Polynomial-time solvable special cases of anticlustering

Joshua Kokol

A thesis presented for the degree of
Bachelor of Science

Algorithmic Bioinformatics
Heinrich Heine University Düsseldorf
Germany
31st January, 2022

Abstract

Forming groups out of data is an essential problem in many areas, like psychology, biology and computer science. With clustering, we can separate data points into distinct groups. There are many clustering algorithms but in general, the clustering problem is NP-hard. The same applies to anticlustering. In anticlustering, the goal is to minimize the dissimilarity between anticlusters and maximize heterogeneity within each anticluster. Imagine you want to prepare an exam with two different versions, so the students cannot copy from their neighbor. Your data points are the possible questions in the exam and your anticlusters will be the actual exams. The goal is to distribute the questions in a way, that the two different exams are as similar in difficulty as possible. This is one example of a problem with two anticlusters. We will analyze different polynomial-time algorithms for clustering and transform these into algorithms, which solve the anticlustering problem in polynomial-time.

Contents

1 Introduction

Distributing a pool of elements into groups is an essential problem in many areas, like psychology, biology and computer science. However, there are many different goals that want to be achieved by partitioning elements. Most often, a division into homogeneous groups is desired, which corresponds to the problem of clustering. This implies, that elements within a group should be as similar as possible, while they also should be as dissimilar as possible to elements within other groups. The quality of these conditions can be measured in many different ways, which we call the objective function of a given clustering algorithm. There already exists a variety of clustering algorithms to solve these kinds of problems. However, there are some cases, where high intra-group dissimilarity and low inter-group dissimilarity is desired. This problem was first called anticlustering by Späth [1] and Valev [2].

A clustering problem is characterized by the number of clusters over which the data should be distributed and the feature dimension. The clustering problem with K clusters and datapoints with M dimensions is referred to as a K-clustering problem with an M-dimensional feature space.

For clustering and anticlustering there are many objective functions, which determine the rules on how to distribute the data into the clusters. The two objective functions we will look at are diversity and dispersion. The diversity objective tries to minimize the sum of all distances in one cluster. For the anticlustering case, it therefore tries to maximize this sum and is also often referred to as 'Max Diversity'. The dispersion objective tries to minimize, respectively maximize for the anticlustering case, the minimum distance between two elements in one cluster. For the anticlustering case, this objective function is also often referred to as 'Max Dispersion'.

The clustering problem and the anticlustering problem are both NP-hard problems, which makes it difficult to find an optimal solution for bigger data [3]. Let's take a 2-clustering algorithm and try to find an optimal solution with the additional constraint, that the cluster sizes should be equally big. This could be achieved through complete enumerations, which would have a run-time of

$$O\left(\binom{N}{\binom{N}{2}}\right), \tag{1}$$

with N being the number of data points. There are faster algorithms than complete enumeration of clustering and anticlustering. Additionally, there are heuristics that try to find feasible solutions as close as possible to the optimal solution in significantly less time. A heuristic that will be compared to the algorithms presented in this thesis is bicriterion iterated local search (BILS), which is explored in detail in Martin Breuers Bachelor Thesis [4].

There are special cases of clustering, for which there are polynomial-time exact algorithms. For example, Brucker [5] describes an algorithm for the 2-clustering special case with an arbitrary amount of feature dimensions and the dispersion objective. The algorithm works for any feature space, like for example the euclidean space. Because anticlustering is the opposite of the clustering problem with regards to dispersion and diversity, it stands to reason that this algorithm and several others described by Brucker [5] can be transformed into anticlustering algorithms and still solve the problem in polynomial-time. In this thesis, we will focus on transforming these algorithms and later perform an evaluation of the run-time.

We will implement three different algorithms for two special cases of anticlustering, which solve the problems in polynomial time for a given objective function. With all these algorithms, we assure a balanced distribution of data points, which means, that all anticlusters will contain the same amount of elements. If the amount of data points is not divisible by the number of clusters, we relax this constraint such that two clusters can deviate in their size by one element.

2 Problem Specification

For anticlustering, there are different objective functions that can strongly influence the distribution in the anticlusters. An objective function $f : \Pi \rightarrow \mathbb{R}^+$ aims to find a feasible partition π^* such that $f(\pi^*) = \max(f(\pi) | \forall \pi \in \Pi)$. A partition represents one possible distribution of the elements over all clusters. The optimal partition π^* is the one that maximizes the given objective function. The objective function associates a positive real number with each partition. [4]

2.1 Preliminaries

In this section, we will go into detail on the preliminaries of this thesis. This thesis builds on the clustering problem and the derived anticlustering problem, first coined by Späth [1] and Valev [2]. An anticlustering problem is called K-anticlustering problem, when the desired number of anticlusters is K. An (anti)cluster is a set c_k of data points $x_1, \dots, x_S \in X$, where X is the set of all N data points $x_1, \dots, x_N$. These data points will be represented through M-dimensional vectors, which will follow the rules of the euclidean space. All following implementations will focus on the euclidean space, where a distance between two points is defined as:

$$d(i,j) = ||j-i||_2 = \sqrt{(j_1-i_1)^2 + \cdots + (j_n-i_n)^2} = \sqrt{\sum_{i=1}^{n} (j_i - i_i)^2}$$

We will use an example data set, which we want to anticluster to explain the general procedure. First, we will describe the input data. The data will be given as an NxM data matrix, with N rows for the data points and M columns for the dimension of the data.

	x	y
1	3	2
2	1	4
3	0	1
4	1	3
5	3	4
6	2	3

Table 1: six 2-dimensional datapoints in a 6x2 matrix

These N data points, or elements, need to be distributed over K clusters c, where $k \in 1, \dots, K$ represents the index of a cluster c. A partition $\pi = (c_1, \dots, c_K)$ represents a complete distribution of N elements over K clusters, with every element being in only one cluster.

$$\bigcup c_k = X, \bigcap c_k = \emptyset \tag{2}$$

The anticlustering problem can be modeled as a graph structure. This is used in the algorithms by Brucker and Papenberg. Let $G = (V, E)$ be an undirected graph with vertices $V = (v_1, \ldots, v_N) = X$ and a set of edges $E = (e_1, \ldots, e_L)$ that connect pairs of vertices. The weight of an edge e_l between vertices i and j is the euclidean distance $d(i, j)$ between these points. The NxN matrix D is the distance matrix between all vertices $v_i \in V$ and describes the dissimilarity between two elements.

$$D = \begin{bmatrix} 0 & 2.83 & 3.16 & 2.24 & 2 & 1.41 \\ 2.83 & 0 & 3.16 & 1 & 2 & 1.41 \\ 3.16 & 3.16 & 0 & 2.24 & 4.24 & 2.83 \\ 2.24 & 1 & 2.24 & 0 & 2.24 & 1 \\ 2 & 2 & 4.24 & 2.24 & 0 & 1.41 \\ 1.41 & 1.41 & 2.83 & 1 & 1.41 & 0 \end{bmatrix}$$

In many cases, there are multiple optimal solutions for one anticlustering problem. In figure 1, two optimal solutions for the dispersion objective of the 2-anticlustering special case are shown. One of these solutions was generated using the adapted Brucker algorithm, examined in section 3, which always generates optimal solutions, as proven in Brucker's paper [5]. The second one was then generated through trial and error since the optimal dispersion value for this instance was given through the Brucker algorithm. In the figure, the two most distant points switch their anticluster allocation. This does not influence the dispersion value, because of how far away these points are from every other point.

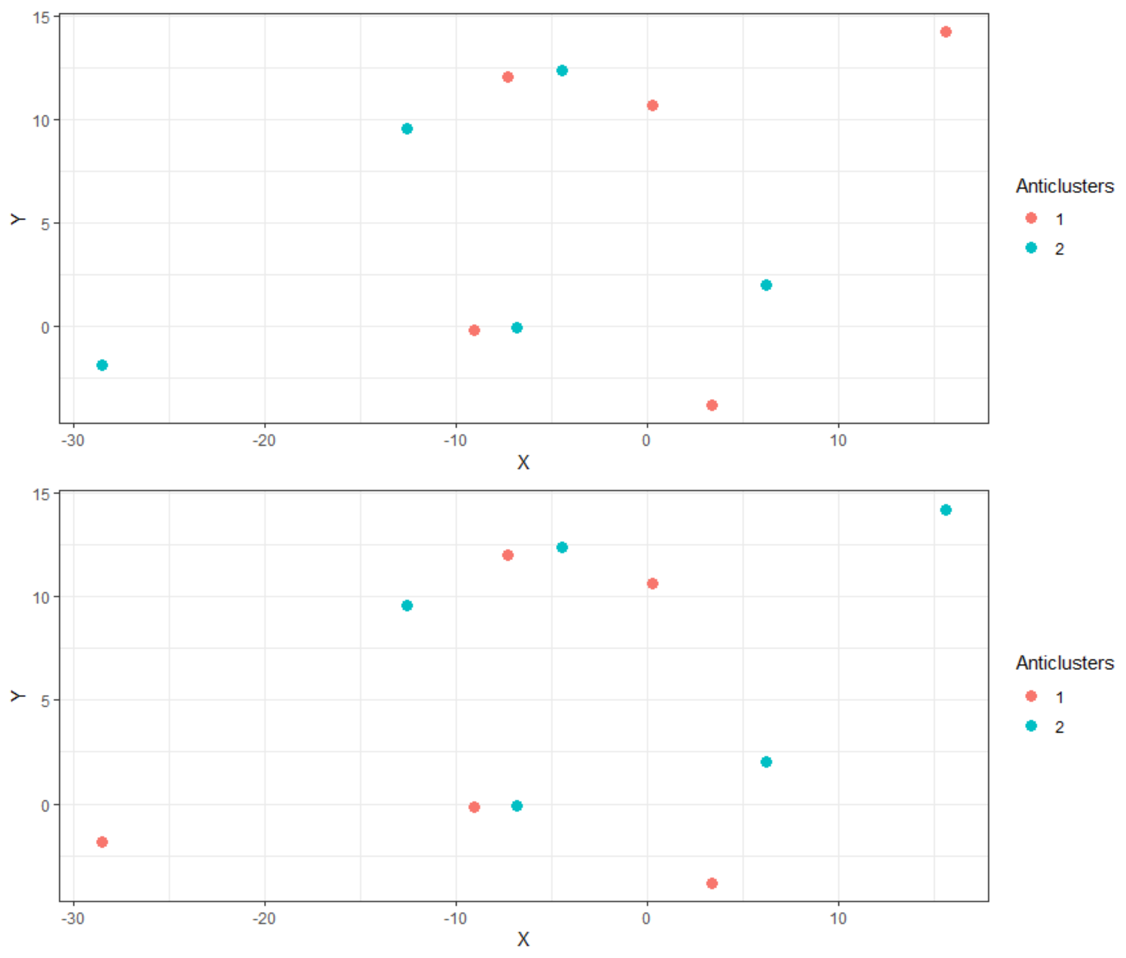

Figure 1: Different optimal solutions for a 2-anticlustering problem

The BILS heuristic and also the exchange method, which will be compared to the algorithms implemented in this paper, are part of the anticlust package for R made by Papenberg et al. [6]. We will use anticlust version 0.6.1 for this thesis. For the computations we will use a computer with an Intel(R) Core(TM) i7-7700K CPU 4.20GHz processor and 16 GB RAM.

2.2 The Dispersion Objective

The first objective function we will look at is 'Max Dispersion'. In anticlustering, we want to maximize the dispersion objective, which describes the minimum difference of two elements in one anticluster, while in clustering the dispersion will be minimized [7]. The dispersion objective is computed as the minimum distance between two vertices of the same anticluster c_k. The dispersion value of a partitioning π is therefore:

$$dispersion(\pi) = \max_{k \in 1,\ldots,K} \min_{i,j \in c_k; i<j} d_{ij} \tag{3}$$

A use case can be the assignment of students to learning groups [7] or the distribution of questions on two versions of an exam. Since the maximum dispersion objective describes the worst-case pairwise dissimilarity over all groups, it is a good objective function for the exam problem. This is due to the fact, that the questions should be as dissimilar as possible in one exam, to cover as many different areas as possible for each student. In most cases, the difference between two data points is described through the distance between these data points, on some M-dimensional space. Because for most cases where we use the dispersion objective only the distance is of interest, the actual dimension of the feature space has only a limited impact on the run time of the algorithm by Brucker [5]. This is a convenient feature of the dispersion objective since the dimension of the feature space can be an ejecting factor for the run time of a possible solution.

2.3 The Diversity Objective

'Max Diversity' is another objective function, which distributes data into anticlusters. It corresponds to the sum of the distances within groups. Therefore it is an extension of the dispersion objective, which aims to enhance the quality of the solution [8]. By maximizing these distances we solve an anticlustering problem and when minimizing, the resulting partition will be a distribution in clusters. The diversity is computed as the sum of all the weights of edges with vertices in one cluster c_k:

$$diversity(\pi) = \sum_{k=1}^{K} \sum_{i,j \in c_k; i<j} d_{ij} \tag{4}$$

A use case for the anticlustering problem is the distribution of students into equal groups. Each student has different attributes like age, gender and more, which influence the distribution of the algorithm [9]. This problem could be used for classroom division, to get a diverse environment for students to learn and grow in, or division into sports teams, to eliminate any unfair advantage or disadvantage to other teams.

3 2-anticlustering algorithm

In this section, we will elaborate on the 2-clustering algorithm, for an arbitrary M-dimensional space with the dispersion objective function, as described in Brucker's paper [5]. For the clustering case, this problem can be solved in polynomial time. We will describe the necessary changes for the anticlustering case, derive the run-time and test the algorithm on different sets of data.

3.1 Introduction

The clustering algorithm takes advantage of a feature of graphs, namely bipartitioning. A subgraph $G_{sub} = (V_1, V_2, E)$ of another Graph $G = (V, E)$ with vertices V and edges E is bipartite, if $\forall$ edges $e = (v, w) \in E \mid v \in V_1$ and $w \in V_2$ or $w \in V_1$ and $v \in V_2$. This means, that all edges are only between V_1 and V_2, and no edges are within one set. To solve the clustering problem, Brucker [5] proposed an algorithm, which constructs bipartite subgraphs $G_i = (A_i, C_i, E)$ from which the optimal partitions B_1^* and B_2^* can be derived.

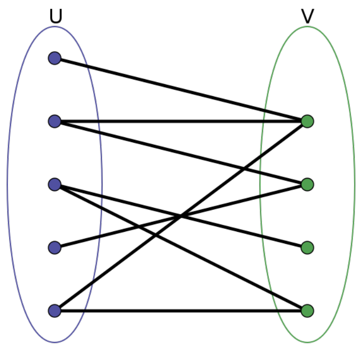

Figure 2: Incomplete bipartite graph https://commons.wikimedia.org/wiki/File:Simple-bipartite-graph.svg

$$B_1 * \longleftarrow \bigcup A_i, B_2^* \longleftarrow V\ B_1^* \tag{5}$$

The general idea of the algorithm is to put elements in different clusters if they are far away from each other, therefore ensuring that these distances do not influence the dispersion value. At the beginning he sorts the distances between all vertices $v_1, ..., v_r$ such that

$$d_{i_1 j_1} \geq ... \geq d_{i_r j_r}. \tag{6}$$

After that, he describes an algorithm for computing these bipartite subgraphs and proves the polynomial-time solvability for the 2-clustering case. For the proof, he uses the distance between two vertices to measure a lower bound for the value of the objective function. This is a crucial realization for the anticlustering problem, because with this if one could easily maximize the dispersion instead of minimizing it, one would get an optimal solution regarding the dispersion objective of the anticlustering case.

The algorithm by Brucker is not the only clustering algorithm that explores the idea of graph partitioning to find a polynomial-time algorithm. Caraballo et. al. [10] describe an algorithm that uses the same idea however, they use a maximum spanning tree to compute the optimal solution.

3.2 Implementation

For the anticlustering case, the algorithm has to be modified. Due to the fact, that we want to maximize the dispersion, instead of minimizing it, a possible way could be to reverse the order of distances to

$$d_{i_1 j_1} \leq \ldots \leq d_{i_r j_r}.$$

Just with this modification, the algorithm produces optimal solutions for the anticlustering case in regards to the dispersion objective. The algorithm works as follows: First, the distances will be sorted as explained before and the first i and j will be added to A_1 and C_1 respectively. The following implementation is written in R and describes the first setup for the algorithm.

```
d <- d[order(d$d),]
i <- 1
j <- 1
Ai <- list(d[i, 2])
A <- list(Ai)
Ci <- list(d[j, 3])
C <- list(Ci)
k <- 2
```

The variable k describes the current amount of A_i and C_i plus 1. It is used to instantiate new A_i and C_i if the right conditions are met. The dataframe d, contains all the distances between two elements and their indices. d[t, 1] is the distance between the elements written in d[t, 2] and d[t, 3].

distances	i	j
0.52343	30	72
0.59991	2	44
0.91728	2	13
2.00134	17	99
...	...	...

The lists A and C itself will contain lists, which will represent the A_i and C_i. The algorithm of Brucker [5] initially uses a graph notation to solve the problem however, this only benefits for the proof that the algorithm solves the special case in polynomial time and not for the actual implementation. Therefore we will not use graphs for the implementation of the anticlustering version. The algorithm runs, until every i and j is ether in A or C and until that condition is met, it iterates over all distances $d_{i_2 j_2}$ to $d_{i_r j_r}$. The actual decision, how i and j are distributed over which A_i or C_i, is determined by five if statements. These statements will not be explained in detail, because the most interesting is the first one and the others just move the elements between the A_i and C_i under certain conditions, which are listed below in table 3.2. The first if statement can be described as follows:

If the current i and j are both not yet in any A_i or C_i, i will be added to a newly instantiated A_k and j to C_k. After that, the counter k is set higher. Here is the implementation of this if statement in R:

```
contained <- FALSE
for(var in 1:k){
  if(d[v, 2] %in% A[[var]] | d[v, 2] %in% C[[var]] |
   d[v, 3] %in% A[[var]] | d[v, 3] %in% C[[var]]){
    contained = TRUE
  }
}
if(!contained){
  A[[k]] <- d[v, 2]
  C[[k]] <- d[v, 3]
  k <- k + 1
  next
}
```

All the if statements are listed here in a simplified version:

1 If i_v and j_v are neither in an A_i or C_i for $1 \leq i \leq k$, then $A_k \leftarrow (i_v)$; $C_k \leftarrow (j_v)$; $k++$

2 if $i_v \in A_s$, $j_v \in A_t$ or $i_v \in C_s$, $j_v \in C_t$ for some $s \neq t$ then $A_s \leftarrow A_s \cup C_t$; $C_s \leftarrow C_s \cup A_t$; $A_t = C_t \leftarrow \varnothing$

3 if $i_v \in A_s$, $j_v \in C_t$ or $i_v \in C_s$, $j_v \in A_t$ for some $s \neq t$ then $A_s \leftarrow A_s \cup A_t$; $C_s \leftarrow C_s \cup C_t$; $A_t = C_t \leftarrow \varnothing$

4 If $i_v \in A_s$ and j_v nowhere then $C_s \leftarrow C_s \cup (j_v)$ (respectively if i and j are swapped)

5 If $i_v \in C_s$ and j_v nowhere then $A_s \leftarrow A_s \cup (j_v)$ (respectively if i and j are swapped)

In the original algorithm, Brucker describes it with a graph and also adds edges in every if statement, which we omitted since it had no impact on the computation.

However, there is one issue with the Brucker algorithm that we have not touched on since now. The algorithm does not necessarily distribute the data into equal-sized (anti)clusters. This is because of statements 4 and 5, since in these statements only one of i and j is distributed into a cluster. This is a problem, because in applications we mostly want clusters with equal size, for example for test-question distribution. Also the BILS heuristic always only looks at equal-sized groups, therefore a comparison would be pointless. Since the dispersion objective is being used, we are only interested in the smallest distance between two elements in the same cluster, therefore there is a simple solution, where we change the cluster assignment of some elements which do not influence the dispersion objective. After the algorithm is complete, it returns the cluster distribution as a vector. Then we count the elements in the clusters and if they are not equal we will execute this R code:

```
disp <- dispersion_objective(Vdata[,1:m], cppvector)
for(c in 1:length(cppvector)){
  if(cppvector[c] == 1){
    cppvector[c] = 2
    newDisp <- dispersion_objective(Vdata[,1:m], cppvector)
    if(newDisp < disp){
      cppvector[c] = 1
    }
  }
  if( (table(cppvector)[1] -1) <= table(cppvector)[2]){
    break
  }
}
```

This is now for the case that the anticluster with label 1 is bigger than the other one. The function dispersion_objective computes the dispersion objective for the current instance $cppvector$

in $O(N^2)$. If the new dispersion is smaller than the current dispersion, then the current element will not change its cluster. This code guarantees an equal anticluster distribution with optimal dispersion and has an overall run time of $O(N^3)$.

The complete implementation as well as the version written in C++ can be found on the Git repository, which is linked at the end of this thesis (Chapter B). An implementation in C++ is used to later compare this implementation with algorithms from the anticlust package, which are written in C. However the R implementation is easier to understand, therefore all code excerpts in this thesis are written in R. The reason we also implemented a C++ version, is that C and C++ are considerably faster than R. So to get comparable results of algorithms, they have to be in the same language. In table 2 you can see the time difference between R and C++ for different amounts of data points.

datapoints	R - Time in sec.	C++ - Time in sec.
10	0.002447897	0.000012944
100	2.111562505	0.001984816

Table 2: Time differences - R vs. C++

3.3 Conclusion

The anticlustering version of the Brucker algorithm also solves the special case for two anticlusters in polynomial time. The run time of the algorithm depends on multiple factors. First, the number of the distances between all possible pairs of elements, which are $\binom{N}{2}$ many. However, the for-loop over all distances only runs $\binom{N}{2}-(N-1)$ times, because it stops if all i and j are distributed in either A or C. Every time a vertex is not contained in A or C, it will be added to either one and because every vertex is contained N times in the distance-matrix, every vertex is distributed at the latest after $\binom{N}{2}-(N-1)$ loops. The second factor for the run time is the maximum number of G_i generated. The maximum amount of G_i can be $\frac{N}{2}$, since a new G_i is only created, if the vertices i and j of the currently examined distance are neither in an A_i or C_i. Since a vertex can only be in one A_i or C_i the worst case would be if the first $\frac{N}{2}$ distances contain all N elements. The sorting algorithm for the distances has a run time of $O(\binom{N}{2} * \log(\binom{N}{2}))$. This combined with the other loops and the balancing of the cluster sizes the run-time of this algorithm is

$$
\begin{aligned}
O((\binom{N}{2}-(N-1))*((\frac{(N-1)^2}{2}+\frac{N-1}{2})+\binom{N}{2}*\log(\binom{N}{2})+N^3)\\
=O((1+\frac{N*(N-1)}{2}-N)*(\frac{(N-1)^2}{4}+\frac{n-1}{2})+\binom{N}{2}*\log(\binom{N}{2})+N^3)\\
=O((-N+\frac{N^2-n}{2}+1)*(\frac{N^2-2N+1}{4}+\frac{2N-2}{4})+\binom{N}{2}*\log(\binom{N}{2})+N^3)\\
=O((-N+\frac{N^2-N}{2}+1)*\frac{N^2-1}{4}+\binom{N}{2}*\log(\binom{N}{2})+N^3)\\
=O(\frac{N^4-3N^3+N^2+3N-2}{8}+\binom{N}{2}*\log(\binom{N}{2})+N^3)\\
=O(N^4)
\end{aligned}
\tag{7}
$$

3.4 A different approach

The issue with the transformation of the implementation of the algorithm from Brucker is that it produces more bipartite mappings than it has to. An optimal solution can also be derived much faster with the approach of Papenberg, which he developed during the supervision of this thesis. The idea is the same as Brucker's, to use bipartite graphs to separate the vertices with the smallest distances. However, in Papenbergs implementation the algorithm terminates after the x smallest distances are no bipartitioning anymore, for a graph $G_i = (V, E)$ with edges E of the x smallest distances.

The procedure of the algorithm is as follows:

1. Initiate an empty, unweighted graph $G = (V, E)$
2. Compute the minimial distance between all elements
3. Add all elements that have the distance D between them as nodes in V
4. Add an edge e to E between all nodes that have the minimum distance D between them
5. check if the graph is bipartit
    - 5.1. If **no**: the max dispersion has been found and every unassigned node can be sorted to a random cluster
    - 5.1. If **yes**: set D to the next lower distance and restart at (4)

The algorithm repeats steps (3)-(5) for a maximum of $\left(\frac{N}{2}-1\right)$ times because after that every vertex is already part of a pair in E. A bipartitioning can generally be found in $O(N * log(N))$, therefore the algorithm has a run-time of

$$O\left(\left(\frac{N^2}{2}-N\right)*log(N)+\binom{N}{2}*\log\left(\binom{N}{2}\right)\right)=O\left(N^2*log(N)+\binom{N}{2}*\log\left(\binom{N}{2}\right)\right), \quad (8)$$

with $O\left(\binom{N}{2}*\log\left(\binom{N}{2}\right)\right)$ being the run time of a sorting algorithm for the distances. This is considerably faster than the transformed implementation of Brucker for the anticlustering case. In figure 3 the runtime of the algorithms for growing data sets is visualized. The data used for this analysis was 2-dimensional normal distributed. For each amount of data points, the mean of ten iterations is used to determine the run time. For a small number of data points, the algorithm of Brucker is faster, however, with about 120 data points and more, the implementation of Papenberg is faster due to the slower growth of the curve.

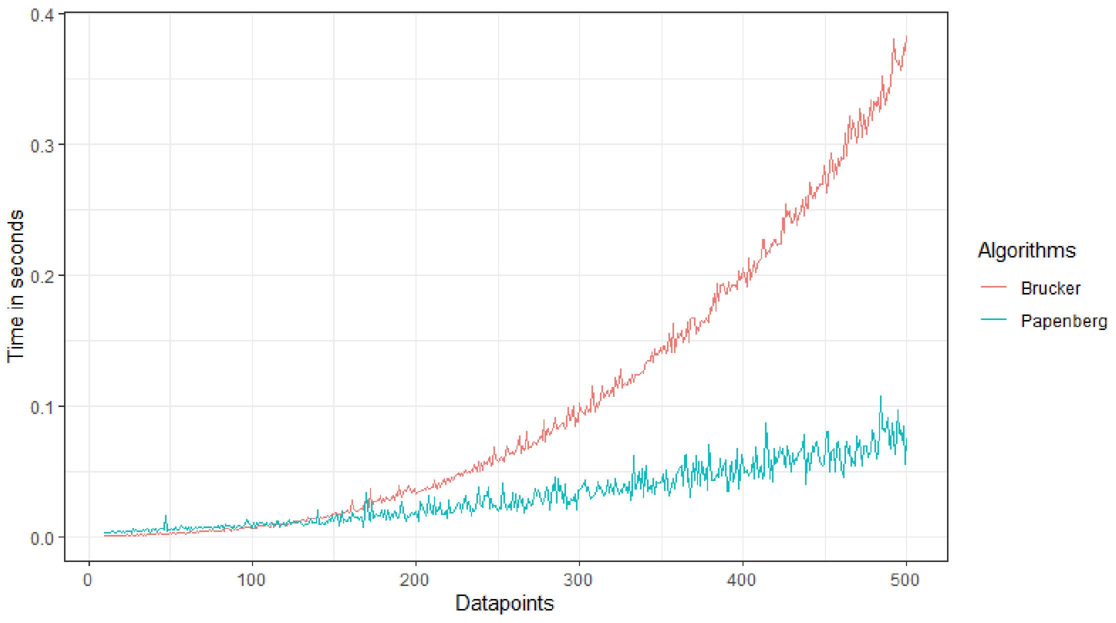

Figure 3: A run-time comparison between the Brucker and Papenberg implementation

3.5 Proof of correctness

The dispersion objective concerns the minimum distance between two elements in an anticluster. Therefore all other distances in an anticluster are irrelevant. Through this property, it is possible to solve special cases in polynomial time. The algorithm of Papenberg and also the algorithm of Brucker use this property. Papenbergs algorithm goes from the smallest distance between two elements to the largest distance. However, it terminates before it can reach the largest distance. It terminates if no bipartitioning can be found between all elements that are currently looked at. These elements are all previous elements and the ones with the current minimum distance between them. Therefore one may conclude, that the algorithm terminates after three or more elements are connected in a circle, as seen in figure 4, made by Martin Papenberg.

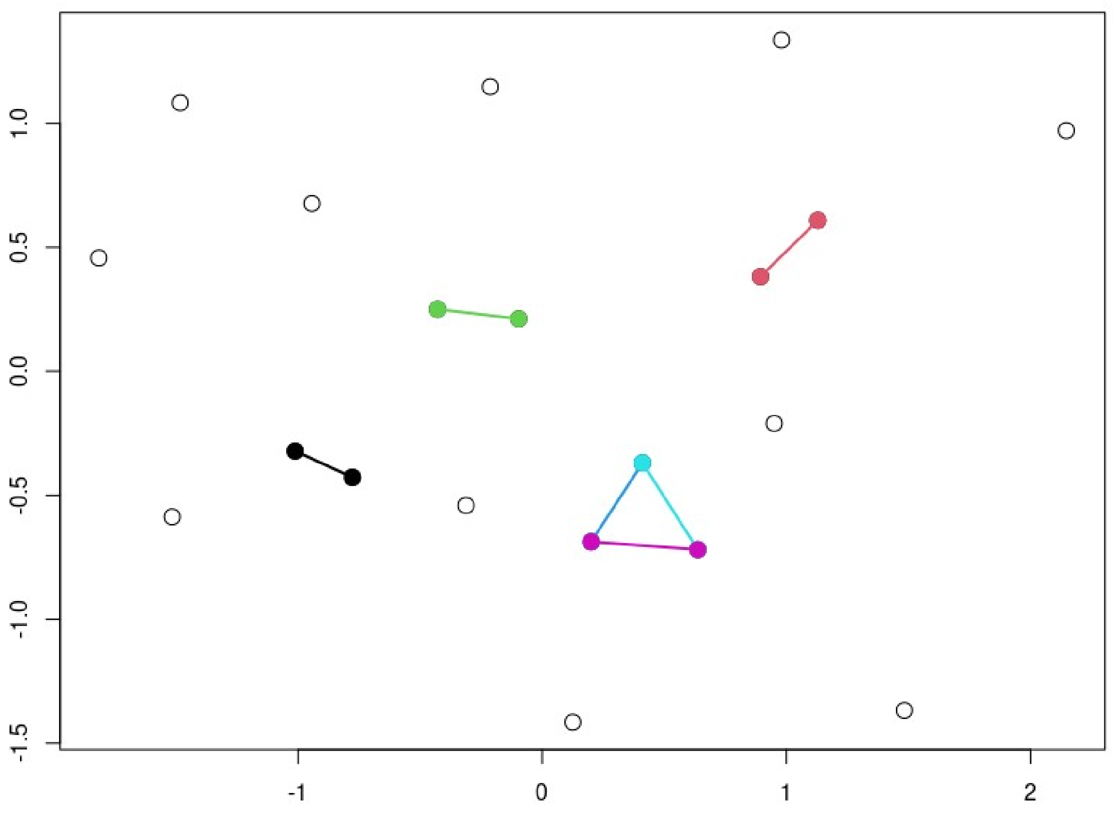

Figure 4: A graph, that is not bipartite anymore

All other points can be distributed to any cluster. The check for a bipartitioning is polynomial-time solvable since we only need to check all edges. If then some edges form a circle, it is no longer a bipartitioning. However a partitioning into three or more groups is not polynomial-time solvable, therefore it is not possible to find an optimal solution for the K-anticlustering case with an M-dimensional feature space in polynomial time.

4 K-anticlustering algorithm

For the K-clustering problem in the 1-dimensional Euclidean space, Brucker [5] describes two polynomial-time algorithms. Both algorithms use the shortest path problem for a network W = (V, E, d) with vertices V, edges E and distances d as basis and only differ in the calculation of the distances d. For the anticlustering case, we will look at a solution for the max dispersion objective because the algorithms of Brucker can not be transformed into an anticlustering algorithm. The algorithm of Brucker searches for a point on the number line, where the data can be split into the right cluster and the left cluster. This is not possible for the anticlustering case, since the anticlusters can not be separated in left and right but rather in alternating order.

4.1 Introduction

The anticlustering algorithm we will evaluate maximizes the dispersion of the data. On the 1-dimensional euclidean space, this is trivial, because every point has up to two direct neighbors - left and right. This is advantageous for us because in the dispersion objective, we are just interested in the minimum distance between two elements in one group. This objective is maximized when we divide the anticlusters alternately among the data points. For the 2-anticlustering case and ascending sorted data points:

data points	0.234	0.456	0.491	0.71	...
cluster	1	2	1	2	...

For the K-clustering case and ascending sorted data points:

data points	0.234	0.456	0.491	...	1.221	1.258	1.4	...
cluster	1	2	3	...	K-1	K	1	...

4.2 Implementation

An algorithm for distributing the data points in ascending order to the anticlusters, is solvable in $O(N)$ if we exclude the run time of a sorting algorithm. The following implementation of this algorithm is written in R.

```
exact_dispersion_one_dim <- function(Data, K){
    Data <- Data[order(Data$V1), ]
    Data$anticluster <- rep_len(c(1:K), nrow(Data))
    return(Data)
  }
```

The dataframe 'Data' contains the datapoints and k is the number of desired anticlusters. The function returns a dataframe with column 'V1' being the values of the data points in ascending order and column 'anticluster' being the index of the corresponding anticluster.

	V1	anticluster
1	0.87	1
2	0.91	2
3	1.03	1
...	...	...
N	5.56	2

Table 3: The returned dataframe with K=2

4.3 Proof of correctness

To prove this method, we will first look at the 2-anticlustering problem in the 1-dimensional euclidean space. Assume the anticluster distribution sorts two neighboring elements x and y to the same cluster. There are two possibilities following this distribution.

1. $d(x, y)$ is the minimum distance which determines the result of the dispersion problem
2. $d(x, y)$ is not the minimum distance

For the second case, it would not matter if x and y are in different anticlusters or the same one because they do not influence the result of the diversity problem. In the first case however, it would influence the dispersion and therefore would result in a higher dispersion value if the elements were in different anticlusters.

For the K-anticlustering problem, the same rules hold. Imagine the data points are alternately distributed over the K anticluster. The dispersion value is now given through two points i and j, which are allocated to the same anticluster c_k. Given through the alternating distribution there are $K-1$ data points p_l between i and j with $1 \leq l \leq K-1$ which are all in different anticlusters. The interval between i and j are therefore contains all K anticlusters and anticluster c_k is represented two times through i and j. If one would now switch the cluster allocations of either i or j, there are two possibilities:

1. i or j switch with a data point within the interval

2. i or j switch with a data point outside the interval

If the first case is true, the dispersion value decreases because now i and j are closer to each other. In the second case, there is now a new anticluster, which is represented by the points i_{new} and j_{new} two times in the interval between i and j. The dispersion is now given by the distance of these two elements and is thus necessarily lower than before. This new dispersion value is now the upper limit and the allocation of all other data points can only reduce the dispersion value and not increase it.

This property only works with elements on the number line, or in other words on a 1-dimensional feature space, because an alternating distribution is in higher dimensions not feasible since it depends on the angle of which one views the data. It is also not possible to reduce the dimensions using the distance between elements. For example, for three points in the 2-dimensional space, it is not possible to reduce them onto 1-dimension regarding their distances unless they are on a straight line in the 2-dimensional space.

4.4 Conclusion

For the 1-dimensional euclidean K-anticlustering problem, we can generate optimal solutions for the dispersion objective $O(N + N * log(N))$ with n being the number of data points and $N * log(N)$ being the run time of an arbitrary sorting algorithm.

5 Comparison with BILS

In this section, we will compare the examined algorithms for the special cases of anticlustering with heuristics and a complete enumeration algorithm. The tests will be regarding the run time and accuracy. Accuracy is especially interesting when compared with the heuristics because it could be that the heuristics also produce an optimal solution for the special cases. The bicriterion iterated local search (BILS) heuristic by Brusco et al. [11], further examined and implemented by Breuer [4] aims to optimize diversity and dispersion at the same time. BILS can be used through the anticlust package by Papenberg et al. [6] and is written in C. The method-call in R looks like this:

```
bicriterion_anticlustering(Vdata, 2)
```

Vdata is a datamatrix that contains the datapoints as rows and features as columns. The BILS heuristic can also use restarts to get closer to an optimal solution, however, this requires more time. In the following examinations, we are using no restarts. The second argument passed to the function represents the number of anticlusters.

For BILS and the other algorithms, we use the method

```
dispersion_objective(Vdata, anticlusters)
```

which returns a value to evaluate the quality of the solution in regard to the dispersion objective. The parameter "anticlusters" is a vector that contains the grouping for the different anticlusters. The algorithms written for this paper all compute an optimal solution, therefore we only use this function to examine the accuracy of BILS.

All examinations and comparisons will be made with normal-distributed data points, unless it is mentioned otherwise.

5.1 2-anticlustering

In figure 5 the time difference between BILS and the Brucker algorithm is illustrated. Notably, the BILS version is considerably slower than the Brucker version. On top of that, the BILS algorithm is a heuristics that does not guarantee an optimal solution.

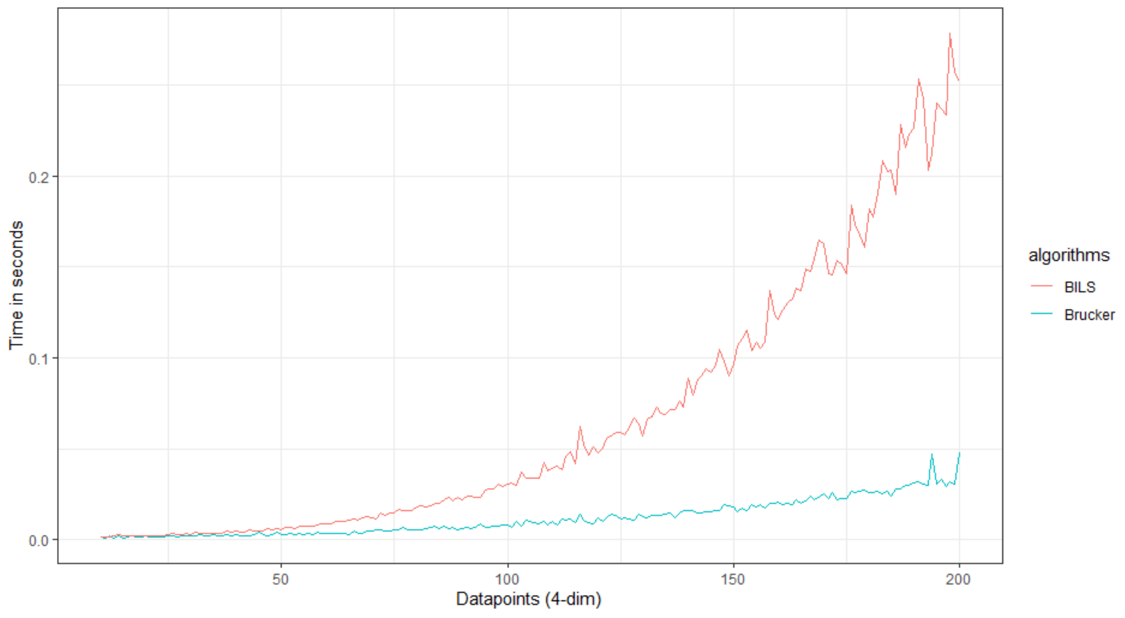

Figure 5: A run-time comparison between BILS and the Brucker algorithm

For figure 5 we took the average of 10 iterations for every number of data points. Notably, the BILS algorithm has much higher fluctuations in the run-time than the adapted Brucker algorithm made for this paper. The times BILS finds an optimal solution average around 50 percent and does not improve or deteriorate for an increasing amount of data points. Breuer [4] states in his bachelor thesis that the quality of solutions of the BILS heuristic decreases with more data points. However, it seems that the number of times that BILS finds an optimal solution does not change, only the quality of suboptimal solutions is impacted by the number of data points. In section A a detailed comparison between BILS and the implementation of Breuer's algorithm is listed.

Since the Brucker algorithm has to sort the distances between all data points in ascending order, the run time is also partly dependent on the sorting of the distances and the computation of the distances. This can be seen in figure 6, which describes the number of feature dimensions on the x-axes. This run time comparison is made with an instance of 50 data points and each observation of feature dimensions is made of the average over five iterations. With about 100 feature dimensions and more, BILS performs better than the Brucker algorithm. However, this margin changes with the number of data points, as seen in table 4. The values seen in table 4 are estimated by taking the average out of ten iterations over each amount of feature dimension. The run-time is very inconsistent over the different iterations, as seen in figure 6, therefore the values in table 4 are only approximations after whom the BILS algorithms perform better in at least 50% of the cases. For higher amounts of data points the Brucker algorithm is better if the number of feature dimensions is high. This happens, because the Brucker algorithm performs generally much better than the BILS heuristic for larger amounts of data points, as seen in figure 5, which counteracts the exponential amount of distances the

Brucker algorithm has to look at.

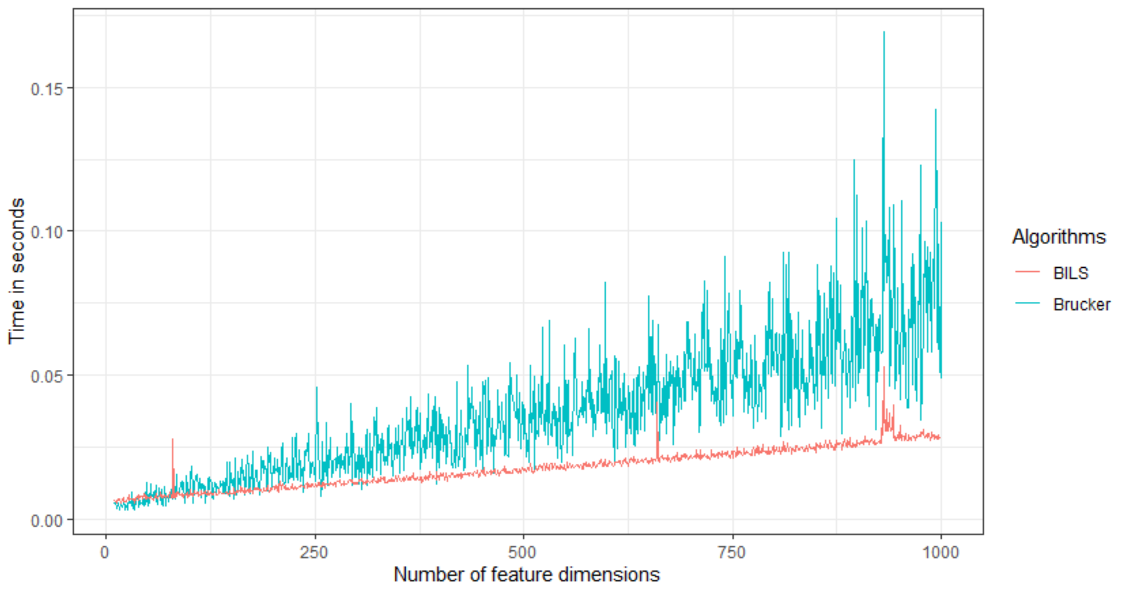

Figure 6: A run-time comparison between BILS and the Burcker algorithm with regard to feature dimensions of an instance with 50 datapoints

Datapoints	10	50	100	150	200	250
Dimensions	10	115	260	314	376	492

Table 4: The estimated threshhold, after which BILS performs better than Brucker

5.2 K-anticlustering

For the 1-dimensional euclidean problem the run time proceeds similarly to the 2-anticlustering case. For an expanded comparison the exchange method is included in the evaluation. The exchange method is also an approximation algorithm and can be called through the anticlust package in R.

```
anticlustering(Vdata, K, objective = "dispersion", method = "exchange")
```

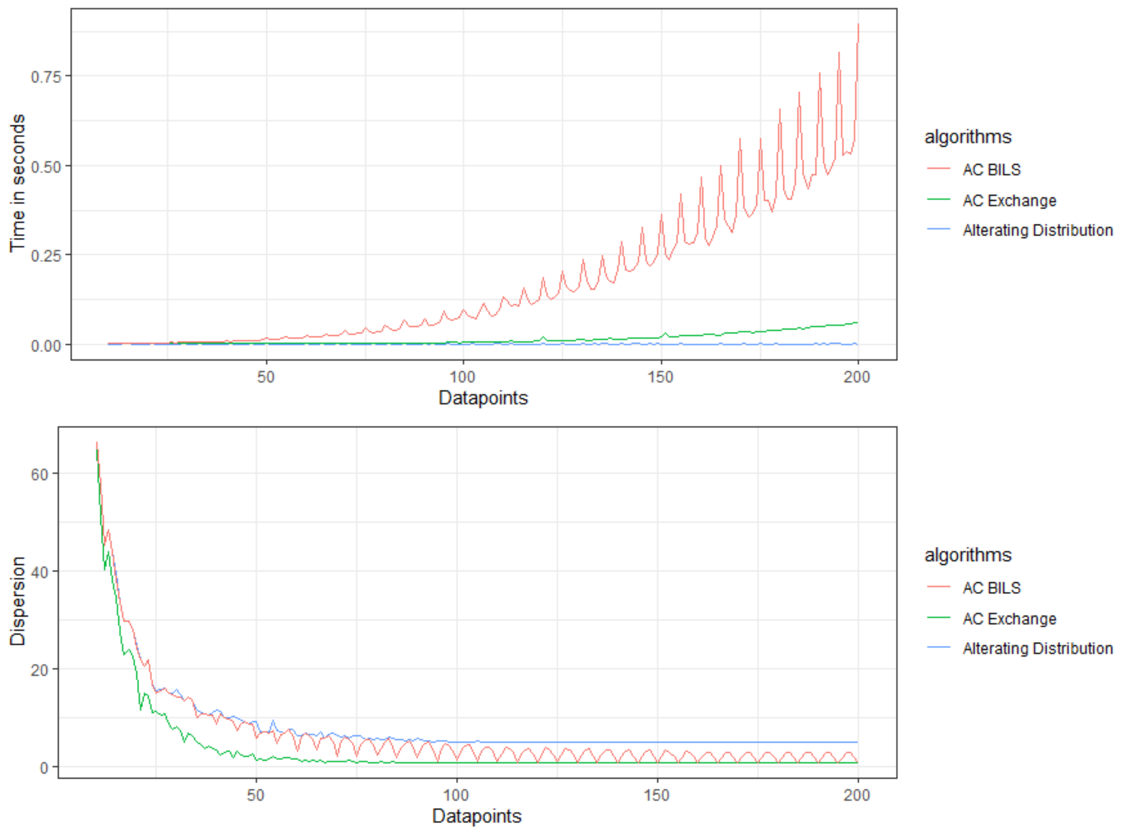

Figure 7: A run-time and dispersion comparison between algorithms for the 1-dimensional Euclidean space with five anticlusters

Figure 7 shows the 5-anticlustering problem with ten iterations for each amount of data points. The data points are normal distributed with zero mean and variance of ten. The BILS heuristic is considerably slower when the number of data points is dividable by the number of anticlusters. That means that the data points can be separated into anticlusters of the same size. This is also visible in figure 7, which is for the 5-anticlustering case and has "bumps" at every fifth value. Breuer encountered this phenomenon in his bachelor thesis too and explained, that many partitions of the corresponding Pareto set are not unique [4]. That means, that they share the same diversity and dispersion with other partitions. When the clusters can be separated into equal size it is more likely that the partitions are the same or at least similar, which leads to the fact that they are all stored in the Pareto set which again slows down the algorithm. Breuer also found that if the data points are integers, the accuracy of the BILS heuristic drops. This also occurs every time the data points can be separated into equal sized anticlusters, as seen in figure 7, which shows the dispersion for different amounts of data points for the 5-anticlustering 1-dimensional euclidean special case. For small amounts of data points the BILS heuristic finds optimal solutions most of the time. However with fifty data points and more the pattern develops, where the diversity is considerably worse when the number of data points is dividable by the number of clusters.

The run time of the exchange algorithm does not seem to be influenced by the amounts of anticlusters. However, the BILS heuristic has a high fluctuation of the run time for different amounts of anticlusters as seen in table 5.

Clusters	2	3	4	5	6	7	8
Alternating	0.00009	0.00008	0.0001	0.00008	0.0001	0.0002	0.00009
Exchange	0.0023	0.0024	0.0021	0.0020	0.0020	0.0019	0.0019
BILS	0.0437	0.0469	0.0805	0.0926	0.0677	0.0694	0.0695

Table 5: Time consumption for different amounts of clusters (average out of 100 each)

6 Diversity

The algorithms presented in section 2 and section 3 all focus on the dispersion objective. However, the diversity objective has many use cases, due to its balanced data distribution. Therefore it would be very beneficial for a polynomial special case to exist. The dispersion objective has the advantage, that it only depends on the minimum distance between two elements in one cluster, whereas the diversity objective the sum of all distances on one cluster contemplates. This is a big issue in regard to finding a polynomial-time solvable special case. There are possibilities to compute the diversity objective in polynomial time with dynamic programming, however, this is only pseudo-polynomial time, by balancing the runtime with memory allocation [12]. For the clustering problem, there are also special cases that can be solved in polynomial time for the diversity objective, like ℓ-Diversity, with ℓ being the minimum number of elements in one cluster, described by Li et. al. [13]. This algorithm is optimal for the 2-clustering problem and uses bipartite matching.

6.1 2-anticlustering

Although the BILS heuristic has no approximation factor, it usually generates good approximations for special cases with either a small k or a small feature space. The fact, that the BILS heuristic tries to optimize diversity and dispersion at the same time, lets BILS find more sophisticated solutions than, for example, Papenberg's algorithm from section 3.4. However, the algorithm by Brucker also approximates the diversity objective. In figure 8 a direct comparison between the BILS heuristic and the Brucker algorithm is displayed.

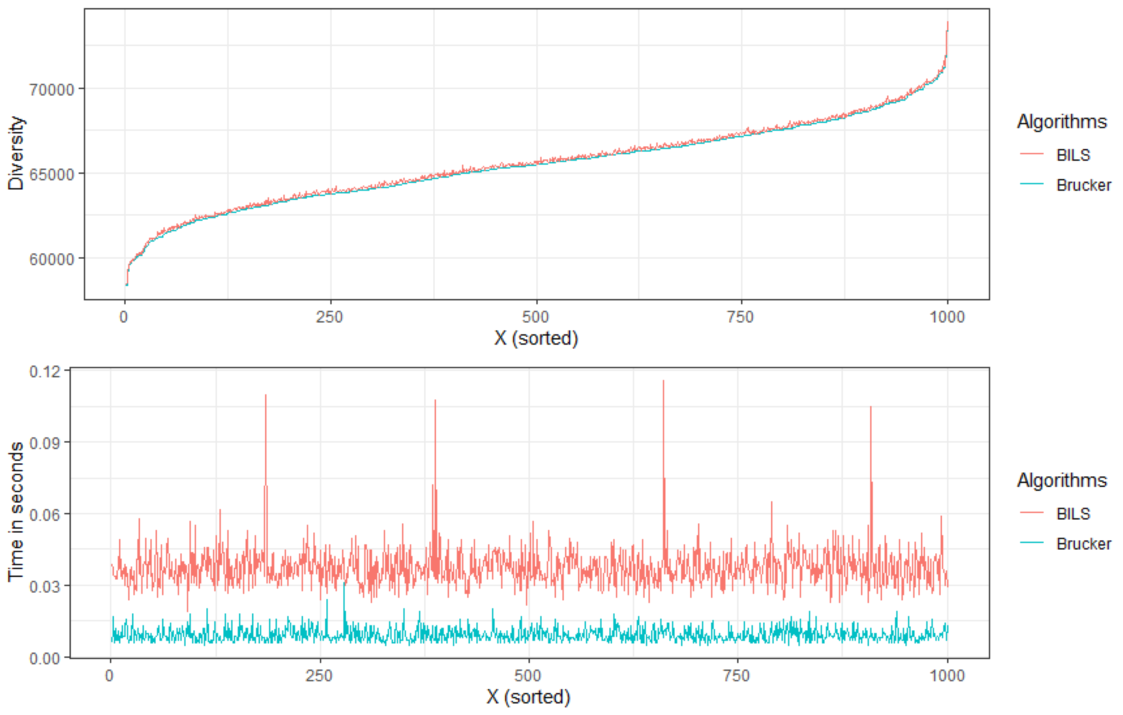

Figure 8: Comparison between BILS and Brucker for the diversity objective with 100 4-dimensional datapoints - sorted by diversity of the Brucker algorithm

The results of BILS and the Brucker algorithm are always close together, however, the solution of the BILS heuristic is almost always a little bit better. In table 7, the BILS heuristic and the Brucker algorithm are compared for different amounts of data points regarding the diversity objective. The reason BILS finds better solutions, especially for higher amounts of data points, is due to the balancing of the anticlustering sizes we have to do for the Brucker algorithm. If we would not balance the anticluster sizes, the Brucker algorithm would find a better solution for the diversity objective in 60 percent of the cases and only in 20 percent it would be worse than BILS. However, the solutions of the BILS heuristic and the Brucker algorithm are always very close. Therefore the Brucker algorithm is, next to always finding an optimal solution for the dispersion objective, a good approximation algorithm for the diversity objective. It is not as good as the BILS heuristic since we have to balance the anticluster sizes, but it may be possible to find solutions that are as good as the ones BILS finds, or even better if one would alter the algorithm for balancing the anticluster sizes to also consider the diversity objective. However, whether the algorithm has an approximation factor requires further examination. The Brucker algorithm is significantly slower than the Papenberg algorithm, but Papenberg's algorithm does not approximate the diversity objective, because after a bipartition is found, it divides the remaining elements into random clusters.

6.2 K-anticlustering

The algorithm for the 1-dimensional anticlustering special case optimizes diversity. The alternating distribution of the data points is the best distribution for the diversity objective in the 1-dimensional special case. If we have an instance $X = (1, 3, 6, 9, 10, 11)$ of six 1-dimensional data points and are trying to distribute them over two anticlusters, the optimal solution for the diversity and dispersion objective would be:

Datapoints	1	3	6	9	10	11
Anticlusters	1	2	1	2	1	2

This distribution has a diversity value of:

$$|6-1| + |10-1| + |10-6| + |9-3| + |11-3| + |11-9| = 34$$

If one would now change the anticluster distribution, there would be two values in each anticluster, that are closer together.

Datapoints	1	3	6	9	10	11
Anticlusters	1	1	2	2	1	2

This distribution has a diversity value of:

$$|3-1| + |10-1| + |10-3| + |9-6| + |11-6| + |11-9| = 28$$

The change does not always make the diversity result worse, however, it cannot make it better than the alternating distribution. Assuming this does not hold for the K-dimensional case and we have an alternating distribution and switch two elements of clusters C_1 and C_2, then

$$\begin{gathered}
\sum_{\substack{x_1,y_1\in C_1\\x_1<y_1}} d(x_1,y_1)+\sum_{\substack{x_2,y_2\in C_2\\x_2<y_2}} d(x_2,y_2)+\cdots+\sum_{\substack{x_K,y_K\in C_K\\x_K<y_K}} d(x_K,y_K)\\
<\sum_{\substack{x_1,y_1\in C_1\\x_1<y_1\\x_1=y_1\neq x_{1_i}}} d(x_1,y_1)+\sum_{\substack{x_1\in C_1\\x_1\neq x_{1_i}}} d(x_1,x_{2_j})+\sum_{\substack{x_2,y_2\in C_2\\x_2<y_2\\x_2=y_2\neq x_{2_j}}} d(x_2,y_2)+\sum_{\substack{x_2\in C_2\\x_2\neq x_{2_j}}} d(x_2,x_{1_i})+\\
\sum_{\substack{x_3,y_3\in C_3\\x_3<y_3}} d(x_3,y_3)+\sum_{\substack{x_4,y_4\in C_4\\x_4<y_4}} d(x_4,y_4)+\cdots+\sum_{\substack{x_K,y_K\in C_K\\x_K<y_K}} d(x_K,y_K)\\
\Leftrightarrow \sum_{\substack{x_1,y_1\in C_1\\x_1<y_1}} d(x_1,y_1)+\sum_{\substack{x_2,y_2\in C_2\\x_2<y_2}} d(x_2,y_2)\\
<\sum_{\substack{x_1,y_1\in C_1\\x_1<y_1\\x_1=y_1\neq x_{1_i}}} d(x_1,y_1)+\sum_{\substack{x_1\in C_1\\x_1\neq x_{1_i}}} d(x_1,x_{2_j})+\sum_{\substack{x_2,y_2\in C_2\\x_2<y_2\\x_2=y_2\neq x_{2_j}}} d(x_2,y_2)+\sum_{\substack{x_2\in C_2\\x_2\neq x_{2_j}}} d(x_2,x_{1_i})\\
\Leftrightarrow \sum_{x_1\in C_1} d(x_1,x_{1_i})+\sum_{x_2\in C_2} d(x_2,x_{2_j})<\sum_{\substack{x_1\in C_1\\x_1\neq x_{1_i}}} d(x_1,x_{2_j})+\sum_{\substack{x_2\in C_2\\x_2\neq x_{2_j}}} d(x_2,x_{1_i})
\end{gathered}\tag{9}$$

This inequality however is not right, since it is not possible to increase the diversity value. In figure 9 one can see, that the number of elements that are closer and further away from the switched points even out. One anticluster may have an increase in its sum of distances, however, then the other anticluster must have a decrease. Either this evens out the increase in the other anticluster or even exceeds it.

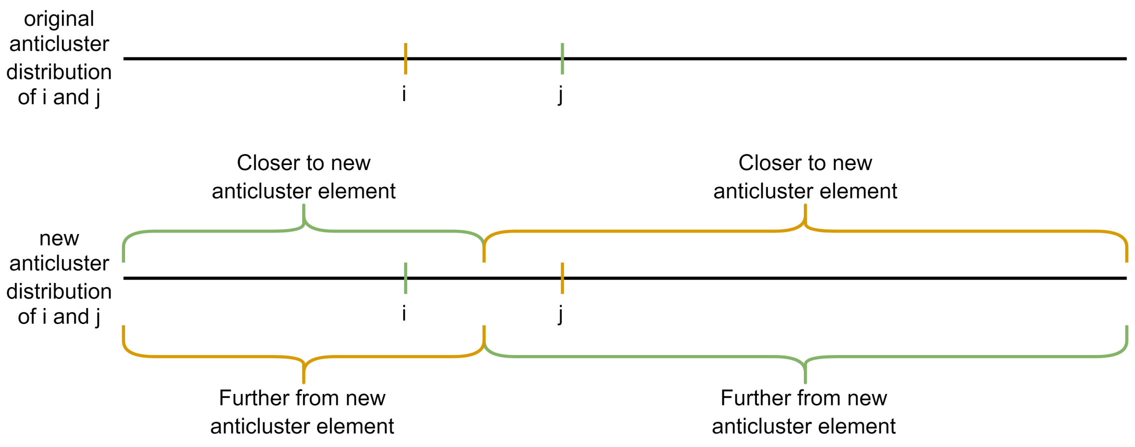
original anticluster distribution of i and j
i
j
Closer to new anticluster element
Closer to new anticluster element
new anticluster distribution of i and j
i
j
Further from new anticluster element
Further from new anticluster element

Figure 9: Impact of change of the anticluster allocation of i and j

7 Discussion

We have now evaluated and implemented three different algorithms for this thesis. First the algorithm of Brucker [5] uses the bipartitioning of graphs to find an optimal solution for the dispersion objective and an approximation for the diversity objective for the 2-anticlustering case with an arbitrary amount of dimensions. For the same special case, the algorithm developed by Papenberg also optimizes the dispersion objective, however, it does not approximate the diversity objective. On the other hand, the algorithm by Papenberg is considerably faster than the algorithm of Brucker. The dispersion objective does not necessarily create balanced distributions of data points. It is for example possible to get a solution as shown in figure 10 with the Papenberg algorithm. This solution uses the Papenberg algorithm until there is no bipartitioning anymore. After that, the remaining data points are sorted manually to get this distribution. This solution is basically a distribution in clusters, with some deviations. The algorithm by Brucker however ensures a more diverse distribution. If one concerns the applications of these algorithms for the distribution of questions in tests made for psychological research with people, it is desired to have high diversity, but maybe also high dispersion to ensure that there is not one question that is too similar to another test-set [14]. Therefore an algorithm that at least approximates diversity and dispersion - like the BILS heuristic - is desirable. For these cases the algorithm by Brucker would be more useful, however, if the data gets very big it might be beneficial to use the algorithm by Papenberg due to its fast performance. After finding a case where the smallest x data points do not form a bipartitioning anymore, the algorithm sorts the rest of the data points into random clusters. Therefore it is rather unlikely to get a distribution like in figure 10. Especially for big amounts of data, the distribution will be very diverse due to its randomness.

The 2-anticlustering special case is also a case, which could find many applications. For example, as stated above, distributing test questions into two tests, one before and one after a psychological experiment. The fact that the number of dimensions of the feature space has a limited impact on the run time also contributes to the applicability of the special case, because data, like test questions, may have many more distinct features that determine their allocation.

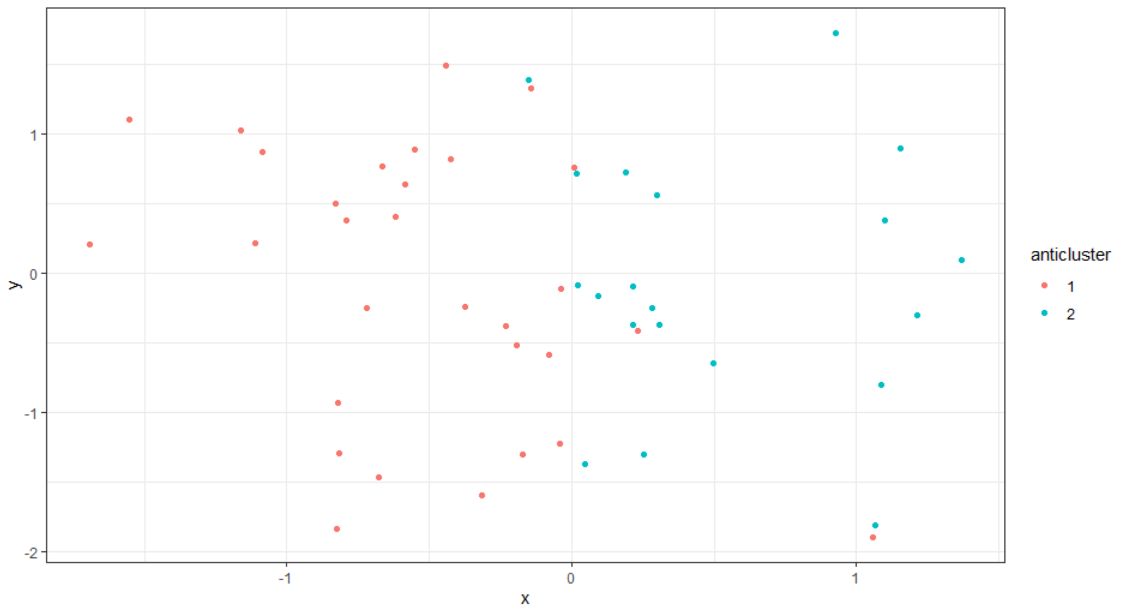

Figure 10: A possible solution for the algorithm of Papenberg with maximizes the dispersion

The algorithm by Brucker [5] is an algorithm originally designed for the 2-clustering special case. However, it is not always the case that the anticlustering version is just the opposite of the clustering problem. This can be seen through the algorithm for the 2-clustering case, described by Bock [15], with the objective function:

$$\max\left(\frac{|c_1| * |c_2|}{N} * ||\bar{x}_{c_1} - \bar{x}_{c_2}||_2\right) \text{ with clusters } c_1 \text{ and } c_2 \text{ and } \bar{x} = \sum \frac{x_i}{N}$$

Bock uses a hyperplane to separate the data, which is generated through (M-1) data points. With this method, there are $(2^M - 2) * \binom{N}{M}$ partitions with M being the dimension of the data. However, this method is not viable for the anticlustering case due to the data points not being separable through a hyperplane in the anticlustering case. The objective function described by Bock is basically the diversity objective function because it sums up all the distances. It may be possible to find another polynomial algorithm that solves this special case with this objective function, but it is difficult to exclude partitions from the beginning without looking at them. This is something Bock's algorithm is able to do for the clustering version.

The K-anticlustering special case with the 1-dimensional euclidean feature space is more limited with regard to finding applications. For this special case, we described an algorithm that distributes the anticlusters over the ascending sorted data in alternating order. This algorithm computes the solution in linear time, plus the run time of a sorting algorithm, which is typically $O(N * log(N))$. It is the fastest algorithm of all the algorithms presented in this paper. In addition, it also optimizes diversity and dispersion at the same time. It is questionable if it is even possible to find an algorithm that optimizes both objective functions for the 2-anticlustering case in polynomial time, because the two objective functions differ so much in their requirements, that it is difficult to find an optimal solution for both in higher dimensions. However, in the process of making this thesis, we did not find any research that explicitly proves, that it is not possible for higher dimensions with two clusters and we have also seen that the algorithm by Brucker, in fact, does approach an optimal solution for the diversity objective, therefore it might be feasible to connect the algorithm of Papenberg with an algorithm that finds an optimal solution for the 2-anticlustering special case in regards to diversity. This algorithm could start after no bipartitioning can be found anymore, since then an optimal dispersion value is already found.

For the 1-dimensional case, it will be difficult to find a research area that uses this special case. It is usually the case that data has multiple dimensions because otherwise, the data is just too simple to extract any meaningful information out of it.

8 Acknowledgement

I would like to thank Prof. Gunnar Klau for giving me the opportunity to write this bachelor thesis and for his support and feedback. I am also very grateful to Martin Papenberg for his support and the discussions in our weekly meetings, which have greatly helped me in the course of writing this thesis.

References

[1] H Späth. “Anticlustering: maximizing the variance criterion”. In: *Control and Cybernetics* (1986), pp. 213–218.

[2] Venceslav Valev. “Set partition principles”. In: *Transactions of the Ninth Prague Conference on Information Theory, Statistical Decision Functions, and Random Processes,(Prague, 1982)* (1983), p. 251.

[3] Teofilo F. Gonzalez. “On the computational complexity of clustering and related problems”. In: *System Modeling and Optimization. Lecture Notes in Control and Information Sciences, vol 38*. Ed. by R.F. Drenick and F. Kozin. Berlin, Heidelberg: Springer, 1982. URL: https://doi.org/10.1007/BFb0006133.

[4] Martin Breuer. “Using anticlustering to maximize diversity and dispersion: Comparing exact and heuristic approaches”. In: (2020).

[5] Peter Brucker. “On the complexity of clustering problems”. In: *Optimization and operations research* (1978), pp. 45–54.

[6] Martin Papenberg. *anticlust*. 2019. URL: https://github.com/m-Py/anticlust.

[7] Elena Fernández, Jörg Kalacsis, and Stefan Nickel. “The maximum dispersion problem”. In: *Omega - The International Journal of Management Science* (2013).

[8] Guangyi Zhang and Aristides Gionis. *Maximizing diversity over clustered data*. 2020. URL: https://epubs.siam.org/doi/pdf/10.1137/1.9781611976236.73.

[9] Jacques Desrosiers, Nenad Mladenovíc, and Daniel Villeneuve. “Design of balanced MBA student teams”. In: *Journal of the Operational Research Society (56.1)* (2005), pp.60–66.

[10] Luis Evaristo Caraballo, José-Miguel Díaz-Báñez, and Nadine Kroher. *A Polynomial Algorithm for Balanced Clustering via Graph Partitioning*. 2018. URL: https://arxiv.org/pdf/1801.03347.pdf.

[11] Michael J Brusco, J Dennis Cradit, and Douglas Steinley. “Combining diversity and dispersion criteria for anticlustering: A bicriterion approach”. In: *British Journal of Mathematical and Statistical Psycholog* (2019).

[12] Tibor Szkaliczki. “Optimal Clustering with Sequential Constraint by Using Dynamic Programming”. In: *The R Journal Vol. 8/1* (2016).

[13] Jian Li, Ke Yi, and Qui Zhang. *Clustering with Diversity*. 2020. URL: https://arxiv.org/pdf/1004.2968v2.pdf.

[14] Martin Papenberg and Gunnar W Klau. “Using anticlustering to partition data sets into equivalent parts”. In: *Psychological methods* ().

[15] H.H. Bock. *Automatische Klassifikation*. Göttingen: Vandenhoeck @ Ruprecht, 1974.

A Tables

In the figure below, each row is calculated by the average of 100 iterations with normal distributed data points.

dimensions	data points	Brucker - dispersion	Brucker - time	BILS - dispersion	BILS - time
2	10	7.167087896	0.001639712	7.143645550	0.001765275
2	50	1.705965342	0.003341427	1.705965342	0.006899939
2	100	1.020978	0.008285217	1.020978	0.03462168
3	10	11.71546448	0.001525605	11.70491717	0.001750903
3	50	4.516541803	0.003376303	4.507480624	0.006655586
3	100	3.09389271	0.00839741	3.08797530	0.03215027
4	10	14.731321	0.001474364	14.686186	0.001507246
4	50	7.305770	0.003208127	7.303520	0.006120982
4	100	5.554850	0.008960285	5.549420	0.03161701
5	10	18.453977	0.001516407	18.401895	0.001555934
5	50	10.063395	0.002936928	10.051350	0.005845835
5	100	8.050842	0.008294218	8.031285	0.03103205
6	10	22.314658	0.001266215	22.282723	0.001547663
6	50	12.982315	0.003827138	12.944430	0.005958827
6	100	10.792532	0.008701644	10.778779	0.03024050
7	10	24.480064	0.001614966	24.438130	0.001668186
7	50	15.675066	0.003564525	15.571754	0.005712645
7	100	12.659529	0.009120035	12.606731	0.02901287
8	10	27.228035	0.001395807	27.184014	0.001605866
8	50	18.004774	0.003500826	17.962598	0.005985312
8	100	15.316060	0.009042275	15.241009	0.02886919
9	10	29.989437	0.001465425	29.919934	0.001565995
9	50	20.291382	0.003460228	20.152215	0.005854776
9	100	17.663237	0.009186375	17.558047	0.02767511
10	10	32.596895	0.001716044	32.477345	0.001726263
10	50	22.726633	0.003337321	22.619030	0.005938587
10	100	19.690223	0.009337652	19.594376	0.02760492

Table 6: Runtime and accuracy comparison between Brucker and BILS for the 2-anticlustering special case - dispersion

In the figure below, each diversity is the average of 100 iterations and the last three colums are the ratio of these 100 iterations for the individual statement. The data is normal distributed and 2-dimensional.

data points	Brucker - diversity	BILS - diversity	Brucker better (%)	equal (%)	BILS better (%)
10	367.8642	374.7669	1	13	86
20	1657.7536	1674.4932	1	0	99
30	3831.0412	3853.3206	0	0	100
40	6920.3747	6950.4355	0	0	100
50	10898.1240	10940.4784	0	0	100
60	15633.8436	15680.8171	0	0	100
70	21210.4544	21262.3959	0	0	100
80	27639.9260	27691.2779	0	0	100
90	35494.1594	35565.9179	0	0	100
100	43759.0626	43835.5921	0	0	100
110	53053.5809	53131.0318	0	0	100
120	63125.4221	63204.0050	0	0	100
130	74446.4636	74541.9722	0	0	100
140	85442.8287	85537.9143	0	0	100
150	98980.8261	99084.1150	0	0	100
160	112393.1286	112513.4389	0	0	100
170	127021.2349	127139.4651	0	0	100
180	141745.1371	141862.2181	0	0	100
190	159257.9248	159395.8260	0	0	100
200	176399.3584	176543.9753	0	0	100

Table 7: Runtime and accuracy comparison between Brucker and BILS for the 2-anticlustering special case - diversity

B Source Code

The source code for this thesis is accessible at: https://gitlab.cs.uni-duesseldorf.de/albi/albi-students/ba-joshua-kokol