Universität Augsburg

Multidimensional Clustering Approaches for Pareto-Frontiers

J. Kastner, M. Endres

Report 2017-02 June 7, 2017

Institut für Informatik
D-86135 Augsburg

Multidimensional Clustering Approaches for Pareto-Frontiers

Johannes Kastner, Markus Endres

Institut für Informatik, Universität Augsburg, D-86135 Augsburg, Germany
{johannes.kastner,markus.endres}@informatik.uni-augsburg.de

Abstract. In Data Mining large and increasing sets of data are becoming more and more common. In order to avoid losing the overview on these data-sets, preference queries are a very popular method to reduce quantities of data to high relevant information. Together with clustering methods like k-means, confusing sets of objects can be constituted and presented clearer in order to get a better overview. In this report we present on the one hand the Pareto-dominance as a very suitable and promising approach to cluster objects over better-than relationships. In order to meet someones desires, one can tip the balance of the final results to the more favored dimension if no decision for allocating objects is possible. On the other hand we introduce based on the Pareto-dominance an advanced clustering approach exploiting the Borda Social Choice voting rule to manage distances of different domains by equally weights during the clustering process.

Keywords Skyline, Pareto-Frontier, Clustering, k-Means, Borda, Social Choice

1 Introduction

Recommendations in online shopping, movie-on-demand services like Amazon or Netflix, or music-streaming services like Spotify or Deezer are becoming more and more popular. Since the data is growing, too, it is very difficult to keep track of the presented results, because the users are overwhelmed with partly unwanted results.

One solution to get less confusing recommendations is to exploit the users preferences in order to get only some best matches using Pareto-frontiers. But for more detailed preference-based wishes, the number of results which are returned, known as Pareto-optima, are growing aswell. Clustering these results is a very promising opportunity to present less but representative results to the users. For example, in a recommender system which addresses people with a similar taste of movies and series, e.g., Netflix or Amazon Instant Video, the following use-case is very common.

Example 1 *Assume user Alex wants to watch a movie. He prefers movies with a possible low* running time *and a recent release year at the same time. The movies presented to Alex are so-called Pareto-optima on a Pareto-frontier or Skyline. An object is a Pareto-optima if no other object dominates this object w.r.t. the given preferences. In Fig. 1, e.g., P_4 has a more recent release year than P_{14} and a lower running time at the same time. So P_4 dominates P_{14} in both dimensions at the same time w.r.t. Alex' preferences, while P_3, e.g., is dominated by P_4 only w.r.t. the running time. Since the set of Pareto-optima can also get very enormous and confusing, clustering is a very promising approach to compress this set and express it with a smaller appropriate set of representatives. Another approach is to mask out undesirable results, e.g., users like P_8 who have a more recent release year, but unfortunately a very high running time. These two attempts can only be reached with some kind of clustering like k-means.*

However, since we have two dimensions with different domains, it is hardly possible to achieve a useful outcome without great afford, because the dimensions should be set into relation to each other. In our example it is very circumstantial to set this range of minutes for the runningtime in relation to only a smaller range of release-years. Since every user has diverse requirements in such a recommender system, it is by far not sufficient to use the basic k-means clustering algorithm along with the well-known Euclidean distance.

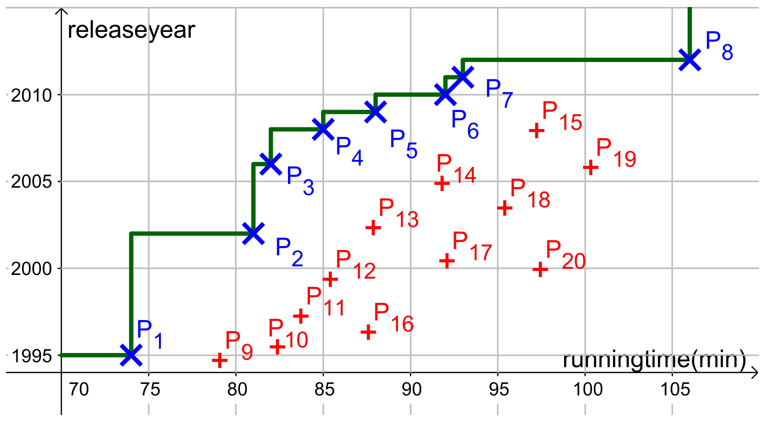

Fig. 1: Pareto-frontier of users with preferably lowest runningtime and preferably recent release-year.

The first approach presented in this report uses the Pareto-dominance in order to cluster more efficient regarding quality of the final clusters and runtime over better-than relationships unlike mapped Euclidean distances, which have to be adjusted inconveniently for each use case.

Alex' movie-search from Example 1, where only two preferences are considered, is a common use case. But more often the following use case, where more preferences are considered, is very conventional, too.

Example 2 *Assume Alex wants to watch a movie again. His most favored* actors *Tom Hanks, Johnny Depp, Bruce Willis, Natalie Portman, Mila Kunis, Samuel L. Jackson, Benedict Cumberbatch, Scarlett Johansson and Kate Beckinsale should possibly get consulted. Since it is late in the evening Alex possibly wants to watch movies, which* runtime *should be* around 120 minutes. *Moreover Alex is in the mood for* movie genres *like Action- and Comedymovies, Thrillers and Science-Fiction. At the same time he favors recommendations for movies, which have a possibly* high user-rating *and were released* around the year 2000. *The possibly best matches can be seen in Table 1.*

We used Pareto-dominance for the cluster allocation in the basic k-means clustering algorithm to find a cluster, which dominates all other clusters for each object in our former workshop paper [8] in a 2-dimensional set of objects like in Example 1. Using Pareto-dominance avoids the process of adjusting the domains and setting them into relation to each other before the clustering process. Moreover the user can influence on the clustering process by weighting one of the dimensions, if there is more than one Pareto-optimal cluster for each object, to gain the allocation to one and only one cluster.

In Example 2 the Pareto-dominance clustering approach presented in [8] stretches to its limits, because of five dimensions, which have to be considered. So our Pareto-dominance clustering approach needs a more precise decision criterion for the cluster allocation of the given objects, than the Pareto-dominance, which leads often to the appearance of Pareto-optima. We decided to consider not only the closest centroids for each dimension, but rather the second-, third-, etc. closest centroids, as well in contrast to the Pareto-dominance approach. Finally we used the Borda Social Choice voting rule [13] in order to assign each point to one and only one cluster by ordering and weighting the distances from each object to each cluster for all dimensions. Eventually the highest sum over all dimensions decides which centroid is the most appropriate. Using Borda has the advantage that all distances for each dimension and cluster-centroids are considered, while the Pareto-dominance only adduces the closest centroids in each dimension. So the opportunity of more than one suitable cluster centroid is minimized. Furthermore no adjustments before the clustering process are needed, too.

The rest of the report is organized as follows: We explain the basic background knowledge in Section 2. Our Pareto-dominance clustering approach and our Borda Social Choice Clustering approach iare described and compared in Section 3 and 4. After that, we discuss experiments, where runtime, number of iterations and quality of the final clusters are considered and compared to other clustering algorithms in Section 5. We present related work in Section 6. Finally in Section 7 the results are summarized and we give an outlook on future work.

Table 1: Table of Alex' 5-dimensional movie-search with preference scores for the Pareto-frontier exposition.

ID	Movie	rank	time	year	actorscore	genrescore
0	The Shawshank Redemption	9.3	142	1994	0.000	0.000
1	Pulp Fiction	9.0	154	1994	0.033	0.250
2	The Dark Knight	9.0	152	2008	0.000	0.400
3	Inception	8.8	148	2010	0.000	0.600
4	Goodfellas	8.8	146	1990	0.006	0.167
5	The Silence of the Lambs	8.7	118	1991	0.000	0.200
6	American History X	8.6	119	1998	0.000	0.000
7	Leon	8.6	110	1994	0.012	0.200
8	Memento	8.6	113	2000	0.000	0.250
9	Terminator 2	8.6	137	1991	0.000	1.000
10	Terminator 2	8.6	154	1991	0.000	1.000
11	Terminator 2	8.6	152	1991	0.000	1.000
12	Reservoir Dogs	8.4	099	1992	0.000	0.250
13	The Avengers	8.3	143	2012	0.024	0.333
14	Kill Bill: Vol. 1	8.2	112	2003	0.000	0.500
15	V for Vendetta	8.2	132	2005	0.009	0.500
16	The Sixth Sense	8.2	107	1999	0.018	0.200
17	Slumdog Millionaire	8.1	120	2008	0.000	0.200
18	Black Swan	8.1	108	2010	0.030	0.200
19	Kill Bill: Vol. 2	8.0	137	2004	0.014	0.500
20	District 9	8.0	112	2009	0.000	1.000
21	Iron Man	7.9	126	2008	0.011	0.500
22	The Hunger Games	7.2	142	2012	0.000	0.750

2 Background

Before explaining our clustering frameworks, we describe some important concepts. One basic point for all clustering approaches is the k-means clustering algorithm. Another important point is the preference framework in order to shape desire-based queries, that should be clustered. Furthermore we used several distances measures and the Borda Social Choice voting rule in order to allocate objects to clusters. The most important of them are described in this section, too.

2.1 Clustering

k-means Since k-means clustering is the base for our approaches, we shortly describe the algorithm, as it is presented in [7], cp. Algorithm 1.

1. Find an initial partition for the cluster centroids by choosing a random point of set X for each of the k centroids in line 2.
2. Calculate for each point the distances to all centroids by Function 1. The point is being allocated to the closest centroid in line 9 of Alg. 1, by using, e.g., the Euclidean distance (line 5 in Func. 1).
3. Recalculate the centroids by averaging the contained points in Alg. 1, line 14.
4. Proceed with step 2 until two succeeding clusterings are stable, which means that all clusters from the last iteration contain the equal set of points as in the current iteration (Alg. 1, line 6).

k-means++ The k-means++ algorithm is a version of the basic k-means clustering algorithm where the initial partition is not chosen arbitrary by a random, but by a randomized seeding technique in order to speed up the clustering process. In particular the possibly best centroids for the initial partition should be found in order to reach more accurate clusterings faster in comparison to using the basic k-means clustering algorithm. The k-means algorithm as presented in Algorithm 1 is modified in line 2, as it is mentioned in [1].

1. Arbitrary choose a point of the set X as first cluster centroid c_0.
2. For each further cluster centroid c_i choose $x \in X$ with a probabiltity of $p(x) = \frac{D(x)^2}{\sum_{x \in X} D(x)^2}$ where $D(x)^2$ is the shortest squared Euclidean distance from a point to the already chosen closest centroid until we have k centroids.
3. Proceed with the k-means clustering algorithm in line 3 of Algorithm 1.

In Example 3 the k-means clustering algorithm is presented with Euclidean distance as distance measure.

Example 3 (k-means euclidean norm)
Consider the users in Figure 2, shown as points on a Pareto-frontier. Now we want to get three clusters.

– *Iinitialize the centroids of the three desired clusters randomly with P_2, P_5 and P_8 for C_0, C_1 and C_2.*
– *Calculate for each user $P_1, ..., P_8$ the distances to all three centroids using the Euclidean distance. P_1, P_2 and P_3 are allocated to C_0, while the users P_4, P_5, P_6 and P_7 are assigned to C_1. Finally C_3 receives the user P_8.*
– *The new centroids W_0, W_1 and W_2 of each cluster are averaged based on the points contained in the clusters. The result is shown in Figure 2.*

2.2 Preferences

Preferences represent wishes which should be fulfilled. In database systems a preference $P = (A, <_P)$ is modeled as a strict partial order on the domain of A, *dom*(A). The term $x <_P y$ can be described as "I like y more than x". In order to construct preferences there are several *preference constructors* published in [9, 10], which are mentioned in the following Subsections.

Algorithm 1 k-means-clustering

Input: Set $X = \{x_i \mid i = 1, ..., n\}$ of d-dimensional points $x_i = (x_{i1}, ..., x_{id})$ of size n, set of k cluster centroids $C = \{c_j \mid j = 1, ..., k\}$.
Output: Sets $E_k = \{e_j \mid j = 0, ..., m\}$ of clustered d-dim. points $e_i = (e_{i1}, ..., e_{id})$.

```
 1: function K-MEANS($X, C$)
 2:     $C \leftarrow random(X, k)$                          // Random initial partition for k centroids
 3:     $E \leftarrow \emptyset$                                  // Current clustered set
 4:     $E' \leftarrow \emptyset$                                 // Clustered set from last iteration
 5:     $equals \leftarrow false$                             // Variable for stop-criterion
 6:     while !$equals$ do                       // Termination-criterion: Old Set equals current set
 7:         for $x_i \in X$ do                                // Iterate through all points
 8:             $id \leftarrow getClosestCentroidId(x_i, C)$      //  Get id of closest cluster
 9:             $E_{id} \leftarrow E_{id} \cup \{x_i\}$         // Add the current point to the closest centroid
10:         end for
11:         $equals \leftarrow checkClusterings(E, E')$           // Check if clusters are equals
12:         $E' \leftarrow E$                          // Save the current clustering for next iteration
13:         for $E_i \in E$ do
14:             $c_i \leftarrow recalculateCentroid(E_i)$        // Recalculate centroids of each cluster
15:         end for
16:     end while
17:     return $E$
18: end function
```

Function 1 Closest centroid for traditional distances

Input: d-dim. point $x_i = (x_{i1}, ..., x_{id})$, set of k centroids $C = \{c_j \mid j = 1, ..., k\}$.
Output: *id* of the closest centroid, the point gets allocated to.

```
 1: function GETCLOSESTCENTROIDID($x_i, C$)
 2:     $dist \leftarrow maxDistance$                            // Distance to closest centroid
 3:     $id \leftarrow 0$
 4:     for $c_j \in C$ do                                   // Iterate through all clusters
 5:         $d \leftarrow distance(x_i, c_j)$            // Current distance between point and cluster
 6:         if $d < dist$ then
 7:             $dist \leftarrow d$                // Replace closest distance if current distance is closer
 8:             $id \leftarrow j$                      // Replace id if current id has closer distance
 9:         end if
10:     end for
11:     return $id$
12: end function
```

Base Preferences The numerical BETWEEN($A, [low, up]$) preference for example prefers values between a *lower* and an *upper* bound. If no value in $[low, up]$ is present, return the values with the lowest distance to the interval. LOWEST(A) and HIGHEST(A) correspond to the *minimum* and *maximum* preference, respectively. If we restrict the attention to LOWEST/HIGHEST as input preferences, then Pareto preference queries coincide with the traditional Skyline queries [3]. Sub-constructors of BETWEEN like AROUND, HIGHEST, LOWEST, MORE THAN and LESS THAN can be described as BETWEEN aswell, e.g. $AROUND_d(A, z) := BETWEEN_d(A, z, z)$. The other numerical constructors are self-explaining.

There are also preference constructors for categorical domains, e.g., LAYERED$_m(A, (L_1, \ldots, L_m))$, where L_i partitions $dom(A)$ with disjoint layers. Thereby L_1 contains the most preferred values, L_2 the second preferred values, etc.

Pareto Preference The most important preference is the well-known Pareto preference which models equal importance where base preferences are used to build Pareto preferences in an intuitive way. A Pareto preference $P := P_1 \otimes P_2 = (A_1 \times A_2, <_P)$ with preferences $P_i = (A_i, <_{P_i})$ and tuples $x = (x_1, x_2), y =$

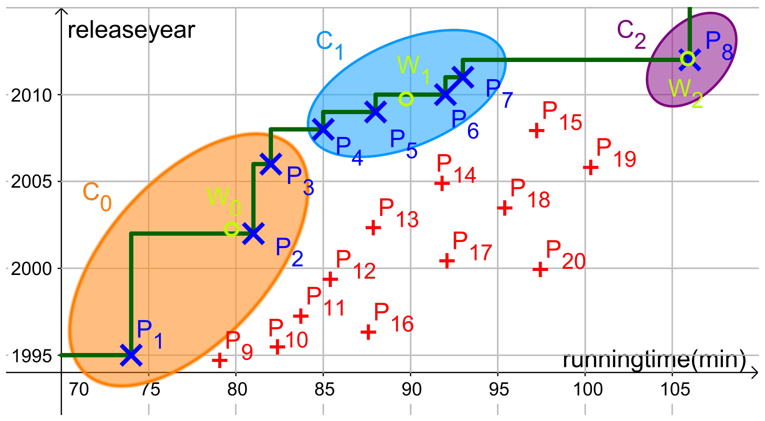

Fig. 2: k-means clustering of users with the Euclidean norm after the first iteration.

$(y_1, y_2) \in dom(A_1) \times dom(A_2)$ is defined as follows:

$$(x_1, x_2) <_P (y_1, y_2) \Leftrightarrow (x_1 <_{P_1} y_1 \wedge (x_2 <_{P_2} y_2 \vee x_2 = y_2)) \vee (x_2 <_{P_2} y_2 \wedge (x_1 <_{P_1} y_1 \vee x_1 = y_1)) \quad (1)$$

2.3 Similarity Measures

There are several distance measures presented in [2], which we use in our paper to calculate similarities between Objetcs.

Euclidean distance The Euclidean distance can be seen in Eq. 2. Given two points x_i, x_j with d dimensions, the particular squared distances regarding each dimension are summed up and rooted after that.

$$dist_{ij} = \sqrt{\sum_{k=1}^{d} (x_{ik} - x_{jk})^2} \quad (2)$$

Canberra norm In order to set the focus on distances using small domains as well, the Canberra distance is a very promising measure. It sums up the absolute fractional distances of two d-dimensional points x_i, x_j in relation to the range of the focussed dimension for all dimensions, cp. Eq. 3.

$$dist_{ij} = \sum_{k=1}^{d} \frac{|x_{ik} - x_{jk}|}{(x_{ik} + x_{jk})} \quad (3)$$

Jaccard Jaccard is a measure for similarity of sets. In order to handle categorical preferences in our framework as well, we need to express categorical preferences with numerical values. Given two d-dimensional sets of objects A and B, the similarity of these two sets is calculated as follows, as it is presented in [15].

$$J(A, B) = \frac{A \cap B}{A \cup B} \quad (4)$$

where $A \cup B$ represents the number of simultaneous presence in both sets, while $A \cap B$ represent the union set of the two sets.

Example 4 *Remember Alex' favourite genres from our starting example as Object* $x_{Alex} = \{Action(Ac), Comedy(Co), Thriller(Th), Sci-Fi(Sc)\}$ *and the movies as Objects* x_j *with their genres in Table 1. Now in Table 2 Jaccard-Scores for some movies are calculated.*

2.4 Borda Social Choice Voting Rule

For our approach regarding objects with more than two dimensions Borda Social Choice is a promising method, because every candidate receives equal weighted votes from each voter, as it is defined in [13].

Given m candidates C_i, $i = 1...m$ and d voters V_j, $j = 1...d$ where every voter votes for each candidate. Each voter V_j has to allocate the votings v_{jk} from a pairwise distinct set of $k = 0...m-1$. After all voters assigned their votes, the votes for each candidate are summed up as it can be seen in Eq. 5, while the Borda winner is determined as mentioned in Eq. 6.

$$bordasum_{C_m} = \sum_{j=1}^{d} v_{jm} \tag{5}$$

$$bordawinner_{C_m} = \max\{bordasum_{C_m}\} \tag{6}$$

If we depict this approach to our clustering-framework we replace the candidates with the available clusters and the voters with the d-dimensional object which should be allocated to one and only one cluster. Finally each object votes for each cluster and after the voting Eq. 5 determines the sum of all votes for each cluster and Eq. 6 the winner, which got the most votes.

Finally, dimensions, which would not be considered because of a smaller domain, get equal weighted votes like the other dimensions and have a higher influence on the clustering process.

Table 2: Calculation Jaccard score in a movie scenario. Action(Ac), Crime(Cr), Thriller(Th), Drama(Dr), Adventure(Ad), Sci-Fi(Sc)

Movie	Genres	Jaccard-Score
Pulp Fiction	{Cr,Th}	0.2
The Dark Knight	{Ac, Cr, Th, Dr}	0.33
The Hunger Games	{Ac, Sc, Ad, Th }	0.4

Table 3: Votings in Borda Social Choice.

	C_1	C_2	C_3	...	C_m
V_1	v_{11}	v_{12}	v_{12}	...	v_{1m}
V_2	v_{21}	v_{22}	v_{23}	...	v_{2m}
V_3	v_{31}	v_{32}	v_{33}	...	v_{3m}
...	...	...	...	...	...
V_d	v_{d1}	v_{d2}	v_{d3}	...	v_{dm}

3 Solution for the 2-Dimensional Case

In this section we describe the Pareto-dominance framework in detail for two-dimensional use cases. While a Pareto preference is used to determine the importance of preferences, the *Pareto-dominance* in our approach is used to *allocate an object to the possibly best cluster*, which is not dominated by other clusters w.r.t. the distances of the individual objects, by using the Euclidean norm for one-dimensional distances. Moreover the Pareto-dominance can additionally be used to find the centroids closest point on the Pareto-frontier as new centroid in each cluster.

3.1 Cluster Allocation

Consider the Pareto-frontier in Figure 3, presenting users w.r.t. a possibly high music-matching score and a possibly close distance. The aim is to get three promising clusters C_1, C_2 and C_3. The points P_2, P_5 and P_8 are chosen as the cluster centroids. After that, for each point $P_1, ..., P_{10}$ the particular distances of the x- and the y-dimension to the cluster centroids are calculated as it can be seen in Table 4. Furthermore the y-dimension, representing the music-matching score is chosen as more important than the x-dimension at the appearance of Pareto-optimal cluster-centroids. This so-called one-dimensional clustering realizes, that a decision is arrived for each object at the cluster allocation. In Figure 3 a clustering after the first iteration is shown.

Table 4: Distances of each user to each cluster centroid. $C_1 := P_2, C_2 := P_5, C_3 := P_8$, x-dim. = distance, y-dim. = music matching score.

	$d_{x_{C_1}}$	$d_{y_{C_1}}$	$d_{x_{C_2}}$	$d_{y_{C_2}}$	$d_{x_{C_3}}$	$d_{y_{C_3}}$
P_1	**27.44**	**0.01**	120.72	0.06	264.60	0.09
P_2	**0.00**	**0.00**	93.27	0.05	237.16	0.09
P_3	**1.66**	**0.01**	91.61	0.04	235.50	0.07
P_4	**41.27**	0.04	52.00	**0.01**	195.89	0.05
P_5	93.27	0.05	**0.00**	**0.00**	143.89	0.04
P_6	141.27	0.06	**48.00**	**0.016**	95.89	0.023
P_7	150.88	0.09	**57.61**	0.04	86.28	**0.00**
P_8	237.16	0.09	143.89	0.04	**0.00**	**0.00**
P_9	311.32	0.09	218.05	0.04	**74.16**	**0.00**
P_{10}	323.08	0.10	229.81	0.06	**85.92**	**0.02**

Now we shortly explain the allocation of the given users to the clusters:

- The users P_1 and P_3 are assigned to cluster C_1, because the centroid of C_1 dominates the other two centroids regarding the distances in both dimensions.
- P_4 has *2 Pareto-optima*, because of the closer distance to C_1 regarding the x-dimension and to C_2 regarding the y-dimension. Hence C_1 and C_2 are Pareto-optimal w.r.t. to the x- and y-dimension. Now the one-dimensional clustering tips the balance to C_2.
- P_6 and P_8 are allocated to Cluster C_2, because of the closer distances to the centroid of C_2 in both dimensions.
- P_7 is closer to C_2 concerning the x-dimension, but has a smaller distance to the centroid of C_3 regarding the y-dimension. This ensures that P_7 is allocated to C_3.
- P_8 and P_{10} are allocated to cluster C_3, because of the existence of only one Pareto-dominant cluster centroid namely C_3.

In order to explain our approach in detail we draw the attention to Figure 4, which shows a snippet of the Pareto-frontier of Figure 3. While P_4 is assigned to C_1 regarding the smaller Euclidean distance of 41.27 unlike a distance of 52.00 to C_2, the Pareto-dominance approach shows its versatility. The user can influence

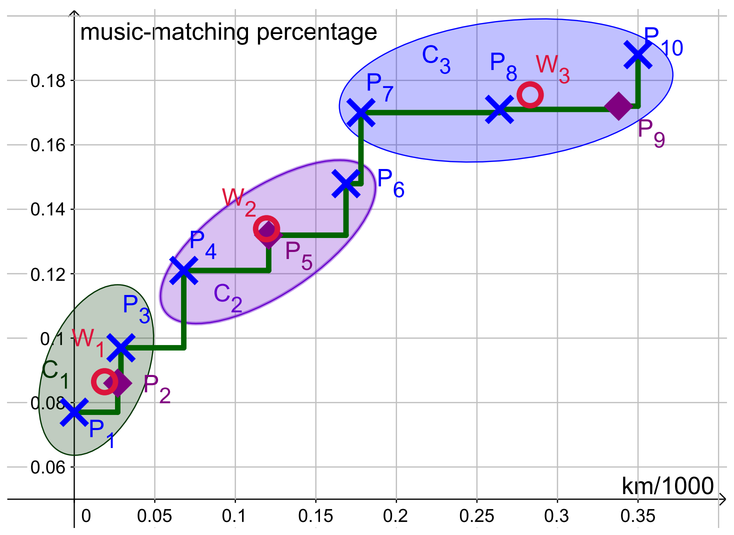

Fig. 3: Clustering of users with the Pareto-dominance after the first iteration.

on the clustering by choosing one dimension as the more important at the appearance of Pareto-optima. If he chooses the x-dimension, as more important P_4 will be assigned to C_2. This ensures that each user will be allocated to one and only one cluster, to avoid overlapping and imprecise clusters. This example shows, that a Pareto-dominant clustering combined with a one-dimensional clustering at the appearance of Pareto-optima tends to a hierarchical clustering. Users with similar scoring values w.r.t. the music matching score are clustered together, unlike in the basic k-means clustering. In particular cluster C_1 and C_2 contain users with very similar music-matching scores, where the range between the two boundary points is very small unlike the k-means clustering approach. So if P_7 is allocated to C_2 and P_4 to C_1 the users contained in the clusters are not as similar as in our approach.

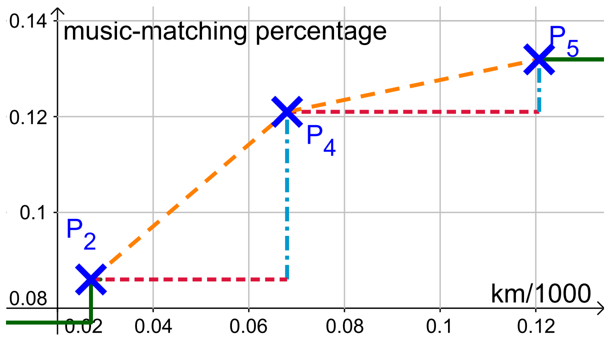

Fig. 4: Camparison of Pareto-dominance and Euclidean Distance. $C_1 := P_2, C_2 := P_4$.

3.2 Cluster Centroids

After each user is allocated to a cluster, the cluster-centroids are recalculated regarding the contained users. For each cluster all values for the x-dimension and y-dimension are averaged, which can be seen in Figure 3 as W_1, W_2 and W_3.

For each cluster-centroid W_1, W_2 and W_3 the closest Pareto-dominant user in each cluster is selected as the new cluster-centroid for the next iteration. In order to find these users, the particular distances regarding the two dimensions are calculated, which can be seen in Table 5. Figure 3 shows the calculated centroids before assigning and the new centroids aswell.

- P_2 is the new cluster centroid of C_1, because of the closest distance of each x- and y-dimension to W_1.
- For cluster C_2 P_5 is allocated as new centroid because of the closer distances in both dimensions, too.
- P_8 and P_9 both are Pareto-optima for the allocation of the cluster centroid of C_3, because P_9 is closer to W_3 regarding the y-dimension and P_8 regarding the x-dimension. The one-dimensional clustering determined by Bob tips the balance to P_8 as new cluster centroid.

Table 5: Particular distances of recalculated cluster-centroids W_1, W_2, W_3 to the points in each cluster.

	dx_{W_1}	dy_{W_1}	dx_{W_2}	dy_{W_2}	dx_{W_3}	dy_{W_3}
P_1	18.848	0.010	—	—	—	—
P_2	**8.595**	**0.001**	—	—	—	—
P_3	10.253	0.010	—	—	—	—
P_4	—	—	50.667	0.012	—	—
P_5	—	—	**1.331**	**0.002**	—	—
P_6	—	—	49.331	0.014	—	—
P_7	—	—	—	—	104.730	0.004
P_8	—	—	—	—	**18.451**	0.004
P_9	—	—	—	—	55.709	**0.003**
P_{10}	—	—	—	—	67.4726	0.0120

3.3 Complexity

The first implementation of our Pareto-dominant clustering approach was realized as a Java program. The clustering algorithm reaches a complexity of $\mathcal{O}(n \cdot c)$ where n is the number of points that should be clustered in c clusters. In each iteration for every point n the distances to each cluster c are calculated in $\mathcal{O}(c)$ and the best centroid is chosen subsequently after the distance-calculation.

4 Solution for the *d*-Dimensional Case

In this section we present our Borda clustering framework in detail, which is a better solution for clustering multi-dimensional objects. While the Pareto-dominance works well for a two-dimensional clustering, for higher dimensions Pareto-dominance is not suitable because of a higher probability of the occurence of Pareto-optima.

Thus we need to find a decision criterion for the allocation of each point to one and only one cluster. For this the Borda Social Choice voting rule is a very promising approach, which allows more influence of smaller domains and incorporates each distance as a weighting, whether they are the best regarding one dimension or not.

4.1 Cluster Allocation

Remember the k-means clustering algorithm from Section 2.1. We modified Algorithm 1, in order to realize a clustering with Borda Social Choice as decision criterion for the cluster allocation. Line 8 of Algorithm 1 is replaced by the function presented in Function 2 in order to perform the Borda voting rule.

In Function 2 the Borda votes for each point x_i to each cluster is assigned as explained in Eq. 5. For this an array is created in line 2 in order to sum up the votes for the clusters. Thus the distances in each dimension between the considered point and the cluster centroid are calculated, which is performed in Func. 3. For each dimension these distances are calculated and saved together with the id in an object-based data-structure in line 6 and 7. After that the object is set on the current last position of the array in line 8 and sorted with *insertion sort* between line 10 and 22 to the right position according to the distances (line 12), in order to avoid a further sorting if all objects would be added unsorted to the array.

While the last distance is sorted to the right position in the array, the votes for each cluster are determined (line 17) and summed up in the array (line 19) for the absolute votes, in order to avoid another iteration through the array afterwards. After the array with the summed up votes is returned from Function 3, it is sorted with *insertion sort* in line 3 of Function 2 in order to find the Borda winner as explained in Eq. 6. If there is only one Borda winner (line 5), the id of the centroid is returned.

We noticed after some preliminary tests, especially for higher dimensions and higher number of clusters, that there are some problems regarding the convergence, which will be explained more explicitly in Section 4.2. We decided to add a specific decision criterion for the cluster allocation at the appearence of more than one Borda winner, in order to solve this problem. Our approach looks back to the last iteration, in order to check if the centroid was allocated to one of the Borda winners (Func. 2, line 10). If not, one of the centroids ids, which is represented in the Borda winners, is randomly chosen in line 19. Finally the k-means clustering continues in line 9 of Function 1.

4.2 Convergence

After some preliminary tests, we found out, that especially for growing numbers of dimensions, our former approach was not terminating, which leads to alternating between two or more clusters. So in addition to our Borda Clustering Framework we decided to adjust Step 1 of the k-means clustering algorithm in Algorithm 1 (line 2) by using k-means++. Using k-means++ ensures especially in higher dimensions that our approach terminates.

One more problem concerning the initial partition is resulting in empty clusters. If the first centroid was randomly chosen, the probability that a direct neighbor of the first centroid will be chosen is very slight. Especially, if there are, e.g., different movies with almost the same specifications from our starting example in Table 1 like Inception (ID 3) and Goodfellas (ID 4) or duplicate movies except for the runningtime, which results because of different ranked versions, like Terminator 2, the possibility is given, that these movies both are chosen as cluster centroids. If so the order of the cluster centroids decides, that the first of the regarding clusters will be populated with objects, while the following neighbor or duplicate will stay empty. K-means++ minimizes these problems and furthermore cares that the runtime and the number of iterations will decrease compared to the Borda Clustering Framework without using k-means++.

Function 2 Closest centroid for Borda Social Choice voting rule

Input: d-dim. point x_i, clusterset C, cluster-id last iteration id_{last}.
Output: id of the closest cluster for point x_i.

```
function GETBESTCLUSTERID($x_i, C, id_{Last}$)
    $bordaVals[] \leftarrow calculateBordaVals(x_i, C)$                    // calculate Borda values
    $insertion\_sort(bordaVals[])$                                          // Sort Borda values
    $counter \leftarrow 1$
    if $bordaVals[0].val > bordaVals[1]$ then                              // Only one Borda winner
        return $bordaVals[0].id$                                            // Return id of first element
    end if
    if $bordaVals[0].val == bordaVals[1]$ then                             // More than one Borda winner
        while $bordaVals[counter].val == bordaVals[0].val$ do
            if $bordaVals[counter].id == id_{last}$ then                   // Check $id_{last}$
                return $id_{last}$                                          // Return id of element from last iteration
            end if
            $counter \leftarrow counter + 1$
            if $counter >= bordaVals[].Size()$ then                        // All centroids Borda winner
                return $bordaVals[random(0, counter - 1)].id$               // Return random id
            end if
        end while
    end if
    return $bordaVals[random(0, counter - 1)].id$                           // Return random id
end function
```

Function 3 Calculate the Votes for each centroid

Input: d-dim. point x_i, clusterset C.
Output: bordaVals for each centroid.

```
function CALCULATEBORDAVALS($x_i, C$)
    $bordaVals[] \leftarrow bordaVals[C.size()]$
    for $i = 0$; $x < x_i.getDims().size()$; $i \leftarrow i + 1$ do
        $bordaObj[] \leftarrow bordaObj[C.size()]$                          // Array for voters
        for $k = 0$; $k < C.size()$; $k \leftarrow k + 1$ do                // For each voter(centroid)
            $val = \mid x_i.getDims()[i] - c_j.getDims()[i] \mid$           // Calculate distance
            $obj \leftarrow newbordaObj(val, c_j.id)$                       // Init. object for voter
            $bordaObj[k] = obj$                                             // Set object on last available position
            $voteDiff \leftarrow k + 1$                                     // Difference for Votes
            for $j = k; j >= 0; j \leftarrow j - 1$ do                      // Start insertion_sort
                if $j > 0$ then                                             // Check if not last item is sorted
                    if $obj.val < bordaObj[j-1].val$ then                   // Compare obj. pairwise
                        $swap(bordaObj[j-1], bordaObj[j])$                  // Swap objetcs
                    end if
                end if
                if $k == C.size() - 1$ then                                 // Last Element is sorted
                    $val \leftarrow bordaObj[].size() - voteDiff$           // Determine Votes
                    $id \leftarrow bordaObj[j].id$                          // Fetch id of Object to sum up
                    $bordaVals[id].val \leftarrow bordaVals[id].val + val$  // Sum up the value
                    $voteDiff \leftarrow voteDiff - 1$                      // Reduce voteDiff for growing values
                end if
            end for
        end for
    end for
    return $bordaVals[]$
end function
```

A problem regarding the convergence in higher dimensions was solved as follows. For each iteration we save the clusterid for each point, the point got allocated after line 9 in an array, which will be created after line 4 in Algorithm 1. Now if there are more than one Borda winners in line 8 of Function 2, it is checked if the ID of the centroid from the last iteration is represented in any of these Borda winners. If so, the object goes to the same cluster as in the last iteration. Finally preliminary benchmarks showed, that this solution ensures that the clusters are becoming stable in a less number of iterations.

4.3 Example

We want to present our approach based on Alex' movie query from Example 2, which refers to the snippet of a movie data set in Table 1. The advanced Borda approach with k-means++ and the lookback to the last iteration at the appearance of more than one Borda winner will be considered.

Initial Partition with k-means++

- The Movie Leon (ID 7) is chosen randomly as centroid C_1. After that the distances for each point to the current centroid C_1 are calculated as explained in Sec. 2.1 Step 2, which can be seen in Table 6.
- The movie is set as new centroid, which exceeds a determined random value while iterating through the set. Since the sum of all previous movie-values for Iron Man(ID 21) and itself is 34921.6, it becomes the new centroid C_2, because a determined random-value for this centroid was 34775.
- For C_3 The Shawshank Redemption (ID 0) is determined likewise C_2 after updating the distances shown in Tab. 6. If a current distance is smaller than a previous one, the current is set and the probability to be chosen decreases.

Table 6: Creating the initial cluster partition with kmeans++.

ID	$D(x)^2$	$\sum$	ID	$D(x)^2$	$\sum$
01	6896.62	6896.62	**21**	**205.80**	**34921.6**
00	1445.30	7206.92	22	1822.62	36744.20
	...		04	2720.41	39464.61
20	0140.00	34715.78	15	0052.50	39517.11

Allocation with Borda Social Choice

- We show the allocation for the movies Pulp Fiction (ID 1) and The Hunger Games (ID 22) to the initialized centroids C_1, C_2 and C_3 with Borda Social Choice, as it can be seen in Table 7.

Table 7: Cluster allocation for some movies. Dim_a := *actorscore*, Dim_g := *genrescore*, Dim_r := *rating*, Dim_y := *release_year*, Dim_t := *runnting_time*

	Pulp Fiction (ID 1)			The Hunger Games (ID 22)		
	$D(C_1)$	$D(C_2)$	$D(C_3)$	$D(C_1)$	$D(C_2)$	$D(C_3)$
Dim_a	0.017 (2)	0.029 (1)	0.032 (0)	0.016 (0)	0.004 (1)	0.000 (2)
Dim_g	0.008 (2)	0.290 (0)	0.287 (1)	0.508 (0)	0.210 (2)	0.213 (1)
Dim_r	0.575 (1)	0.920 (0)	0.400 (2)	1.225 (1)	0.880 (2)	1.400 (0)
Dim_y	0.875 (1)	0.600 (2)	3.000 (0)	6.875 (0)	5.400 (1)	3.000 (2)
Dim_t	19.38 (1)	27.20 (0)	10.40 (2)	7.375 (1)	15.20 (0)	1.600 (2)
$\sum$	**7**	3	5	2	6	**7**

- Pulp Fiction has the closest distance of 0.017 to C_1 w.r.t the actorscore, so the voting will be 2. C_2 is the second-closest, so it receives a voting of 1 while C_3 has the largest distance and receives 0. For all other dimensions the votings are assigned likewise.
- After all dimensions got processed, the votings for each centroid are summed up as mentioned in Eq. 5. Finally C_1 receives a total of 7 votes, while C_3 gets 5 votes and C_2 only 3 votes. According to Eq. 6 Pulp Fiction will go to the Borda winner C_1.
- In contrast to Borda, using Pareto-dominance would provide the following result. We would have 3 different Pareto-optima, because C_1 would dominate the other centroids w.r.t the actor- and genre-score, C_2 would dominate the others w.r.t the year and C_3 regarding the other both dimensions. Even if we would consider the number of dominating dimensions, there would be a draw between C_1 and C_3 because of 2 dominating dimensions. Borda finally provides a distinct result compared to the Pareto-dominance.
- If there would be a draw, e.g. for the movie The Hunger Games considering C_2 and C_3, which would get a total of 7 votes, the cluster-ID of the previous iteration would tip the balance to one of these both centroids if the ID is represented in the current set of Borda winners. Else the movie will be allocated by random. Finally The Hunger Games goes to C_3 because C_3 is the Borda winner with a total of 7 votes.
- The algorithm continues with the recalculation of the centroids and the check of the termination criterion as mentioned in Section 2.1.

The final clusterings after the fifth iteration can be seen in the Figures 5, 6, 7 and 8. In this figures the dimension for rank is chosen as fixed dimension for all other dimensions in order to compare them easily. Finally in the two-dimensional illustrations several clusterings around the centroids C_Q, C_P and C_R can be seen.

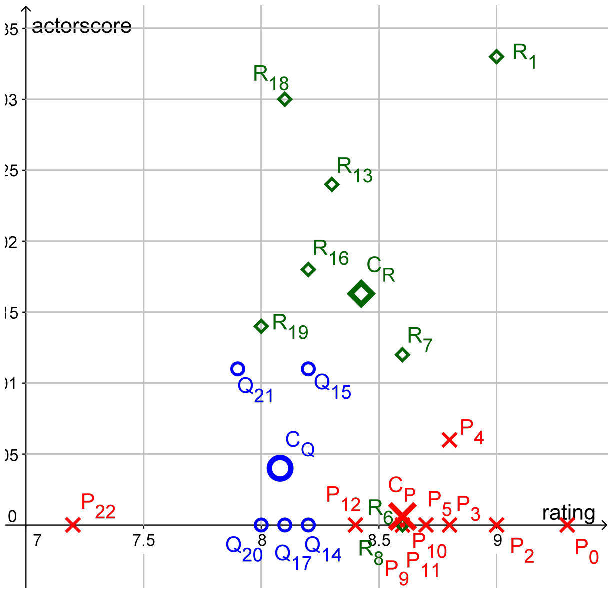

Fig. 5: Final clustering of Ex. 2 w.r.t the dimensions rank and actorscore.

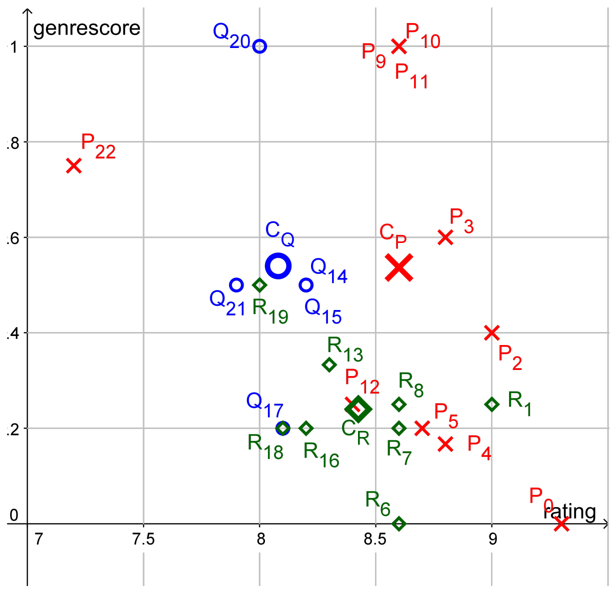

Fig. 6: Final clustering of Ex. 2 w.r.t the dimensions rank and genrescore.

Note that we have five dimensions to be considered, which means that in some of them outliers can be recognized, while they can be very close in other dimensions. In detail:

- Except for some outliers in Fig. 6 and 8 all blue displayed movies are very close to the centroid C_Q.
- While the green displayed movies are spread especially in Fig. 7 and 8, clusters can be recognized in Fig. 5 and 6 for centroid C_R.

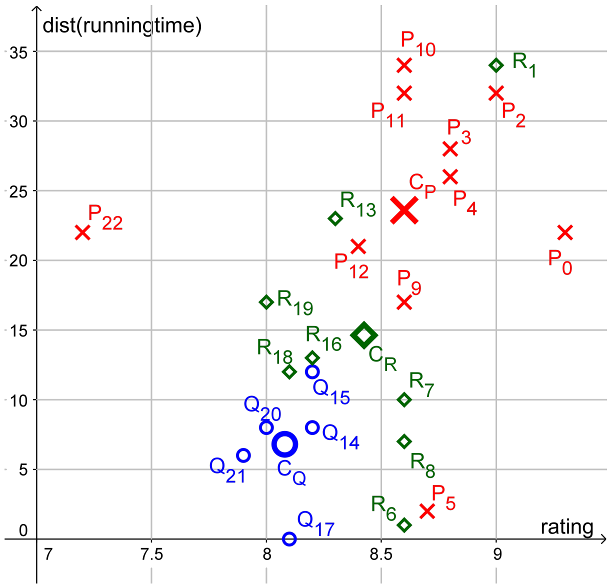

Fig. 7: Final clustering of Ex. 2 w.r.t the dimensions rank and runningtime.

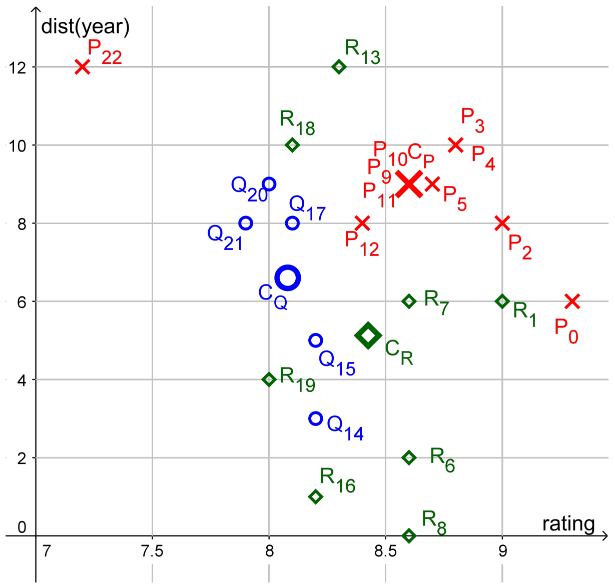

Fig. 8: Final clustering of Ex. 2 w.r.t the dimensions rank and year.

- For C_P clusters can be identified in Fig. 5 and 8 by the red displayed movies with only two aberrations regarding P_{22} and P_0, while they are spread in the other dimensions.
- All in all the final configuration of the clusters is intuitive and comprehensible, which leads to a adequate quality becuase of similar movies. For Example both versions of Terminator 2(ID 9, 10) are clustered together. As previously mentioned, the similar movies Goodfellas (ID 4) and Inception (ID 3) are clustered together as well, because they have an equal rank and equivalent release year, a very similar score for the actors and similar running times.

4.4 Complexity

Our algorithm reaches a complexity of $\mathcal{O}(n \cdot (d \cdot c \cdot c^2 + c^2 + c))$ where n is the number of d-dimensional points that should be clustered in c clusters. For each point n the distances of each dimension d for each cluster c is calculated in $\mathcal{O}(d \cdot c)$. These distances are sorted by *insertion sort* with a complexity of $\mathcal{O}(c^2)$ for the worst case, while the list of the summed up votes is sorted with *insertion sort* in $\mathcal{O}(c^2)$, too . Finally the list is iterated one more time in order to check if there is more than one Borda winner in $\mathcal{O}(c)$. Hence we get a complexity of $\mathcal{O}(n \cdot d \cdot c^3)$.

5 Experiments

In this section we briefly describe the benchmark settings and present results of experiments regarding runtime, number of iterations and quality of the clustering approaches compared to the basic k-means approach with traditional distance measures both for our Pareto-dominance clustering approach and our Borda Social Choice clustering approach.

5.1 Benchmark Settings

The first implementation of our Pareto-dominant clustering was realized as a Java program. We also implemented a database internal approach based on PG/PL-SQL in PostgreSQL[1], because we want to show the benefits of clustering in relational databases. For both approaches we varied the number of points and the number of desired clusters. To compare the runtimes, we created several synthetic anticorrelated sets of two-dimensional Pareto-optimal points. In order to gain averaged reliable data, clusterings were performed in test rows with varying numbers of repeats w.r.t. the number of points.

For the multi-dimensional Borda Social Choice Clustering approach we ran our experiments on a Intel Xeon machine with 2.53 GHz and a cache-size of 8192 kB. For our benchmarks we created anticorrelated sets of multidimensional objects, too and varied the number of dimensions, the number of objects per set and the number of desired clusters. We investigated the runtimes and number of iterations of our approach compared to the basic k-means with Euclidean distance and Canberra as distance measures. Furthermore results of our experiments with k-means++ are also presented.

5.2 Benchmarks Pareto-dominance

Runtime The benchmarks of the Java implementation in Figure 9 show that the approach using the Pareto-dominance are mostly similar regarding the runtime compared to the basic k-means approach using the Euclidean distance. For constant numbers of clusters and growing number of points the runtime of the averaged clustering is growing for both approaches. Whereas for constant numbers of points and growing clusterings there are some aberations at $k = 7$ for 15000 for both approaches. Especially if points at the border of the cluster switch between two clusters, the runtime is growing. All in all the clustering approach using the Pareto-dominance is nearly efficient as using the Euclidean distance.

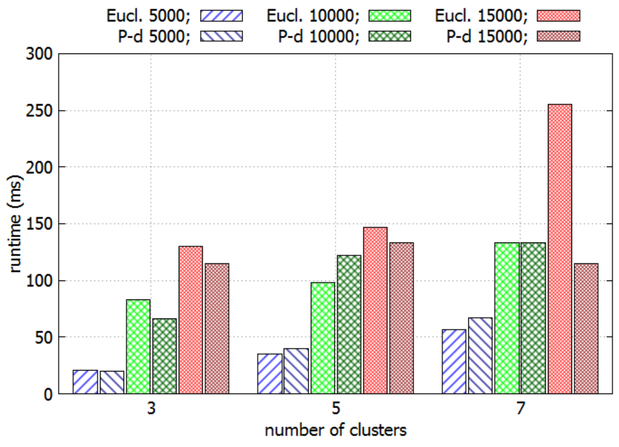

Fig. 9: Java benchmarks of the basic k-means clustering algorithm with Euclidean distance(Eucl) and Pareto-dominance(P-d).

[1] https://www.postgresql.org/

The benchmarks for the database approach in Figure 10 shows different behavior. The Pareto-dominance approach is of factor 2 slower than the approach using the Euclidean distance, because of the use of expensive database operations like Joins or Group by operations, which are not needed using the Euclidean distance. Especially the tests with $k = 5, 7$ and sets with 15000 points show aberations for the Pareto-dominance. But all in all for growing number of points and growing number of clusters the runtime is growing, too. Finally our approach is slower than the basic k-means clustering. But the effort of normalizing the users w.r.t. the two dimensions should be considered as a time-consuming process in each use case.

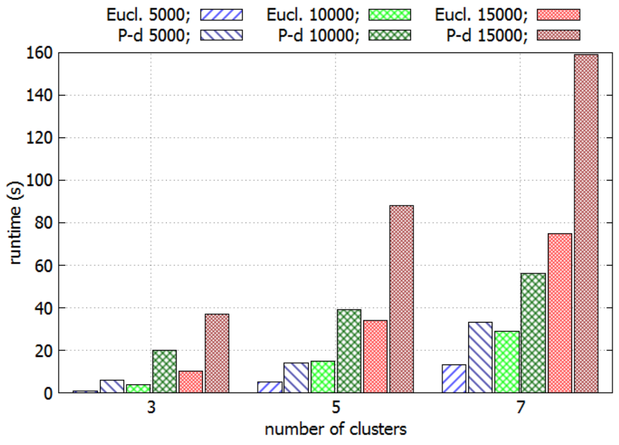

Fig. 10: *PostgreSQL benchmarks of the basic k-means clustering algorithm with Euclidean distance(Eucl) and Pareto-dominance(P-d).*

Iterations The number of iterations w.r.t. the number of desired clusters and the number of points can be seen in Figure 11. For both frameworks, the number of iterations is similar. For growing number of desired clusters, the number of iterations is growing for both approaches as well, except for the sets of $k = 5, 7$ with 15000 points using the Pareto-dominance. In contrary to our expectations for this experiment the number of points in the sets has no influence on the number of necessary iterations to achieve a stable clustering.

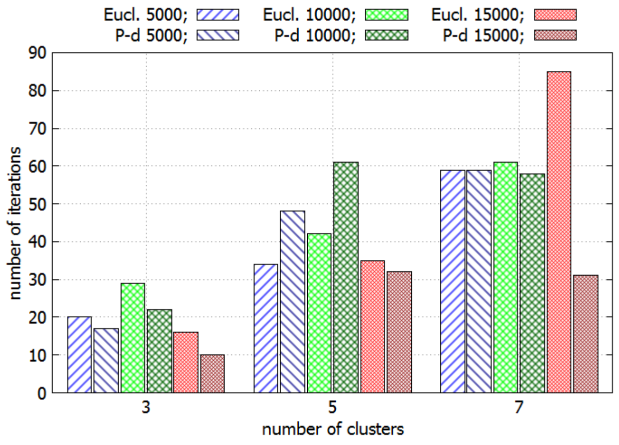

Fig. 11: *Comparison of Euclidean distance (Eucl.) and Pareto-dominance(P-d) regarding iterations.*

5.3 Benchmarks Borda Social Choice

Since Euclidean is the most common distance for k-means, we want to show that our approach terminates at least as fast as k-means and needs the same or less numbers of iterations until termination. Furthermore we investigated Canberra to gain useless reference values for our approach, which should be dominated by them of our approach w.r.t. the runtime and the number of iterations as well. In order to receive a faster runtime and less iterations until stable clusterings, we want to show the utility of k-means++ for our Borda approach w.r.t. the runtime and number of iterations.

Runtime

– In the 3-dimensional testrow in **Figure 12** for growing number of clusters and sets of input objects (5000, 10000, 15000) the runtime is growing, too. Our approach (Borda) works in equal time compared to k-means with Euclidean (Eucl.) and Canberra (Canb.) for small numbers of clusters. For 7 and 9 clusters our approach is slower independent of the number of input objects because of a higher complexity of our approach. Benefits of a faster runtime for k-means++ (Borda++) is in most cases hardly recognizable.
– In **Figure 13**, For growing numbers of dimensions our approach reaches a better runtime compared to the Euclidean distance except for high number of clusters in all benchmarked sets of 5000, 10000 and 15000 object. In some cases our approach terminates faster than k-means using Canberra, e.g., the testrow with 7 clusters, but all in all our approach mostly reaches an equal runtime in a 5-dimensional space.
– A similar behavior illustrates the testrow for a 9-dimensional set of objects in **Figure 14**. While both our approaches terminate in similar time compared to k-means with Canberra for 3 and 5 clusters, they are a lot faster than k-means with euclidean distance. The trends for growing runtimes w.r.t. the number of clusters and objects can be noticed in 9-dimensional space, too.

Iterations

– For growing numbers of clusters and growing numbers of objects per set the numbers of needed iterations until termination is ascending for all dimensions as it can be seen in **Figures 15, 16** and **17**.
– Both our versions of k-means reach a stable clustering in clearly less iterations, especially for higher number of clusters and bigger sets of objects.
– While the number of iterations is growing fast for growing numbers of dimensions for k-means using Euclidean and Canberra, our Borda approach only needs less additional iterations compared to smaller dimensions.
– K-means++ has only small effects on the iterations of k-means with Borda Social Choice for the cluster allocation.

Finally our approach works at most in equal time compared to k-means with both of the other distances, but for higher numbers of clusters it needs more time until termination because of the higher complexity of the Borda voting rule. Moreover our approaches need only a fractional part of iterations until termination for all testrows.

5.4 Quality

In this subsection we want to compare the quality of clusters between the basic k-means clustering algorithm and our Pareto-dominance clustering approach in order to show that the stable clusters are different but similar and thus useful for clustering objects of Pareto-frontiers.

Considering the example from Figure 3, which shows the most occurent stable clusterings after the second and last iteration. We performed a test-series based on the k-means clustering, which were compared to a testrow with 100 clusterings of our approach. In order to compare these clusters we use precision and recall. Precision represents all desired and delivered objects (correct alarms) in relation to all delivered objects (correct alarms & false alarms), whereas recall represents the desired and delivered objects (correct alarms) in relation to all desired objects (correct alarms & false dismissals).

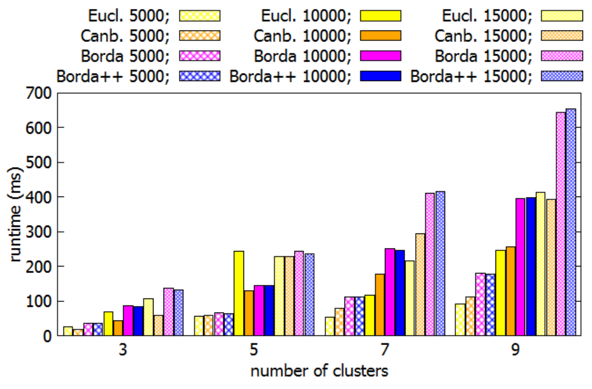

Fig. 12: Runtime for 3 dimensions.

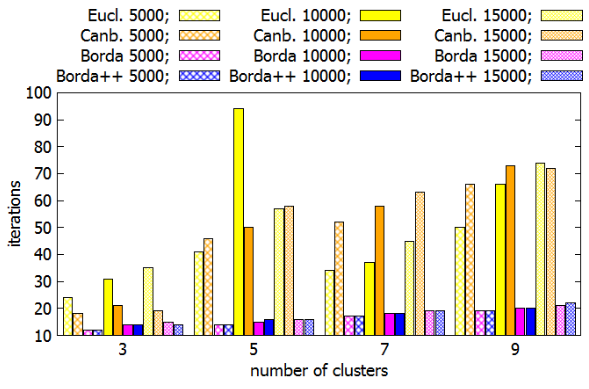

Fig. 15: Iterations for 3 dimensions.

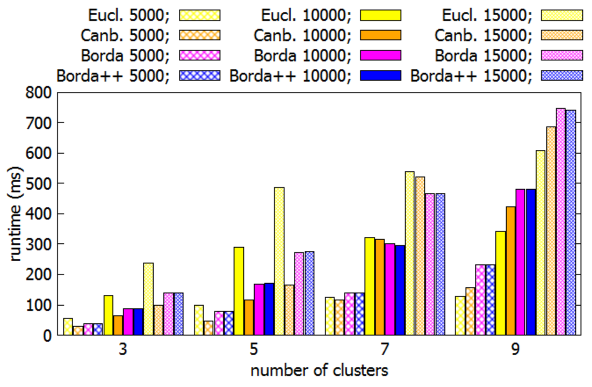

Fig. 13: Runtime for 5 dimensions.

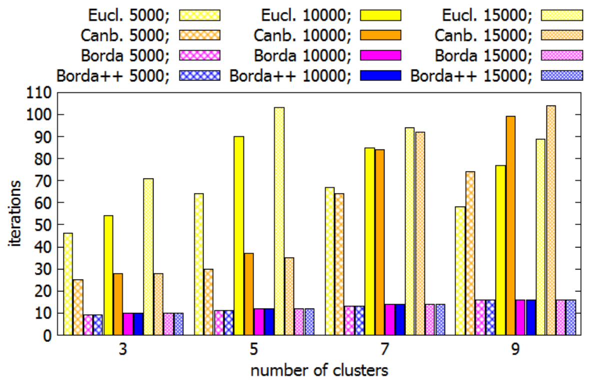

Fig. 16: Iterations for 5 dimensions.

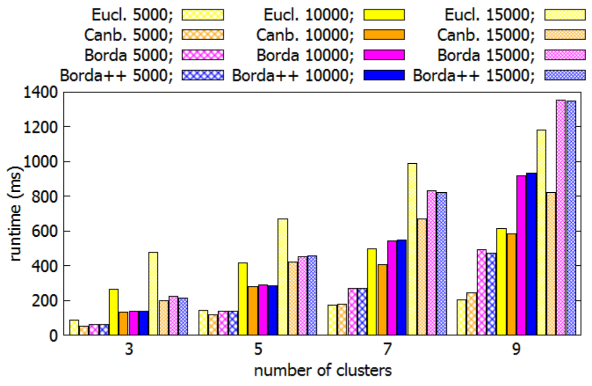

Fig. 14: Runtime for 9 dimensions.

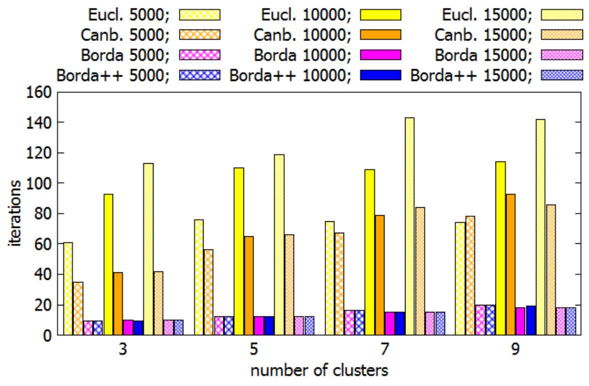

Fig. 17: Iterations for 9 dimensions.

Table 8 shows the values for precision and recall for the most occurent Pareto-dominant clustering on the base of the k-means clusters. Precision and recall both show for all clusters very high values with only two user, which switch between the clusters. Thus the quality of the Pareto-dominant clustering is in this case mostly high, but not equal to the k-means clustering with Euclidean distance. In Table 9 the test series show the precision and recall scores for two more clusterings and the frequency in the test series of 100 clusterings, which get averaged. Finally in this case there is one very present clustering (1), which is a very satisfying clustering as mentioned in Section 4. Both other clusterings have very high scores for precision. The scores for

Table 8: Precision-Recall model in detail for the most occurent clustering in Figure 3.

	C_1	C_2	C_3
false alarms (fa)	$\{\}$	$\{P_4\}$	$\{P_7\}$
false dismissals (fd)	$\{P_4\}$	$\{P_7\}$	$\{\}$
correct alarms (ca)	$\{P_1, P_2, P_3\}$	$\{P_5, P_6\}$	$\{P_8, P_9, P_{10}\}$
Precision: $\frac{ca}{ca+fa}$	1	0.66	0.75
Recall: $\frac{ca}{ca+fd}$	0.75	0.66	1

recall especially for the third cluster for the clustering 2 and 3 are very low, which shows that only a few users based on the k-means approach were assigned to this cluster.

Table 9: Averaged Precision-Recall model for 100 testrows and the three occurent clusterings of users in Figure 3.

Clustering	freq.	P_{C_1}	P_{C_2}	P_{C_3}	R_{C_1}	R_{C_2}	R_{C_3}
1	78	1.00	0.67	0.75	0.75	0.67	1.00
2	7	1.00	0.60	1.00	1.00	1.00	0.33
3	15	0.80	0.60	1.00	1.00	0.75	0.33
Averaged	100	0.97	0.65	0.81	0.81	0.70	0.85

6 Related Work

Since clustering is are very common topic in Data Mining, there are several other approaches considering clustering and Pareto-dominance, which were published in the last years.

A very early approach considering a Pareto-efficient clustering was published in [5] where more then one criterion for clustering was consulted. This paper presents on the one hand a modified relocation algorithm, and on the other hand a modified agglomerative algorithm. These approaches aim finding a Pareto-dominant clustering that dominates all other clusterings, while the approach presented in our report uses Pareto-dominance in order to allocate an object to a specific cluster.

In [6] a k-means clustering-based technique was published, where a so-called SkyClustering method is working within a Skyline-computation in SQL on a relational database in order to compress a large Pareto-optimal set of objects to explore the diversity of a Skyline.

In order to prevent a large set of Pareto-optimal objects, in highdimensional space in [4] only a few dimensions k are considered for the Skyline computation. This approach attaches weight to less, but maybe more important dimensions on objects.

Another approach to handle with Skylines was presented in [14], where supervised alternative clusterings are introduced. The main focus of this paper is to find clusterings of good quality starting from given negative clusterings which should be as different as possible and at the same time a Pareto-optimal solution. If the solution is not satisfying, the Pareto-frontier will be reclustered with the traditional k-means clustering, unlike the Pareto-dominant clustering presented in our report.

In [12] Subspace Clustering is considered to mask out dimensions in high dimensional data by a so called feature selection, which reduces the dimensions by removing irrelevant and redundant ones. So overlapping clusterings are found in subspaces. In contrast to previous work, we exploit Borda Social Choice to weight the distances to clusters in each dimension according to the distances for the cluster allocation for each object.

7 Conclusion and Outlook

In this report we presented on the one hand a novel Pareto-dominance based clustering framework on Pareto-frontiers for two-dimensional uses cases. Our framework provides several reasons for using this to manage large confusing sets of tuples with explicitly different domains. First, one can influence the result of the clustering by attaching a weight to a more important dimension in order to cluster at least over one dimension at the appearance of Pareto-optima. Second, tuples can now be clustered over better-than relationships in order to avoid adjustments for utilization in different uses cases. Third, our preliminary benchmarks show that a Pareto-dominant clustering can be realized in adequate time. The quality of our approach is satisfying, because the stable clusters distinguish from them of the basic k-means clustering, but are still as similar as possible, especially regarding the affiliation of similar points w.r.t. the one-dimensional clustering.

On the other hand we introduced a novel clustering framework eploiting the Borda Social Choice voting rule. Especially for high-dimensional applications our framework handles large and dizzying sets of objects. The users don't need to care about the normalization of the domains, because Borda Social Choice for the cluster allocation consults each dimension equally by weighting the distances to the clusters. Furthermore our benchmarks show that our approach terminates in comparative runtime to k-means clustering with traditional measures and needs less iterations until a stable clustering is reached. Our Borda Clustering framework can be used to manage a variety number of multi-dimensional preference-based use cases with diverse domains, e.g., movie-search, hotel-booking, car-purchase, etc.

In summary both our approaches presented in this report are promising, because of satisfying results of our experiments. Since Pareto-dominance works well for clustering objects in smaller number of dimensions, Borda Social Choice is more convenient for uses cases with at least three dimensions.

Future work includes the optimization of our approaches in order to cluster faster compared to k-means using traditional metrics. Afterwards we will integrate our approaches into the commercial analytical database EXASolution [11].

Acknowledgements

This work has been partially funded by the German Federal Ministry for Economic Affairs and Energy according to a decision by the German Bundestag, grant no. ZF4034402LF5.

References

1. D. Arthur and S. Vassilvitskii. K-means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.
2. S. Bandyopadhyay and S. Saha. *Unsupervised Classification*. Springer Verlag, Berlin Heidelberg, Germany, 2013.
3. S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proceedings of ICDE '01*, pages 421–430, Washington, DC, USA, 2001. IEEE.
4. C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding K-dominant Skylines in High Dimensional Space. In *Proceedings of SIGMOD '06*, pages 503–514, New York, NY, USA, 2006. ACM.
5. A. Ferligoj and V. Batagelj. Direct Multicriteria Clustering Algorithms. *Journal of Classification*, 9(1):43–61, 1992.
6. Z. Huang, Y. Xiang, B. Zhang, and X. Liu. A Clustering Based Approach for Skyline Diversity. *Expert Syst. Appl.*, 38(7):7984–7993, July 2011.
7. A. K. Jain. Data Clustering: 50 Years Beyond K-means. *Pattern Recogn. Lett.*, 31(8):651–666, June 2010.
8. J. Kastner, M. Endres, and W. Kießling. A Pareto-Dominant Clustering Approach for Pareto-Frontiers. In *Proceedings of the Workshops of the EDBT/ICDT 2017 Joint Conference), Venice, Italy, March 21-24, 2017*, volume 1810 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.
9. W. Kießling. Foundations of Preferences in Database Systems. In *Proceedings of VLDB '02*, pages 311–322, Hong Kong, China, 2002. VLDB.
10. W. Kießling, M. Endres, and F. Wenzel. The Preference SQL System - An Overview. *IEEE Data Eng. Bull.*, 34(2):11–18, 2011.

11. S. Mandl, O. Kozachuk, M. Endres, and W. Kießling. Preference Analytics in EXASolution. In *Proceedings of the Sixteenth Conference on "Database Systems for Business, Technology, and Web" (BTW), 4.-6.3.2015 in Hamburg, Germany.*, pages 613–632.
12. L. Parsons, E. Haque, and H. Liu. Subspace Clustering for High Dimensional Data: A Review. *SIGKDD Explor. Newsl.*, 6(1):90–105, June 2004.
13. F. Rossi, K. B. Venable, and T. Walsh. *A Short Introduction to Preferences Between Artificial Intelligence and Social Choice*. Morgan & Claypool Publishers, 2011.
14. D. T. Truong and R. Battiti. A Flexible Cluster-Oriented Alternative Clustering Algorithm for Choosing from the Pareto Front of Solutions. *Machine Learning*, 98(1):57–91, 2015.
15. R. Xu and D. C. Wunsch II. *Clustering*. John Wiley & Sons, Inc.,, Hoboken, New Jersey, USA, 2009.