Maximizing dispersion for anticlustering

Max Diekhoff

A thesis presented for the degree of
Bachelor of Science

Algorithmic Bioinformatics
Heinrich Heine University Düsseldorf
Germany
3rd February, 2023

Acknowledgments

I would like to express my gratitude to my first assessor Prof. Dr. Gunnar Klau for proposing the captivating topic for this thesis and my second assessor Dr. Martin Papenberg for always providing guidance. Furthermore, I would like to thank my daily supervisor Khoa Tran Nguyen for many productive meeting and valuable help in writing this thesis.

Finally, I want to thank my family, especially my parents, for always supporting me in every conceivable way and Milena, without whom I would not even have tried to start this journey.

Abstract

Partitioning elements into K anticlusters with high between-group homogeneity and high heterogeneity within groups is called the K-anticlustering problem in m-dimensional feature space that arises in experimental psychology and other fields. In this thesis we focus on max dispersion as the objective function, for which the K-anticlustering problem is NP-hard for $K \geq 3$ and $m \geq 2$. We explore how an exact polynomial time algorithm for 2-anticlustering can be adapted for K-anticlustering using a modified ILP for *equitable graph K-coloring*. The algorithm was tested with synthetic normally distributed datasets for different K and m, and against the *bicriterion iterated local search* (BILS) heuristic. The results show that the proposed algorithm can solve the K-anticlustering problem in a reasonable timeframe for a variety of input specifications, e.g. 3-anticlustering for N=10000 and 6-anticlustering for N=500 in 10 minutes each. We show that the algorithm can be a viable alternative for the BILS heuristic, as it runs faster for 3-anticlustering and just two to five times longer for 10-anticlustering where BILS does not find the optimal result in most cases. Finally, we propose some ideas how this algorithm can be further improved.

Contents

1 Introduction

Partitioning a pool of elements into groups is a very common problem in multiple areas, which has led to numerous approaches motivated by different goals. Within that, clustering is a very common task that aims to form distinct, well-separated groups that include similar elements.

Sometimes, the opposite is required, to form groups with high between-group similarity and high within-group heterogeneity. This is known as anticlustering, a term that was independently introduced by Späth [1] and Valev [2]. Anticlustering can be useful for a variety of cases, for example, splitting a group of experimental subjects in psychological research into similar focus groups or drafting multiple exams from a pool of questions with the same difficulty level that cover different topics.

The anticlustering problem is defined by the number of anticlusters in which the data should be separated and the distances between the data points. An anticlustering problem with K anticlusters and data points with m features is called an K-anticlustering problem with m-dimensional feature space [3].

Similar to clustering, there are different objective functions for anticlustering to evaluate the quality of solutions. In this thesis, we will focus on dispersion as the objective function for anticlustering, which is defined by the minimum difference between two elements within the same anticluster. The anticlustering problem with this objective function is often referred to as *max dispersion* [4]. Max dispersion is especially desirable when we want to ensure that every pair of elements within the same anticluster is as different as possible. A good example for that is the exam problem mentioned earlier. We want the questions in every exam to cover different topics, therefore we cannot have similar questions in the same exam, which is supposed to be achieved by the dispersion objective.

The K-anticlustering problem for max dispersion is NP-hard for $K \geq 3$ and $m \geq 2$ [5], therefore finding an optimal solution for large inputs can be computationally infeasible, and it is desirable to develop good heuristics and algorithms that find optimal solutions in reasonable time. In his bachelor thesis, Joshua Kokol implemented and discussed an exact algorithm for max dispersion for the 2-anticlustering problem under supervision by Martin Papenberg [6].

In this thesis, our primary research question is to examine how that algorithm can be adapted for the general K-anticlustering problem. Additionally, we will test how the computation times behaves for different K and m and compare the algorithm to the *bicriterion iterated local search* (BILS) heuristic by Brusco et al. [4] to study if the algorithm is a viable alternative for anticluster-applications.

In Section 2, we present the theoretical concepts required for a complete comprehension of this thesis, and in Section 3 we describe our algorithm, give a proof of correctness and show some implementation specifications in detail.

The results of our tests and how they were conducted are presented in Section 4 with a focus on the computation time needed for different parameters and a comparison to the BILS heuristic [4] from the 'anticlust' R-package presented by Papenberg and Klau in 2021 [3].

In Section 5 we critically discuss the methods and results, specifically how higher dimensionality impacted runtimes and which specifications we considered for the ILP formulation.

Finally, the last section contains a synopsis of our findings and outlines possible future work.

2 Preliminaries

In this section, we will present the concepts that are used in this thesis.

We will discuss *Integer Linear Programming*, *Graph K-coloring* and the problem formalization for *K-anticlustering*.

2.1 Integer Linear Programming

An *Integer Linear Program* (ILP) is a mathematical program for optimization and can be written as follows:

$$min\ c^T x \tag{1}$$

$$s.\ t.\ Ax \leq b \tag{2}$$

$$x \in \{0, 1\} \tag{3}$$

In its basic form, an ILP consists of an *objective function* (1) to be minimized, linear inequality *constraints* (2), and variables (3) that are restricted to be non-negative integers. The solution space is defined by the constraints. Solving an ILP means optimizing the objective function within the solution space. To accomplish this, an ILP solver determines values for the variables x that satisfy the constraints. The inputs for an ILP are given as matrix A and vectors b and c. There are several different ways to express an ILP, which can be transformed into one another.

2.2 Graph K-coloring

Graph coloring is one of the most well-known optimization problems in the field of graph theory. In its classical form, we want to assign a color to each vertex $v \in V$ from a graph $G = (V, E)$ in such a way that no adjacent vertices are assigned the same color, which is called a *vertex coloring*. It is often desirable to know the least amount of colors needed, which is known as the *chromatic number* $\mathcal{X}$.

The corresponding decision problem is to decide whether a proper coloring with the maximum of K colors exists for a given graph G. This is known as *graph K-coloring*.

A special case for graph coloring is the *equitable coloring problem*, a term and problem first introduced by Meyer in 1973 [7], where an *equity constraint* is added to ensure that two color classes cannot differ by more than one in size. Like many optimization problems on graphs, the equitable coloring problem is NP-hard [8]. A graph coloring that satisfies this equity constraint is called an *equitable K-coloring* and the *equitable chromatic number* $\mathcal{X}_{eq}(G)$ is the minimum K for which G has an equitable K-coloring [9]. Equitable K-coloring is very close to the problem we need to solve in our algorithm.

2.3 K-anticlustering

The input for K-anticlustering consists of the number of anticlusters K and an $N \times N$ distance matrix D that contains the pairwise distances between data points, $x_1, ..., x_N \in X$ where X is the set of all N data points. Those data points have m features and will be represented as m-dimensional vectors in Euclidean space. This is necessary because we need to be able to measure a distance between our data points to calculate D, for which we will use the Euclidean distance that is defined as follows:

$$d(p,q) = \sqrt{\sum_{i=1}^{m} (q_i - p_i)^2}$$

We define an anticluster as a set $c_k \subset X$ of data points $x_1, \ldots, x_l \in X$. A partition of our input is defined as $\pi = c_1, \ldots, c_K$ with the following properties:

$$\bigcup_{i=1}^{K} c_i = X \tag{4}$$

$$c_i \cap c_j = \emptyset \quad \forall i, j \in \{1, \ldots, K\}, i \neq j \tag{5}$$

$$|c_i - c_j| \leq 1 \ \forall i, j \in \{1, \ldots, K\}, i \neq j \tag{6}$$

Restrictions 4 and 5 ensure that every data point is assigned to exactly one anticluster. Therefore, every partition π of X is a feasible solution for the K-anticlustering problem. Additionally, the last restriction 6 ensures that all anticlusters differ by at most one in size, a property required by many anticlustering-applications [3].

In this thesis, we will focus on dispersion as the objective function for K-anticlustering. An objective function $f :\rightarrow \mathbb{R}_0$ assigns a non-negative real number to each partition π.

The dispersion objective is the maximization of the minimum distance between any two elements in the same anticluster. The dispersion value of a partition π is defined as:

$$\text{dispersion}(\pi) = \max_{k \in 1,\ldots,K} \min_{x_i, x_j \in c_k, i<j} d(x_i, x_j)$$

To illustrate this, we want to give a simple example and the optimal 3-anticlustering for max dispersion for the nine data points in Table 1. The pairwise distance between the data points are given as distance matrix D.

$$D = \begin{bmatrix} 0 & 2.24 & 1.41 & 7.21 & 7.07 & 6.08 & 7.07 & 7.07 & 8.06 \\ 2.24 & 0 & 3 & 8.54 & 8.06 & 5.10 & 6.08 & 6.70 & 8.49 \\ 1.41 & 3 & 0 & 5.83 & 5.66 & 5.39 & 6.32 & 6 & 6.71 \\ 7.21 & 8.54 & 5.83 & 0 & 1.41 & 7.28 & 7.62 & 5.83 & 3.61 \\ 7.07 & 8.06 & 5.66 & 1.41 & 0 & 6.08 & 6.32 & 4.47 & 2.24 \\ 6.08 & 5.10 & 5.39 & 7.28 & 6.08 & 0 & 1 & 2.24 & 5.10 \\ 7.07 & 6.08 & 6.32 & 7.62 & 6.32 & 1 & 0 & 2 & 5 \\ 7.07 & 6.71 & 6 & 5.83 & 4.47 & 2.24 & 2 & 0 & 3 \\ 8.06 & 8.49 & 6.71 & 3.61 & 2.24 & 5.10 & 5 & 3 & 0 \end{bmatrix}$$

	m_1	m_2
x_1	1	3
x_2	2	1
x_3	2	4
x_4	5	9
x_5	6	8
x_6	7	2
x_7	8	2
x_8	8	4
x_9	8	7

Table 1: nine data points with two features in a 9x2 matrix

The optimal dispersion value for this example is 5.10 with the anticlusters $c_1 = \{x_2, x_5, x_7\}$, $c_2 = \{x_3, x_6, x_9\}$ and $c_3 = \{x_1, x_4, x_8\}$. The anticlusters are visualized as triangles in Figure 1.

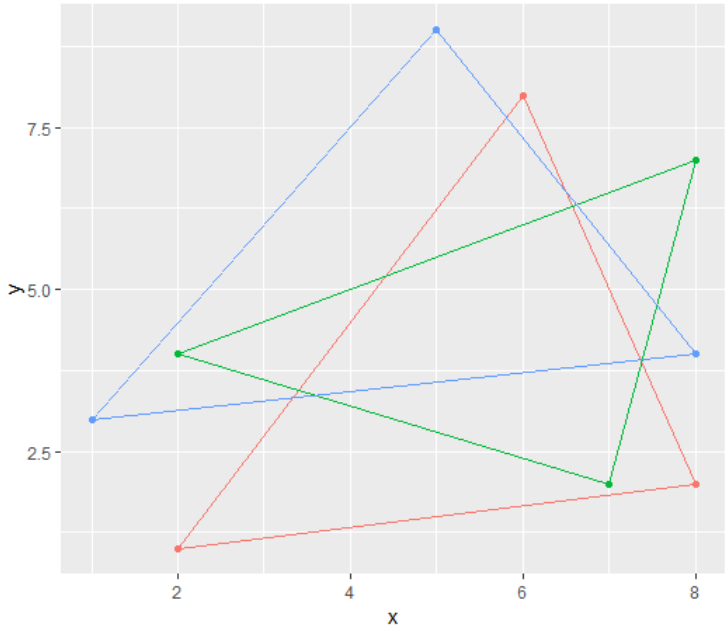

Figure 1: Anticlusters with optimal dispersion value

3 Methods

In this section, we will go into detail how we solve the K-anticlustering problem for max dispersion for general $K \in \mathbb{N}$.

3.1 K-anticlustering dispersion algorithm

As Joshua Kokol described in his bachelor thesis, it is possible to calculate the optimal dispersion value for the 2-anticlustering problem in polynomial time [6]. The algorithm uses graph theory by modelling data points as nodes $v \in V$, and consecutively adds edges to the graph between the nodes with minimal distance until there is no bipartioning anymore. If the graph is no longer bipartite, the distance that was looked at in the last step is the optimal dispersion value.

To expand this idea for the K-anticlustering problem, we need a termination condition for the general case, corresponding to the bipartitioning for the 2-anticlustering problem. It is possible to use a modified version of equitable graph K-coloring for this situation, which we will show in Section 3.2.

The steps for our algorithm are the following, with the $N \times N$ distance matrix D and the number of anticlusters K as input:

1. Initiate an empty, undirected, unweighted graph $G = (V, E)$
2. Order the distances in ascending order
3. Set current distance d to minimal distance
4. Add all elements with distance d to another element as nodes to V
5. Add edges between all nodes whose corresponding elements have pairwise distance d between each other to E
6. Check if G has an equitable graph K-coloring
   (a) if yes: Set d to next shortest distance and go to step 4
   (b) if no: optimal dispersion is d and the optimal clusters are retrieved from the previous iteration

We call every repetition of steps (4) to (6) one iteration of our algorithm.

To be able to retrieve the anticlusters the results of the last successful K-coloring are stored in step (6). The retrieving is done by grouping elements with the same color in the same anticluster and randomly assigning elements that were not included in the graph to anticlusters in a way that satisfies the size constraint.

With equitable K-coloring being an NP-hard problem, the computation time of this algorithm is dominated by the check in step (6) and how often it has to be executed. At worst the number of iterations is in $O(N^2)$, when all distances are looked at one by one.

3.2 K-coloring ILP

As we stated in the previous subsection, we need to check if our graph G has an equitable K-Coloring to determine if the optimal dispersion value was reached. Jabrazilov and Mutzel [10] describe the classical assignment-based ILP model for a graph $G = (V, E)$ where every vertex $v \in V$ is assigned a color i as following.

For each color $i \in \{1, \ldots, H\}$ we introduce assignment variables w_i, with $w_i = 1$ if a color is assigned and $w_i = 0$ otherwise. For each node $v \in V$ and color $i \in \{1, \ldots, H\}$, we introduce assignment variables $x_{v,i}$ with $x_{v,i} = 1$ if a vertex v is given color i and $x_{v,i} = 0$ otherwise.

Since it is not known how many colors will be needed at the time of modelling, an upper bound H for the number of colors is needed. H can be set as the result of a heuristic that gives an upper bound or at $|V|$, which is the worst case for graph coloring. Given that the model can be written as:

$$\min \sum_{1 \leq i \leq K} w_i \tag{7}$$

$$s.t. \sum_{i=1}^{H} x_{v,i} = 1 \qquad \forall v \in V \tag{8}$$

$$x_{u,i} + x_{v,i} \leq w_i \qquad \forall (u, v) \in E, i = 1, \ldots, H \tag{9}$$

$$x_{v,i}, w_i \in \{0, 1\} \qquad \forall v \in V, i = 1, \ldots, H \tag{10}$$

The objective function (7) ensures that the number of used colors is minimized, and therefore the result gives the chromatic number of G. The equations (8) make sure that every vertex gets assigned exactly one color. Constraints (9) ensure that for every edge $(u, v) \in E$ the connected vertices u and v are assigned different colors and that no unused color is assigned to a vertex. The assignment variables are binary, which is reflected in (10).

If the number of vertices $|V|$ is used as upper bound H, the number of variables is in $O(|V|^2)$ and the number of constraints in $O(|V|^3)$ for this ILP. For the K-anticlustering problem, we do not need to know the chromatic number of G, but only if G is K-colorable. To adapt the ILP to the necessities of our problem, we choose K as upper bound H which makes the ILP unsolvable if the graph is not K-colorable, which is all we need to know for our terminating condition.

Additionally, we also need constraints to make sure, that every color i is not assigned more than $\lceil \frac{N}{K} \rceil$-times, so we can retrieve anticlusters that differ at most one in size from the solution.

$$\lfloor \frac{N}{K} \rfloor \leq \sum_{v \in V} x_{v,i} \leq \lceil \frac{N}{K} \rceil \quad i = 1, \ldots, K \tag{11}$$

To do so, we adapt the constraints (11) proposed by Diaz et al. [11] to formulate an ILP for equitable K-coloring. Since only the data points needed are added as nodes to the graph, we just need the upper bound for the number of nodes assigned to any color.

With these changes, we get the following ILP:

$$\min \sum_{1 \leq i \leq K} w_i \tag{12}$$

$$st. \sum_{i=1}^{K} x_{v,i} = 1 \qquad \forall v \in V \tag{13}$$

$$x_{u,i} + x_{v,i} \leq w_i \qquad \forall (u,v) \in E, i = 1, \ldots, K \tag{14}$$

$$\sum_{v \in V} x_{v,i} \leq \lceil \frac{N}{K} \rceil, \qquad i = 1, \ldots, K \tag{15}$$

$$x_{v,i}, w_i \in \{0,1\} \qquad \forall v \in V, i = 1, \ldots, K \tag{16}$$

As we can see, the number of variables is now in $O(K \cdot |V|)$ and the number of constraints in $O(|V| \cdot (K \cdot |E|))$, which is a vast improvement for reasonable values for K.

3.3 Proof of Correctness

The dispersion objective is defined as the minimal distance between two elements within any anticluster. We use this characteristic by going from the smallest to the largest distance between two elements. As long as we find an equitable K-coloring in a given step, we have the possibility to put every pair of elements that has distance at or below the current distance in different anticlusters, with every color representing an anticluster. Not finding a solution for the graph K-coloring ILP in an iteration means that we have to put at least two elements with the current distance in the same group. Since all elements with lower distance between them can be assigned to different anticlusters and all other distances are higher, that current distance d is the optimal dispersion value and the anticlusters can be retrieved from the last ILP solution by randomly adding data points not represented in the graph to anticlusters in a way that the data points are equally distributed.

4 Results

We used data with the following characteristics: Every test input consists of the distances between N elements in matrix form and the number of anticlusters K. The N data points for the input data were generated by creating normally distributed numbers from $[0, 10]$ for all m feature dimensions, and the distance matrix D by calculating pairwise Euclidean distances between those data points. We used fixed random seeds in the process of creating the input sets to ensure reproducibility.

We implemented and ran our algorithm using R and Gurobi [12], a math programming solver that comes with an R interface and was suited to solve our ILPs. We used the R programming language, thereby our algorithm can be included in the anticlust package [3].

All tests are run on a computer with an Intel® Core™i7-12700H CPU (Intel Corporation, Santa Clara, CA, USA) at 4.7 GHz with 32 GB of RAM.

For every test, we set a wall time of two days and stored the following information:

- number of data points n
- number of anticlusters K
- number of feature dimensions m
- runtime in seconds
- number of iterations needed
- dispersion objective

4.1 3-anticlustering

In this section, we evaluate the performance in runtime of our 3-anticlustering algorithm for different input sizes and feature dimension numbers.

At first, we examined up to which input size N the algorithm would run within a reasonable time. To do so, we started with $N = 6o$ data points, conducted tests for ten different inputs and increased N by 60 until the computation time reached ten minutes.

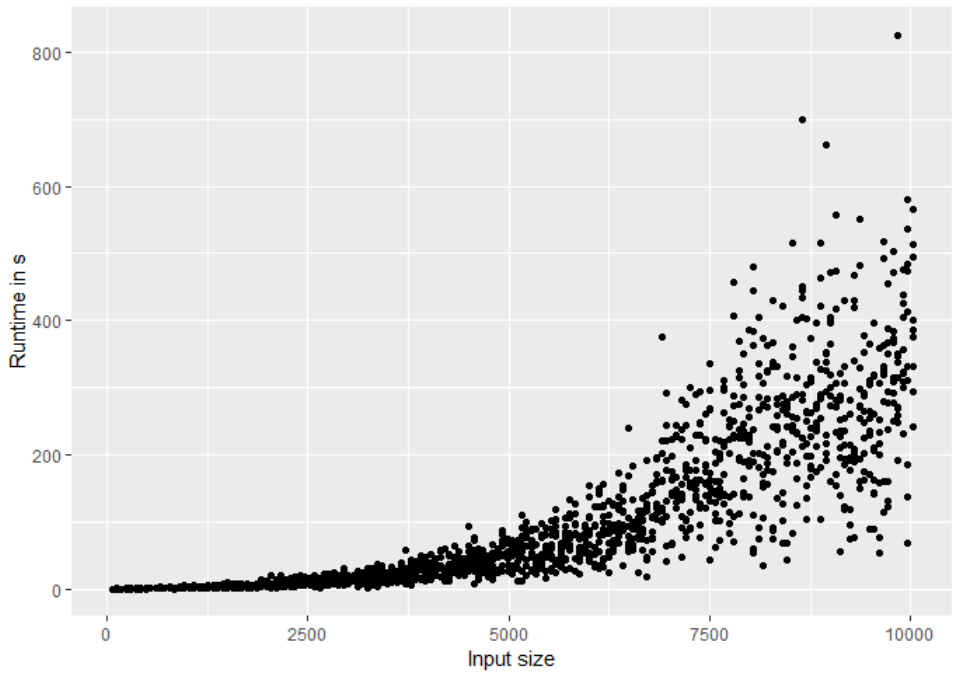

Figure 2: Runtimes for 3-anticlustering with two features

Figure 2 shows the results for 3-anticlustering in 2-dimensional input space, and we observe that input sizes up to 10000 can be run in ten minutes on our hardware for most cases. Furthermore, it becomes clear that the runtimes can vary extremely for different inputs of similar size, e.g. from around 80 seconds up to over 800 for inputs near 10000 in size.

We then conducted ten tests for every input size $N \in 60, 120, 180, \ldots, 1200$ for $m \in 1, 2, 5, 10, 50$ features to compare the computation time needed.

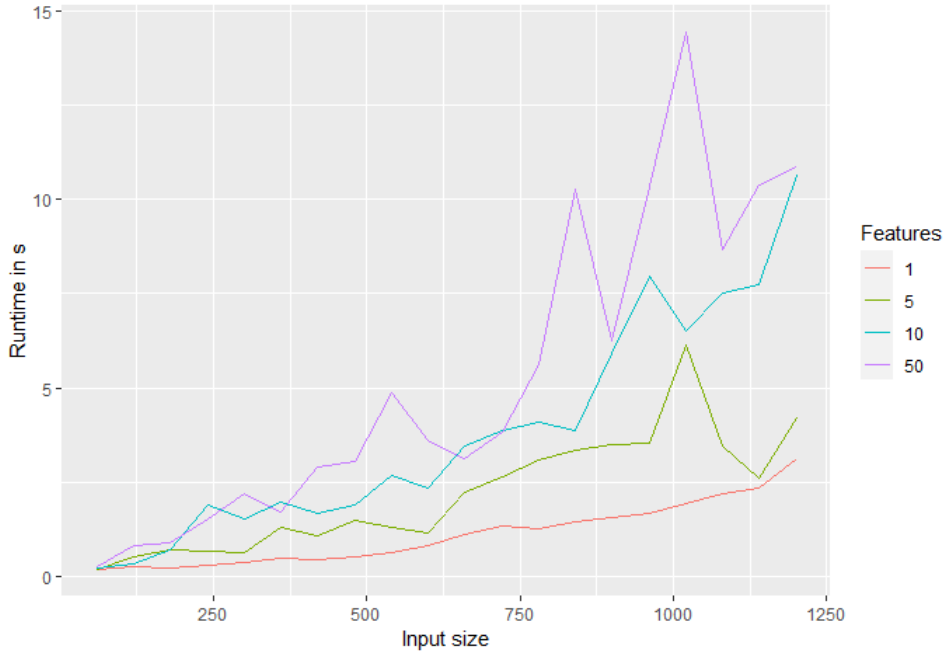

Figure 3: Runtime comparison for different number of features for 3-anticlustering

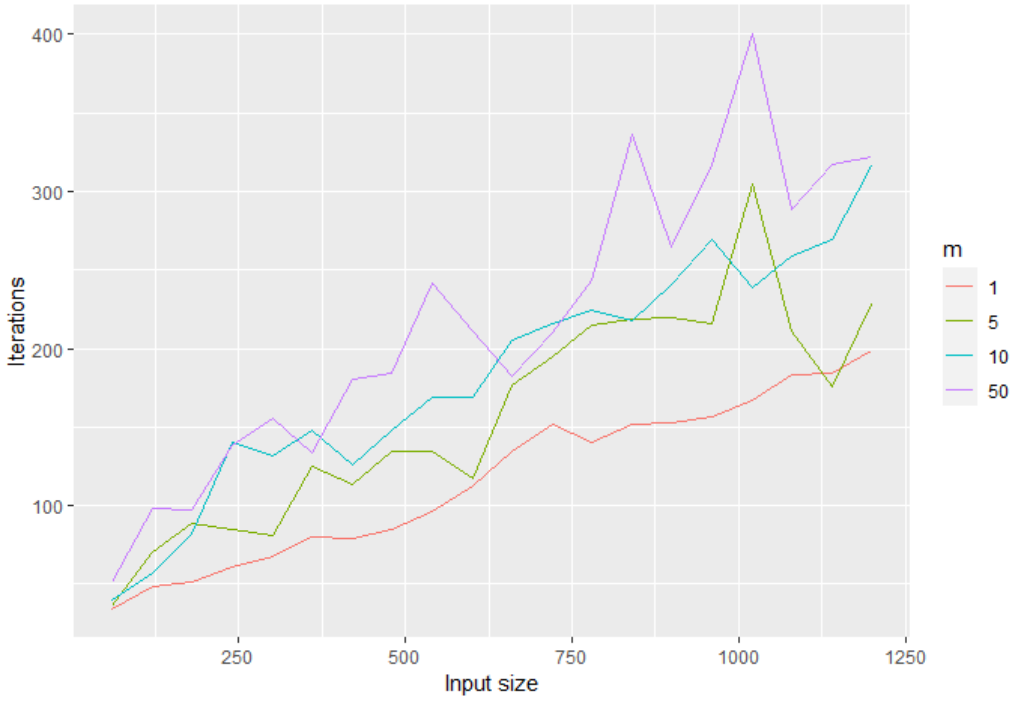

Figure 4: Iterations needed for different number of features for 3-anticlustering

Figure 3 shows the results of these tests, where we computed the runtime averages for every input size, and Figure 4 shows the average number of iterations needed for these tests. We can observe that the runtime and number of iterations increases with more features, which was unexpected, since in theory the number of features should be irrelevant for the runtime of our algorithm. Furthermore, we can see very similar curves for runtime and number of iterations for all number of features, which suggests a strong correlation between runtime and number of iterations.

This correlation might be explained with the distribution of data points in the input sets. If there are more than K data points close to each other with most other distances being greater, the algorithm would terminate after very few iterations and therefore with a short runtime. Conversely, if the data points are mostly evenly distributed within the input space, the algorithm needs more iterations and consequently longer runtimes. This could also explain the high variance in computational time we observed for 3-anticlustering in two-dimensional feature space.

4.2 4- to 7-anticlustering

We will now take a look at runtimes for different numbers of anticlusters K.

We set the number of features to $m = 2$ and conducted test for $N \in \{60, 120, 180, \ldots\}$ until the computational time exceeded ten minutes for $K = 4, 5, 6$ and for $N \in \{42, 84, 126, \ldots\}$ for $K = 7$. We chose different N for $K = 7$ to get anticlusters of the same size and to get more results, since we expected the runtime to grow faster than for smaller K.

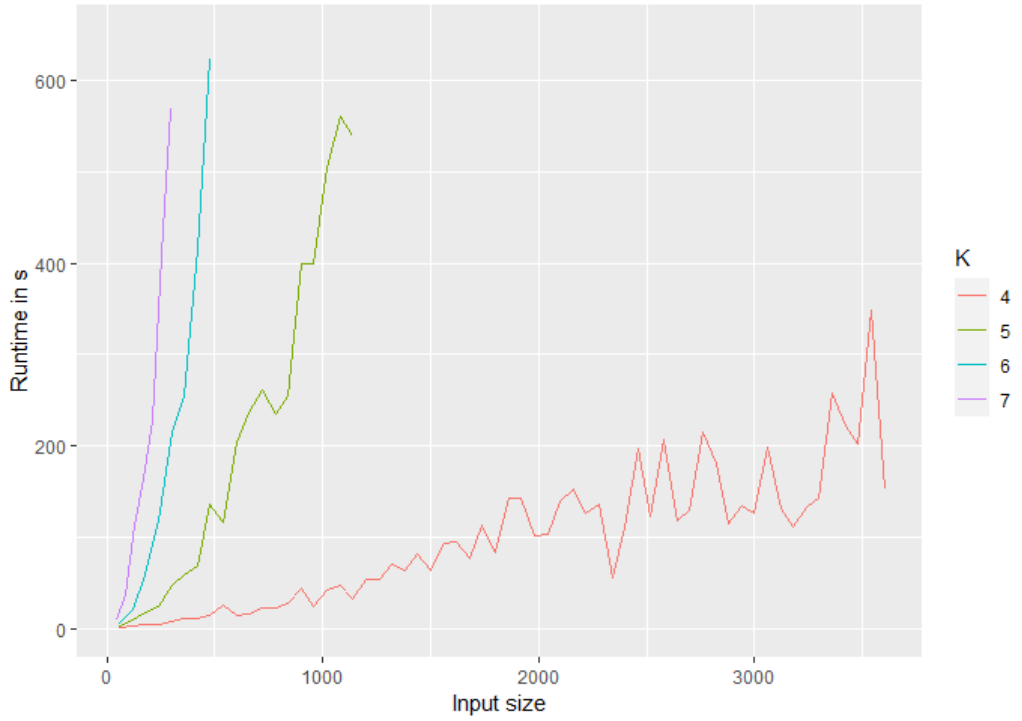

Figure 5: Runtime comparison for different numbers of anticlusters

Figure 5 shows the average runtimes for different input sizes for those K. We can solve the 5-anticlustering problem up to $N = 1200$ in ten minutes, and the 6- and 7-anticlustering problem for input sizes of 500 and 350 respectively. The test for 4-anticlustering was the only one reaching the wall time of two days and stopped at $N = 3600$ with average computation times of three to five minutes for the larger data sets.

The runtimes clearly grow with a higher number of anticlusters. This was expected, since increasing the number of anticlusters impacts the runtime of our algorithm negatively in two different ways: First, the more anticlusters we have, the more colors we allow for the equitable K-coloring, therefore we will need more iterations of our algorithm until no equitable K-coloring is found. Second, the size of the ILPs to be solved by our algorithm scales with K and the number of edges, which increases in every iteration, as we have stated in Section 3.3.

4.3 Comparison to BILS heuristic

In the following section, we compare our algorithm with the BILS heuristic developed by Brusco et al. [4]. We used the version included in the anticlust package [3] that was implemented by Martin Breuer in his bachelor thesis [5].

We compared the computational times needed and measured the accuracy of the Brusco heuristic for both 3-anticlustering and 10-anticlustering with 10 features. We used 10000 repetitions for every run of the heuristic, a value recommended by the authors [4]. For 3-anticlustering we examined input sizes from $N \in \{60, 120, \ldots, 1800\}$ and $N \in \{20, 30, \ldots, 100\}$ for 10-anticlustering and performed 20 tests for every input size. To compare the computation

times the average of those 20 tests was calculated for every input size.

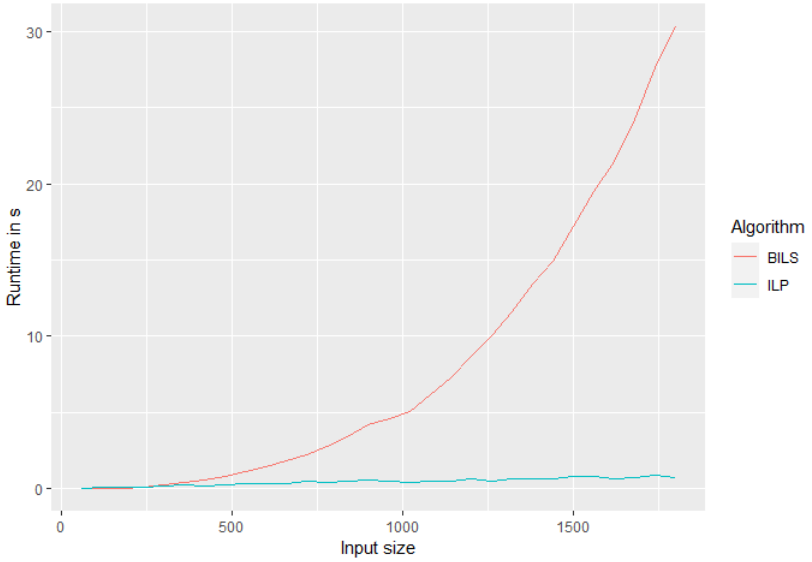

Figure 6: Runtimes for 3-anticlustering with ten features

For 3-anticlustering the runtimes of our algorithm are consistently as fast or faster than those of the BILS heuristic for inputs up to $N = 1800$ in size as we can see in Figure 6.

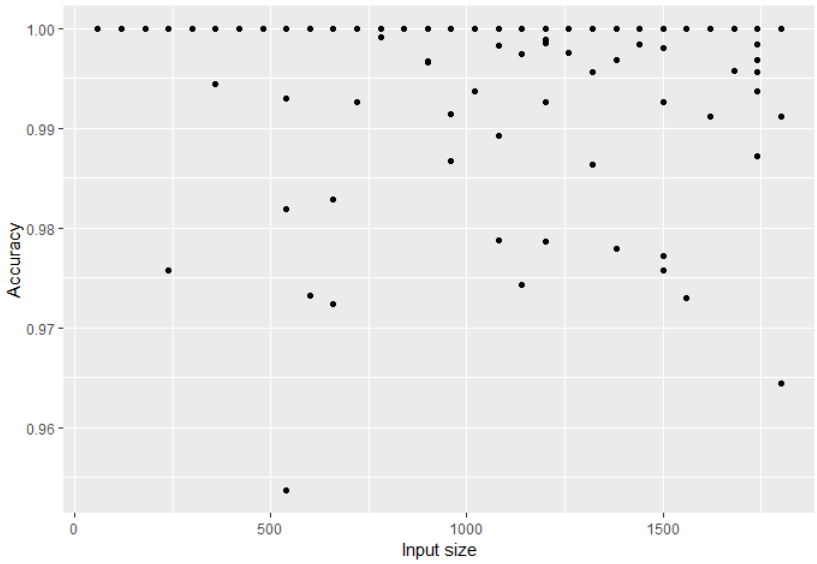

Figure 7: BILS accuracy for 3-anticlustering

Figure 7 shows that the BILS heuristic finds the optimal dispersion value in most cases for these input specifications, however there are non-optimal solutions for as few as $N = 240$ data points and the frequency of non-optimal solutions seems to increase with input size. The worst accuracy observed was 0.954 for $N = 540$.

Given these results, our algorithm performs better for 3-anticlustering and appears to be the better option over the BILS heuristic for at least these input specifications.

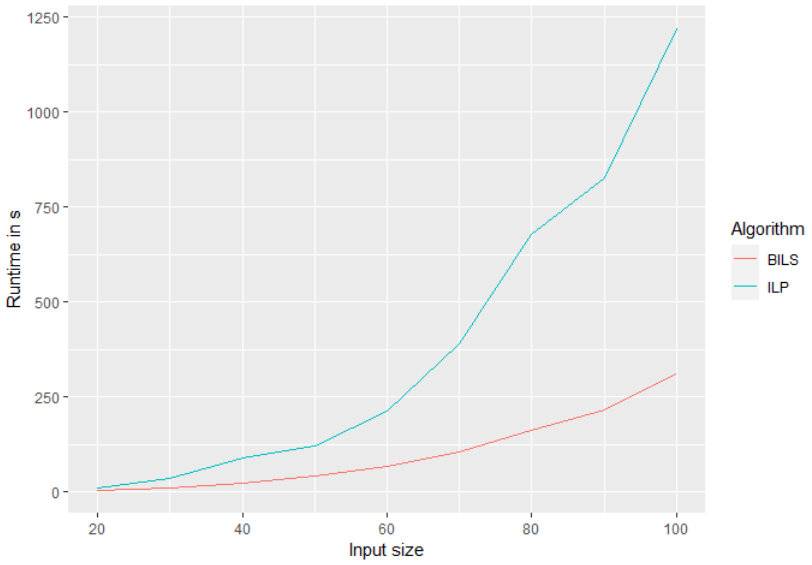

Figure 8: Runtimes for 10-anticlustering with ten features

For 10-anticlustering we can observe the inverse effect with the BILS heuristic consistently outperforming our algorithm in computational time by factor two to five for the input sizes from $N \in \{20, 30, \ldots, 100\}$ as shown in Figure 8.

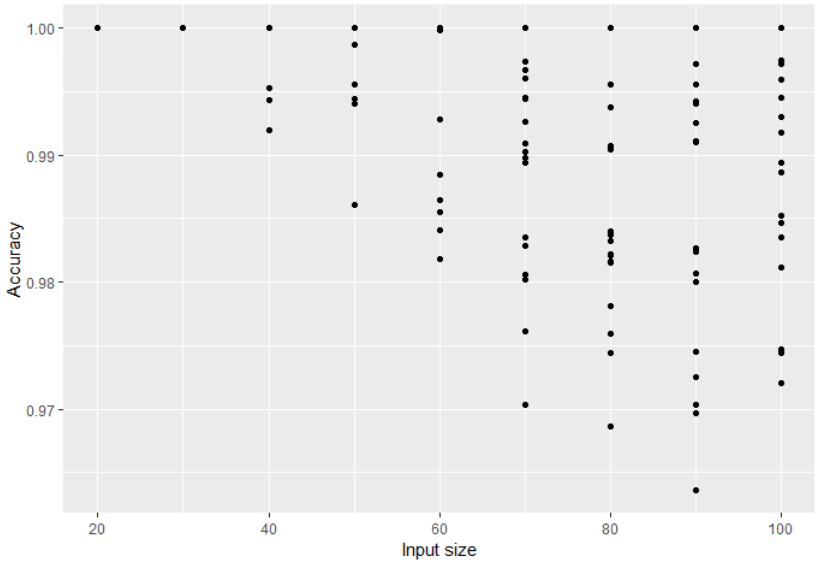

Figure 9: BILS accuracy for 10-anticlustering

Figure 9 shows that the BILS heuristic finds the optimal dispersion value for all tests with $N = 20, 30$, but the number of non-optimal solutions increased with input size similar to 3-anticlustering. For $N \geq 70$ the BILS heuristic provided non-optimal solutions more than half of the time with accuracies ranging mostly from 0.97 to 1.

We can conclude that there is no clearly better option between our algorithm and the BILS heuristic for 10-anticlustering, but a trade-off between computation time and accuracy that has to be evaluated depending on the anticlustering-application.

From those results for 3- and 10-anticlustering we presume that the BILS heuristic deals better with a higher number of anticlusters than our algorithm regarding runtime.

5 Discussion

In the following section, we will critically discuss the methods and results presented in this thesis.

We achieved our main goal in implementing an algorithm that can solve the K-anticlustering problem in a reasonable timeframe for a variety of input specifications. The results show that our algorithm is a viable or better alternative to the BILS heuristic for the K-anticlustering problem depending on the input specifications. Therefore, this algorithm can be useful for many anticluster-applications.

5.1 Dimensionality

As we have observed in Section 4.1 the computation time needed for our 3-anticlustering increases with more features for our synthetic input data, although the algorithm only uses the distance matrix as input and therefore more features should not increase computation time in theory.

We believe this is due to the *curse of dimensionality* a term first introduced by Bellman in 1961 in the context of *dynamic programming* [13] and nowadays commonly used in many fields to refer to different effects observed while analyzing data in high-dimensional spaces. One of these effects is that the distances between data points can converge to the same value for high dimensionality for commonly used metrics, which was explored by Beyer et al. in the context of the *nearest neighbor* problem, and they demonstrated the effect for as few as ten dimensions [14]. This is nowadays often called *distance concentration* and was examined for the Euclidean distance we used among other distance metrics by Aggarwal et al. in 2001 [15].

Distance concentration could explain why our algorithm needs more iterations to find a solution the more features the data points have. When the pairwise distances between data points become more and more similar, it seems reasonable to think that more iterations are needed until there is no more equitable K-coloring.

5.2 ILP formulations

In Section 3.3 we described the ILP used in our algorithm, which takes the assignment-based ILP formulation for equitable K-coloring with two main changes. We decided to use the assignment-based ILP model for two main reasons: First, the assignment variables enable the modelling of data points as nodes in a graph and thus retrieving anticlusters from the ILP solution is very easy.

Second, the *symmetry problem*, one of the two main drawbacks described by Malaguti and Toth [16] for assignment-based ILPs, is very diminished in our case. As we stated in Section 3.2 in its classical form the assignment-based ILP needs an upper bound H for the number of colors. This can lead to exponentially many equivalent solutions, as there are $\binom{H}{\mathscr{K}}$ possibilities to select the needed colors from the allowed H [10]. However, since we do not

need to know the chromatic number $\mathscr{X}$ and allow only K colors in our ILP, the effects of this are very diminished in our case as reflected in the very fast runtimes especially for 3- and 4-anticlustering.

We added additional constraints for symmetry breaking proposed by Méndez-Díaz et al. [11] in early developed and observed an increase in computational time, presumably because the cost of the additional constraints outweighed the benefits since we only allow K colors. It might still be interesting to study for which number of anticlusters the symmetry problem starts to affect our algorithm, and whether symmetry breaking has a positive effect in those cases.

6 Conclusions

In this thesis, we introduced an exact algorithm to solve the NP-hard K-anticlustering problem for max dispersion. We used an ILP to solve the decision problem for a special case of equitable graph coloring to determine whether the optimal dispersion value was reached.

We have shown that this algorithm can solve the K-anticlustering problem for large inputs in a reasonable timeframe given its NP-hardness and can be used as a viable alternative for the BILS heuristic we compared it to.

For future work on this topic, there are several approaches conceivable to improve the algorithm further. Since not all anticlustering-applications require anticlusters of the same size, it is desirable to make it possible to select custom upper bounds for the size of anticlusters. Additionally, there might be algorithmic ideas to speed up our algorithm by minimizing the number of iterations. In the current algorithm, only the results of the last two ILPs are actually needed, the second to last to form the anticlusters and the last to know the optimal dispersion value, and it would be interesting to study how this point can be reached faster. This could be done if there is a faster way to determining the result of the equitable K-coloring decision problem, in which case the assignment based ILP would only be solved once to build the anticlusters.

Additionally, it would be interesting to study if ideas from search algorithms can improve this algorithm. We are looking for the optimal dispersion value in an ordered array of distances, and the modified equitable K-coloring ILP can be used to determine whether a given distance is higher than the optimal dispersion value. This is very close to the specifications for searching within a sorted array, so it is not far to seek that ideas from search algorithms could decrease the number of iterations needed. It should be noted however that the time needed to build and solve the ILPs increase the larger the chosen distance is, which has to be taken into account when deciding which search algorithms could work best.

We also discussed the curse of dimensionality and distance concentration in Section 5.1 and how the use of the Euclidean distance might have a negative effect on our algorithm for higher dimensionality. Aggarwal et al. discussed multiple distance metrics in the context of distance concentration and proposed that fractional distance metrics provide more meaningful results and improve effectiveness for clustering algorithms [15]. It could be worth exploring if this improvement also applies to the K-anticlustering problem.

Lastly, the topic of graph coloring is very well studied and there are a lot of ideas to either improve the standard graph coloring ILP, for example in [9] or completely different ways to approach graph coloring. Jabrazilov and Mutzel discussed *set covering* based ILP models and *partial-ordering* based ILP models as alternatives to the assignment based approach we used [10].

In conclusion, the presented algorithm provides a workable solution for many real-life K-anticlustering problems for max dispersion and is a solid base for future improvements.

References

[1] H Späth. “Anticlustering: maximizing the variance criterion”. In: *Control and Cybernetics* 15 (1986), pp. 213–218.

[2] V 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] M Papenberg and G Klau. “Using Anticlustering to Partition Data Sets Into Equivalent Parts”. In: *Psychological Methods* 26 (2021), pp. 161–174.

[4] M Brusco, J D Cradit, and D Steinley. “Combining diversity and dispersion criteria for anticlustering: A bicriterion approach”. In: *British Journal of Mathematical and Statistical Psychology* 73.3 (2020), pp. 375–396.

[5] M Breuer. “Using anticlustering to maximize diversity and dispersion: Comparing exact and heuristic approaches”. BA thesis. Heinrich Heine University Düsseldorf, 2020.

[6] J Kokol. “Polynomial-time solvable special cases of anticlustering”. BA thesis. Heinrich Heine University Düsseldorf, 2022.

[7] W Meyer. “Equitable Coloring”. In: *The American Mathematical Monthly* 80 (1973), pp. 920–922.

[8] H Furmaczyk and M Kubale. “The Complexity of Equitable Vertex Coloring of Graphs”. In: *Journal of Applied Computer Science* 13 (2005), pp. 95–106.

[9] I Méndez-Díaz, G Nasini, and D Severín. “A tabu search heuristic for the Equitable Coloring Problem”. In: *CoRR* 1405.7020 (2014).

[10] A Jabrayilov and P Mutzel. “New Integer Linear Programming Models for the Vertex Coloring Problem”. In: *CoRR* 1706.10191 (2017).

[11] I Méndez Daz, G Nasini, and D Severn. “A Linear Integer Programming approach for the Equitable Coloring Problem”. In: *Information Sciences* (2004), pp. 2–5.

[12] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: https://www.gurobi.com.

[13] Richard Bellman. “Dynamic programming”. In: *Science* 153.3731 (1966), pp. 34–37.

[14] Kevin S. Beyer et al. “When Is ”Nearest Neighbor” Meaningful?” In: *International Conference on Database Theory*. 1999.

[15] C Aggarwal, A Hinneburg, and D Keim. “On the Surprising Behavior of Distance Metric in High-Dimensional Space”. In: *Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6* (Feb. 2002).

[16] E Malaguti and P Toth. “A survey on vertex coloring problems”. In: *International Transactions in Operational Research* 17.1 (2010), pp. 1–34.

A Source Code

The source code, result data and the RScripts used for the tests are accessible at:
https://gitlab.cs.hhu.de/albi/albi-students/ba-max-diekhoff