Finding Diverse, High-Value Representatives on a Surface of Answers *

You Wu[‡†] Junyang Gao[†] Pankaj K. Agarwal[†] Jun Yang[†]
[‡]Google Research, [†]Duke University

wuyou@google.com {jygao,pankaj,junyang}@cs.duke.edu

ABSTRACT

In many applications, the system needs to selectively present a small subset of answers to users. The set of all possible answers can be seen as an elevation surface over a domain, where the elevation measures the quality of each answer, and the dimensions of the domain correspond to attributes of the answers with which similarity between answers can be measured. This paper considers the problem of finding a diverse set of k high-quality representatives for such a surface. We show that existing methods for diversified top-k and weighted clustering problems are inadequate for this problem. We propose k-DHR as a better formulation for the problem. We show that k-DHR has a submodular and monotone objective function, and we develop efficient algorithms for solving k-DHR with provable guarantees. We conduct extensive experiments to demonstrate the usefulness of the results produced by k-DHR for applications in computational lead-finding and fact-checking, as well as the efficiency and effectiveness of our algorithms.

1 Introduction

The problem of returning a selective subset of answers to users has been studied by many research communities in many contexts, such as recommender systems [18, 6] and top-k queries with diversification [2, 3, 14]. For many applications, the answers can be modeled as points on an elevation surface over an underlying domain. The elevation captures an answer's quality or relevance to the query, while the dimensions of underlying domain correspond to various answer attributes by which we can measure the similarities among answers. Most of the time, users are interested in examining only a few answers. Naturally, we want to select those answers with high quality. Second, we want the selected answers to be diverse, because users learn less from answers that are similar (close to each other in the underlying domain). Finally, in many scenarios, we further want the selected answers to be "representative," and avoid outliers for which close-by answers have far lower qualities. In other words, we would like to use a small number of high-value points to represent the high-value regions of the surface.

To see why these criteria are important, consider the following example from *computational lead-finding and fact-checking* [16], a novel application in journalism where we analyze data to find interesting leads, or counterarguments to claims that may be factually correct but still misleading (e.g., due to cherry-picking). Here, the surface is formed by a large number of factual claims that one can generate from a dataset.

Example 1 (Revisiting Giuliani's Adoption Claim). *A real-life example of "correct but still misleading" claim considered in [16] was Giuliani's claim in 2007 that "adoptions went up 65 to 70 percent" in the New York City "when he was the mayor." An effective way of countering such a claim, used frequently in practice by journalists and professional fact-checkers, is to find claims of the same form, which, with slightly different views of the same data (time series of New York City adoption totals in this case), lead to much weaker or even opposite conclusions. For example, one such counterargument produced by the algorithm in [16] is:*

> *(CA-1) Comparing Giuliani's first and second 4-year terms, i.e., 1994-1997 and 1998-2001, leads to 1% decrease.*

Here, each possible counterargument is characterized by three attributes: length of the two windows being compared, distance between the windows, and the position (starting point) of the second window. The quality of each counterargument is measured by a combination of how much it weakens the original claim's conclusion (e.g., from "up 65 to 70 percent" to "1% decrease"), and how natural and contextually relevant it is to the original claim (e.g., how well the two windows of comparison reflect "before" and "after" Giuliani's contribution). Hence, the set of all possible counterarguments can be seen as an elevation surface over a 3-d domain.

Many counterarguments are not directly comparable to each other. Even if we return only Pareto-optimal counterarguments as in [16], there are too many answers, e.g., just to list a few:

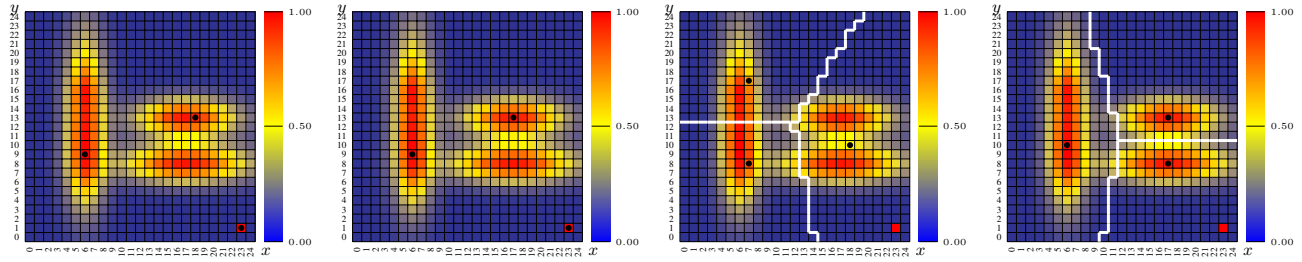
> *(CA-2) Comparing the two 4-year windows of 1995-1998 and 1999-2002, leads to 19% decrease.*
>
> *(CA-3) Comparing the two 2-year windows of 1996-1997 and 2000-2001, leads to 24% decrease.*
>
> *(CA-4) Comparing the last year of Giuliani's first and second half of his term, i.e., 1997 and 2001, leads to 32% decrease.*

One obvious problem is that these answers are not too different from each other. For example, the 8 years covered by (CA-1) and the 8 years covered by (CA-2) share 6 years in common.

*Work on this paper is supported by NSF grants CCF-15-13816, CCF-15-46392, IIS-14-08846, and IIS-13-20357, by ARO grant W911NF-15-1-0408, by grant 2012/229 from the U.S.-Israel Binational Science Foundation, and a Google Faculty Research Award. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the funding agencies.

†Most of the work was conducted when the author was at Duke University.

Proceedings of the VLDB Endowment, Vol. 10, No. 7



(a) MMR (b) Graph-based diversification (c) Weighted k-means (d) k-DHR

Figure 1: Comparison of 3 answers chosen on a synthetic piece-wise constant surface by different methods. Elevation (value) is represented by color. Chosen answers are marked with black dots.

Another interesting problem is that counterarguments themselves can be countered. For example, the following claim is a counterargument to (CA-1), and vice versa:

> *(CA-5) Comparing second half of Giuliani's term against the 4 years preceeding his term, the adoptions increased by 99.34%.*

Hence, professional fact-checkers carefully avoid selecting outliers as counterarguments. Even if a counterargument is highly effective by itself, it will be considered nitpicking if most of its neighbors—obtained by slight perturbations of its attributes—are not.

To recap, there are three objectives we want to achieve with our selection of answers from a surface, where the elevation represents answer quality (henceforth also referred to as *value*), and the underlying domain captures similarity among answers:

(*Utility*) Each selected answer should have high value.

(*Diversity*) Selected answers should not be too similar.

(*Representativeness*) Selected answers should represent high-value neighborhoods.

Existing Methods While our problem is related to diversified top-k queries and weighted clustering, existing methods for these problems cannot be readily applied to achieve all three of our objectives above. To illustrate, consider a piecewise-constant surface shown in Figure 1, generated using a mixture of four 2-d Gaussian kernels with similar peak values, plus a single spike at the lower-right corner $(23, 1)$, which has the highest value overall but is located in a low-value region. The high-value (vertically oriented) region on the left is a mixture of two Gaussian kernels with close-by centers, while the two high-value regions (horizontally oriented) on the right corresponds to two separate Gaussian kernels with the same peak value (slightly lower than the peak values of the high-value region on the left). If the goal is to choose $k = 3$ representatives, ideally, we should choose one for each of the three high-value regions, and the lonesome spike should be avoided. Let us see what existing methods will do for this example.

Diversified Top-k We first consider two methods representative of the work on diversified top-k queries: the *maximal marginal relevance (MMR)* method [2], and *graph-based diversification* [14]. MMR iteratively finds the next answer to return that optimizes a linear combination of its value (for utility) and its minimum distance to answers already picked (for diversity). The trade-off is controlled by a parameter $\lambda \in (0, 1]$. Graph-based diversification assumes a distance threshold δ and looks for k answers with the highest total value (for utility) that are at least distance δ away from each other (for diversity). (More details can be found in the technical report version of this paper [17].)

Picking parameters λ for MMR and δ for graph-based diversification is tricky, but with some trials and errors, we end up with our best-effort solutions for MMR and graph-based diversification in Figures 1a and 1b, respectively. Both methods managed to choose answers with high individual value, but both failed to avoid the spike at $(23, 1)$, which should not be surprising because neither methods considers *representativeness* of their answers.

Beyond the lack of representativeness, there are other problem with these methods in how they achieve diversity. For MMR, because diversity (distance) is part of its linear optimization objective function, it tends to choose answers far away from previously chosen ones. In Figure 1a, MMR first picked $(23, 1)$ and then $(6, 9)$. For the third answer, MMR picked $(18, 13)$ over $(17, 13)$, even though the latter has a higher value and is the peak of the surrounding high-value region, simply because of distances to the first two chosen answers. For the same reason, MMR favored picking from the upper one of the two high-value regions on the right, even though the lower region is bigger and peaks at the same value.

For graph-based diversification, there are scenarios where no choice of the distance threshold δ can lead to desired result. Let us ignore the distraction of outliers by removing the spike at $(23, 1)$ (Figure 2). Again, if we are asked to choose 3 answers, it is best to pick the centers of the three high-value regions. However, as we will show below, no choice of δ would allow graph-based diversification to pick the two peaks on the right, located at $(17, 8)$ and $(17, 13)$, simultaneously. The problem is that there exists two candidates $(6, 9)$ and $(6, 16)$ from the same high-value region on the left, whose values are slightly higher than $(17, 8)$ and $(17, 13)$ and whose distance is longer than that between $(17, 8)$ and $(17, 13)$. If we set δ to be small enough such that $(17, 8)$ and $(17, 13)$ are not mutually exclusive as candidates, graph-based diversification will still favor picking $(6, 9)$ and $(6, 16)$ simultaneously, hence preventing $(17, 8)$ and $(17, 13)$ to be both chosen among the 3 answers. Basically, even for a relatively simple surface, a single δ cutoff is not sufficient to capture what constitutes "close enough" across different parts of the surface.

Weighted Clustering Another line of work related to our goal is weighted clustering. By considering each answer's value as its weight, weighted clustering algorithms can be used to partition the surface into the desired number of regions and find a representative from each region. Figure 1c shows the result of applying weighted k-means clustering to our example; here, we mark the cluster boundaries using white lines, and we pick the weighted centroid of each cluster (rounded to the nearest integral point) as the answer representing the cluster. Unfortunately, weighted k-means clustering recognizes the two separately high-value regions on the right as a single cluster, while partitioning the skinny high-value region on the left into two clusters. Moreover, while the weighted k-means objective function considers the values of all answers in a cluster, it gives no special consideration to the value of representative for the cluster. Thus, there is no guarantee that the representative has high value. In fact, we see in Figure 1c that k-means

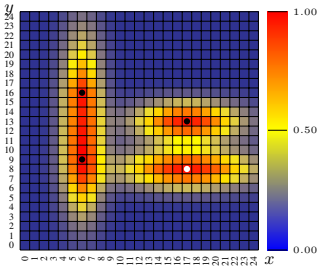


Figure 2: Example where no setting of distance threshold δ for graph-based diversification works; the white dot should have been chosen among the 3 representatives, but can never be chosen.

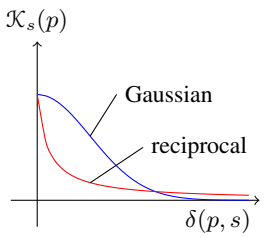


Figure 3: Impact as function of distance, according to Gaussian and reciprocal kernels.

clustering is prone to picking a low-value representative for a cluster wherein peaks are away from the (unweighted) geographical cluster center, because distances from the representative are heavily penalized (squared in the objective function).

In summary, for the three objectives introduced earlier, diversified top-k methods target *utility* and *diversity*, but not *representativeness*. Weighted clustering methods such as k-means achieves *representativeness* and *diversity* via partitioning, but they do not account for the *utility* of representatives. We also saw limitations of diversity control in MMR (due to fixed linear weighting of distance by λ) and graph-based diversification (due to fixed distance threshold δ), as well as difficulty in setting their parameters. The same limitations would also apply if we add consideration for representativeness to MMR and graph-based diversification in analogous ways, e.g., by defining representativeness over a fixed-radius neighborhood and including it as a separately weighted term in the objective function.

Our Contributions We define the k-DHR problem for finding a ***D***iverse set of k ***H***igh-value ***R***epresentative points from a surface. This new problem formulation accounts for *utility*, *diversity*, and *representativeness* simultaneously. A key feature of our formulation—inspired by clustering—is that it achieves diversity by partitioning the surface and selecting one representative per partition. k-DHR allows users to tune the desired degree of diversity using an *impact kernel*, which, together with clustering, enables k-DHR to adapt to the local surface features much better than methods that rely on weights or thresholds that are inflexible and difficult to set. For example, Figure 1d shows the 3 representatives found by k-DHR, which are intuitively better than the solutions by the three existing methods discussed earlier.

We study techniques for efficiently solving the k-DHR problem. While k-DHR is NP-hard, we show that its optimization objective function is submodular and monotone, allowing us to develop Greedy, a greedy algorithm with $(1 - \frac{1}{e})$-approximation. We propose a number of techniques to further improve Greedy's performance, culminating in an algorithm we call Greedy$^+$. In addition, we present two alternative algorithms, LocalSearch (based on local search) and EM (based on the expectation-maximization technique). These two algorithms provide additional references of comparison; LocalSearch can also be used by other algorithms as a fast post-processing step to ensure local optimality.

Finally, we apply k-DHR to the practical problem of computational lead-finding for journalism, and evaluate the usefulness of k-DHR solutions on real-world data. We also perform extensive experiments to study the efficiency of our proposed algorithms.

2 The k-DHR Problem

Given a non-negative real-valued function g with a finite set $\mathcal{D}$ of inputs, our goal, intuitively, is to select a small set of elements from $\mathcal{D}$ that are "diverse" and "representative" of the high-value regions of $\mathcal{D}$. We formalize the problem below.

Given a candidate *representative* $s \in \mathcal{D}$, we define the *impact of s on the elements of $\mathcal{D}$* using a function $\phi_s : \mathcal{D} \mapsto \mathbb{R}_{\geq 0}$:

$$\phi_s(p) = g(s) \cdot \mathcal{K}_s(p), \tag{1}$$

where $\mathcal{K}_s : \mathcal{D} \mapsto \mathbb{R}_{\geq 0}$ is the *impact kernel* function. The impact of s on p is proportional to $g(s)$, the value of s, and is discounted by $\mathcal{K}_s(p)$. Generally speaking, the impact of s peaks at $p = s$, and diminishes for p further away from s. We will discuss concrete choices of $\mathcal{K}_s$ later.

Given a set $S \subseteq \mathcal{D}$ of representatives, we define the *impact of S on an element $p \in \mathcal{D}$*, denoted $\Phi_S(p)$, as the maximum impact on p among all individual representatives:

$$\Phi_S(p) = \max_{s \in S} \phi_s(p). \tag{2}$$

Intuitively, each $p \in \mathcal{D}$ is represented by some $s \in S$ having the maximum impact on p. Let $\mathcal{D}_S(s)$ denote the subset of elements in $\mathcal{D}$ represented by s, which we call the *max-impact region* of s, define as follows.

$$\mathcal{D}_S(s) = \{p \in \mathcal{D} \mid \phi_s(p) = \Phi_S(p)\}. \tag{3}$$

Basically, Φ_S is the upper envelope of functions $\{\phi_s\}_{s \in S}$, and $\{\mathcal{D}_S(s)\}_{s \in S}$ is also known as the *maximization diagram* of $\{\phi_s\}_{s \in S}$.[1]

Given k, the desired number of representatives, we want to choose the subset $S \subseteq \mathcal{D}$ of size k to maximize the *weighted total impact* of S on all elements of $\mathcal{D}$, defined as:

$$\mathcal{G}(S) = \sum_{p \in \mathcal{D}} g(p) \cdot \Phi_S(p). \tag{4}$$

The higher the value of p, the more weight it carries in the total, reflecting the intuition that high-value regions of $\mathcal{D}$ demand better coverage by the representatives.

To recap, the *k-(D)iverse (H)igh-value (R)epresentatives* (k-DHR) problem is formally defined as follows.

(The k-DHR Problem) *Given function $g : \mathcal{D} \mapsto \mathbb{R}_{\geq 0}$ and a natural number $k > 0$, find:*

$$S^* = \arg\max_{S \subseteq \mathcal{D}, |S| = k} \mathcal{G}(S), \text{ where} \tag{5}$$

$$\mathcal{G}(S) = \sum_{p \in \mathcal{D}} \left(g(p) \cdot \max_{s \in S} \left(g(s) \cdot \mathcal{K}_s(p) \right) \right). \tag{6}$$

If $\mathcal{D}$ is a subset of $\mathbb{R}^d$, a natural choice of the impact kernel would be the (unnormalized) d-dimensional Gaussian kernel with Mahalanobis distance:

$$\mathcal{K}_s(p) = \exp\left\{ -\frac{C}{2} \delta^2(p, s) \right\} \tag{7}$$

where parameter $C \in \mathbb{R}_{\geq 0}$ and covariance matrix Σ (with determinant 1) are user-defined, and $\delta^2(p, s) = (p - s)^T \Sigma^{-1} (p - s)$ is the squared Mahalanobis distance between p and s. We will discuss the setting of C as well as other possible impact kernels in the remainder of this section.

[1] The effective partitioning of the $\mathcal{D}$ by $\{\mathcal{D}_S(s)\}_{s \in S}$ is analogous to the Voronoi diagram—the latter is the minimization (as opposed to maximization) diagram of the distances to (as oppose to the impacts of) representatives.

2.1 Achieving the Three Objectives

Achieving Representativeness We note that representativeness is built into the definition of k-DHR—each representative naturally represents elements in its max-impact region. By weighing the impact on each element p by p's value in the objective function $\mathcal{G}$, k-DHR focuses more on high-value regions of the surface. Furthermore, doing so discourages picking high-value outliers as representatives, because they have low-value neighborhoods that weigh down their contribution to $\mathcal{G}$. In contrast, as discussed in Section 1, MMR and graph-based diversification, do not consider representativeness, so they are prone to picking outliers.

Achieving (Tunable) Diversification By dividing $\mathcal{D}$ into max-impact regions, each with one representative, k-DHR achieves diversification in a data-driven manner without resorting to any fixed distance threshold (like graph-based diversification) or weight on distance (like MMR). Once k-DHR picks a representative, it becomes redundant to have other representatives from the same "neighborhood." Here, the extent of neighborhood naturally depends on k as well as the presence of other nearby, high-value candidate representatives.

Nonetheless, k-DHR allows users to control the degree of diversity through the impact kernel $\mathcal{K}$. In particular, for the Gaussian-Mahalanobis kernel in Eq. (7), setting C larger implies a looser diversification requirement. Consider two extreme cases for example.

- $C = 0$. We have $\mathcal{K}_s(p) = 1$ and $\phi_s(p)$ simplifies to $g(s)$. Here, the impact of a representative s on p does not decrease as p moves away from s. Hence, it suffices to pick $\arg\max_{s \in \mathcal{D}} g(s)$, the highest-value element, to represent all of $\mathcal{D}$ (having additional representatives in S would not increase $\mathcal{G}(S)$ further). We can view this case as having the strongest diversification requirement, where the single representative "pushes out" all other candidates.
- $C \to +\infty$. We have $\mathcal{K}_s(p) = 1$ if $p = s$, or 0 otherwise. In this case, a representative has no impact on any element other than itself, and $\mathcal{G}(S)$ simplifies to $\sum_{s \in S} g(s)^2$. Hence, the optimal solution S^* would consist of the k elements in $\mathcal{D}$ with the highest values, regardless of distances among them. We can view this case as having no diversification requirement at all.

Intuitively, changing the impact kernel effectively adjusts the notion of how close two representatives are. A small C increases the competition from nearby high-value candidate representatives, and the objective function $\mathcal{G}$ of k-DHR naturally penalizes picking nearby representatives, hence encouraging diversity. Moreover, for the Gaussian-Mahalanobis kernel, changing C effectively performs uniform scaling of the co-variance matrix Σ. Diversity control along different directions is made possible by tuning individual values of Σ, hence changing the shape of the impact kernel. Typically, C is folded into Σ. We keep them separate for clarity as the they are used to control different things.

Tunable diversity is necessary for encoding expert knowledge and/or user preference. Even for the same dataset, the desired degree of diversity may vary depending on the context of interest. For example, in the historical data on employment (which we shall experiment later in Section 4.2), one may be interested in trends at different temporal scales: the desired degree of diversity when looking for claims in a specific election cycle will inevitably differ from that when looking across generations of population.

Finally, it is worth emphasizing again how k-DHR achieves tunable diversity differently from MMR and graph-based diversification discussed in Section 1. The other two methods also allow users to specify the two extreme cases above (corresponding to $C = 1$ and $C \to +\infty$ for k-DHR).[2] However, for the more interesting cases between the two extremes, these methods control the trade-off differently and lead to different results. As we have seen in Section 1 and will see in Section 4.2, MMR's linear weighting of distance by a fixed λ tends to push representatives farther away than necessary; graph-based diversification's fixed distance threshold δ may not suit different regions of the surface. The ways these methods apply their parameters do not allow them to adapt to local features of the surface; finding the appropriate parameters for a given dataset is thus difficult. In contrast, k-DHR's kernel-based formulation is more flexible and does not impose a hard constraint. Instead, diversity is achieved via clustering into max-impact regions, which adapts better to the local surface features.

Achieving Utility k-DHR achieves utility by accounting for the value of a representative in its impact (Eq. (1)). Doing so encourages selection of high-value representatives, but thanks to other components of the objective function, it does not lead to simply selecting elements with the highest values. For example, instead of picking another high-value element close to but largely "shadowed" by an existing representative, k-DHR is more likely to pick a faraway element whose value is not as high but brings a larger marginal contribution to $\mathcal{G}$.

In contrast, classical weighted clustering methods, such as k-means, k-median, and k-medoids, do not consider the weights of the representatives (centroids/medians/medoids). In fact, as we will show in Section 3.1, the k-median problem can be reduced to an instance of k-DHR. Taking the analogy a little further, we can view weighted clustering as considering values of elements in computing the weighted total impact (i.e., using the $g(p)$ term in Eq. (6)), but not considering the values of the representatives (i.e., not using $g(s)$ in Eq. (6)). As we have seen in Section 1 and will see in Section 4.2, this lack of consideration for the values of the representatives themselves leads to low result utility.

2.2 Choice of Impact Kernel

We introduced the Gaussian-Mahalanobis impact kernel (Eq. (7)) earlier in this section. Here we discuss some other choices of the impact kernel for k-DHR.

Gaussian vs. Reciprocal One alternative to the Gaussian kernel is the *reciprocal kernel* $\mathcal{K}_s(p) = \frac{1}{\delta(p,s)+C}$, where δ remains the Mahalanobis distance function and C is a regularization term (to avoid division by zero when $p = s$ or overly rewarding p's very close to s). Intuitively, the impact is inversely proportional to distance. However, compared to a Gaussian kernel, which gives a smooth decrease of impact in the close neighborhood of s, the reciprocal kernel yields a sharp decrease around s (Figure 3). For this reason, we find the reciprocal kernel less favorable than the Gaussian kernel in our applications.

Mahalanobis vs. Geodesic Distance Another option is to keep using a Gaussian kernel, but with a different distance function. Since we view g as a surface, a good choice is the *geodesic distance*, which measures the length of the shortest path between two points on the surface. Practically, suppose $\mathcal{D}$ is a set of discrete points. Let $\mathcal{N} \subseteq \mathcal{D} \times \mathcal{D}$ be a symmetric relation defining all pairs of (immediate) neighboring points of $\mathcal{D}$. Abusing the notation a little, let $\mathcal{N}(p) = \{q \mid (p,q) \in \mathcal{N}\}$ be the set of neighbors of

[2]The strongest diversification requirement can be achieved in MMR by setting $\lambda = 0$, and in graph-based diversification by setting $\delta \to +\infty$; all three methods return only the high-value element as the single representative. The case of no diversification can be achieved in MMR by setting $\lambda = 1$, and in graph-based diversification by setting $\delta = 0$ for the graph-based approach; all three methods return the k elements with the highest values.

p, and assume $|\mathcal{N}(p)|$ is bounded by a small constant. Consider a weighted undirected graph $G = (\mathcal{D}, \mathcal{N}, W)$, where for $(p, q) \in \mathcal{N}$, the edge weight is

$$W(p,q) = \sqrt{\delta^2(p,q) + (\alpha \cdot (g(p) - g(q)))^2}.$$

Here, $\delta(\cdot,\cdot)$ is the Mahalanobis distance function, and α is a normalization constant where

$$\alpha = \frac{\max_{p,q\in\mathcal{D}} \delta(p,q)}{\max_{p\in\mathcal{D}} g(p) - \min_{q\in\mathcal{D}} g(q)}.$$

Then, the (approximate) geodesic distance $\delta_G(p,q)$ between any $p, q \in \mathcal{D}$ is defined as the graph distance between p and q on G.

Geodesic distance has some modeling advantages over Mahalanobis distance, because the former accounts for changes in "elevation" (value). For example, consider two high-value regions separated only by a thin but deep valley. Under geodesic distance, k-DHR may choose two representatives, one from each region. Under Mahalanobis distance in $\mathcal{D}$, k-DHR will be less likely to do so, because it ignores the effort of crossing elevation changes and thinks that the two representatives are close.

On the other hand, from the efficiency perspective, Mahalanobis distance is more appealing. Computing geodesic distance on demand for an arbitrary pair of points takes $\Omega(|\mathcal{D}|)$ time in the worse case [8, 9]—as opposed to $O(1)$ time for Mahalanobis distance. Computing geodesic distances between one point and all other points takes $O(|\mathcal{D}|)$—same as that for Mahalanobis distances, but at the cost of $O(|\mathcal{D}|)$ additional space. In practice, for data and applications we tested, we found Mahalanobis distance to give comparable results much more quickly, as we will show in Section 4.

Kernel for Domains with Mutually Non-Representable Elements In practice, there are scenarios where it makes no sense for two elements of $\mathcal{D}$ to represent each another. For instance, suppose $\mathcal{D}$ includes both numerical and categorical attributes. (A concrete example will be presented in Section 4.1.) Two elements that do not agree on the categorical attributes may not be comparable in general, so selecting one in the answer does not give any representation to the other. Although we have not explicitly addresses such scenarios earlier, k-DHR can handle them gracefully with appropriate definitions of the impact kernel, as shown below.

Let the problem domain $\mathcal{D}$ be a finite subset of a d-dimensional space $\mathcal{P}$, consisting of two orthogonal subspaces which we call the *representing subspace* (denoted $\mathcal{P}_{\mathsf{R}}$) and the *non-representing subspace* (denoted $\mathcal{P}_{\mathsf{NR}}$). Formally, $\mathcal{D} \subseteq \mathcal{P} \subseteq \mathcal{P}_{\mathsf{R}} \times \mathcal{P}_{\mathsf{NR}}$, and $\dim(\mathcal{P}_{\mathsf{R}}) + \dim(\mathcal{P}_{\mathsf{NR}}) = d$. Intuitively, if two elements of $\mathcal{D}$ differ in their values for the non-representing subspace, they should not represent each other—we define their impact on each other to be 0; otherwise, they can represent each other and we compute their impact on each other using their projections onto the representing subspace. Formally, for each $p \in \mathcal{D}$, let $\pi_{\mathsf{R}}(p)$ and $\pi_{\mathsf{NR}}(p)$ denote the projections of p on $\mathcal{P}_{\mathsf{R}}$ and $\mathcal{P}_{\mathsf{NR}}$, respectively; let $\mathcal{K}^{\mathsf{R}}$ be a kernel defined over $\mathcal{P}_{\mathsf{R}}$ (such as the Gaussian-Mahalanobis kernel). We can simply define the impact kernel for $\mathcal{D}$ as $\mathcal{K}_s(p) = \mathcal{K}^{\mathsf{R}}_{\pi_{\mathsf{R}}(s)}(\pi_{\mathsf{R}}(p))$ if $\pi_{\mathsf{NR}}(p) = \pi_{\mathsf{NR}}(s)$, or 0 otherwise.

We will further discuss in Section 3 the algorithmic implications for domains with non-null non-representing subspaces.

3 Algorithms

We now discuss how to solve the k-DHR problem. We begin by establishing the hardness of k-DHR. Then, we present several algorithms for k-DHR: LocalSearch, EM, Greedy, as well as a version called Greedy$^+$ with a number of efficiency improvements. We conclude this section with a discussion on the relative merits of the four algrorithms, and on issues such as how these algorithms can help users choose appropriate k, and how they handle domains with mutually non-representable elements.

3.1 NP-Hardness of k-DHR

We show the NP-hardness of the decision version of the k-DHR problem by reduction from the well-known k-median problem, shown to be NP-hard in the plane under Euclidean distance [12].

(The k-median Problem in 2D) *Given a set $\mathcal{D}$ of n points in $\mathbb{R}^2$, a parameter k, and a value Δ, the k-median problem finds a k-subset S of $\mathcal{D}$ such that $\sum_{x\in\mathcal{D}} \min_{s\in S} \|x - s\| \le \Delta$, where $\|\cdot\|$ is the Euclidean norm.*

Lemma 1. *The k-DHR problem (decision version) is NP-hard.*

Proof. Given a set $\mathcal{D}$ of n points, a parameter k, and a value Δ, the k-median problem can be reduced to the decision version of k-DHR as follows. Consider an instance of k-DHR where $\mathcal{D}$ is the same, $g(p) = 1$ for all $p \in \mathcal{D}$, and $\mathcal{K}_s(p) = M - \|p - s\|$. Here, M is a constant no smaller than $\max_{p,q\in\mathcal{D}} \|p - s\|$. The k-median problem is equivalent to finding a k-subset S of $\mathcal{D}$ such that $\mathcal{G}(S) = \sum_{p\in\mathcal{D}}(M - \|p - s\|) \ge nM - \Delta$. Since the k-median problem is NP-hard, k-DHR (the decision version) is NP-hard. □

NP-hardness in Other Settings We used a different kernel in the reduction for the sake of simplicity. We note that this kernel approximates the linear kernel and the proof can be adapted. The k-means problem, where one uses squared Euclidean distance instead of the Euclidean distance, is also known to be NP-hard. Furthermore, $M \cdot (1 - \|p-q\|^2/M)$ approximates Gaussian kernel with appropriate choices of C and Σ. Therefore one could prove the NP-hardness for the Gaussian kernel as well, although the derivation becomes more complicated.

3.2 The LocalSearch Algorithm

Starting with an arbitrary k-subset S of $\mathcal{D}$, LocalSearch repeatedly makes local improvements—i.e., replacing a representative of S with one of its "neighbors" in $\mathcal{D}$—that increases the objective function $\mathcal{G}(S)$ the most, until no more improvements can be made.

We use the same neighbor relation $\mathcal{N}$ defined in Section 2.2 on pairs of elements in $\mathcal{D}$. As a concrete example, if $\mathcal{D}$ is a set of d-dimensional integral points (as for WAC claims to be discussed in Section 4.1), it is natural to define two points $(x_1, \ldots, x_d)$ and $(x'_1, \ldots, x'_d)$ to be neighbors iff $\sum_{i=1}^{d} |x_i - x'_i| = 1$. Given a set of representatives S, LocalSearch considers another set S' as a candidate for local improvement iff $S' = S \setminus \{s\} \cup \{s'\}$ for some $s \in S$ and $s' \in \mathcal{N}(s)$. Let candidates(S) denote set of candidates for improving S. Given a initial set S of k representatives, LocalSearch works as follows:

- Repeat until return:
  a. Find $S_{\text{new}} \leftarrow \arg\max_{S' \in \mathsf{candidates}(S)} \mathcal{G}(S')$.
  b. If $\mathcal{G}(S_{\text{new}}) \le \mathcal{G}(S)$, return S. Otherwise, $S \leftarrow S_{\text{new}}$.

Evaluating the objective function $\mathcal{G}$ for a given set of k representatives takes $O(k|\mathcal{D}|)$ time, where we compute the impact of each representative on each element of $\mathcal{D}$. Each step of LocalSearch involves evaluating $\mathcal{G}$ over all candidates in candidates(S). For the concrete example of $\mathcal{N}$ over d-dimensional integral points above, each element has $O(d)$ neighbors, so $|\mathsf{candidates}(S)| = O(kd)$. Hence, the overall complexity is $O((\#\text{ iterations}) \cdot k^2 d|\mathcal{D}|)$.

While LocalSearch will always find a locally optimal solution, it can (and will likely) miss the globally optimal solution. We can run LocalSearch multiple times, each time with a different, randomly selected set of k representatives. Standard techniques such

as simulated annealing can help, but a key concern remains that a large number of runs may be required to produce high-quality solutions for complex surfaces that arise in practice (as we shall see in experiments in Section 4).

3.3 The EM Algorithm

Our next approach applies the expectation-maximation (EM) technique [4]. Intuitively, EM alternates between two steps: a) given the current set S of k representatives, we assign each element of $\mathcal{D}$ to its best representative in S (one with the largest impact); b) given the current assignment of elements in $\mathcal{D}$ into k "clusters," we pick the best representative for each cluster (one that would generate the highest weighted total impact for the cluster).[3] Given an initial S, EM works as follows:

- Repeat until return:
  a. Partition $\mathcal{D}$ into clusters $\{\mathcal{D}_S(s)\}_{s\in S}$ (Eq. 3).
  b. For each cluster $\mathcal{D}_S(s)$, find new representative $\mathsf{newrep}(s) = \arg\max_{s'\in\mathcal{D}} \sum_{p\in\mathcal{D}_S(s)} g(p)\cdot\phi_{s'}(p)$.
  c. $S_{\text{new}} \leftarrow \{\mathsf{newrep}(s) \mid s \in S\}$. If $S = S_{\text{new}}$, return S. Otherwise, $S \leftarrow S_{\text{new}}$.

In (a), partitioning $\mathcal{D}$ into clusters takes $O(k\cdot|\mathcal{D}|)$ time—it involves computing the impact of each of the k representatives on each element $p\in\mathcal{D}$, and then for each p picking the representative with the maximum impact on p. In (b), for each cluster, finding the best representative involves enumerating every possible candidate s', and evaluating $\sum_{p\in\mathcal{D}_S(s)} g(p)\cdot\phi_{s'}(p)$ for each s'. The time complexity to process each cluster is $O(|\mathcal{D}|\cdot|\mathcal{D}_S(s)|)$. Summing over all clusters, we get a time complexity of $O(|\mathcal{D}|^2)$ for (b). Overall, the cost of EM is dominated by (b), and the total time complexity is $O((\#\text{ iteration})\cdot|\mathcal{D}|^2)$.

Like LocalSearch, the quality of the solution by EM can be affected by the initial choice of S. Thus, in practice, we run EM multiple times, each time with a different, randomly selected set of k representatives.

Compared with LocalSearch, EM steps are more expensive, and does not guarantee local optimality. However, our experiments show that when the number of runs is accounted for, EM can find solution with equal or better quality than LocalSearch within the same time frame; see Section 4 for details.

3.4 The Greedy Algorithm

An obvious drawback of the previous two algorithms is their lack of guarantee on the (global) optimality of their solutions. We now present a greedy algorithm capable of producing a solution that is at least $(1-1/e)\approx 62\%$ of the optimal in terms of the objective function value. In the following, we first sketch out a basic version of Greedy, then show its approximation ratio, and finally, discuss techniques for improving its efficiency that culminate in an improved version of the algorithm called Greedy$^+$.

Greedy starts out with an empty solution set $S_0 = \varnothing$. In the $(i+1)$-th step, Greedy greedily adds one more representative s to grow the solution from S_i to S_{i+1}. This representative s is chosen as the remaining unpicked element that maximizes the objective function $\mathcal{G}(S_i\cup\{s\})$. Greedy returns S_k after k steps.

[3]Weighted clustering algorithms often use the expectation-maximation technique as well. As explained in Section 2.1, the key difference lies in the criteria for assigning elements to clusters and picking representatives for clusters. Weighted clustering only considers distances and weights (values) of elements, while we also consider the values of representatives. This difference also leads to a subtlety: when we look for the best representative for a cluster, we look beyond elements within the cluster for candidates, as it is possible for the best representative to be outside the current cluster.

Choosing the representative in each step in a straightforward manner takes $O(|\mathcal{D}|^2)$ time, so the overall complexity of Greedy is $O(k|\mathcal{D}|^2)$. In more detail, each step considers $O(|\mathcal{D}|)$ candidates. For each candidate s, we need to evaluate $\mathcal{G}(S_i\cup\{s\})$. By remembering $\Phi_{S_i}(p)$ for each element $p\in\mathcal{D}$, we can easily compute $\Phi_{S_i\cup\{s\}}(p)$ in constant time. Therefore, it takes $O(|\mathcal{D}|)$ time to evaluate $\mathcal{G}(S_i\cup\{s\})$ for a given s (and to update $\Phi_{S_i}(p)$ to $\Phi_{S_{i+1}}(p)$ for each p once s is chosen).

To establish the quality of solution by Greedy, we first show the submodularity [15] and monotonicity of the objective function $\mathcal{G}$. (Note that there are other equivalent definitions of submodularity; we use only one below.)

Definition 1 (*Submodularity*)**.** *Let Ω be a set. A set function $f: 2^\Omega\mapsto\mathbb{R}$, where 2^Ω denotes the power set of Ω, is said to be* submodular *if for all $X\subseteq\Omega$ and $x_1,x_2\in\Omega\setminus X$, we have*

$$f(X\cup\{x_1\})+f(X\cup\{x_2\})\geq f(X\cup\{x_1,x_2\})+f(X).$$

Lemma 2. *$\mathcal{G}$ is submodular.*

Proof. We need to show that for all $S\subseteq\mathcal{D}$ and $s_1,s_2\in\mathcal{D}\setminus S$:

$$\mathcal{G}(S\cup\{s_1\})+\mathcal{G}(S\cup\{s_2\})\geq\mathcal{G}(S\cup\{s_1,s_2\})+\mathcal{G}(S).$$

Consider $P_1=\mathcal{D}_{S\cup\{s_1\}}(s_1)$ and $P_2=\mathcal{D}_{S\cup\{s_2\}}(s_2)$. First of all, it is clear that for all $p\in\mathcal{D}\setminus(P_1\cup P_2)$, $\Phi_{S\cup\{s_1\}}(p)=\Phi_{S\cup\{s_2\}}(p)=\Phi_{S\cup\{s_1,s_2\}}(p)=\Phi_S(p)$, i.e., including either or both of s_1 and s_2 would not change the contribution of p to the objective function $\mathcal{G}$.

For $p\in P_1\cup P_2$, we have

$$\max_{i=1,2}\{\Phi_{S\cup\{s_i\}}(p)\}=\Phi_{S\cup\{s_1,s_2\}}(p),$$
$$\min_{i=1,2}\{\Phi_{S\cup\{s_i\}}(p)\}\geq\Phi_S(p).$$

Adding up the equality and the inequality above, we have

$$\Phi_{S\cup\{s_1\}}(p)+\Phi_{S\cup\{s_2\}}(p)\geq\Phi_{S\cup\{s_1,s_2\}}(p)+\Phi_S(p).$$

Summing the inequality above for all $p\in\mathcal{D}$, we get

$$\mathcal{G}(S\cup\{s_1\})+\mathcal{G}(S\cup\{s_2\})\geq\mathcal{G}(S\cup\{s_1,s_2\})+\mathcal{G}(S).$$

Hence $\mathcal{G}$ is submodular. □

Lemma 3. *$\mathcal{G}$ is monotone, i.e., for all $S_1,S_2\subseteq\mathcal{D}$ such that $S_1\subseteq S_2$, we have $\mathcal{G}(S_1)\leq\mathcal{G}(S_2)$.*

The proof for Lemma 3 is trivial and thus omitted.

It was shown in [13] that a greedy algorithm provides a $(1-1/e)$-approximation for maximizing a monotone submodular set function with cardinality constraint. The greedy algorithm is defined as growing the solution set by picking its elements one at a time, namely the one that provides the maximum marginal contribution to the maximization objective. Greedy is an instantiation of this algorithm for the k-DHR problem.

Theorem 1. *Let S^* be the optimal solution to the k-DHR problem, and S^G be the solution by Greedy, we have*

$$\mathcal{G}(S^G)\geq\left(1-\frac{1}{e}\right)\cdot\mathcal{G}(S^*).$$

Note that this result holds for any choice of the impact kernel (or the distance function that it uses), because submodularity and monotonicity of $\mathcal{G}$ are inherent to the formulation of k-DHR and do not depend on $\mathcal{K}$ (our proofs of Lemmas 2 and 3 only uses the fact that $\Phi_S(p)$ maximizes $\phi_s(p)$ over all $s\in S$).

3.5 From Greedy to Greedy$^+$

Next, we present a number of techniques for improving the efficiency of Greedy, by utilizing the properties of the objective function and the solution set.

Lazy Update Not all elements are promising candidates for the solution set. For example, within a same neighborhood in $\mathcal{D}$, elements with higher values are more promising than those with lower values. However, the basic Greedy considers all elements as possible candidates indiscriminately. Motivated by this observation, we introduce the lazy update technique, which utilizes results remembered from previous steps to prioritize candidates for consideration, making it possible to choose the next representative without exhaustively re-examining all candidates.

Consider the $(i+1)$-th step. At this point we have already chosen S_i. For a remaining candidate $p \in \mathcal{D} \setminus S_i$, let $\Delta_{p,i}$ denote the extra utility that would be obtained by choosing p as the $(i+1)$-th representative; i.e., $\Delta_{p,i} = \mathcal{G}(S_i \cup \{p\}) - \mathcal{G}(S_i)$. The $(i+1)$-th representative to be chosen is $\arg\max_{p \in \mathcal{D} \setminus S_i} \Delta_{p,i}$.

We use a priority queue Q to record the extra utilities we computed during the algorithm. Each entry of Q has the form $\langle p, j, \Delta_{p,j} \rangle$, and records the extra utility of candidate p we computed after step j of the greedy algorithm. At any time, Q contains at most one entry for each p, recorded when p's extra utility was computed the last time. We prioritize Q based on its entries' extra utility component.

Observe that for any p, $\Delta_{p,i_1} \geq \Delta_{p,i_2}$ for any $i_1 < i_2$. In other words, the earlier a candidate p is added to the solution set, the more p can contribute to $\mathcal{G}$. Therefore, in the $(i+1)$-step, the (potentially stale) extra utility recorded for p in Q serves as an upper bound on $\Delta_{p,i}$. To choose the $(i+1)$-th representative, we repeatedly retrieve the top entry $\langle p, j, \Delta_{p,j} \rangle$ from Q. If $j = i$, we know p should be chosen. Otherwise, we update $\Delta_{p,j}$ to $\Delta_{p,i}$, and reinsert the updated entry $\langle p, i, \Delta_{p,i} \rangle$ into Q.

To update $\Delta_{p,j}$ to $\Delta_{p,i}$, we only need to consider elements in $\mathcal{D}_{S_j \cup \{p\}}(p) \bigcap \left(\bigcup_{s \in S_i \setminus S_j} \mathcal{D}_{S_i}(s) \right)$. Intuitively, there is no need to consider elements outside $\mathcal{D}_{S_j \cup \{p\}}(p)$ because they did not even contribute to $\Delta_{p,j}$ and therefore will not contribute to $\Delta_{p,i}$. There is also no need to consider elements outside $\bigcup_{s \in S_i \setminus S_j} \mathcal{D}_{S_i}(s)$ because their contributions to $\Delta_{p,j}$ are unaffected by changes from S_j to S_i and therefore remain the same to $\Delta_{p,i}$. Thanks to the geometric structure of the max-impact regions, such elements can be enumerated efficiently without scanning the entire $\mathcal{D}$. See our technical report [17] for additional details.

To initialize Q at the beginning of the greedy algorithm, let $Q = \{\langle p, -1, g(p) \cdot \mathcal{K}_p(p) \cdot G \rangle \mid p \in \mathcal{D}\}$, where $G = \sum_{q \in \mathcal{D}} g(q)$. The extra utility component here is a safe upper bound on $\Delta_{p,0}$, because $\Delta_{p,0} = \mathcal{G}(\{p\}) = \sum_{q \in \mathcal{D}} g(q) \cdot g(p) \cdot \mathcal{K}_p(q) \leq g(p) \cdot \mathcal{K}_p(p) \sum_{q \in \mathcal{D}} g(q) = g(p) \cdot \mathcal{K}_p(p) \cdot G$.

Overall, the lazy update technique can potentially save us a lot of work on considering many low-priority candidates, such as those with low values and those close to existing representatives.

Domains with Non-Representing Subspace Recall from Section 2.2 that in practice, some domains have elements that are mutually non-representable. We now show how to improve the greedy algorithm for the case where $\mathcal{D} \subseteq \mathcal{P} = \mathcal{P}_{\mathsf{R}} \times \mathcal{P}_{\mathsf{NR}}$ and the non-representing subspace $\mathcal{P}_{\mathsf{NR}}$ is non-null. The high-level intuition is the following. Since elements that differ in $\mathcal{P}_{\mathsf{NR}}$ cannot represent each other, we can group elements by their projections on $\mathcal{P}_{\mathsf{NR}}$ and process each group using an instance of Greedy. We execute these Greedy instances in parallel, requesting each instance to generate the next representative when needed.

Formally, for $X \subseteq \mathcal{P}$ and $\check{p} \in \mathcal{P}_{\mathsf{NR}}$, let $X_{\check{p}} = \{p \in X \mid \pi_{\mathsf{NR}}(p) = \check{p}\}$ denote the subset of X whose projection onto $\mathcal{P}_{\mathsf{NR}}$ is $\check{p}$. The definition of $\mathcal{K}$ using $\mathcal{K}^{\mathsf{R}}$ for domains with non-representing subspaces in Section 2.2 leads readily to the following:

Lemma 4. *Let S^* be the optimal solution to k-DHR on $\mathcal{D} \subseteq \mathcal{P}_{\mathsf{R}} \times \mathcal{P}_{\mathsf{NR}}$. For any $\check{p} \in \mathcal{P}_{\mathsf{NR}}$ where $\mathcal{D}_{\check{p}} \neq \varnothing$, $S^*_{\check{p}}$ is the optimal solution to the $|S^*_{\check{p}}|$-DHR problem on the domain $\mathcal{D}_{\check{p}}$.*

The above lemma allows us to solve the k-DHR problem on the entire domain $\mathcal{D}$ as independent sub-problems of $k_{\check{p}}$-DHR on sub-domains $\mathcal{D}_{\check{p}}$, except that the choice of $k_{\check{p}}$ for each $\check{p}$ is not known a priori. However, this issue can be addressed by the incremental nature of Greedy. We run an instance of Greedy for each $\mathcal{D}_{\check{p}}$, and we pick our next (global) representative to be the best among the next (local) representative chosen by each of the Greedy instances. Local representatives that are not picked remain in consideration when we pick the next global representative.

Suppose there are m distinct projections of $\mathcal{D}$ on $\mathcal{P}_{\mathsf{NR}}$, leading to m partitions of $\mathcal{D}$ of similar sizes. The overall time complexity of the above strategy is $O(\frac{k+m}{2} \frac{|\mathcal{D}|}{m^2})$, where picking a next local representative takes $O(\frac{|\mathcal{D}|}{m^2})$ time. A total of $k+m$ local representatives will be computed, including one extra representative for each of the m partitions, on top of the k representatives needed by the global solution.

Limiting the Set of Candidates The complexity of Greedy depends heavily on the number of candidates considered in each step. Even with lazy update, we are not ruling out the possibility for any element of $\mathcal{D}$ to be in the solution. To further improve efficiency, however, we consider the approach of limiting the set of candidates from the outset. One simple strategy is to choose as candidates all local maxima of the surface (defined using the same neighbor relation $\mathcal{N}$ introduced in Section 2.2). A disadvantage of this strategy is that it does not permit flexible control of the number of candidates.

Another strategy is weighted sampling, where we choose each element $p \in \mathcal{D}$ as a candidate with probability proportional to $g(p)$; we can control the number of candidates. (If $\mathcal{D}$ has a non-null non-representing subspace, we also observe the constraint that we pick at least one candidate from each sub-domain $\mathcal{D}_{\check{p}}$.) To reduce the chance that a particular set of candidates leads to a poor solution, we can run the greedy algorithm multiple times, each time with a different random sample as the set of candidates. We experimentally evaluate these strategies in Section 4.

The Combined Algorithm In Algorithm 1 we present Greedy$^+$, the improved version of Greedy that incorporates all techniques described above. This algorithm is given a candidate set $\mathcal{C} \subseteq \mathcal{D}$. To handle a domain with a non-null representing subspace $\mathcal{P}_{\mathsf{NR}}$, we create, for each distinct projection $\check{p}$ on $\mathcal{P}_{\mathsf{NR}}$, a priority queue $Q_{\check{p}}$ (Lines 2–4), to run an instance of the greedy algorithm with lazy update to solve the sub-problem on $\mathcal{D}_{\check{p}}$. We then use a merge queue M for storing the next local representative from each sub-problem and for choosing the next best global representative. The computeUpdate subroutine for updating $\Delta_{p,j}$ to $\Delta_{p,i}$ is given at the end of Algorithm 1; the enumeration of points on Lines 22 and 25 are performed efficiently using the power diagram structure of max-impact regions (see [17] for details).

3.6 Discussion

Among LocalSearch, EM, and basic Greedy, LocalSearch is clearly the fastest with each of its steps ($O(k^2 d|\mathcal{D}|)$ vs. $O(|\mathcal{D}|^2)$ for EM and Greedy). On the other hand, Greedy has a fixed number of steps (k). More importantly, Greedy deterministically produces a solution with theoretically guaranteed quality, so only one run is needed. In contrast, LocalSearch and EM require multiple runs

Algorithm 1: Greedy$^+(g, \mathcal{D}, k, \mathcal{C})$.

$\check{\mathcal{D}} \leftarrow \{\pi_{\mathsf{NR}}(p) \mid p \in \mathcal{D}\}$;
foreach $\check{p} \in \check{\mathcal{D}}$ **do**
  $Q_{\check{p}} \leftarrow \{\langle p, -1, g(p)\cdot\mathcal{K}_p(p)\cdot G\rangle \mid p \in \mathcal{C}_{\check{p}}\}$, where $G = \sum_{q\in\mathcal{D}_{\check{p}}} g(q)$, as a priority queue organized by each entry's third component;
  $k_{\check{p}} \leftarrow 0$; $S_{\check{p},0} \leftarrow \varnothing$;
$M \leftarrow \{\langle \check{p}, Q_{\check{p}}.\mathsf{removeMax}()\rangle \mid \check{p} \in \check{\mathcal{D}}\}$, as a priority queue organized by the third component within each entry's second component;
$S \leftarrow \varnothing$;
for $i \leftarrow 1$ **to** k **do**
  while true do
    if $M.\mathit{empty}()$ **then break**;
    $\langle \check{p}, \langle p, j, \Delta_{p,j}\rangle\rangle \leftarrow M.\mathsf{removeMax}()$;
    if $j = k_{\check{p}}$ **then**
      $S_{\check{p},k_{\check{p}}+1} \leftarrow S_{\check{p},k_{\check{p}}} \cup \{p\}$; $S \leftarrow S \cup \{p\}$;
    else
      $Q_{\check{p}}.\mathsf{insert}(\mathsf{computeUpdate}(\langle p, j, \Delta_{p,j}\rangle, k_{\check{p}}, \mathcal{D}_{\check{p}}, S_{\check{p},i}, S_{\check{p},k_{\check{p}}}))$;
    if $\neg Q_{\check{p}}.\mathit{empty}()$ **then**
      $M.\mathsf{insert}(\langle \check{p}, Q_{\check{p}}.\mathsf{removeMax}()\rangle)$;
    if $j = k_{\check{p}}$ **then**
      $k_{\check{p}} \leftarrow k_{\check{p}} + 1$; **break**;
return S;
def computeUpdate$(\langle p, j, \Delta_{p,j}\rangle, i, \mathcal{D}, S_j, S_i)$ **begin**
  // note that $\mathcal{D}$, S_j, S_i here may refer to a sub-problem
  if $j = -1$ **then**
    $\Delta_{p,i} = \sum_{q\in\mathcal{D}_{S_i\cup\{p\}}(p)} g(q)\cdot\phi_p(q)$;
  else
    $\Delta_{p,i} \leftarrow \Delta_{p,j}$;
    foreach $q \in \mathcal{D}_{S_j\cup\{p\}}(p) \bigcap \left(\bigcup_{s\in S_i\setminus S_j} \mathcal{D}_{S_i}(s)\right)$ **do**
      $\Delta_{p,i} \leftarrow \Delta_{p,i} - g(q)\cdot\left(\min\{\Phi_{S_i}(q), \phi_p(q)\} - \Phi_{S_j}(q)\right)$;
  return $\langle p, i, \Delta_{p,i}\rangle$;

with different seeds. Greedy$^+$ makes each step of Greedy much faster, especially by using a size-limited candidate set. The trade-off, however, is that solution quality becomes susceptible to the choice of the candidate set (just as LocalSearch and EM are affected by the choice of seeds). Thus, multiple runs of Greedy$^+$ may be needed. We will examine this trade-off closely with experiments in Section 4.

A nice feature of LocalSearch is that it always finds a locally optimal solution. Note that we can give this advantage to other algorithms as well, simply by feeding their solutions to LocalSearch as seeds and let LocalSearch improve them to local optima. Since LocalSearch steps are fast, this post-processing step adds very little overhead.

Compared with LocalSearch and EM, Greedy and Greedy$^+$ also hold advantages on the important practical issues of helping users choose appropriate k, and handling domains with mutually non-null non-representing subspace. We discuss these issues next.

Choosing k The formulation of k-DHR assumes that k is given, but in practice, given an input surface, it can be tricky for users to determine the value of k. One approach is to look for signs in the solutions indicating that the given k is too big. Namely, suppose for some representative s in the solution set S, $s \notin \mathcal{D}_S(s)$. In other words, s, as a representative, can in fact be represented better by another representative. If we can find such an s, that means we are picking too many representatives. For Greedy and Greedy$^+$, since the algorithms effectively produces solutions for $k = 1, 2, \ldots$ incrementally, we can simply perform the test at the end of every step and terminate as soon as we find k to be too big. For LocalSearch and EM, however, we can perform this test only at the end of the algorithm; if k is deemed too big, we have to run the algorithm from scratch with a smaller k.

A more methodical approach for choosing k, which we apply in Section 4, is to observe how the value of the objective function $\mathcal{G}$ improves as we increase k. If further increases in k would lead to only small marginal improvement, we will have found a good value for k. Again, the incremental nature of Greedy and Greedy$^+$ make them ideal for implementing this approach. LocalSearch and EM will be much more expensive to use.

Handling Domains with Non-Representing Subspace A non-null non-representing subspace $\mathcal{P}_{\mathsf{NR}}$ is problematic for LocalSearch, because poor seeding can be particularly detrimental. Recall from Section 3.5 that we can partition elements of $\mathcal{D}$ by their projection onto $\mathcal{P}_{\mathsf{NR}}$. Because elements from different partitions are not neighbors of each other, if LocalSearch starts with k seed representatives from a specific subset of the partitions, it will never consider any representatives from other partitions or change the number of representatives allocated to each partition. A huge number of independent runs may be needed to land on a reasonable solution.

EM also has trouble with a non-null $\mathcal{P}_{\mathsf{NR}}$. During the execution of EM, the current set S of representatives likely come from a small subset of the partitions of $\mathcal{D}$; elements outside these partitions have no preference in attaching themselves to any representative $s \in S$ because s always has zero impact on all of them. Although more flexible than LocalSearch, EM's search is less directed and less effective than Greedy. Greedy$^+$, with its optimization for domains with non-representing subspace, is even more efficient.

4 Experiments

We begin this section by describing the real-life application scenarios and datasets that we evaluate k-DHR and our algorithms with. Then, Section 4.2 focuses on assessing the k-DHR problem formulation and the usefulness of its solutions. Section 4.3 focuses on evaluating the efficiency of our proposed algorithms.

All algorithms were implemented in C++. We conducted all experiments on a machine with Intel Core i7-2600 3.4GHz processor and 7.8GB of memory.

4.1 Application Scenarios and Datasets

In [16], a factual claim based on structured data is modeled as a parametrized query, where the parameters control the context or particular view of the data from which the claim draws its conclusion. The goal of computational lead-finding is to find, given a dataset and a parametrized query template, settings of parameters that lead to interesting claims. k-DHR helps us find k claims that are interesting (high-utility), diverse, and representative. In this setting, $\mathcal{D}$ is set of all possible parameter settings, and g can be based on the query result as well as the *sensibility* of the parameter setting (see [16] for details). In the following, we describe two types of claims studied in [16], namely the *Window Aggregate Comparison (WAC)* claims and the *Time Series Similarity (TSS)* claims. We then describe the datasets that we use to evaluate the application of k-DHR to the above two claim types.

Window Aggregate Comparison (WAC) Claims A WAC claim is defined as a query q over a sequence of n positive numbers $x_1, x_2, \ldots, x_n$ (e.g., New York City adoption totals by year) and parametrized by three variables: 1) the *window length* w, 2) the *anchor time* t, and 3) the *lead* d. The query compares the sum of

values in two windows of the same length w with distance d between their index positions in the input sequence, where the second window ends at position t; i.e., $q(w,t,d) = \left(\sum_{i\in(t-w,t]} x_i\right) / \left(\sum_{i\in(t-d-w,t-d]} x_i\right)$.

For the k-DHR problem, $\mathcal{D}$ in general consists of points of $\mathbb{N}^3$ in a convex polytope constrained by $w \in [1, n-1]$, $t \in [w+1, n]$, and $d \in [1, t-w]$. For experiments in this paper, we fix $w = 1$, because otherwise it would be difficult to visualize the surface and assess the usefulness of solutions intuitively. Hence, $\mathcal{D}$ becomes a set of $O(n^2)$ 2D integral points, where the non-representing subspace is null. For the elevation function g in k-DHR, we start with q and normalize its range to $[0, 1]$ using the logistic function parametrized by the standard deviation of q on $\mathcal{D}$ (see [17] for details).

Time Series Similarity (TSS) Claims A TSS claim is a query q over a collection of m time series $X_1, X_2, \ldots, X_m$ of the same length n (e.g., legislators' voting records in the US Congress). The query compares the similarity between two of the m sequences within an interval $[a, b]$; i.e., $q(u, v, a, b) = \text{sim}_{[a,b]}(X_u, X_v)$. For the k-DHR problem, the full $\mathcal{D}$ is 4-dimensional and has size $O(m^2n^2)$. To obtain the elevation function g for k-DHR, we start with q and again normalize its range to $[0, 1]$.

In practice, we are often given a time series X_u of interest and only interested in finding claims comparing X_u with some other X_v; furthermore, claims involving different X_v's do not represent each other. Therefore, in this case, $\mathcal{D}$ has size $O(mn^2)$ and a non-representing subspace with attribute v. In other situations, we may be given both X_u and X_v, and only interested in finding claims comparing them over different time periods. In that case, $\mathcal{D}$ becomes structurally the same as that of WAC, i.e. $O(n^2)$ 2-d integral points with null non-representing subspace.

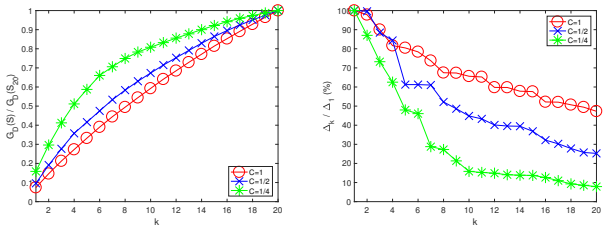
Datasets We use three real-life datasets in conjunction with the two claim types above for evaluation.

UNEMP This dataset is a small time series of US yearly unemployment rate,[4] from 1948 to 2012 ($n = 65$). It is used for generating a small surface for WAC claims used for intuitively assessing solution quality in Section 4.2. A representative chosen by k-DHR for this surface should translate into a robust factual claim quoting an increase in unemployment rate between two years in history. To make interpretation easier, we let t in WAC start at 1948 (instead of 1), and we fix $w = 1$ as discussed earlier. To help visualization, we further restrict $\mathcal{D}$ to a rectangular region $\{(t, d) \mid t \in [1981, 2012] \wedge d \in [1, 33]\}$ (without this restriction $\mathcal{D}$ would have been triangular, which is awkward to visualize).

STOCK This dataset contains the price history of the Yahoo! stock, with daily prices from January 1996 to July 2016 (with 253 trading days a year). It is suitable for generating WAC claims. Fixing $w = 1$, we define $\mathcal{D}$ by varying the range of t and d as follows. We restrict the anchor time t to be within the last ℓ years of the time series, and d to the range $(0, \ell]$. The size of $\mathcal{D}$ is hence $253^2 \cdot \ell^2$. By varying ℓ, we get $\mathcal{D}$ of different sizes, which we use to test the scalability of our algorithms.

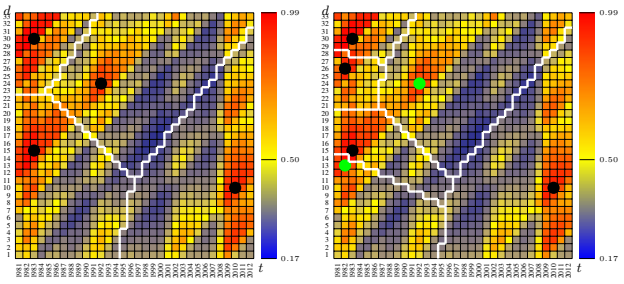
VOTE This database of US Congressional voting records contains a time series of votes for each of the US legislators. We focus on votes in the House from 2001 to 2014 by one of the $m = 153$ legislators whom we consider "recognizable" (with least one appearance on Sunday talk shows). We restrict the time granularity of comparison to whole months, resulting in $n = 154$. We consider two variants of the problem of generating TSS claims pertaining to former House Representative James Marshall (by fixing u to him):

[4]Derived by taking the average monthly unemployment rate data by calendar years. `http://data.bls.gov/timeseries/LNS140000000`



(a) (Normalized) $\mathcal{G}(S_k)$ vs. k (b) Δ_k/Δ_1 vs. k

Figure 4: How Greedy solution quality on *UNEMP* improves with k, for different values of kernel parameter C.



(a) $C = 1/2$, $k = 4$ (b) $C = 1$, $k = 6$; last two in green

Figure 5: Greedy solutions on *UNEMP* under different values of kernel parameter C.

- *VOTE-PELOSI*: By further fixing v to former House Speaker Nancy Pelosi, we get a $\mathcal{D}$ with dimensions (a, b) and total size $\binom{n}{2} \approx 1.2 \times 10^4$; the non-representing subspace is null.
- *VOTE-ALL*: By allowing v to vary, we get a $\mathcal{D}$ with dimensions (v, a, b), where the non-representing subspace involves dimension v. $\mathcal{D}$ effectively consists of m slices of 2-d surfaces, so the total size is $m\binom{n}{2} \approx 1.8 \times 10^6$.

For all experiments, we use a Gaussian-Mahalanobis impact kernel with identity covariance matrix, unless specified otherwise.

4.2 Case Study: WAC Claims on *UNEMP*

We use this case study to assess the flexibility and effectiveness of the k-DHR problem formulation. We use Greedy for k-DHR in experiments for this case study, because Greedy is deterministic and has provable guarantee on its solution quality (we will study its efficiency as well as other algorithms for k-DHR later in Section 4.3).

Choosing k, and Effect of Kernel Parameter C We first apply the strategy described in Section 3.6 to choose the approximate k for *UNEMP*. We run Greedy for 20 steps, which incrementally produces solutions $S_1, S_2, \ldots, S_{20}$ to k-DHR for $k = 1, 2, \ldots, 20$. Figure 4a shows the solution quality as measured by the value of the objective function $\mathcal{G}(S_k)$ (normalized by $\mathcal{G}(S_{20})$), while Figure 4b shows the marginal utility of each additional representative, as a percentage of the utility of the first representative (i.e., Δ_k/Δ_1, where $\Delta_k = \mathcal{G}(S_k) - \mathcal{G}(S_{k-1})$. We see that in general, the overall solution quality increases with k, but marginal improvement decreases. A good value of k can be chosen such that the next greedily selected representative gives significantly lower marginal improvement.

These figures also show the behavior under three different settings of the impact kernel parameter: $C = 1$, $1/2$ and $1/4$. For $C = 1/2$, as we can see from Figure 4b, the marginal contribution Δ_5 of the fifth representative is only 60% of Δ_1 (as opposed to $> 80\%$ by the fourth representative); therefore, $k = 4$ would be appropriate for $C = 1/2$, which gives the solution shown in Figure 5a. In contrast, for $C = 1$, Figure 4b shows that $k = 4$

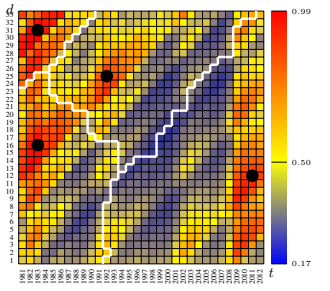
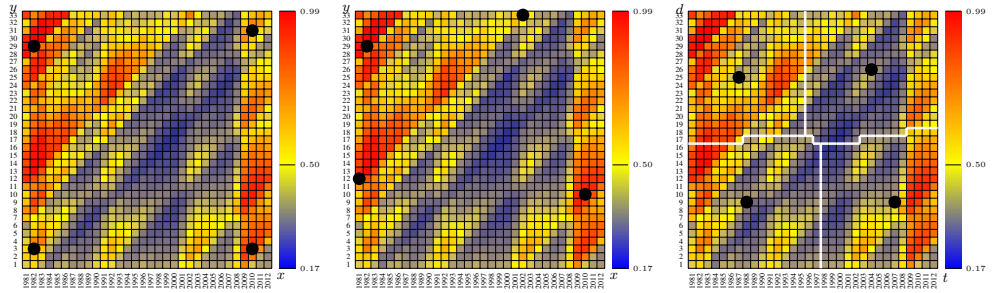


Figure 6: Solution to k-DHR by Greedy under geodesic distance; other settings same as Figure 5a.



(a) MMR (b) Graph-based diversification (c) Weighted k-means

Figure 7: Four representatives chosen from *UNEMP* by previous methods.

is not enough, as the next representative still gives good marginal improvement. This observation can be confirmed intuitively in Figure 5b, where the first four representatives, marked by black dots, are not enough to cover all high-value regions of the surface; the next two representatives, marked by green dots, continue to improve the solution.

Recall from Section 2.1 that a smaller C indicates a stronger preference towards diversity. Indeed, we see in Figure 4 that a smaller C translates to faster rate of decrease in marginal improvement, because a stronger diversity requirement implies that fewer representatives are needed. Figure 5 also shows the effect of C clearly: selection of representatives is intuitively more diverse in Figure 5a than in Figure 5b (even if we just compare the first four representatives chosen).

In the ensuing discussion, we fix $C = 1/2$ and $k = 4$. To give a better sense of the usefulness of the k-DHR solution, we translate the four representatives to plain English:

- The unemployment rate in 1983 increased by as much as 169.8% compared to that of 1968.
- The unemployment rate in 1983 increased by as much as 228.2% compared to that of 1953.
- The unemployment rate in 1992 increased by as much as 110.5% compared to that of 1968.
- The unemployment rate in 2010 increased by as much as 142.6% compared to that of 2000.

Kernel with Geodesic Distance In Section 2.2 we discussed the use of geodesic instead of Mahalanobis distance in the impact kernel. Now we see how it affects the solution quality for *UNEMP*. Figure 6 shows the 4-DHR representatives under geodesic distance. Compared with Figure 5a, which was obtained under Mahalanobis distance for the same setting, we see only very minor differences in their choices of representatives. We also see that geodesic distance is able to find boundaries of max-impact regions that better match the nuanced features of the surface, but in this case they have little impact on the solution. There are cases where the merit of geodesic distance is more noticeable; we show such a case with a synthetic surface in [17].

Another consideration is algorithm efficiency. With geodesic distance, we experimented with two implementation options:

- Use $O(|\mathcal{D}|)$ extra space, and compute pairwise geodesic distance in each step in $O(|\mathcal{D}|^2 \log|\mathcal{D}|)$ time. For *UNEMP*, Greedy finished in 11.52 seconds with this option.
- Use $O(|\mathcal{D}|^2)$ extra space, and precompute pairwise geodesic distance in $O(|\mathcal{D}|^2 \log|\mathcal{D}|)$ time. The time complexity of each Greedy step remains $O(|\mathcal{D}|^2)$. Greedy finished in 3.55 seconds with this option.

In contrast, Greedy with Mahalanobis distance finished in 1.32 seconds, significantly faster than with geodesic distance. For bigger problems, the efficiency advantage of Mahalanobis distance becomes even greater.

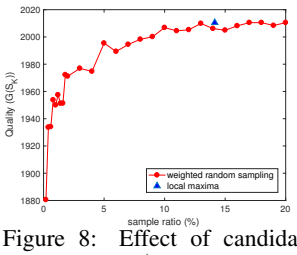


Figure 8: Effect of candidate set on Greedy$^+$ solution quality (*STOCK*, $\ell = 5$).

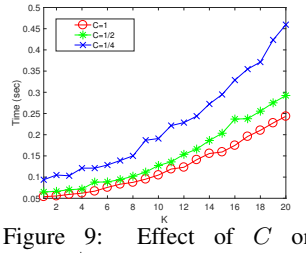


Figure 9: Effect of C on Greedy$^+$ efficiency (using all local maxima as candidates) on *VOTE-ALL*.

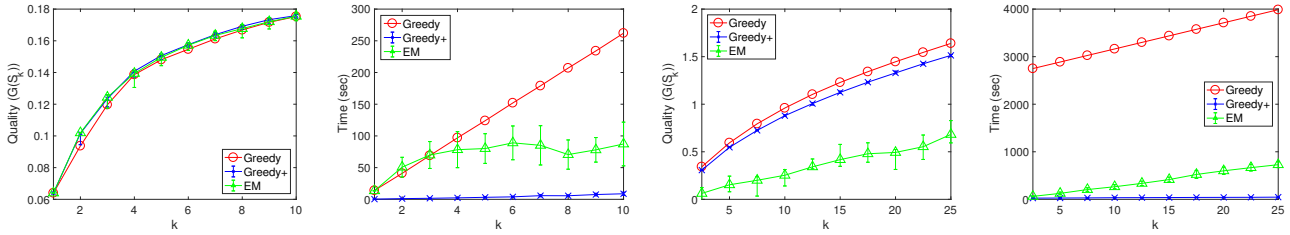
Comparison with Previous Methods Next, we apply the three previous methods discussed in Section 1—MMR, graph-based diversification, and weighted k-means clustering—to *UNEMP* for $k = 4$. MMR and graph-based diversification require parameter tuning. For MMR, we chose its parameter λ to be the smallest value such that the marginal contribution of the $(k+1)$-th (i.e., 5th) representative is non-negative. For graph-based diversification, there is no obvious way to choose the distance threshold δ automatically for a complex surface. We first ran weighted k-means to get a rough idea of how close the representatives are, and then set δ to be the smallest distance between two cluster centroids.

Figures 7 shows the solutions by the three previous methods. In Figures 7a and 7b, we see that both MMR and graph-based diversification picked diverse representatives with high individual values, but did a poor job with representativeness. Some of their choices, e.g., those located in the upper-right corner, do not represent a large high-value region. In Figure 7c, we see that weighted k-means failed to produce a meaningful partitioning for this fairly complex surface, and failed to pick representatives with high utility. In comparison, the solution to k-DHR in Figure 5a or 6 are intuitively much better than previous methods in achieving utility, diversity, and representativeness simultaneously.

4.3 Algorithm Effectiveness and Efficiency

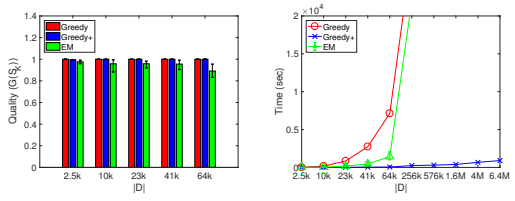
For the purpose of evaluating algorithm effectiveness and efficiency, we use the larger *VOTE* and *STOCK* datasets, which induce larger input surfaces to k-DHR. Before embarking on the comparison of all k-DHR algorithms, we will first investigate the effect of candidate set generation on Greedy$^+$, and the effect of kernel parameter C on algorithm efficiency.

Choosing Candidates for Greedy$^+$ Section 3.5 discussed two possible ways of using a small subset of $\mathcal{D}$ as the initial candidate set for Greedy$^+$: 1) using all local maxima and 2) weighted random sampling. Figure 8 shows how these two methods compare



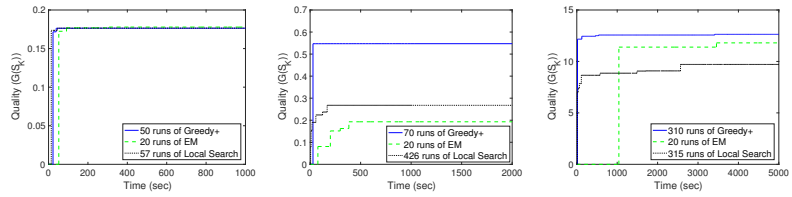
(a) Effectiveness (*VOTE-PELOSI*) (b) Efficiency (*VOTE-PELOSI*) (c) Effectiveness (*VOTE-ALL*) (d) Efficiency (*VOTE-ALL*)

Figure 10: Performance of k-DHR algorithms on *VOTE* as k varies.



(a) Effectiveness (b) Efficiency

Figure 11: Performance of k-DHR algorithms when varying the size of input data (derived from *STOCK*).



(a) *VOTE-PELOSI* (b) *VOTE-ALL* (c) *STOCK* ($\ell = 1$)

Figure 12: Best solution quality vs. time spent for LocalSearch, EM, and Greedy$^+$ on different datasets ($k = 10$).

in terms of solution quality, as measured by the objective function value. The dataset is *STOCK* with $\ell = 5$, for which the size of $\mathcal{D}$ is $253^2 \cdot 5^2 \approx 1.6 \times 10^6$. This surface is very noisy; about 14% of the elements in $\mathcal{D}$ correspond to local maxima. For weighted random sampling, we vary the sample size between 0.1% and 20% of $|\mathcal{D}|$. Each point in the plot is the average over 5 independent runs. Not surprisingly, the solution quality improves as the sample size increases, and becomes comparable to that using all local maxima starting around 10%. For the smaller *VOTE-PELOSI* and *VOTE-ALL* surfaces, it turns out a much lower sample ratio would suffice (see [17] for results). In ensuing experiments, we fix the candidate set size to be 1% of $|\mathcal{D}|$ for Greedy$^+$ with weighted random sampling, unless specified otherwise.

Effect of Kernel Parameter *C* on Algorithm Efficiency In Section 4.2 we have seen how C controls the diversity of k-DHR solutions. Now, we examine its effect on algorithm efficiency. In Figure 9, we show the running time for the first k steps of Greedy$^+$ (using all local maxima as candidates) for *VOTE-ALL*, under three different settings of C. We see that larger C leads to faster running times. There is a natural explanation. Recall from Section 2.1 that a smaller C indicates a stronger preference towards diversity, with representatives having larger impacts on faraway elements, which intuitively makes the problem harder. In the case of Greedy$^+$ with lazy updates, a smaller C basically leads to more updates being triggered.

Comparison of Algorithms when Varying *k* We compare the performance of EM, Greedy, and Greedy$^+$ on *VOTE-PELOSI* and *VOTE-ALL*, as we vary k. For all these algorithms, we apply LocalSearch as a post-processing step, as discussed in Section 3.6 (we will study how LocalSearch performs as a standalone algorithm at the end of this section). Greedy is deterministic. For EM and Greedy$^+$, we perform 20 runs for each problem setting with random seeding (we will further study the relationship between the number of runs and solution quality at the end of this section). In the effectiveness (as measured by solution quality) plots (Figures 10a and 10c), each plot point for EM and Greedy$^+$ shows the average solution quality over the 20 runs, plus an error bar showing the minimum and maximum qualities among the runs. In the efficiency (as measured by running time) plots (Figures 10b and 10d), each plot point for EM and Greedy$^+$ shows the average running time per run, plus an error bar representing the standard deviation.

On *VOTE-PELOSI*, in terms of effectiveness, Figure 10a shows that the best solution qualities of EM and Greedy$^+$ are comparable, and slightly better than that of Greedy. But among the multiple runs, Greedy$^+$ is more consistent than EM, as shown by narrower error bars. In terms of efficiency, Figure 10b shows that the running time of Greedy increases linearly in k. EM is slightly slower than Greedy when k is small, but becomes faster than Greedy when k increases, because its number of steps required for convergence does not increase linearly in k. Greedy$^+$ is clearly much faster than both Greedy and EM.

Next, consider the larger, more complex surface *VOTE-ALL* with a 1-d non-representing subspace. In term of effectiveness, Figure 10c shows that while Greedy and Greedy$^+$ remain comparable, EM produces notably poorer solutions. The reason lies in the difficulty of handling domains with non-representing subspace in EM, as discussed in Section 3.6, which is not an issue for Greedy or Greedy$^+$. In terms of efficiency, Figure 10d shows that Greedy is very slow because of the non-representing subspace, particularly at the very first step. Greedy$^+$ remains much faster than the other two algorithms.

Overall, Greedy$^+$ runs much faster than Greedy and EM while producing solutions of comparable or better quality.

Comparison of Algorithms when Varying Input Size We use *STOCK* to test the scalability of the proposed algorithms as we vary the input size. By letting $\ell = 0.2, 0.4, 0.6, 0.8, 1, 2, 3, 5, 8, 10$, we generate ten 2-d surfaces of sizes 2.5k, 10k, 23k, 41k, 64k, 256k, 576k, 1.6M, 4.1M, and 6.4M. The non-representing subspace is null here. Here, Greedy$^+$ uses all local maxima as candidates, and as Greedy, runs once for each problem setting; EM runs 20 times per setting with random seeding. Again, LocalSearch is applied as a post-processing step for all three algorithms.

In terms of effectiveness, Figure 11a compares the solution qualities of the three algorithms for the five smaller input surfaces where all three algorithms finished with 2 hours. As the input surfaces differ, we normalize the objective function values the using that achieved by Greedy. There is no observable difference between

Greedy and Greedy$^+$, but the quality of EM decreases slightly as the input size increases.

In terms of efficiency, Figure 11b shows that EM is faster than Greedy, but neither managed to finish within 6 hours when the input size goes beyond 254k. On the other hand, Greedy$^+$ scales much better than Greedy and EM, while generating solutions of comparable or higher quality. On the largest surface of size 6.4M, Greedy$^+$ finished within 20 minutes. Note that we have not considered I/O-efficiency of our algorithms when the input does not fit in memory, but it is possible to adapt Greedy$^+$ such that it makes one pass over its disk-resident intermediate data in each step. Other techniques such as parallelization and surface simplification are also interesting directions for future work.

Solution Quality vs. Time Spent While Greedy and Greedy$^+$ with all local maxima as candidates are deterministic, LocalSearch (running independently), EM, and Greedy$^+$ with randomly sampled candidates are non-deterministic and dependent on the initial seeding of the solution or candidate set. In practice, we perform multiple runs of these algorithms until we find a good solution or run out of time, so it is important to understand how fast they are able to improve over time. In Figure 12, we run LocalSearch, EM, and Greedy$^+$ repeatedly and record the best solution quality obtained so far at a particular time. Each plot is thus a step function, where each step up corresponds to the completion of a run that produced a new best solution.

On *VOTE-PELOSI* (Figure 12a), we see that the three algorithms eventually converged to a similar quality. However, Greedy$^+$ and LocalSearch managed to squeeze in more runs within the time limit, and converged faster than EM. On the larger, more complex *VOTE-ALL* (Figure 12b), we see that Greedy$^+$ not only converged much faster, but also to a higher quality than EM and LocalSearch, again because the difficulty of handling non-presenting subspace for EM and LocalSearch as discussed in Section 3.6. On *STOCK*, Greedy$^+$ again converged fastest and to the highest quality among the three algorithm, due to the much larger input size (as opposed to non-representing subspace). In conclusion, Greedy$^+$ can produce equal or better solutions faster than LocalSearch and EM. For this reason, Greedy$^+$ is our algorithm of choice in practice.

5 Related Work

The lead-finding problem studied in this paper is closely related to a large body of work on diversification of results for recommender systems and top-k queries.

Top-k diversification As discussed in Section 1 and 4.2, MMR [2] operates in an iterative and greedy fashion. Optimality was defined as the trade-off between a candidate item's relevance to the query item (utility), and its relevance to existing solution items (diversity). [3] later built on MMR and studied the diversified top-k problem over bounded regions.

The graph-based approach [14] uses a hard threshold to define similar items, and force them to be mutually exclusive in the solution set as a way to ensure diversity.

As illustrated throughout this paper, while these two methods focus on the quality of solution points only, we consider the relationship between the chosen representatives and other points, thus taking into account solution representativeness. We also took a more flexible approach towards diversification.

Weighted clustering We have also compared the k-DHR problem with the popular weighted k-means clustering algorithm [11, 10]. We have shown that weighted k-means does not strive for high-value representatives.

The classification of weighted clustering algorithms was studied in [1]. Weighted k-means belongs to the class of *weight sensitive* weighted clustering algorithms.

Diversification in recommender systems The utility-diversity trade-off also frequently appears in recommender systems. Using collaborative filtering, the trade-off can be represented as a linear combination of two different probability re-distribution methods [18]. Diversification is achieved by prefering "weak-ties".

Spatial diversification Item relevance in recommender systems can be regarded as item distance in a metric space, or parameter space distance in our context. [7, 5] studied the problem of k-Nearest Diverse Neighbor (kNDN) from a geometric perspective.

6 Conclusion

In this paper, we have studied the problem of finding diverse high-quality representatives on surface data. We have shown that existing methods designed for diversified top-k do not account for representativeness. Their diversify control also suffer from "no-one-size-fit-all". On the other hand, clustering methods, such as k-means, do not aim at finding high-quality individuals to represent high-value regions. We present the k-DHR problem that targets all three aspects of good representatives on a surface, which takes a different approach towards diversity. We show the advantage of k-DHR over the other methods on both synthetic and real data.

References

[1] M. Ackerman, S. Ben-David, S. Brânzei, and D. Loker Weighted clustering. *AAAI*, 2012, 858–863.

[2] J. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. *SIGIR*, 1998, 335–336.

[3] I. Catallo, E. Ciceri, P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Top-k diversity queries over bounded regions. *TODS*, 38(2), 2013.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society Series B (methodological)*, 39(1):1–38, 1977.

[5] J. R. Haritsa. The KNDN problem: A quest for unity in diversity. *IEEE DEB*, 32(4):15–22, 2009.

[6] M. Hasan, A. Kashyap, V. Hristidis, and V. J. Tsotras. User effort minimization through adaptive diversification. *SIGKDD*, 2014, 203–212.

[7] A. Jain, P. Sarda, and J. R. Haritsa. Providing diversity in k-nearest neighbor query results. *PAKDD*, 2004, 404–413.

[8] R. Kimmel, A. Amir and A. M. Bruckstein. Finding shortest paths on surfaces using level sets propagation. *PAMI*, 17(6):635–640, 1995.

[9] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. *PNAS*, 95(15):8431–8435, 1998.

[10] S. P. Lloyd. Least squares quantization in pcm. *Info. Theory*, 28(2):129–137, 1982.

[11] J. MacQueen. Some methods for classification and analysis of multivariate observations. *BSMSP*, 1(14):281–297, 1967.

[12] N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM*, 13(1):182–196, 1984.

[13] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functionsi. *Mathematical Programming*, 14(1):265–294, 1978.

[14] L. Qin, J. X. Yu, and L. Chang. Diversifying top-k results. *VLDB*, 5(11):1124–1135, 2012.

[15] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24. Springer, 2003.

[16] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu. Toward computational fact-checking. *VLDB*, 7(7):589–600, 2014.

[17] Y. Wu, J. Gao, P. K. Agarwal, and J. Yang. Finding diverse, high-value representatives on a surface of answers. Technical report, Duke University, 2016. `http://db.cs.duke.edu/papers/WuGaoEtAl-16-diverse_reprs.pdf`.

[18] T. Zhou, Z. Kuscsik, J-G Liu, M. Medo, J. R. Wakeling, and Y-C Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *PNAS*, 107(10):4511–4515, 2010.