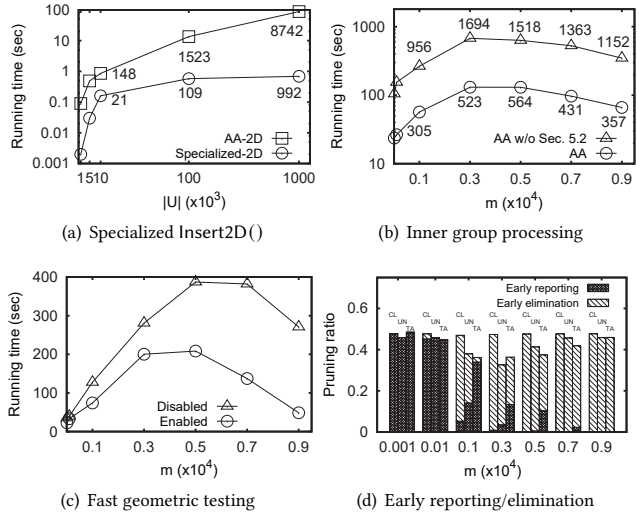
Crossbreeds: In Figure 15(b), we consider the *budgeted* CO, where a maximum-coverage product $\boldsymbol{p}$ is to be created, subject to a budget B on creation cost. We model the cost by the L_2 norm, i.e., the square root of the summed squared values in $\boldsymbol{p}$, and plot the running time of the adapted AA for CL, TA, and UN users, on IND products. The top corner of the product space (i.e., the ultimate product) has creation cost 2, thus we test up to $B = 1.9$ which is close to that extreme. AA performs well in all cases, e.g., for the largest budget $B = 1.9$, it takes 179s, 209s, and 238s for CL, TA, and UN, respectively.

Regarding *thresholded* IS, i.e., the second crossbreed in Section 5.5, our adaptation first processes mIR and then invokes a convex optimization solver to derive the cost-optimal $\boldsymbol{p'}$ in R. The latter step adds insignificant overhead, yielding a very similar overall performance to the experiments on vanilla mIR, in Sections 6.1 and 6.2.

6.4 Effectiveness of Optimizations

In this section, we investigate the effect of several optimizations. By default, we use CL users and IND products.

In Figure 16(a), we compare the specialized with the generic AA in $d = 2$, by varying $|\mathcal{U}|$. The specialized AA for $d = 2$ is one to two orders of magnitude faster. E.g., it achieves a 128-fold improvement for $|\mathcal{U}|$ = 1M. Next to selected points in the chart, we also report



(a) Specialized Insert2D() (b) Inner group processing

(c) Fast geometric testing (d) Early reporting/elimination

Figure 16: Effectiveness of optimizations

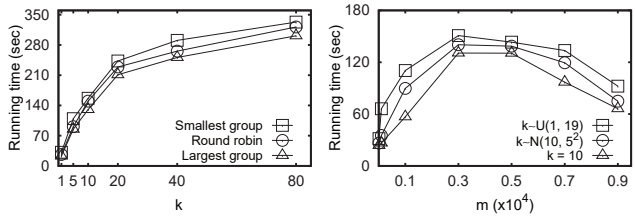
the number of generated cells. For better visualization, we omit that number for $|\mathcal{U}| < 10$K, but the reduction is drastic for those values too, i.e., the specialized AA produces 13 vs. 57 cells for $|\mathcal{U}| = 1$K, and 15 vs. 113 for $|\mathcal{U}| = 5$K. The results attest to the significance of the four benefits we elaborated at the end of Section 5.4.

Next, we assess the power of the inner group processing technique in Section 5.2. Recall that AA could determine a cell c as completely covering or not covering a user group G in some cases, but that is not enough (and not all AA does). The major bottleneck still regards the undecided cases (like c_3 in Figure 3), which require the consideration of individual halfspaces within G. For that task, we developed the inner group processing technique. In Figure 16(b), we compare AA with and without this enhancement, revealing that it is responsible for a 4.4 to 5.8 times speedup. To offer more insight, reducing the number of containment tests was one of the main motivations in Section 5.2. Therefore, next to selected points in Figure 16(b), we report the number of containment tests performed, demonstrating a 2.7 to 3.2 times reduction.

In Figure 16(c), we assess the effectiveness of the fast geometric testing described in Section 5.3. In particular, we run AA with and without it, while varying m. The running time comparison reveals that this lightweight filter-and-refine optimization offers AA a speedup of up to 5.6 times.

In Figure 16(d), we investigate the power of early reporting and early elimination. Specifically, we vary m and plot the ratio of produced cells that are reported or eliminated early. This ratio is between 46% and 48% for CL, between 36% and 49% for TA, and between 33% and 46% for UN. We also present the breakdown of the ratio into early reporting and early elimination. When m is small, a cell is easier to report early. Conversely, when m is large, a cell is harder to report, but easier to eliminate early. Hence, as m grows, the effectiveness of early reporting gives way to the effectiveness of early elimination. As a result, they complement each other, delivering a combined pruning that is consistently strong across the board.

Next, we consider choices related to grouping, and assess its effectiveness in more general scenarios. Having chosen a cell c



(a) Group selection strategy (b) Users with diverse k values

Figure 17: Grouping-related investigation

to process, AA inserts to it the largest group in $c.\mathcal{G}$ (Line 9 in Algorithm 2). The rationale is that the larger the group, the more aggressively it can push c towards early reporting or early elimination. In Figure 17(a), we consider AA variants which choose for insertion (i) the smallest group in $c.\mathcal{G}$, or (ii) the next group in a (randomly initialized) round robin order. Our strategy is the fastest. Round robin is the runner up, being 6% to 17% slower. The smallest group strategy, which is the exact opposite of ours, is the slowest. The results justify our choice.

Due to the need for a common top-k-th product, diverse k values among users may affect the effectiveness of grouping and, in turn, the performance of AA. In Figure 17(b), we compare the running time of AA when (i) $k = 10$ for all users, (ii) k for different users is drawn uniformly from range $[1, 20)$, and (iii) k for different users is drawn from a normal distribution with mean 10 and standard deviation 5. We observe that AA's performance is only moderately affected by the diversity in k, and infer that the effectiveness of grouping is rather robust to users with individual k values.

7 CONCLUSION

In this paper, we introduce the mIR problem, which identifies the region where any product would be in the top-k for a desired fraction of a user population. mIR finds application in understanding preference/market dynamics, effective marketing, product improvement, etc. In addition to its direct applications, mIR can be adapted to solve exactly some practical top-k influence problems that were previously either not completely resolved, or only heuristically addressed. We develop an algorithmic mIR framework, which offers exact answers and practical response times, both for mIR and for the previous problems. A direction for future work is to consider highly dynamic user sets (e.g., users currently online) for applications such as online advertising. Incremental result maintenance and approximation are likely avenues to deal with the real-time nature of these applications.

ACKNOWLEDGMENTS

Bo Tang and Mingji Han were supported by the National Science Foundation of China (NSFC No. 61802163), the Education Department of Guangdong (Grant No. 2020KZDZX1184), and the Guangdong Provincial Key Laboratory (Grant No. 2020B121201001). Kyriakos Mouratidis was supported by the Singapore Management University Lee Kong Chian Fellowship. The authors would like to thank professors Jing Jiang and Hady Lauw of Singapore Management University for discussions and pointers to the literature on preference learning.

REFERENCES

[1] Hotel dataset. http://www.hotels-base.com.
[2] House dataset. http://www.ipums.org.
[3] lpsolver. http://lpsolve.sourceforge.net/5.5/.
[4] NBA dataset. http://www.basketball-reference.com.
[5] qhalf. http://www.qhull.org/html/qhalf.htm.
[6] qhull. http://www.qhull.org.
[7] TripAdvisor Data Set. http://www.cs.virginia.edu/~hw5x/dataset.html.
[8] P. K. Agarwal and M. Sharir. Arrangements and their applications. *Handbook of computational geometry*, pages 49–119, 2000.
[9] A. Asudeh, A. Nazi, N. Koudas, and G. Das. Maximizing gain over flexible attributes in peer to peer marketplaces. In *PAKDD*, pages 327–345, 2019.
[10] A. Asudeh, A. Nazi, N. Zhang, and G. Das. Efficient computation of regret-ratio minimizing set: A compact maxima representative. In *SIGMOD Conference*, pages 821–834, 2017.
[11] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.
[12] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer-Verlag TELOS, 2008.
[13] A. Blum, J. C. Jackson, T. Sandholm, and M. Zinkevich. Preference elicitation and query learning. *J. Mach. Learn. Res.*, 5:649–667, 2004.
[14] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
[15] Y. Cai, Y. Tang, and N. Mamoulis. Maximizing a record's standing in a relation. *IEEE Trans. Knowl. Data Eng.*, 27(9):2401–2414, 2015.
[16] T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discret. Comput. Geom.*, 16(4):369–387, 1996.
[17] Y.-C. Chang, L. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith. The onion technique: Indexing for linear optimization queries. In *SIGMOD Conference*, pages 391–402, 2000.
[18] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10(4):377–409, 1993.
[19] M. A. Cheema, Z. Shen, X. Lin, and W. Zhang. A unified framework for efficiently processing ranking related queries. In *EDBT*, pages 427–438, 2014.
[20] P. Ciaccia and D. Martinenghi. Reconciling skyline and ranking queries. *PVLDB*, 10(11):1454–1465, 2017.
[21] K. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Computational Geometry*, 3(4):185–212, 1993.
[22] M. Das, G. Das, and V. Hristidis. Leveraging collaborative tagging for web item design. In *KDD*, pages 538–546, 2011.
[23] H. Edelsbrunner, R. Seidel, and M. Sharir. On the zone theorem for hyperplane arrangements. *SIAM J. Comput.*, 22(2):418–429, 1993.
[24] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, page 102–113, 2001.
[25] Y. Gao, Q. Liu, G. Chen, B. Zheng, and L. Zhou. Answering why-not questions on reverse top-k queries. *PVLDB*, 8(7):738–749, 2015.
[26] S. Ge, L. H. U, N. Mamoulis, and D. W. Cheung. Efficient all top-k computation - A unified solution for all top-k, reverse top-k and top-m influential queries. *IEEE Trans. Knowl. Data Eng.*, 25(5):1015–1027, 2013.
[27] S. Ge, L. H. U, N. Mamoulis, and D. W. Cheung. Dominance relationship analysis with budget constraints. *Knowl. Inf. Syst.*, 42(2):409–440, 2015.
[28] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDB J.*, 16(1):5–28, 2007.
[29] V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A system for the efficient execution of multi-parametric ranked queries. In *SIGMOD Conference*, pages 259–270, 2001.
[30] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comp. Surveys*, 40(4):11:1–11:58, 2008.
[31] M. S. Islam and C. Liu. Know your customer: computing k-most promising products for targeted marketing. *VLDB J.*, 25(4):545–570, 2016.
[32] K. G. Jamieson and R. D. Nowak. Active ranking using pairwise comparisons. In *NIPS*, pages 2240–2248, 2011.
[33] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142, 2002.
[34] J. Koh, C. Lin, and A. L. P. Chen. Finding k most favorite products based on reverse top-t queries. *VLDB J.*, 23(4):541–564, 2014.
[35] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD Conference*, pages 201–212, 2000.
[36] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang. DADA: a data cube for dominant relationship analysis. In *SIGMOD Conference*, pages 659–670, 2006.
[37] C. Lin, J. Koh, and A. L. P. Chen. Determining k-most demanding products with maximum expected number of total customers. *IEEE Trans. Knowl. Data Eng.*, 25(8):1732–1747, 2013.
[38] H. Lu and C. S. Jensen. Upgrading uncompetitive products economically. In *ICDE*, pages 977–988, 2012.
[39] Y. Lu, C. Zhai, and N. Sundaresan. Rated aspect summarization of short comments. In *WWW*, pages 131–140, 2009.
[40] N. Mamoulis, M. L. Yiu, K. H. Cheng, and D. W. Cheung. Efficient top-k aggregation of ranked inputs. *ACM Trans. Database Syst.*, 32(3):19, 2007.
[41] M. Miah, G. Das, V. Hristidis, and H. Mannila. Standing out in a crowd: Selecting attributes for maximum visibility. In *ICDE*, pages 356–365, 2008.
[42] R. D. C. Monteiro and I. Adler. Interior path following primal-dual algorithms. part II: convex quadratic programming. *Math. Program.*, 44(1-3):43–66, 1989.
[43] K. Mouratidis and B. Tang. Exact processing of uncertain top-k queries in multi-criteria settings. *PVLDB*, 11(8):866–879, 2018.
[44] K. Mouratidis, J. Zhang, and H. Pang. Maximum rank query. *PVLDB*, 8(12):1554–1565, 2015.
[45] K. Mulmuley. On levels in arrangements and voronoi diagrams. *Discrete & Computational Geometry*, 6:307–338, 1991.
[46] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. J. Xu. Regret-minimizing representative databases. *PVLDB*, 3(1):1114–1124, 2010.
[47] V. Padmanabhan, S. Rajiv, and K. Srinivasan. New products, upgrades, and new releases: A rationale for sequential product introduction. *Journal of Marketing Research*, 34(4):456–472, 1997.
[48] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.
[49] Y. Peng, R. C. Wong, and Q. Wan. Finding top-k preferable products. *IEEE Trans. Knowl. Data Eng.*, 24(10):1774–1788, 2012.
[50] A. Popescu and O. Etzioni. Extracting product features and opinions from reviews. In *HLT/EMNLP*, pages 339–346, 2005.
[51] L. Qian, J. Gao, and H. V. Jagadish. Learning user preferences by adaptive pairwise comparison. *PVLDB*, 8(11):1322–1333, 2015.
[52] B. Tang, K. Mouratidis, and M. L. Yiu. Determining the impact regions of competing options in preference space. In *SIGMOD Conference*, pages 805–820, 2017.
[53] B. Tang, K. Mouratidis, M. L. Yiu, and Z. Chen. Creating top ranking options in the continuous option and preference space. *PVLDB*, 12(10):1181–1194, 2019.
[54] Y. Tao, V. Hristidis, D. Papadias, and Y. Papakonstantinou. Branch-and-bound processing of ranked queries. *Inf. Syst.*, 32(3):424–445, 2007.
[55] Y. Tao, D. Papadias, X. Lian, and X. Xiao. Multidimensional reverse k NN search. *VLDB J.*, 16(3):293–316, 2007.
[56] Y. Tao, X. Xiao, and J. Pei. Efficient skyline and top-k retrieval in subspaces. *IEEE Trans. Knowl. Data Eng.*, 19(8):1072–1088, 2007.
[57] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Reverse top-k queries. In *ICDE*, pages 365–376, 2010.
[58] A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis. Identifying the most influential data objects with reverse top-k queries. *PVLDB*, 3(1):364–372, 2010.
[59] A. Vlachou, C. Doulkeridis, K. Norvag, and Y. Kotidis. Branch-and-bound algorithm for reverse top-k queries. In *SIGMOD Conference*, pages 481–492, 2013.
[60] Q. Wan, R. C. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng. Creating competitive products. *PVLDB*, 2(1):898–909, 2009.
[61] H. Wang, Y. Lu, and C. Zhai. Latent aspect rating analysis on review text data: a rating regression approach. In *KDD*, pages 783–792, 2010.
[62] R. C. Wong, M. T. Özsu, A. W. Fu, P. S. Yu, L. Liu, and Y. Liu. Maximizing bichromatic reverse nearest neighbor for Lp -norm in two- and three-dimensional spaces. *VLDB J.*, 20(6):893–919, 2011.
[63] X. Wu, Y. Tao, R. C. Wong, L. Ding, and J. X. Yu. Finding the influence set through skylines. In *EDBT*, pages 1030–1041, 2009.
[64] M. Xie, R. C. Wong, and A. Lall. 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.
[65] M. Xie, R. C. Wong, J. Li, C. Long, and A. Lall. Efficient k-regret query algorithm with restriction-free bound for any dimensionality. In *SIGMOD Conference*, pages 959–974, 2018.
[66] G. Yang and Y. Cai. Querying improvement strategies. In *EDBT*, pages 294–305, 2017.
[67] J. Yang, Y. Zhang, W. Zhang, and X. Lin. Influence based cost optimization on user preference. In *ICDE*, pages 709–720, 2016.
[68] J. Yang, Y. Zhang, W. Zhang, and X. Lin. Cost optimization based on influence and user preference. *Knowl. Inf. Syst.*, 61(2):695–732, 2019.
[69] S. Yang, M. A. Cheema, X. Lin, and Y. Zhang. SLICE: reviving regions-based pruning for reverse k nearest neighbors queries. In *ICDE*, pages 760–771, 2014.
[70] M. L. Yiu and N. Mamoulis. Multi-dimensional top-k dominating queries. *VLDB J.*, 18(3):695–718, 2009.
[71] A. Yu, P. K. Agarwal, and J. Yang. Processing a large number of continuous preference top-k queries. In *SIGMOD Conference*, pages 397–408, 2012.
[72] A. Yu, P. K. Agarwal, and J. Yang. Top-k preferences in high dimensions. *IEEE Trans. Knowl. Data Eng.*, 28(2):311–325, 2016.
[73] J. Zhang, K. Mouratidis, and H. Pang. Direct neighbor search. *Inf. Syst.*, 44:73–92, 2014.
[74] Z. Zhang, C. Jin, and Q. Kang. Reverse k-ranks query. *PVLDB*, 7(10):785–796, 2014.
[75] Z. Zhou, W. Wu, X. Li, M. Lee, and W. Hsu. MaxFirst for MaxBRkNN. In *ICDE*, pages 828–839, 2011.