Comparing Diversity Heuristics

Marina Drosou and Evaggelia Pitoura
Computer Science Department
University of Ioannina, Greece
{mdrosou, pitoura}@cs.uoi.gr

Abstract

Result diversification has recently attracted much attention as a means of increasing user satisfaction in recommender systems and web search. Selecting diverse items, however, has been proven to be an NP-hard problem. In this paper, we focus of a common interpretation of diversity, namely content diversity, and present a number of heuristics that have been proposed in the literature for tackling this problem. We also provide a comprehensive experimental evaluation of their performance.

1 Introduction

Today, most user searches have an exploratory nature, in the sense that users are mostly interested in retrieving various pieces of information about their search topic. Therefore, recently, *result diversification* has attracted considerable attention as a means of enhancing user satisfaction in recommender systems and web search (e.g. [18, 14]). Also, it can counteract the over-specialization problem, i.e. the retrieval of too homogeneous results. Consider, for example, a user who wants to buy a car and submits a web search to retrieve information. A *diverse* result, i.e. a result containing various brands and models with different horsepower and other technical characteristics is intuitively more informative than a result that contains a homogeneous result containing only cars with similar features.

Diverse items may be generally defined in three different ways, namely in terms of (i) *novelty*, i.e. items that contain new information when compared to previously seen ones (e.g. [5, 17]), (ii) *coverage*, i.e. items that belong to different categories (assuming that a taxonomy of categories exists, e.g. [2]) and (iii) *content* (or *similarity*), i.e. items that are dissimilar to each other (e.g. [16]).

In this paper, we tackle the problem of selecting k diverse information pieces (or *items*) among the results that satisfy the information needs of the users. We focus on content diversity among the various items and seek to process them in such a way as to locate items that are distant (dissimilar) to each other.

The problem of selecting k diverse results out of the available ones is related to the p-dispersion problem which has been shown to be NP-hard [7, 8]. Due to the high complexity of the problem, we employ heuristics that have been proposed in the literature in various research fields. We consider four categories of heuristics: (i) *greedy*, (ii) *interchange*, (iii) *neighborhood* and (iv) *clustering* heuristics. We implement all heuristics and perform an experimental evaluation on both synthetic and real datasets to demonstrate the efficiency and effectiveness of each of them.

The rest of this paper is structured as follows. In Section 2, we define the k-diversity problem, while in Section 3, we present a suite of heuristics for its efficient solution. Section 4 evaluates the performance of the various heuristics. Section 5 briefly reviews related work and, finally, Section 6 concludes this paper.

2 Problem Definition

There are various forms of diversity. Here, we focus on content diversity, so that users receive dissimilar information. Specifically, given a set[1] P of n items, we aim at selecting k items out of them, such that, the average pairwise distance between the selected items is maximized. More formally:

[1] In this work, we use the term “set” loosely to denote a set with *bag semantics* or a *multiset*, where the same item may appear more than once in the set.

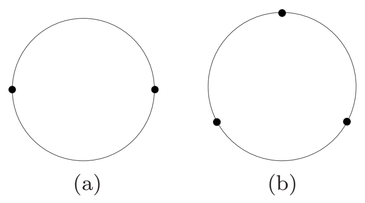

Figure 1: Most diverse points of a circle for (a) $k = 2$ and (b) $k = 3$.

Definition 1 (k-diversity): *Let $P = \{p_1, \ldots, p_n\}$ be a set of n items and k an integer with $k \leq n$. Let also $d(p_i, p_j)$, $p_i, p_j \in P$, be a distance measure between items p_i and p_j. The k-diversity problem is to locate a set S^*, $S^* \subseteq P$, such that:*

$$S^* = \operatorname*{argmax}_{\substack{S \subseteq P \\ |S|=k}} f(S), \text{ where } f(S) = \frac{1}{k(k-1)} \sum_{i=1}^{k} \sum_{j>i}^{k} d(p_i, p_j) \text{ and } p_i, p_j \in S \tag{1}$$

The problem of selecting the k items having the maximum average pairwise distance out of n items is similar to the p-*dispersion* problem. This problem, as well as a number of its variations (e.g. select p out of n items so that the minimum distance between any two of them is maximized), have been extensively studied in operations research and are in general known to be NP-hard [7, 8]. Thus, to solve large instances of the problem, we need to rely on heuristics. A number of heuristics have been proposed in the literature (e.g. [8]), ranging from applying exhaustive algorithms to adapting traditional optimization techniques.

A general issue that hinders the development of efficient incremental solutions to the problem is that the $k-1$ most diverse items of a set P are not necessarily a subset of its k most diverse items. For example, consider as items the points on the circumstance of a circle and their euclidean distances. The two furthest apart points are (any) two antidiametric ones. However, no antidiametric points belong to a set of three most diverse points (Figure 1).

In the following section, we describe various families of heuristics for tackling the k-diversity problem.

3 Content Diversity Heuristics

In this section, we describe a suite of heuristics for locating solutions to the k-diversity problem. These heuristics have been employed for solving variations of the problem in more than one research areas. We classify the examined heuristics into four categories: (i) *greedy*, (ii) *interchange*, (iii) *neighborhood* and (iv) *clustering* heuristics (Figure 2). In the following sections, we describe heuristics in each category and discuss their weaknesses and strong points.

3.1 Greedy Heuristics

The greedy heuristics are the ones most commonly used since they are intuitive and some of them are also relatively fast (e.g. [18, 2, 11, 4]). Greedy heuristics make use of two sets: the set P of candidate items and a set S which contains the selected ones. Items are iteratively moved from P to S and vice versa until $|S| = k$ and $|P| = n - k$. There are two main variations: (i) the Greedy Construction heuristic and (ii) the Greedy Deletion one.

Algorithm 1 Greedy Construction Heuristic

Input: The set of candidate items P and the number of wanted items k.
Output: A set S with k most diverse items.

```
1: find $p_i, p_j$, s.t. $d(p_i, p_j) = \max\{d(p_i, p_j) : p_i, p_j \in P, p_i \neq p_j\}$
2: $S \leftarrow \{p_i, p_j\}$
3: while $|S| < k$ do
4:     find $p_i \in P \backslash S$, s.t. $setdist(p_i, S) = \max\{setdist(p_j, S) : p_j \in P \backslash S\}$
5:     $S \leftarrow S \cup \{p_i\}$
6: end while
7: return $S$
```

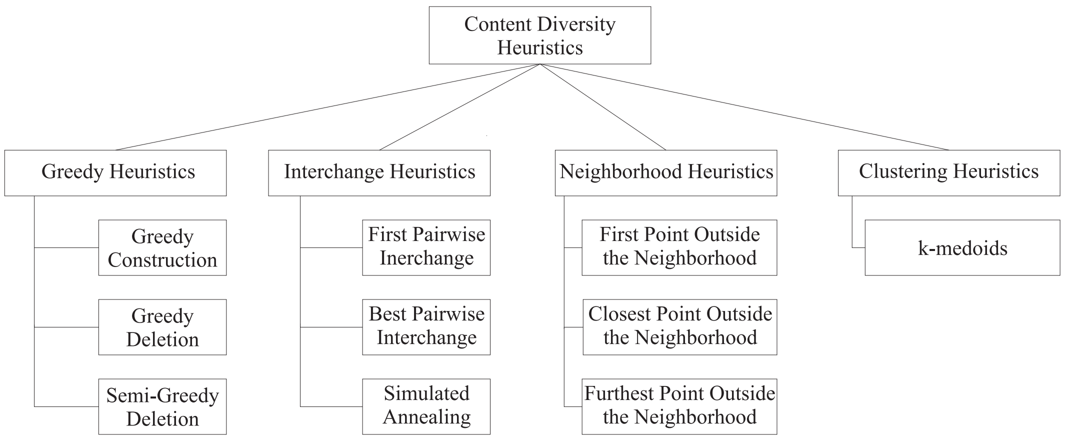
Content Diversity Heuristics
Greedy Heuristics
Interchange Heuristics
Neighborhood Heuristics
Clustering Heuristics
Greedy Construction
Greedy Deletion
Semi-Greedy Deletion
First Pairwise Inerchange
Best Pairwise Interchange
Simulated Annealing
First Point Outside the Neighborhood
Closest Point Outside the Neighborhood
Furthest Point Outside the Neighborhood
k-medoids

Figure 2: Classification of content diversity heuristics.

In the *Greedy Construction* (GC) heuristic, S is initialized with the two furthest apart items in P. Then, at each iteration, one more item is added to S. The item that is added is the one that has the maximum item-set distance from S (Algorithm 1). We define the *item-set distance*, denoted $setdist(p_i, S)$, between an item p_i and a set of items S as the average distance between p_i and the items in S, that is:

$$setdist(p_i, S) = \frac{1}{|S|} \sum_{p_j \in S} d(p_i, p_j) \tag{2}$$

The complexity of GC in terms of distance computations is $O(n^2)$, where n is the size of P. However, this is due to the first step of the algorithm where the two furthest apart items have to be located. The rest of the algorithm requires $O(k^2 n)$ distance computations. Therefore, to reduce the execution time of GC, we also consider the variation where S is initialized by selecting the two furthest apart items from a randomly selected subset of P with size equal to h, $h < n$, instead of using the whole set. We will later show that this modification greatly reduces the execution time without affecting the diversity of the selected items considerably.

Algorithm 2 Greedy Deletion Heuristic

Input: The set of candidate items P and the number of wanted items k.
Output: A set S with k most diverse items.

```
1: $S \leftarrow P$
2: while $|S| > k$ do
3:     find $p_i, p_j$, s.t. $d(p_i, p_j) = \min\{d(p_i, p_j) : p_1, p_j \in S, p_i \neq p_j\}$
4:     find $p_z$, s.t. $setdist(p_z, S\backslash\{p_z\}) = \min\{setdist(p_z, S\backslash\{p_z\}) : p_z \in \{p_i, p_j\}\}$
5:     $S \leftarrow S\backslash\{p_z\}$
6: end while
7: return $S$
```

Algorithm 3 Semi-Greedy Deletion Heuristic

Input: The set of candidate items P and the number of wanted items k.
Output: A set S with k most diverse items.

```
1: $S \leftarrow P$
2: while $|S| > k$ do
3:     find $p_i, p_j$, s.t. $d(p_i, p_j) = \min\{d(p_i, p_j)| : p_i, p_j \in S, p_i \neq p_j\}$
4:     randomly set $p_z \leftarrow p_i$ or $p_z \leftarrow p_j$
5:     $S \leftarrow S\backslash\{p_z\}$
6: end while
7: return $S$
```

In the *Greedy Deletion* (GD) heuristic, S is initialized with all items in P. Then, at each iteration, one item is deleted from S, until k of them remain. To delete an item, the two closest items of S are first located and then the one that has the smallest item-set distance from the remaining points in S is chosen for deletion (Algorithm 2). We also consider the *Semi-Greedy Deletion* (SGD) variation, where one of those two items is chosen randomly for deletion (Algorithm 3). Those heuristics are more time consuming than the constuction alternative, since they require $O(n^3)$ distance computations.

3.2 Interchange Heuristics

Interchange heuristics have also been used in the literature for solving the k-diversity problem (e.g. [15, 12]). Generally, these heuristics are initialized with a random solution S and then iteratively attempt to improve that solution by interchanging an item in the solution with another item that is not in the solution. There are two main variations: (i) The *First Pairwise Interchange* (FI) heuristic (Algorithm 4) performs at each iteration the first met interchange that improves the solution, while (ii) the *Best Pairwise Interchange* (BI) heuristic considers all possible interchanges and performs the one that improves the solution the most (Algorithm 5).

Algorithm 4 First Pairwise Interchange Heuristic

Input: The set of candidate items P and the number of wanted items k.
Output: A set S with k most diverse items.

```
1: Set $S$ to be a random solution
2: while at least one change occurs in $S$ do
3:     find $p_i, p_j$, s.t. $d(p_i, p_j) = \min\{d(p_i, p_j) : p_i, p_j \in S, p_i \neq p_j\}$
4:     for all $p_z \in P \backslash S$ do
5:         $S' \leftarrow \{S \backslash \{p_i\}\} \cup \{p_z\}$
6:         $S'' \leftarrow \{S \backslash \{p_j\}\} \cup \{p_z\}$
7:         if $f(S') > f(S)$ and $f(S') \geq f(S'')$ then
8:             $S \leftarrow S'$
9:             break
10:        end if
11:        if $f(S'') > f(S)$ and $f(S'') \geq f(S')$ then
12:            $S \leftarrow S''$
13:            break
14:        end if
15:    end for
16: end while
17: return $S$
```

Algorithm 5 Best Pairwise Interchange Heuristic

Input: The set of candidate items P and the number of wanted items k.
Output: A set S with k most diverse items.

```
1: Set $S$ to be a random solution
2: while at least one change occurs in $S$ do
3:     find $p_i, p_j$, s.t. $d(p_i, p_j) = \min\{d(p_i, p_j) : p_i, p_j \in S, p_i \neq p_j\}$
4:     for all $p_z \in P \backslash S$ do
5:         $S' \leftarrow \{S \backslash \{p_i\}\} \cup \{p_z\}$
6:         $S'' \leftarrow \{S \backslash \{p_j\}\} \cup \{p_z\}$
7:         if $f(S') > f(S)$ and $f(S') \geq f(S'')$ then
8:             $S \leftarrow S'$
9:         end if
10:        if $f(S'') > f(S)$ and $f(S'') > f(S')$ then
11:            $S \leftarrow S''$
12:        end if
13:    end for
14: end while
15: return $S$
```

The worst case complexity of both variations is $O(n^k)$. While each distinct iteration of FI is on average faster than an iteration of BI, BI is more likely to require less iterations to converge. There are two ways to limit the iterations performed by interchange heuristics. We can either set a bound, on the maximum number of possible interchanges to be performed or allow interchanges to continue as long as the solution improves by a given threshold.

Another heuristic that can be classified into this category is *Simulated Annealing* (SA). Simulated Annealing attempts to avoid local maxima by also accepting interchanges that do no improve the solution. Such interchanges are accepted with a probability that decreases as time passes (Algorithm 6). The performance of Simulated Annealing heavily depends on a number of tuning parameters. However, it is generally exponential as well.

Algorithm 6 Simulated Annealing Heuristic

Input: The set of candidate items P, the number of wanted items k and the parameters $FREEZELIMIT \in \mathbb{Z}^+$, $SIZEFACTOR \in \mathbb{Z}^+$, $CUTOFF \in \mathbb{Z}^+$, $TEMPFACTOR \in [0,1]$, $MINPERCENT \in [0,1]$.
Output: A set S with k most diverse items.

```
1: Set $S$ to be a random solution
2: $T \leftarrow \max\{d(p_i,p_j) : p_i,p_j \in P, p_i \neq p_j\}$
3: $iterations \leftarrow 0$
4: while $iterations < FREEZELIMIT$ do
5:    $trials \leftarrow changes \leftarrow 0$
6:    while $trials < SIZEFACTOR$ and $changes < CUTOFF$ do
7:       $trials \leftarrow trials + 1$
8:       $p_i \leftarrow$ a random point in $\{S\}$
9:       $p_j \leftarrow$ a random point in $\{P \backslash S\}$
10:      $S' \leftarrow \{S \backslash \{p_i\}\} \cup \{p_j\}$
11:      $e \leftarrow f(S') - f(S)$
12:      $r \leftarrow$ a random number in $(0,1)$
13:      if $e > 0$ or $r < exp(e/T)$ then
14:         $S \leftarrow S'$
15:         $changes \leftarrow changes + 1$
16:      end if
17:   end while
18:   $T \leftarrow T \cdot TEMPFACTOR$
19:   if $f(S)$ has increased then
20:      $iterations \leftarrow 0$
21:   end if
22:   if $changes/trials < MINPERCENT$ then
23:      $iterations \leftarrow iterations + 1$
24:   end if
25: end while
26: return $S$
```

3.3 Neighborhood Heuristics

Neighborhood heuristics are construction algorithms that start with a solution S containing one random item and then iteratively add items to the solution. The items to be added are chosen based on the notion of *r-neighborhood* of an item $p_i \in P$, $N(p_i, P, r)$, defined as:

$$N(p_i, P, r) = \{p_j \in P : d(p_i, p_j) \leq r\} \tag{3}$$

In other words, all items that have a smaller or equal distance to p_i than r belong to its r-neighborhood. At each iteration, only items outside the r-neighborhoods of all already selected items are considered. Out of these items, one is chosen to be added to the solution. There are several options regarding which item to add. The *First Item Outside the Neighborhood* (FiN) heuristic adds the fist located item in P that is outside those r-neighborhoods, while the *Closest Item Outside the Neighborhood* (ClN) heuristic adds the one that has the smallest sum of distances to the already selected items. Finally, the *Furthest Item Outside the Neighborhood* (FuN) heuristic adds the item that has the largest sum of distances to the already selected items. Note that, the selection of r plays an important role as it restricts the number of items that are considered at each iteration. In fact, given a value of r, a solution S with $|S| = k$ may not even exist. Algorithm 7 shows generic pseudo-code for all the heuristics of this category. These heuristics begin with a random item and iteratively add one more item to the solution until k items have been selected. Therefore, in the worst case, they require $O(k^2n)$ distance computations.

A factor that influences the performance of all neighborhood heuristics is the value of r. To determine an r value leading to good solutions, one must have a view of how items are spread throughout the available space. One time-efficient way to get an estimation for r is to initially set r to be equal with the average distance of k randomly selected items. If a solution with k items cannot be located with this initial value, we can reduce r by a percentage and run the neighborhood heuristic again.

3.4 Clustering Heuristics

The k-diversity problem bears certain resemblances to clustering, in the sense that selecting k diverse items can be seen as selecting k representatives of the initial set P of items, i.e. clustering P into k groups and selecting their medoids. However, the two problems are also different. For example, outliers that are expected to be recovered when selecting diverse items are generally suppressed by clustering algorithms.

Algorithm 7 Neighborhood Heuristics

Input: The set of candidate items P, the number of wanted items k and a radius r.
Output: A set S with k most diverse items.

```
1: $p_i \leftarrow$ random point of $P$
2: $S \leftarrow \{p_i\}$
3: $OUT \leftarrow P \backslash \{p_j : p_j \in N(p_i, P, r)\}$
4: while $|OUT| > 0$ do
5:     select a point $p_i \in OUT$ to be added      // first, closest or furthest
6:     $OUT \leftarrow OUT \backslash \{p_j : p_j \in N(p_i, P, r)\}$
7:     $S \leftarrow S \cup \{p_i\}$
8: end while
9: return $S$
```

Nonetheless, it is interesting to see how clustering algorithms behave when employed to tackle the k-diversity problem. In this work, we employ the widely-used *k-medoids* (KM) heuristic [10] (Algorithm 8). The number of required distance computations is $O(Ik(n-k)^2)$, where I is the number of performed iterations. The number of iterations I is not bounded. Both I and the overall performance of k-medoids heavily depend on the initialization of the algorithm. Usually, concerning clustering algorithms, the method of multiple restarts is employed, where the algorithm is run a number of times and the best located solution is kept.

Algorithm 8 k-medoids Algorithm

Input: The set of candidate items P and the number of wanted items k.
Output: A set S of k medoids.

```
1: Set $S = \{s_1, \ldots, s_k\}$ to be a random solution
2: while at least one change occurs in $S$ do
3:     for all $p_i \in P \backslash S$ do
4:         assosiate $p_i$ with the medoid $s_j$, s.t. $d(p_i, s_j) = \min\{d(p_i, s_j) : p_i \in P, s_j \in S\}$
5:     end for
6:     for all clusters $S_i$ do
7:         update corresponding medoid $s_i \leftarrow p_j^*$, s.t. $p_j^* = \arg\min_{p_j \in S_i} \sum_{p_z \in C_i} d(p_z, p_j)$
8:     end for
9: end while
10: return $S$
```

4 Evaluation

In this section, we compare the heuristics of Section 3 in terms of efficiency and effectiveness and present our evaluation results using both synthetic and real data. All heuristics were implemented in JDK 6. We ran our experiments on a Windows XP PC with a 2.4 GHz Intel Core 2 Processor and 2.00 GB of RAM.

4.1 Synthetic Data

To evaluate the performance of the heuristics, we create synthetic datasets consisting of data items in the Euclidean space, where each dimension takes values in $[0, 1000]$. In our experiments, we vary the size n of the datasets as well as the dimensions D of the items. Our datasets contain items drawn from either a uniform or a normal distribution centered in the middle of $\mathbb{R}^D$ with a standard deviation equal to 150 in all dimensions. We also create datasets that contain items from five different normal distributions with a standard deviation equal to 50 spread throughout the available space. We measure distances among items using the Euclidean distance metric. We report results averaged over ten runs of the heuristics. We use “div” to denote the average diversity of the located solutions. Also, time is measured in milliseconds. Table 1 summarizes the notation used for the heuristics of Section 3.

4.1.1 Greedy Heuristics

We begin our experimental evaluation by examining the behavior of the greedy heuristics. Concerning GC, we examine the cases for which $h = n$ (i.e. initializing the heuristic with the whole set of candidate items), $h = n/2$ and $h = 2$ (i.e. initializing the heuristic with two random items).

Table 1: Heuristics notation.

Notation	**Heuristic**
GC	Greedy Construction
GD	Greedy Deletion
SGD	Semi-Greedy Deletion
FI	First Pairwise Interchange
BI	Best Pairwise Interchange
SA	Simulated Annealing
FiN	First Item Outside the Neighborhood
ClN	Closest Item Outside the Neighborhood
FuN	Furthest Item Outside the Neighborhood
KM	k-medoids
RA	Random Selection

In Table 2, we show results for 4-dimensional datasets of size $n = 400$ when varying k for the GC, GD and SGD heuristics. We also report the average diversity and execution time when randomly selecting k items (denoted RA). Naturally, the average achieved diversity decreases as k increases, since more items have to be selected. We see that GC consistently outperforms both greedy deletion variations, both in terms of achieved diversity and execution time. As k increases, the execution time of GC increases as well, while the execution time of GD and SGD is not affected considerably. However, GD and SGD consume much more time than the GC alternative, without achieving better diversity. We also observe that reducing the initially examined set of items in GC by half (i.e. using $h = n/2$) reduces the execution time without affecting the diversity of the solution. Using $h = 2$ further improves execution time with only a slight decrease in diversity. Finally, GD and SGD consume similar time but GD locates solutions with higher diversity.

Next, we fix $k = 40$ and repeat the experiment for different input sizes (Table 3). As n increases, better solutions can be located by all heuristics since there are more candidate items available to choose from. GC continues to locate solutions with higher diversity. The execution time of GD and SGD increases rapidly for higher values of n (by more than an order of magnitude when tripling the dataset size). The execution time of GC increases as well along with n but only linearly.

Finally, we conduct an experiment for different data dimensionalities (Table 4). The execution time of all heuristics increases slightly along with D, since each distance computation requires more time. However, this increase is not significant. The achieved diversity increases along with D as well, since for a fixed n the data space becomes sparser.

4.1.2 Interchange Heuristics

We continue our experimental evaluation by examining the behavior of the interchange heuristics. We allow FI and BI to run until convergence, since we observed that trying to limit the number of iterations as described in Section 3.2 heavily influences the diversity of the located solutions. For SA, we experimented with a number of configurations. Here, we report results for $FREEZELBIIT = 5$, $SIZEFACTOR = 10(n - k)$, $CUTOFF = 10$, $TEMPFACTOR = 0.9$ and $MINPERCENT = 0.5$.

Table 5 shows results when varying k. Surprisingly, FI locates better solutions than BI, even though one would expect the best-interchange strategy to lead to better solutions. However, it also consumes more time. Generally, SA locates better solutions than the other two heuristics, especially for larger values of k. However, the cost in execution time is much higher. This trade-off can probably be tuned by calibrating SA's parameters. This calibration depends on the input of the heuristic though and is not easy to be achieved.

Table 6 shows results when varying n. It is interesting to notice that, while FI continues to locate better solutions, BI scales better in terms of execution time as n increases. This is not the case when varying the number of dimensions D (Table 7), where both heuristics exhibit similar behavior. SA continues to outperform both FI and BI in terms of achieved diversity at the cost of high execution time.

Table 2: Greedy heuristics:
Average achieved diversity and execution time for $n = 400$ and $D = 4$.

Uniform dataset												
k	**GC** ($h = n$)		**GC** ($h = n/2$)		**GC** ($h = 2$)		**GD**		**SGD**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**
20	1133.67	29.70	1133.16	17.10	1106.35	12.40	1064.03	1121.90	946.47	1111.00	784.18	0.00
30	1103.12	39.20	1102.99	31.20	1088.28	25.00	1026.84	1121.90	921.00	1114.10	790.03	0.00
40	1079.85	57.80	1079.81	45.30	1068.47	43.80	991.92	1111.00	906.20	1103.10	790.45	0.00
50	1061.54	81.10	1061.54	70.30	1053.93	67.10	967.45	1112.50	886.83	1101.60	787.94	0.00
60	1045.47	109.40	1045.47	93.70	1038.77	89.10	955.13	1117.00	881.29	1118.80	768.02	0.00

Normal dataset												
k	**GC** ($h = n$)		**GC** ($h = n/2$)		**GC** ($h = 2$)		**GD**		**SGD**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**
20	692.15	29.70	691.65	15.80	673.56	12.40	666.86	1123.40	618.93	1112.50	399.97	0.00
30	664.62	40.60	664.38	31.40	652.68	23.40	640.13	1112.50	596.27	1114.00	388.15	0.00
40	643.56	59.30	643.56	48.40	637.07	43.70	611.68	1121.70	570.81	1103.10	399.05	0.00
50	626.06	81.30	625.96	71.90	619.97	62.40	587.14	1115.40	561.72	1106.20	397.38	1.50
60	610.84	109.30	610.84	93.80	607.82	93.80	574.63	1123.30	553.29	1109.40	376.52	1.60

Clustered dataset												
k	**GC** ($h = n$)		**GC** ($h = n/2$)		**GC** ($h = 2$)		**GD**		**SGD**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**
20	578.84	28.10	578.46	18.70	570.97	12.50	514.86	1107.70	482.24	1112.70	444.38	0.00
30	560.35	39.20	560.35	31.30	554.45	25.10	486.40	1104.70	463.26	1107.90	436.45	0.00
40	548.80	59.50	548.75	46.70	545.44	46.80	475.20	1112.60	461.82	1104.60	433.01	0.00
50	540.33	79.70	540.30	68.70	537.75	62.40	471.69	1114.20	454.40	1100.00	437.58	1.60
60	533.11	109.30	533.11	93.70	531.11	89.30	470.41	1103.20	455.74	1103.00	432.73	0.00

Table 3: Greedy heuristics:
Average achieved diversity and execution time for $k = 40$ and $D = 4$.

Uniform dataset												
n	**GC** ($h = n$)		**GC** ($h = n/2$)		**GC** ($h = 2$)		**GD**		**SGD**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**
200	1007.47	36.00	1007.47	25.00	999.82	18.80	939.92	246.80	866.55	242.10	770.87	0.00
400	1079.85	57.80	1079.81	45.30	1068.47	43.80	991.92	1111.00	906.20	1103.10	790.45	0.00
600	1113.75	84.40	1113.46	68.80	1103.32	64.00	1017.29	2906.20	911.02	2962.70	789.28	0.00

Normal dataset												
n	**GC** ($h = n$)		**GC** ($h = n/2$)		**GC** ($h = 2$)		**GD**		**SGD**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**
200	560.99	36.00	560.99	25.10	555.66	20.50	529.74	248.30	496.65	243.70	385.09	0.00
400	643.56	59.30	643.56	48.40	637.07	43.70	611.68	1121.70	570.81	1103.10	399.05	0.00
600	691.37	84.40	691.37	70.40	682.96	64.10	654.54	2926.60	622.49	2984.60	391.80	1.50

Clustered dataset												
n	**GC** ($h = n$)		**GC** ($h = n/2$)		**GC** ($h = 2$)		**GD**		**SGD**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**
200	513.77	36.10	513.77	22.00	510.93	18.70	450.23	247.00	438.66	245.40	436.35	0.00
400	548.80	59.50	548.75	46.70	545.44	46.80	475.20	1112.60	461.82	1104.60	433.01	0.00
600	568.90	81.30	568.89	71.90	564.94	64.10	501.99	2895.30	480.43	2909.40	435.33	0.00

Table 4: Greedy heuristics:
Average achieved diversity and execution time for $k = 40$ and $n = 400$.

Uniform dataset												
D	GC ($h = n$)		GC ($h = n/2$)		GC ($h = 2$)		GD		SGD		RA	
	div	time	div	time	div	time	div	time	div	time	div	time
2	760.17	54.70	760.17	42.30	754.08	35.80	606.93	1032.90	564.46	1014.10	534.64	0.00
4	1079.85	57.80	1079.81	45.30	1068.47	43.80	991.92	1111.00	906.20	1103.10	790.45	0.00
6	1309.98	60.90	1310.15	48.50	1300.71	46.90	1274.58	1181.10	1150.14	1193.80	970.41	0.00

Normal dataset												
D	GC ($h = n$)		GC ($h = n/2$)		GC ($h = 2$)		GD		SGD		RA	
	div	time	div	time	div	time	div	time	div	time	div	time
2	466.42	57.80	466.42	42.30	461.97	37.70	403.98	1029.60	387.56	1021.90	260.94	0.00
4	643.56	59.30	643.56	48.40	637.07	43.70	611.68	1121.70	570.81	1103.10	399.05	0.00
6	768.96	59.50	768.91	48.50	760.31	45.30	750.24	1182.80	705.07	1178.00	497.90	0.00

Clustered dataset												
D	GC ($h = n$)		GC ($h = n/2$)		GC ($h = 2$)		GD		SGD		RA	
	div	time	div	time	div	time	div	time	div	time	div	time
2	423.37	53.20	423.37	44.00	421.09	40.60	358.16	1029.60	334.42	1026.70	306.18	0.00
4	548.80	59.50	548.75	46.70	545.44	46.80	475.20	1112.60	461.82	1104.60	433.01	0.00
6	654.61	62.60	654.61	46.90	652.05	46.90	591.68	1182.80	555.03	1192.30	513.24	0.00

Table 5: Interchange heuristics:
Average achieved diversity and execution time for $n = 400$ and $D = 4$.

Uniform dataset								
k	FI		BI		SA		RA	
	div	time	div	time	div	time	div	time
20	1090.42	437.50	1013.87	298.50	1133.96	2117.10	784.18	0.00
30	1017.95	684.40	913.16	512.40	1103.03	4090.60	790.03	0.00
40	1002.14	1064.10	901.68	731.20	1079.72	8651.6	790.45	0.00
50	972.96	1273.30	876.29	992.30	1061.18	9634.40	787.94	0.00
60	940.65	1528.20	853.19	1089.10	1045.44	16082.80	768.02	0.00

Normal dataset								
k	FI		BI		SA		RA	
	div	time	div	time	div	time	div	time
20	672.73	620.40	649.44	442.10	692.06	1884.5	399.97	0.00
30	646.52	1150.00	605.23	726.50	664.41	3926.5	388.15	0.00
40	616.14	1756.20	560.61	998.30	643.51	6581.2	399.05	0.00
50	594.93	2145.40	529.58	1331.20	625.92	9509.5	397.38	1.50
60	583.30	2987.60	501.44	1687.50	610.83	21037.4	376.52	1.60

Clustered dataset								
k	FI		BI		SA		RA	
	div	time	div	time	div	time	div	time
20	504.95	473.60	510.24	220.20	579.02	1751.50	444.38	0.00
30	503.98	892.30	476.33	437.40	560.35	3910.80	436.45	0.00
40	493.88	1490.60	468.81	635.90	548.71	7196.80	433.01	0.00
50	487.08	2295.30	461.69	774.90	540.30	10824.90	437.58	1.60
60	479.56	2929.60	455.81	932.90	533.10	18570.30	432.73	0.00

Table 6: Interchange heuristics:
Average achieved diversity and execution time for $k = 40$ and $D = 4$.

Uniform dataset								
n	**FI**		**BI**		**SA**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
200	929.70	581.30	887.22	510.90	1007.37	3909.30	770.87	0.00
400	1002.14	1064.10	901.68	731.20	1079.72	8651.60	790.45	0.00
600	1011.28	1590.70	878.09	704.60	1113.88	13243.70	789.28	0.00

Normal dataset								
n	**FI**		**BI**		**SA**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
200	532.93	653.00	505.50	596.90	560.66	2842.10	385.09	0.00
400	616.14	1756.20	560.61	998.30	643.51	6581.20	399.05	0.00
600	654.42	3134.50	580.02	1203.20	691.41	12703.30	391.80	1.50

Clustered dataset								
n	**FI**		**BI**		**SA**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
200	469.45	612.50	460.82	362.50	513.78	3639.10	436.35	0.00
400	493.88	1490.60	468.81	635.90	548.71	7196.80	433.01	0.00
600	504.98	1948.40	463.21	609.30	568.77	12115.70	435.33	0.00

Table 7: Interchange heuristics:
Average achieved diversity and execution time for $k = 40$ and $n = 400$.

Uniform dataset								
D	**FI**		**BI**		**SA**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
2	598.53	554.70	552.92	495.40	759.99	5570.20	534.64	0.00
4	1002.14	1064.10	901.68	731.20	1079.72	8651.60	790.45	0.00
6	1270.88	1354.70	1193.36	1056.30	1310.23	6829.70	970.41	0.00

Normal dataset								
D	**FI**		**BI**		**SA**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
2	408.05	1345.20	340.04	618.70	466.28	6775.10	260.94	0.00
4	616.14	1756.20	560.61	998.30	643.51	6581.20	399.05	0.00
6	749.90	1848.50	669.02	956.20	769.04	7988.90	497.90	0.00

Clustered dataset								
D	**FI**		**BI**		**SA**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
2	366.55	901.40	341.25	528.30	423.41	7595.30	306.18	0.00
4	493.88	1490.60	468.81	635.90	548.71	7196.80	433.01	0.00
6	600.73	1804.80	577.17	690.70	654.49	8167.10	513.24	0.00

Table 8: Neighborhood heuristics:
Average achieved diversity and execution time for $n = 400$ and $D = 4$.

Uniform dataset

k	**FiN**		**ClN**		**FuN**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
20	851.09	0.00	653.00	17.30	1109.17	20.40	784.18	0.00
30	860.61	0.00	725.06	21.80	1060.27	35.90	790.03	0.00
40	847.09	3.10	769.25	37.60	1004.71	43.60	790.45	0.00
50	789.18	4.70	557.88	126.50	991.50	96.90	787.94	0.00
60	798.96	3.10	547.02	168.80	1023.24	196.80	768.02	0.00

Normal dataset

k	**FiN**		**ClN**		**FuN**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
20	479.88	1.50	334.05	15.60	672.98	20.30	399.97	0.00
30	471.00	1.60	373.48	26.60	654.44	45.30	388.15	0.00
40	488.45	1.60	402.48	29.70	627.07	70.20	399.05	0.00
50	504.34	0.00	426.78	46.90	595.01	89.10	397.38	1.50
60	500.54	6.30	389.90	93.80	578.85	134.40	376.52	1.60

Clustered dataset

k	**FiN**		**ClN**		**FuN**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
20	404.05	1.60	259.87	23.30	557.74	18.70	444.38	0.00
30	421.75	4.70	333.08	39.20	534.02	37.60	436.45	0.00
40	426.45	11.00	386.67	56.20	514.60	51.60	433.01	0.00
50	444.78	15.60	395.30	81.20	496.27	59.40	437.58	1.60
60	370.97	21.70	280.39	182.80	492.90	142.10	432.73	0.00

4.1.3 Neighborhood Heuristics

Next, we evaluate the performance of the neighborhood heuristics. To determine the input radius r of these heuristics (Equation 3), we use the following approach. We initially set r to be equal with the average distance of k randomly selected items from the dataset. If a solution with k items cannot be located by the heuristic, we reduce r by half and run the heuristic again. Tables 8 to 10 show the results of our experiments. All three neighborhood heuristics are fast to compute (the initial value of r was usually reduced two or three times by all heuristics). We observe that the FiN and ClN heuristics do not manage to locate good solutions for the problem, since the improvement from the random case (RA) is not very significant. FuN on the other hand locates good solutions, comparable to the ones located by the greedy and interchange heuristics. Its execution time is lower than that of the interchange heuristics and can be compared with that of GC, as long as n does not increase much. Generally, neighborhood heuristics have the disadvantage that a good initial value of r must be estimated. In case this is possible, out of the three variations, FuN is the best choice.

4.1.4 Clustering Heuristics

Interestingly, the k-medoids algorithm does not manage to locate good solutions for the problem. In fact, the diversity of the selected items is almost always less than that of the random case (Tables 11 to 13). This, however, is not very surprising. For example, as k increases and more clusters are formed over the same data, some medoids are placed near in space. What is most interesting to notice is that KM is the only heuristic that achieves increasing diversity as k increases, in contrast to all other heuristics. Therefore, KM may have some practical use in highly dense spaces.

4.1.5 Summary

As demonstrated in the previous sections, there is no category of heuristics that outperforms the others. Instead, there are some heuristics (belonging to different categories) that can be considered better than the rest.

Table 9: Neighborhood heuristics:
Average achieved diversity and execution time for $k = 40$ and $D = 4$.

Uniform dataset								
n	**FiN**		**ClN**		**FuN**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
200	768.77	0.00	588.73	31.10	959.88	35.90	770.87	0.00
400	847.09	3.10	769.25	37.60	1004.71	43.60	790.45	0.00
600	883.04	4.70	718.42	73.40	1026.56	70.40	789.28	0.00

Normal dataset								
n	**FiN**		**ClN**		**FuN**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
200	444.01	1.50	427.32	6.20	520.43	25.00	385.09	0.00
400	488.45	1.60	402.48	29.70	627.07	70.20	399.05	0.00
600	883.04	4.70	718.42	73.40	672.17	115.70	391.80	1.50

Clustered dataset								
n	**FiN**		**ClN**		**FuN**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
200	388.93	7.80	305.95	40.60	466.24	21.90	436.35	0.00
400	426.45	11.00	386.67	56.20	514.60	51.60	433.01	0.00
600	425.66	9.20	375.99	84.30	546.68	89.10	435.33	0.00

Table 10: Neighborhood heuristics:
Average achieved diversity and execution time for $k = 40$ and $n = 400$.

Uniform dataset								
D	**FiN**		**ClN**		**FuN**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
2	545.75	1.60	490.33	45.20	633.44	51.60	534.64	0.00
4	847.09	3.10	769.25	37.60	1004.71	43.60	790.45	0.00
6	1029.79	0.00	835.23	50.00	1286.69	67.20	970.41	0.00

Normal dataset								
D	**FiN**		**ClN**		**FuN**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
2	325.74	1.50	248.78	48.40	427.57	82.80	260.94	0.00
4	488.45	1.60	402.48	29.70	627.07	70.20	399.05	0.00
6	582.02	1.50	430.21	46.80	757.04	84.30	497.90	0.00

Clustered dataset								
D	**FiN**		**ClN**		**FuN**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**
2	333.73	9.40	222.24	85.90	393.08	79.60	306.18	0.00
4	426.45	11.00	386.67	56.20	514.60	51.60	433.01	0.00
6	528.17	9.40	396.05	64.10	630.53	64.00	513.24	0.00

Table 11: k-medoids:
Average achieved diversity and execution time for $n = 400$ and $D = 4$.

Uniform dataset				
k	**KM**		**RA**	
	div	**time**	**div**	**time**
20	727.87	9.40	784.18	0.00
30	761.53	12.60	790.03	0.00
40	780.54	15.50	790.45	0.00
50	769.03	21.80	787.94	0.00
60	780.24	37.50	768.02	0.00

Normal dataset				
k	**KM**		**RA**	
	div	**time**	**div**	**time**
20	368.54	14.10	399.97	0.00
30	390.34	11.00	388.15	0.00
40	386.73	15.60	399.05	0.00
50	387.74	23.50	397.38	1.50
60	387.48	37.40	376.52	1.60

Clustered dataset				
k	**KM**		**RA**	
	div	**time**	**div**	**time**
20	422.95	10.90	444.38	0.00
30	431.26	17.20	436.45	0.00
40	430.75	18.70	433.01	0.00
50	432.96	26.60	437.58	1.60
60	430.98	40.60	432.73	0.00

Table 12: k-medoids:
Average achieved diversity and execution time for $k = 40$ and $D = 4$.

Uniform dataset				
n	**KM**		**RA**	
	div	**time**	**div**	**time**
200	748.81	17.30	770.87	0.00
400	780.54	15.50	790.45	0.00
600	768.18	23.40	789.28	0.00

Normal dataset				
n	**KM**		**RA**	
	div	**time**	**div**	**time**
200	391.54	17.20	385.09	0.00
400	386.73	15.60	399.05	0.00
600	386.44	23.20	391.80	1.50

Clustered dataset				
n	**KM**		**RA**	
	div	**time**	**div**	**time**
200	428.91	14.00	436.35	0.00
400	430.75	18.70	433.01	0.00
600	433.63	26.50	435.33	0.00

Table 13: k-medoids:
Average achieved diversity and execution time for $k = 40$ and $n = 400$.

Uniform dataset				
D	**KM**		**RA**	
	div	**time**	**div**	**time**
2	532.64	20.30	534.64	0.00
4	780.54	15.50	790.45	0.00
6	930.42	17.20	970.41	0.00

Normal dataset				
D	**KM**		**RA**	
	div	**time**	**div**	**time**
2	264.77	21.90	260.94	0.00
4	386.73	15.60	399.05	0.00
6	483.88	20.40	497.90	0.00

Clustered dataset				
D	**KM**		**RA**	
	div	**time**	**div**	**time**
2	310.32	17.20	306.18	0.00
4	430.75	18.70	433.01	0.00
6	511.99	17.20	513.24	0.00

Concerning greedy heuristics, Greedy Construction (GC) clearly outperforms the Greedy Deletion alternatives (GD and SGD), not only in execution time, as expected due to its lower complexity, but also in terms of the achieved diversity. It also scales much better as n increases.

As far as interchange heuristics are concerned, Simulated Annealing (SA) locates the best solutions. This, however, is achieved at the cost of a very high execution time. The First Interchange (FI) and Best Interchange (BI) heuristics also locate good solutions at much smaller cost. This, along with the fact that the performance of SA depends on a large number of input parameters that need to be tuned, leads us to the conclusion that FI and BI are better choices. Between the two, FI locates more diverse solutions but BI scales better, especially as n increases. Compared to GC from the previous category, GC locates solutions with comparable diversity to SA at less time than all three interchange heuristics.

Out of the three neighborhood heuristics, Furthest Item Outside the Neighborhood (FuN) locates the best solutions. Its performance lies somewhere between that of GC and FI/BI, while its execution time is quite low. Closest Item Outside the Neighborhood (ClN) does not locate good solutions to the problem. The reason for this is that its performance heavily depends on the choice of r. In fact, the solutions located by ClN are expected to have a diversity close to the value of r. First Item Outside the Neighborhood (FiN) performs much better than ClN but worse than FuN. However, its execution time is very low, something that gives to this heuristic some practical value.

Finally, k-medoids (KM) does not manage to locate good solutions for the problem. It also behaves in a different way than the other heuristics when k increases.

4.2 Real Data

We also evaluate the heuristics of Section 3 using a real automobile dataset from the UCI Machine Learning Repository [1]. The dataset contains information about various cars. Each car is associated with a number of categorical attributes (e.g. brand, body style, fuel system) as well as a number of continuous attributes (e.g. length, horsepower, price). We use $p_i^{(j)}$ to denote the value of j^{th} attribute for the i^{th} item. Let D_{con} be the set of contimuous attributes of the dataset and D_{cat} be the set of categorical ones. We define the distance d between two items of this dataset P as:

$$\begin{aligned} d(p_i, p_j) &= \frac{1}{|D_{con}| + |D_{cat}|} \left[\sum_{x \in D_{con}} \frac{|p_i^{(x)} - p_j^{(x)}|}{\max_{p_a, p_b \in P} |p_a^{(x)} - p_b^{(x)}|} + \sum_{y \in D_{cat}} \delta_{i,j}^{y} \right], \\ & \text{where } \delta_{i,j}^{y} = \begin{cases} 0, & \text{if } p_i^{(y)} = p_j^{(y)} \\ 1, & \text{otherwise} \end{cases} \end{aligned} \quad (4)$$

In Table 14 we show results for all the examined heuristics for different values of k when choosing diverse items from the whole dataset ($n = 160$, $D_{con} = 16$, $D_{cat} = 10$). We observe similar behavior as in the case of synthetic data. Out of the three greedy heuristics, GC performs better, both in terms of achieved diversity and time. The achieved diversity of GC is 10% higher than in the random case on average. For the interchange and neighborhood families, the best diversity is achieved by SA and FuN respectively. FuN, especially, exhibits low execution time as well, even lower than GC for smaller values of k. For small values of k, FI and BI also exhibit good behaviour but GC scales better with time. As far as KM is concerned, once again, it does not locate better solutions than the random case (RA).

5 Related Work

Although diversity has attracted considerable attention recently, it is not a new concept. [3] proposed a linear combination of diversity along with relevance for the enhancement of user satisfaction by information retriecal systems back in 1998, similar to the approaches more recently followed in [18, 6]. In [18], a method for topic diversification is proposed for recommendations based on the intra-list similarity, a permutation-insensitive metric introduced to assess the topical diversity of a given recommendation list. The proposed algorithm also considers the relevance of the candidate items when creating recommendation lists. [6] applies the concept of ranking based on both relevance and diversity in the context of publish/subscribe systems. The notion of diversity is also explored in database systems. Motivated by the fact that some database relation attributes are more important to the user, [14] proposes a method where recommendation lists consisting of database tuples are diversified by first varying the values of higher priority attributes before varying the values of lower priority ones. When the tuples are associated with scores, a scored variation of the method picks more relevant to the query tuples first. [12] tackles the problem of automatically extracting a set of features from the items of interest that is capable of differentiating the items from each other. [16] formulates the k-diversity problem as an optimization problem. Given the $n \times n$ distance matrix of the candidate items, the goal is to find a binary vector of size n that represents which items belong to the most diverse subset of size k. This is a binary problem which has to be relaxed to a real-valued problem to be solved. Recently, [9] defined a set of natural, intuitive axioms that a diversification system is expected to satisfy. The authors prove that all axioms cannot hold simultaneously and show which of them are satisfied by three main diversification methods.

Besides using content diversity, there is also related work based on different definitions of diversity. [15] proposes computing diversity based on the notion of explanations. The explanation of a given item for a user is the set of similar items the user has rated in the past. Distances between two items are measured based on their corresponding explanations. [11] proposes algorithms to capture diverse concepts in text documents, aiming at retrieving diverse sentences that can be used as snippets from search engines. These algorithms are based on coverage, i.e. how many terms of the document appear in the selected sentences, and orthogonality, i.e. how much the terms appearing in such a sentence differ from those appearing in the rest. Coverage is also considered in [2], which aims at locating diverse documents to answer a user query. Given a taxonomy of topics and a probability distribution for topics relevant to the query, the k-diversity problem is formulated as the location of a set of documents with cardinality k that maximizes the probability of as many topics as possible being represented by that set. In [5], a distinction is made between novelty, i.e. avoiding redundancy, and diversity, i.e. resolving ambiguity. Both user queries and documents are broken down into small pieces of information, called nuggets. Then, diverse documents are retrieved based on a probabilistic model of nuggets being contained in the various queries and documents.

6 Conclusions

In this paper, we employed various heuristics used in the literature for tackling the k-diversity problem and applied them on both synthetic and real datasets in order to compare them and provide a comprehensive experimental evaluation for their performance. We saw that, while the k-diversity problem is NP-hard, a number of heuristics exist that can locate good solutions at reasonable time.

Our future plans include combining diversity with other criteria, such as relevance, for the final ranking of results, especially in the case of the publish/subscribe paradigm, where low execution times are critical, due to the on-line nature of those systems. We are also interested in finding other ways besides the heuristics presented here to achieve diversity. For example, employing some spatial tree structure,

Table 14: Average achieved diversity and execution time for the Automobile dataset.

Automobile dataset																										
k	**GC** ($h = n$)		**GC** ($h = n/2$)		**GC** ($h = 2$)		**GD**		**SGD**		**FI**		**BI**		**SA**		**FiN**		**ClN**		**FuN**		**KM**		**RA**	
	div	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**	**div**	**time**
5	0.51	34.30	0.51	9.30	0.48	1.50	0.50	278.30	0.44	268.90	0.51	7.80	0.50	12.60	0.51	220.40	0.38	0.00	0.35	1.60	0.48	1.50	0.29	4.60	0.28	0.00
10	0.46	34.60	0.46	6.20	0.45	3.10	0.44	278.10	0.41	267.20	0.43	26.70	0.40	20.30	0.46	1148.40	0.35	0.00	0.27	3.10	0.45	4.70	0.30	11.00	0.29	0.00
20	0.43	46.70	0.43	21.80	0.42	6.30	0.40	281.10	0.37	270.30	0.38	70.40	0.35	71.90	0.42	3243.90	0.35	1.60	0.26	15.60	0.41	23.20	0.30	14.10	0.29	0.00
30	0.41	59.40	0.41	31.40	0.41	24.90	0.38	281.10	0.36	267.30	0.36	143.80	0.34	272.00	0.41	7281.30	0.34	1.60	0.29	18.80	0.38	28.20	0.30	31.30	0.30	0.00
40	0.40	76.40	0.40	48.40	0.40	35.80	0.36	276.70	0.34	267.20	0.34	226.30	0.33	425.00	0.40	12114.20	0.31	12.50	0.26	45.10	0.37	71.90	0.30	123.40	0.30	0.00
50	0.39	71.90	0.39	71.90	0.39	53.30	0.35	269.00	0.34	259.30	0.33	389.10	0.33	614.00	0.39	17201.60	0.30	15.60	0.27	65.70	0.35	84.20	0.29	265.00	0.29	0.00
60	0.38	86.10	0.38	86.10	0.38	79.80	0.34	268.70	0.33	253.10	0.33	689.00	0.33	843.90	0.38	18322.00	0.31	17.20	0.26	101.30	0.33	93.70	0.29	1750.00	0.30	0.00

such as the R-tree [13], for indexing candidate items and retrieving diverse items by traversing the tree. Such structures could be used in conjunction with neighborhood heuristics.

References

[1] Automobile Dataset. http://archive.ics.uci.edu/ml/datasets/Automobile.

[2] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM*, pages 5–14, 2009.

[3] J. G. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, pages 335–336, 1998.

[4] H. Chen and D. R. Karger. Less is more: probabilistic models for retrieving fewer relevant documents. In *SIGIR*, pages 429–436, 2006.

[5] C. L. A. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR*, pages 659–666, 2008.

[6] M. Drosou, K. Stefanidis, and E. Pitoura. Preference-aware publish/subscribe delivery with diversity. In *DEBS*, pages 1–12, 2009.

[7] E. Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1):48 – 60, 1990.

[8] E. Erkut, Y. Ülküsal, and O. Yeniçerioglu. A comparison of p-dispersion heuristics. *Computers & OR*, 21(10):1103–1113, 1994.

[9] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.

[10] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[11] K. Liu, E. Terzi, and T. Grandison. Highlighting diverse concepts in documents. In *SDM*, pages 545–556, 2009.

[12] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. *PVLDB*, 2(1), 2009.

[13] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis. *R-Trees: Theory and Applications (Advanced Information and Knowledge Processing)*. Springer, September 2005.

[14] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. Amer-Yahia. Efficient computation of diverse query results. In *ICDE*, pages 228–236, 2008.

[15] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, pages 368–378, 2009.

[16] M. Zhang and N. Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *RecSys*, pages 123–130, 2008.

[17] Y. Zhang, J. P. Callan, and T. P. Minka. Novelty and redundancy detection in adaptive filtering. In *SIGIR*, 2002.

[18] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, pages 22–32, 2005.