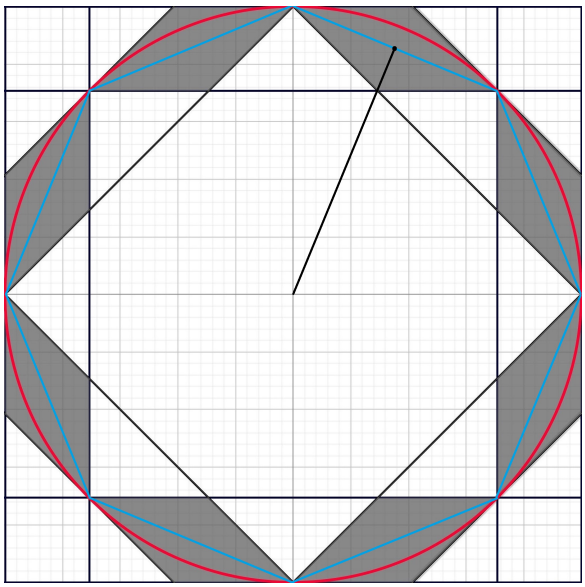
K-hyperplane clustering problem on spatial branch and bound solvers

An improved algorithm for the K-hyperplane clustering problem

Adam Rose



School Of Mathematical Sciences
University of Southampton

Contents

Abstract

The k-hyperplane clustering problem is a data analysis/science problem. Associating a large number of points to k hyperplanes (clusters) with the objective of minimising the total distance between all the points and their associated hyperplane. The objective of this paper is to discuss ideas to improve computation solving time, however the ideas discussed can be generalised to linearising a n-dimensional hypersphere for computers.

1 Introduction

The k-Hyperplane Clustering problem (k-HC) is an optimisation problem in a variety of fields, such as data mining, operations research, and machine learning. The problem in short is associating a (usually) large number points each to a hyperplane (of which there a k-hyperplanes) such that the sum of all the distances between each point and their associated hyperplane is minimal. Whereas all the points and the number hyperplanes k are fixed, the variable which we can control is the orientation of each hyperplane, therefore we can place all the hyperplanes in orientations such that they span the hyper-space as much as possible to minimise the total distance. From linear algebra we know we can describe the hyperplanes orientation as a perpendicular vector which we call $\underline{w_j}$; and because the size/length of $\underline{w_j}$ doesn't matter we fix that $\|\underline{w_j}\|_2 = 1$ for convenience. Now we have a unit hypersphere of which $\underline{w_j}$ could lie anywhere on it, the goal of this paper is to help the computer search over this hypersphere by creating linear lower bound within this hypersphere so that computers can search over the whole hypersphere faster. The ideas and methods discussed in this paper can therefore be generalised to general optimisation of a n-dimensional hypersphere for computers.

There are numerous real world application where k-HC is more applicable than the traditional k-means clustering problem (similar to k-HC but clusters are singular central points (called centroids) instead of hyperplanes). Real world applications of k-HC are centred around data analysis problems usually working with very large data sets/instances. Such examples are: [5] which uses k-local hyperplane distance nearest-neighbor classifier (HKNN); a modified version of k-HC. Clustering is one of the most popular tools for analysing DNA microarray data across various medical fields. Here they are using it to identify co-expression patterns with the NFkB1 in gene expression data from hippocampal tissue samples.
k-HC is also used for time series predictions problem, this is usually a strong non-linear correlation. [6] uses a neural network fusion model based upon k-HC algorithm to partition data into clusters for gradient descent machine learning.
In [4] k-HC is applied in the context of breast cancer prognosis to identify classes of patients with well-separated survival curves. This has also been tested with the k-means clustering algorithms but k-HC provides better results for analysing and grouping the data.

The k-Hyperplane Clustering problem (k-HC) is defined as: given a set of m points $A = (\underline{a_1}, \underline{a_2}, \ldots, \underline{a_m})$ with $\underline{a_i}$ in $\mathbb{R}^n$, partition the set A into k disjoint subset; called clusters, indexed $j \in \{1, \ldots, k\}$. With each cluster being a hyperplane $H_j = \{\underline{x} \in \mathbb{R}^n \mid \underline{w_j} \cdot \underline{x} = \gamma_j\}$; where $\underline{w_j} \in \mathbb{R}^n$ is the perpendicular vector to H_j, and $\gamma_j \in \mathbb{R}$. The objective of the problem is to minimise the sum of the squared 2-norm orthogonal distance of each point and its associated cluster (hyperplane). Hence grouping m points into k clusters such that the total distance between all points $\underline{a_i}$ with its assigned cluster H_j is minimal.
Recall from linear algebra that the minimal distance between a point $\underline{a_i}$ and a hyperplane H_j is the

orthogonal distance $d_{ij} = \|\underline{a_i} - P(\underline{a_i})\|$, where $P(\underline{a_i})$ is the point which corresponds to the orthogonal projection of $\underline{a_i}$ on the hyperplane H_j.

$$d_{ij} = \|\underline{a_i} - P(\underline{a_i})\| = \frac{|\underline{w_j} \cdot \underline{a_i} - \gamma_j|}{\|\underline{w_j}\|_2} \tag{1.1}$$

Where $\|\underline{w_j}\|_2$ is the standard (Euclidean) 2-norm: $\|\underline{w_j}\|_2 = \sqrt{\underline{w_j} \cdot \underline{w_j}} = \sqrt{\underline{w_j}^T \underline{w_j}}$. Derivation of orthogonal distance to the plane see [3]. Note the distances d_{ij} are squared because this problem is a data analysis problem and as such statisticians are concerned with the mean square error from a particular model; the error in this case being the distance d_{ij} from the modelled hyperplane. The result for this application is the points which are further away from their corresponding hyperplane yield a larger contribution to the objective function. Which results in solutions where all points are close to their associated cluster and we don't get e.g. solutions where $m - r$ points are very close to their associated cluster, and r points being very far from their associated clusters.
The formulation for the algorithm (from [1] Page 2, Section 2) is as follows:

$$\begin{array}{lllll}
\min & \sum_{i=1}^{m} y_i^2 & & & (1) \\
\text{s.t.} & \sum_{j=1}^{k} x_{ij} = 1 & i \in \{1,\dots,m\} & & (2) \\
y_i & \geq (\underline{w_j} \cdot \underline{a_i} - \gamma_j) - M(1 - x_{ij}) & i \in \{1,\dots,m\}, & j \in \{1,\dots,k\} & (3) \\
y_i & \geq (-\underline{w_j} \cdot \underline{a_i} + \gamma_j) - M(1 - x_{ij}) & i \in \{1,\dots,m\}, & j \in \{1,\dots,k\} & (4) \\
 & \|\underline{w_j}\|_2 \geq 1 & & j \in \{1,\dots,k\} & (5) \\
 & y_i \geq 0 & i \in \{1,\dots,m\} & & (6) \\
 & \underline{w_j} \in \mathbb{R}^n, \gamma_j \in \mathbb{R} & & j \in \{1,\dots,k\} & (7) \\
 & x_{ij} \in \{0,1\} & i \in \{1,\dots,m\}, & j \in \{1,\dots,k\} & (8)
\end{array} \tag{1.2}$$

For each point $i \in \{1,\dots,m\}$ the distance variable y_i is defined from (1.1). Dictating $\|\underline{w_j}\|_2 = 1$ we can eliminate the 2 norm (which is a non-linear function) from the distance function, $d_{ij} = |\underline{w_j} \cdot \underline{a_i} - \gamma_j|$ (note this doesn't affect the value of d_{ij}). Where $\underline{w_j}$ and γ_j are continuous variables that represent hyperplane H_j for each $j \in \{1,\dots,k\}$
x_{ij} is a binary variable for $i \in \{1,\dots,m\}$ and $j \in \{1,\dots,k\}$ such that it takes value 1 only when $\underline{a_i}$ is assigned to cluster j. Constraint (2) is simply to make sure each $\underline{a_i}$ is only assgined to only one cluster. This allows us to create constraints (3) and (4) which is the distance function $d_{ij} = |\underline{w_j} \cdot \underline{a_i} - \gamma_j|$ as 2 linear functions, with $-M$ if $\underline{a_i}$ isn't assigned to this cluster. M is the diagonal length of smallest hyperrectangle containing all vector $\underline{a_i}$;

$$M = \sqrt{\sum_{r=1}^{n} \left(\max_{i \in \{1,\dots,m\}} \underline{a_i}_r - \min_{i \in \{1,\dots,m\}} \underline{a_i}_r \right)^2}$$

Thus when $x_{ij} = 0$ then constraints (3) and (4) become insignificant such that the constraints are just automatically satisfied. $\|\underline{w_j}\|_2 = 1$ can be relaxed to $\|\underline{w_j}\|_2 \geq 1$ without changing the objective

value because for $\min \sum_{i=1}^{m} y_i^2$ will be minimal for $\min y_i$ which in return will reduce w_j through (3) and (4).

The formulation (1.2) contains continuous and integer variables and a quadratic convex objective function. All constraints are linear except for (5) which is quadratic as well as non-convex. This paper will focus on refining and creating linear constraints which underlie constraint (5) but make it such that modern solvers will be quicker/more-efficient at solving the problem. Modern optimisations solvers iteratively split the nonconvex feasible region into subregions and to find lower bounds (branch and bound) for each subregion by constructing and optimising over a polyhedral envelope. So therefore by creating these linear constraints we are assisting in the development of these polyhedral envelopes. We also relax the the constraint of (5) to $\geq$ rather than $=$ because the optimal solution will lie on the hypersphere i.e. $\|\underline{w_j}\|_2 = 1$ (due to the problem being a minimisation problem) and $=$ would make it impossible to create linear polyhedral envelopes which include more than few finite points intersecting the hypersphere. As a result we will initially get $\underline{w_j}$ points outside the hypersphere but in any case this would only increase the distance y_i.

It is also worth noting that the solvers will use branch and bound over subregions as well as binary variables like (x_{ij}). We will use these ideas to motivate some of linear formulations for the hypersphere.

2 Data instances and control system

In this paper we will improve the formulation of the standard k-HC algorithm (1.2). Comparing the different possible improvements in formulation to the algorithm against other improvements as well as the original standard algorithm. Analysing the data of testing a variety of different instances. Testing will be conducted inside Python where I will be creating the model using Gurobi python API. The testing will be done my personal computer, conducted on 4 threads of a i5-4690k @4.3ghz with 8 gb DDR3 ram.
Now we did limit ourselves to one branch and bound solver - 'Gurobi', to focus on the fundamental algorithms and formulations. However we did not create any formulations (or internal code) that would be particularly advantageous to improving the results for Gurobi and not other solvers. This approach was to make the ideas in this paper more general to all branch and bound solvers rather than optimising it specifically for Gurobi

Before presenting the initial results (table 1) I will briefly explain some of the columns which will be of interest, throughout this paper:

- **Nodes** is the number of nodes Gurobi has branched over and explored in its attempt to solve k-HC. Higher number of nodes is better but we should expect there to be fewer nodes explored as the algorithm improves; due to Gurobi being more effective at each branch (of branch and bound).
- **Time** will be the designated max time allocated to that instance unless the problem is solved (finished) before the time limit.
- **Obj Value** this result is the value of the solution (i.e. the sum of the squared distance). On its own this value is not important to us however through iterations of the code/algorithm the solution may improve; this is especially important for large size problems where finding a definite solution (and finishing) is unlikely.
- **LB** is the global lower bound Gurobi has found so far. This is one of our most important factors as it gives us an indication of if our objective value is a final solution
- **Gap** is the percentage the LB is to the Obj value; with the algorithm is complete once Gap $= 100\%$. We will aim to increase both LB and Gap with future iterations as this is needed for the algorithm to reach a definite solution.

The samples (or instances as we will later refer to as) used for this testing will vary in sizes over number of: points (m), clusters (k), and dimensions (n). The test data is a subset from the samples used in [1] sections 3; which is a set of structured randomly generated instances. I consider 15 instances partitioned into 3 groups: small, medium, and large. With the 3 group classes allocation of time for an optimization solver will be 300, 1200, and 2700 seconds (respectively). With each group having 5 instances there will be slight differences to represent a variety of realistic parameters (m, k, n) for that size instance. This will allow us to evaluate implementations of new algorithms against the parameters as well as create more efficient algorithms for certain parameter ranges e.g $n < 6$, more on this later. Additionally larger instance sizes weren't covered in this paper because my personal system wouldn't run them effectively, instead throughout the paper I have tried to extrapolate from the data to give an indication of what this would be like for larger problems.

Below is table 1 of the selected instances tested throughout this paper, the testing in this table is on the original algorithm and will provide an initial benchmark for future improvements to come: These results obviously don't mean much right now with no comparisons, however this page is

Table 1: Computational results of basic k-HC algorithm (1.2)

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	4130	0.24	0.268	0.268	0%
	16	3	3	Test2	6,369,493	300	0.0754	0	100%
	20	3	3	Test3	5,131,104	300	2.23	0	100%
	24	3	2	Test4	4,750,886	300	3.98	0	100%
	30	2	3	Test5	6,036,877	300	2.63	0.06	97.8%
Medium	500	2	2	Test6	970,463	1200	111.1	0	100%
	500	5	4	Test7	343,223	1200	254.0	0	100%
	500	8	4	Test8	288,984	1200	138.9	0	100%
	500	5	6	Test9	323,681	1200	188.7	0	100%
	500	8	6	Test10	219,902	1200	166.6	0	100%
Large	2000	5	4	Test11	45,759	2700	1647.2	0	100%
	2000	16	4	Test12	6803	2700	3397.1	0	100%
	2000	5	8	Test13	6289	2700	4402.1	0	100%
	2000	16	8	Test14	4653	2700	5959.6	0	100%
	2000	16	12	Test15	4261	2700	3501.0	0	100%

important to note down to compare with later tables (with improved algorithm's computational results) which will be formatted the same way.
There are however some initial things which can be said about the results and what should be expected in later iterations of the formulation. Firstly notice how Test1 is the only one which complete the solve (and the only one which got significant gap), we expect this to finish in roughly the same short amount of time on later iterations; maybe a little slower due to more constraints/calculations. The number of nodes explored is affected by all 3 parameters (m, k, n), and although we have no data to support this we can presume that these parameters also effect the Objective value and Lower bound (LB); and in turn Gap.
There is not enough data to determine how each of the parameters effect each value of the table: Nodes, Obj value, and LB. From this initial testing and some intuition it suggest that number of points m has a huge impact on objective value which makes sense as its the sum of squared distances of each point (so more points would increase the objective value). However I expect the objective value to be significantly lower than it is currently for large m, with a relation such that the proportion increase in the number of point is lower than the proportion increase in Objective value (in most cases).
The number of clusters k is clearly a important parameter on the computers processing, adding significant computing processing to the problem as it is essentially multiplying the problems difficulty, e.g. for $k = 8$ there are 8 $\underline{w_j}$ hypersphere problems which must be solved/found moreover these 8 $\underline{w_j}$ all depend on one another. Arguable equally as important as n which we would expect to be a serious parameter for the algorithm, as adding additionally dimensions increases the problems scale

by 2 (for each dimension).
Throughout this paper we will use the data provided from these and future tables to identify how the parameters effect each of the response variables, as well as importantly how Gurobi is branching over the nodes of the current linear model depending on the algorithm. Which will aid us to develop and create further improvements to the formulations.

3 Norms

The original algorithm 1.2 is not ideal for optimisation solvers due to its nonlinearity in the objective function y_i^2 and variable $\underline{w_j}$. Now although modern optimisation solvers can and will deal with quadratic and polynomial constraints/objectives by creating polyhedral envelops, it is far more efficient at solving linear terms. Changing both of these sets of terms to equivalent linear equations is not possible, however we can use linear functions using the data/parameters to create further constraints to help the solver reach the solution faster.
We will investigate this in later chapters but the core idea is to reduce the computation needed for our non-linear (quadratic) terms $\|\underline{w_j}\|_2 \geq 1$ by creating linear constraints to underlie the quadratics ones, this will assist Gurobi to create polyhedral envelopes closers to the boarder of the hypersphere.

3.1 p-Norm

The most common norm used is the 2-norm, which is the standard length norm used to for vectors $\underline{a} \in \mathbb{R}^n$:

$$\|\underline{a}\|_2 = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2} \tag{3.1}$$

The 2-norm, or Euclidean norm, which may also be defined as the square root of the inner product of a vector with itself $\|\underline{a}\|_2 = \sqrt{\underline{a} \cdot \underline{a}}$. This is the standard and most common function to calculate the length function in $\mathbb{R}^n$ as its a consequence of Pythagorean theorem and is consistent under rotation. We will define a more general p-norm for $p \in \{2, 3, \ldots\}$ and for a vector $\underline{a} \in \mathbb{R}^n$ as:

$$\|\underline{a}\|_p = \sqrt[p]{a_1^p + a_2^p + \ldots + a_n^p} \tag{3.2}$$

Special cases for $p = \{1, \infty\}$ include the 1-norm and infinity norm (respectively). These are very common norms of $\mathbb{R}^n$ defined as the following:

$$\|\underline{a}\|_1 = \sum_{i=1}^{n} |a_i| \qquad = |a_1| + |a_2| + \ldots + |a_n| \tag{3.3}$$

$$\|\underline{a}\|_\infty = \max_{i \in \{1,\ldots,n\}} |a_i| \tag{3.4}$$

These 2 norms are useful for our algorithm because they are the only norms which can be formulated linearly, where as the p-norm (3.2) for $p = \{2, 3, \ldots\}$ is always non-linear (p^{th} root of polynomial; degree $= p$) .
Its not immediately obvious from the formulation but both the one norm $\|\underline{a}\|_1$ and infinity norm $\|\underline{a}\|_\infty$ can be represented by linear constraints and therefore can be calculated linearly, but the linearity of these norms should be motivated by the following figure 1 below:

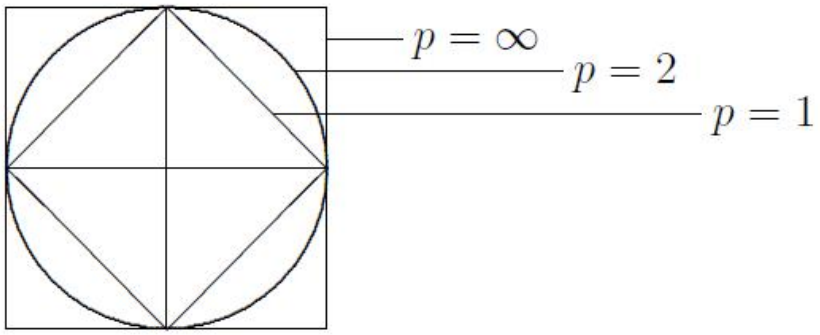


Figure 1: $\underline{a} \in \mathbb{R}^2$ such that $\|\underline{a}\|_p = C$, for some $C \in \mathbb{R}^+$

This shows the 1 and infinity norm compared to the 2 norm in 2 dimensions. Clearly the 1 norm and infinity norm are straight/linear compared to the 2 norm; while not included in the picture is norms $p = \{3, 4, \ldots\}$ which would be between the $p = 2$ and $p = \infty$ norms. This is crucial for improving solving time for the optimisation solvers.

Remark. In 2-D space the 1-norm is a diamond, 2-norm a circle, inf-norm a cube. In general n-D space: the 1-norm is a **hyperoctahedron**, 2-norm a **hypersphere**, inf-norm a **hypercube**.

3.2 Different norm relations

So far we have defined different norms but have yet to relate the norms to one another. In this section we will create the necessary inequalities to relate any norm to another, specifically focusing on the one norm $\|\underline{a}\|_1$ and infinity norm $\|\underline{a}\|_\infty$ compared to the 2-norm $\|\underline{a}\|_2$. This comparison is most important because the one norm $\|\underline{a}\|_1$ and infinity norm $\|\underline{a}\|_\infty$ can be represent linearly, and therefore we can use it to restrict and modify the underlying linear feasible region to assist the polyhedral envelope creation for $\|\underline{w_j}\|_2 \geq 1$ (line 5 of 1.2). This will be discussed further in later chapters.

To handle and manipulate norms we will first state a slightly weaker version of Cauchy–Schwarz inequality:

Theorem 3.1. ***(Cauchy–Schwarz inequality)*** $\forall a, b \in \mathbb{R}^n$*:*

$$|\underline{a} \cdot \underline{b}|^2 \leq (\underline{a} \cdot \underline{a})(\underline{b} \cdot \underline{b}) \tag{3.5}$$

Where $(- \cdot -)$ *is any inner product (e.g dot product).*

Proof omitted

Remark. It immediately follows from the Cauchy-Schwarz that for any norm $\forall a, b \in \mathbb{R}^n$

$$|\underline{a} \cdot \underline{b}| \leq \|\underline{a}\| \|\underline{b}\|$$

$$|\underline{a} \cdot \underline{b}|^{1/2} \leq \|\underline{a}\|^{1/2} \|\underline{b}\|^{1/2}$$

This is because all norms being non negative.

This will help us later prove some equivalence relations between different norms.

Theorem 3.2. *For any $p, q \in \mathbb{N} \cup \{\infty\}$, there are two positive scalars $\alpha, \beta \in \mathbb{R}$ such that:*

$$\alpha\|\underline{w}\|_p \leq \|\underline{w}\|_q \leq \beta\|\underline{w}\|_p \qquad \forall \underline{w} \in \mathbb{R}^n$$

We will omit the proof for the general case as it is not needed, however we will prove specific cases of this theorem relating to $p, q = \{1, 2, \infty\}$ for which we will need later due to the linearity of $\|\underline{w}\|_1$ and $\|\underline{w}\|_\infty$.

Corollary 3.3.

$$\|\underline{w}\|_2 \leq \|\underline{w}\|_1 \leq \sqrt{n}\|\underline{w}\|_2 \qquad \forall \underline{w} \in \mathbb{R}^n \tag{3.6}$$

Proof. Squaring both norms $\|\underline{w}\|_1$ and $\|\underline{w}\|_2$ leads to:

$$\|\underline{w}\|_1^2 = (|w_1| + |w_2| + \ldots + |w_n|)^2 \text{ and } \|\underline{w}\|_2^2 = w_1^2 + w_2^2 + \ldots + w_n^2$$

Now as $(|w_1|)^2 = w_1^2$, and expanding $\|\underline{w}\|_1^2$ we obtain:

$$\|\underline{w}\|_1^2 = |w_1|(|w_1| + |w_2| \ldots + |w_n|) + |w_2|(|w_1| \ldots + |w_n|) + \ldots$$

Expanding this it is clear to see that $\|\underline{w}\|_2^2$ is contained in $\|\underline{w}\|_1^2$ as well as a positive real constant from the product of all cross terms C. In other words:

$$\|\underline{w}\|_1^2 = \|\underline{w}\|_2^2 + C \quad \text{where } C = \sum_{i=1}^{n}\sum_{j=1}^{n} |w_i||w_j| \quad \text{with } i \neq j$$

Clearly $C \geq 0$ therefore $\|\underline{w}\|_1^2 \geq \|\underline{w}\|_2^2$, but therefore as norms are always positive (by definition) we can conclude $\|\underline{w}\|_1 \geq \|\underline{w}\|_2$

Now creating an upper bound on $\|\underline{w}\|_1^2$. We will use the Cauchy-Schwarz inequality 3.1:

$$\|\underline{w}\|_1 = \sum_{i=1}^{n} |w_i| \cdot 1 \leq \left(\sum_{i=1}^{n} |w_i|^2\right)^{1/2} \left(\sum_{i=1}^{n} 1^2\right)^{1/2} = \sqrt{n}\|\underline{w}\|_2$$

Therefore $\|\underline{w}\|_1 \leq \sqrt{n}\|\underline{w}\|_2$ □

Corollary 3.4.

$$\frac{1}{\sqrt{n}}\|\underline{w}\|_2 \leq \|\underline{w}\|_\infty \leq \|\underline{w}\|_2 \qquad \forall \underline{w} \in \mathbb{R}^n \tag{3.7}$$

Proof. The upper bound for the infinity norm is trivial:

$$\|\underline{w}\|_\infty^2 = \max_{i \in \{1,\ldots,n\}} w_i^2 \leq \sum_{i=1}^{n} w_i^2 = \|\underline{w}\|_2^2$$

For the lower bound, using the inequality above we can see that the $\|\underline{w}\|_2^2$ is largest when every entry of $\underline{w}$ is maximum M, i.e. $\underline{w} = (M, \ldots, M)$. Hence $\|\underline{w}\|_2^2 = \sum_{i=1}^{n} w_i^2 = nM^2$. Clearly $\|\underline{w}\|_\infty = M$ and $\|\underline{w}\|_2 = \sqrt{n}M$, hence as this is the maximum of the 2 norm we can conclude $\frac{1}{\sqrt{n}}\|\underline{w}\|_2 \leq \|\underline{w}\|_\infty$. □

4 Infinity Norm

Currently with our algorithm (1.2) has $\|\underline{w}\|_2^2 \geq 1$ which is a quadratic (non-linear) and non-convex region. Gurobi will split this non-convex region into smaller convex subregions by McCormick envelopes [2], it will then branch over all the subregions to find an optimal answer. Gurobi will consider points in the infeasible regions $\|\underline{w}\|_2^2 < 1$ in the process of branching on McCormick envelopes, this is an issue is because branch and bound solvers can not correctly evaluate the boundary of quadratic constraints. The volume between the unit hypercube and unit hypersphere is the feasible region we are concerned with.
Note: when we refer to the 'unit hypercube' we are referring the hypercube of length 2 (with each vertex coordinate value being either -1 or 1) because the bound on $\underline{w}$ between $(-1, 1)$ as we know all optimal solutions lie on $\|\underline{w}\|_2^2 = 1$, therefore we will eliminate Gurobi searching for unoptimised $\underline{w}$ (i.e. outside $(-1, 1)$). We say the unit hypercube is the upper bound as clearly the unit hypersphere lies within the hypercube, and it is easy to dictate that each coordinate is between -1 and 1 with in Gurobi or other branch and bound solvers).

Gurobi will see the linear feasible region (the region made by linear constraints) as the whole unit hypercube which means the region which Gurobi searches over is significantly inefficient with a significant percentage of the that region being infeasible (due to the non-linear constraint $\|\underline{w}\|_2^2 \leq 1$), e.g. in 2 dimensions 79% of the region is infeasible.
Therefore reducing the linear region for Gurobi to search by creating linear constraints underlying $\|\underline{w}\|_2^2 \geq 1$ so that the optimal solution is still valid but increasing the linear feasible region to be closer in-line with the actual feasible region will greatly improve computational results.

Using our corollary 3.4 (comparing the infinity norm to the standard 2 norm) we can use these linear norm constraints to create a lower bound to the quadratic constraint $\|\underline{w}\|_2^2 \geq 1$; note in the proof of this corollary its clear that these inequalities are as tight as possible. We have already spoken about a bound between $(-1, 1)$; for mathematical formality this is the equivalent to $\|\underline{w}\|_\infty \leq 1$; which agrees with $\|\underline{w}\|_\infty \leq \|\underline{w}\|_2$ as $\|\underline{w}\|_2 \geq 1$.
So we have stated above a feasible upper bound using the infinity norm, now using the other side of the inequality in corollary 3.4 we can obtain the lower bound with the infinity norm $\|\underline{w}\|_\infty \geq \frac{1}{\sqrt{n}}$.
Thinking about this geometrically these new constraints are creating a minimal hypercube enclosing the unit hypersphere of $\|\underline{w}\|_2 = 1$ and a maximal hypercube which is enclosed within the unit hypersphere.

Figure 2 shows geometrically the linear feasible region of $\underline{w}$ under these new inf-norm formulations in 2 dimensions:

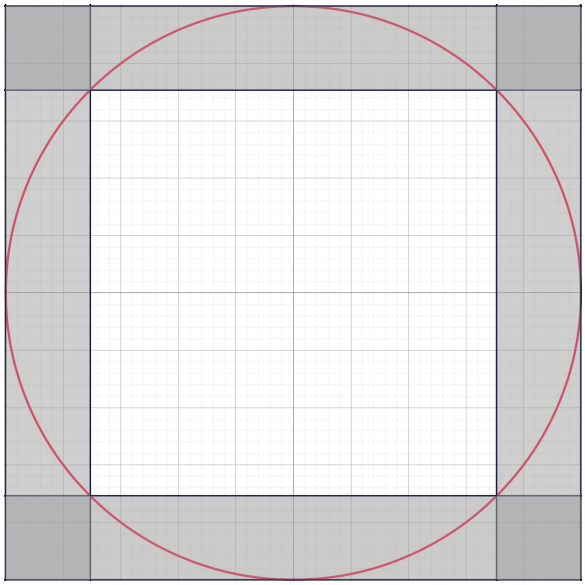


Figure 2: The grey area shows the linear feasible region of $\underline{w}$

The reduction of area (of a quadrant) through this lower bound constraint results in the feasible region for $\underline{w}$ being $1 - \frac{1}{\sqrt{n^n}}$. This is a 50% reduction in area for 2 dimensions, 19% reduction for 3 dimensions, 6% reduction for 4 dimensions. So this constraint reduces the linear feasible region but it has exponentially diminishing effects on the linear feasible region as the dimensions increase. This should not have such a dramatic decrease in computation results because this lower bound removes lots of infeasible region furthermore the volume of the hypersphere within the unit hypercube is also decreasing as the dimension increases.

Remark. Outside upper bounds are less critical to reduce because k-HC is a minimisation problem so optimisation solvers (e.g. gurobi) will focus searching for solutions as close to 0 as possible.

We will still create upper bounds as they improve the optimisation of the algorithm but over a long time frame their effect is negligible. Therefore we will not make computationally complex tight upper bounds which may reduce the linear feasible region but the actual feasible region above the hypersphere is searched less by optimisation solvers due to the number of added equations.

Remark. The inf-norm has an upper bound $\|w_j\|_\infty \leq 1$, but this can be implemented by restricting the domain of the variable between -1 and 1; which is straight forward to do with any optimisation solver.

Formulating the lower bound on our quadratic constraint $\|\underline{w_j}\|_2 \geq 1$ with the inf-norm $\|\underline{w_j}\|_\infty \geq \frac{1}{\sqrt{n}}$ requires creating 2 new sets of binary variables s_{jq} and z_{jq}. We can formulate the infinity norm on $\underline{w_j}$ as a linear relaxation problem to extend onto our original algorithm 1.2:

$$
\begin{array}{lll}
\underline{w_j}_q \geq \frac{1}{\sqrt{n}} - N(1 - z_{jq}) - N(s_{jq}) & q \in \{1, \ldots, n\}, \quad j \in \{1, \ldots, k\} & (1) \\
\underline{w_j}_q \leq -\frac{1}{\sqrt{n}} + N(1 - z_{jq}) + N(1 - s_{jq}) & q \in \{1, \ldots, n\}, \quad j \in \{1, \ldots, k\} & (2) \\
z_{jq} \in \{0, 1\}, \quad s_{jq} \in \{0, 1\} & q \in \{1, \ldots, n\}, \quad j \in \{1, \ldots, k\} & (3) \\
\sum_{q=1}^{n} z_{jq} = 1 & j \in \{1, \ldots, k\} & (4)
\end{array}
\tag{4.1}
$$

Constrains (1) and (2) are covering the positive and negative sides of $\|\underline{w_j}\|_\infty \geq \frac{1}{\sqrt{n}}$ as we can't use absolute functions in our algorithms. N is the tightest constant ($N = 2 + \frac{1}{\sqrt{n}}$) that allows us to let any $\underline{w_j}$ satisfy the constraints (1) and (2) only once for each dimension $q \in \{1, \ldots, n\}$ and cluster $j \in \{1, \ldots, k\}$. The binary variable z_{jr} dictates that the absolute value of only one coordinate of $\underline{w_j}$ is above $\frac{1}{\sqrt{n}}$. Whereas s_{jr} dictates that the one coordinate of $\underline{w_j}$ only satisfies one of (1) and (2), which is equivalent $|\underline{w_j}_q| \geq \frac{1}{\sqrt{n}} - N(1 - z_{jq})$.

Below is the table of computational results for the standard algorithm (1.2) additionally with inf-norm algorithm formulation 4.1:

Table 2: Computational results of Inf-norm algorithm (4.1)

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	2898	0.17	0.268	0.268	0%
	16	3	3	Test2	5,686,137	300	0.0712	0	100%
	20	3	3	Test3	5,222,576	300	1.86	0	100%
	24	3	2	Test4	3,034,202	133.8	3.98	3.98	0%
	30	2	3	Test5	6,137,272	300	2.53	0.60	76.5%
Medium	500	2	2	Test6	1,987,026	1200	111.1	0.3489	99.7%
	500	5	4	Test7	491,360	1200	228.7	0	100%
	500	8	4	Test8	322,721	1200	85.46	0	100%
	500	5	6	Test9	323,791	1200	146.2	0	100%
	500	8	6	Test10	189,576	1200	167.1	0	100%
Large	2000	5	4	Test11	72,885	2700	982.8	0	100%
	2000	16	4	Test12	12,090	2700	561.2	0	100%
	2000	5	8	Test13	4986	2700	1388.2	0	100%
	2000	16	8	Test14	4444	2700	2358.0	0	100%
	2000	16	12	Test15	4055	2700	9783.7	0	100%

This initial improvement of the algorithm compared to the original table 1 is very clear, with 13 of the 15 instances having improved results with the addition of the infinity norm formulation, Instance 'Test10' had very slight adverse results, while 'Test15' had a very adverse result with the objective value 3 times larger than on previous iterations. I can only put this down to being an anomaly as from further algorithm iterations (in later chapters) and testing on this instance I assume the solutions lies on the hypersphere such that many of the $\underline{w_j}$ are far away from the hypercube that the infinity norm makes.
Generally the number of nodes remained the same with a slight increase of nodes on this new formulation. Due to the complexity of the new calculations of this formulation Gurobi shouldn't be able to maintain the number of nodes of the original algorithm. It is clearly branching differently with these new constraints, a lot of these branching (nodes) will be on the many binary variables created in the infinity norm formulations; there are $2 * j * n$ new binary variables. Therefore lots of the branching Gurobi does by dividing the feasible regions into manageable feasible polygons (so Groubi can solve it linearly) will be less in this formulation (than originally) and most of the branching is from the binary variables. Therefore concluding that we can state with confidence that Gurobi was able to branch over the feasible region more effectively making use of the binary variables to help divide up the feasible region.

Importantly the implementation of the infinity norm has created lower bounds (in the results) which were previously non-existent. This is an important criteria for future formulations, lower bound is the going to be the most critical factor. This is because if given enough time most algorithms reach the final objective value solution, however verifying this is the solution value is crucial for the algorithm to conclude. As we create and add formulations we should expect to get the objective values converging to the same value (the final solutions), however Gurobi must increase the lower bound (reducing the gap) to conclude the algorithm. Therefore as we make more improvements we will critique a formulations performance by it's effect on the Lower bound and Gap.

The results make it clear that the infinity norm has improved the calculation, however there is another formulation for infinity norm which is using the combinations of rotations (section 6) and the 1 norm (section 5). We will investigate this later in section 8 and compare it to the results of this formulation.

5 1-Norms

In this chapter we will use the relation between the 1-norm and the 2-norm (corollary 3.3) to reduce the linear feasible region of $\underline{w}$. Building upon the improvements made in the previous chapter we will create bounds from (3.3) in a linear programming formulation; again it is important this is as tight as possible. The lower bound $\|\underline{w_j}\|_1 \geq 1$ and upper bound $\|\underline{w_j}\|_1 \leq \sqrt{n}$ enclose the hypersphere $\|\underline{w_j}\|_2 = 1$. Geometrically this can be thought of as the lower bound creates a maximal hyper-octahedron contained within the hypersphere while the upper bound creates a minimal hyper-octahedron containing the hypersphere. Figure 3 shows geometrically in 2 dimensions the linear feasible region of $\underline{w_j}$ after the 1-norm and inf-norm implementations

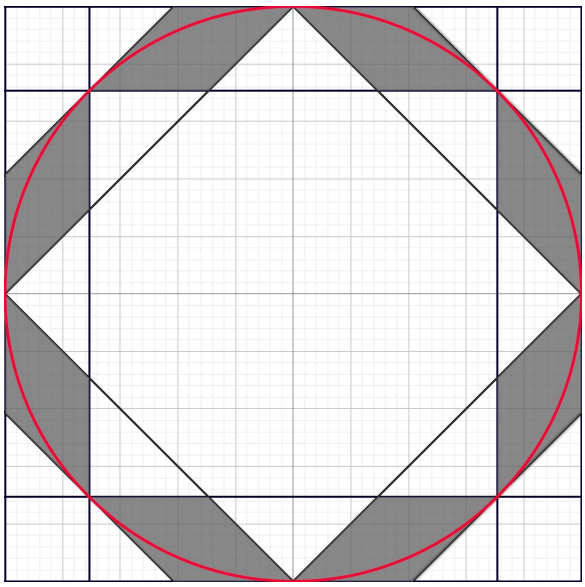


Figure 3: The grey area shows the linear feasible region of $\underline{w}$

Using only the one norm the reduction of area (of a quadrant) in 2 dimensions is 67% and then in 3 dimensions it is 17%. With the inf-norm and 1-norm the reduction of area is 75% in 2 dimensions and 26% in 3 dimensions. Again the results are diminishing as the dimension increase, but importantly we are reducing significant parts of infeasible area for $\underline{w}$. The result of adding the 1-norm formulation is that the linear feasible regions remaining are disjoint convex subsets, which will make it easy for Gurobi to choose what to branch over.
The 1-norm is more complicated computationally than the inf-norm because it is a sum of absolute values. In this chapter we will discuss 2 methods to formulate the 1-norm as a linear programming model. Both methods use different ideas and are arguably situationally better than the other.

5.1 Algebraic formulation

The first of these two methods is the 'algebraic formulation', this is the less computationally demanding of the two. Importantly this doesn't scale exponentially as the dimensions of the problem increase.

The general idea with this formulation is that each coordinate of $\underline{w_j}$ can be written as $\underline{w_j}_q = \underline{\rho_j}_q - \underline{\eta_j}_q$. Both $\underline{\rho_j}_q$ and $\underline{\eta_j}_q$ are non-negative entries, and can be thought of as splitting $\underline{w_j}$ into 2 vectors $\underline{(wPos)_j}$ and $\underline{(wNeg)_j}$. Therefore separating all elements of $\underline{w_j}$ into a positive vector and a negative vector of all non-negative values.
The linear programming formulation of this is as follows:

$$
\begin{array}{lcll}
\underline{w_j}_q & = & \underline{\rho_j}_q - \underline{\eta_j}_q & q \in \{1,\dots,n\} \quad j \in \{1,\dots,k\} \quad (1) \\
\underline{\rho_j}_q & \leq & v_{jq} & q \in \{1,\dots,n\} \quad j \in \{1,\dots,k\} \quad (2) \\
\underline{\eta_j}_q & \leq & 1 - v_{jq} & q \in \{1,\dots,n\} \quad j \in \{1,\dots,k\} \quad (3) \\
 & \underline{\rho_j} \in (\mathbb{R}^+)^n,\ \underline{\eta_j} \in (\mathbb{R}^+)^n & & j \in \{1,\dots,k\} \quad (4) \\
 & v_{jq} \in \{0,1\} & q \in \{1,\dots,n\} \quad j \in \{1,\dots,k\} & (5) \\
1 & \leq & \sum_{q=1}^{n} (\underline{\rho_j}_q + \underline{\eta_j}_q) & j \in \{1,\dots,k\} \quad (6) \\
\sqrt{n} & \geq & \sum_{q=1}^{n} (\underline{\rho_j}_q + \underline{\eta_j}_q) & j \in \{1,\dots,k\} \quad (7)
\end{array}
\tag{5.1}
$$

Clearly line (1) is the separation of positive and negative elements. Lines (2) and (3) dictate that $\underline{\rho_j}_q = 0$ or $\underline{\eta_j}_q = 0$, using a binary variable v_{jq} (5). At least one of these values must be 0 make sure all entries in $\underline{\rho_j}$ and $\underline{\eta_j}$ are non-negative. With this we can now rewrite the 1-norm as $\|\underline{w}\|_1 = \sum_{q=1}^{n} (\underline{\rho_j}_q + \underline{\eta_j}_q)$. Therefore we can apply this to create our lower and upper bound of the 1-norm, $\|\underline{w}\|_1 \geq 1$ and $\|\underline{w}\|_1 \leq \sqrt{n}$.

Remark. This formulation (5.1) is computationally simple enough to add an upper bound (7) without it having a detrimental effect on the solving. From preliminary testing the upper bound improved the results for small/medium instances however on large time frames it was slightly detrimental to results and this trend is likely to continue. Therefore line **(7) will not be used** in the results.

First we will see how this formulation performs by itself; as this will give us a better idea of how Gurobi will use this formulation, importantly we can hopefully gauge a idea of how this formulation is effected by the parameters (m, k, n).
These results are critical to our investigation, as geometrically the 1-norm is an hyper-octahedron which volume wise is smaller than the hypercube of the Inf-norm, however they're both constructed in very different methods. Therefore we will mostly be comparing these results to compare this formulation and the Inf-norm formulation. Initially looking at the geometric aspects of these two formulations we would expect the Inf-norm formulation as the volume is larger and therefore it reduces the linear feasible region more, however its not that simple because both formulations are very

different, with the Inf-norm being constructed with lots more binary variables. Although the 1-norm creates convex linear regions in each quadrant (octant in 3 dimensions and orthant in n-dimensions), whereas the Inf-norm linear region requires more cuts/segmenting to create convex regions for each orthant. Therefore comparing these types of formulations will give us an indication of how Gurobi is evaluating these sets of formulations.

Table 3: Computational results of the algebraic 1-norm (5.1)

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	840	0.15	0.268	0.268	0%
	16	3	3	Test2	965,691	300	0.0712	0.060	15.8%
	20	3	3	Test3	6,286,256	300	1.86	0.81	56.5%
	24	3	2	Test4	576,025	23.6	3.98	3.98	0%
	30	2	3	Test5	5,067,750	300	2.53	1.34	46.9%
Medium	500	2	2	Test6	1,421,260	1200	111.1	7.12	93.6%
	500	5	4	Test7	485,398	1200	227.5	0	100%
	500	8	4	Test8	310,443	1200	74.8	0	100%
	500	5	6	Test9	376,059	1200	173.8	0	100%
	500	8	6	Test10	263,516	1200	118.1	0	100%
Large	2000	5	4	Test11	42,180	2700	1060.1	0	100%
	2000	16	4	Test12	13,237	2700	857.7	0	100%
	2000	5	8	Test13	35,476	2700	1726.2	0	100%
	2000	16	8	Test14	7183	2700	2481.9	0	100%
	2000	16	12	Test15	5939	2700	2824.7	0	100%

The results of this formulation are not clear cut better than the Inf-norm formulation (4.1) however many of the desirable features we are looking for in our improved formulations are better in this formulations, furthermore in areas where it is worse than the Inf-norm the results are close, whereas the Inf-norm which is significantly behind on some instances. The objective values are lower on the Algebraic-1-norm for all small and medium cases except 'Test9', however in large instances Inf-norm out-performed over all instance but 'test15' (which as I discussed previously seems to be a test anomaly for Inf-norm). Along with that the Algebraic-1-norm obtaining lower bounds for Test1 - 6 and which was an improvement on the Inf-norm and moreover they were better LBs, as well as on instances where it reached a Gap = 0 (i.e. a definite solution) the time achieved was less.
As this 1-norm is formulated with less binary variables than Inf-norm (half as many) it would be expected that number of nodes analysed would be significantly less. The data however has a mix of results where its sometimes higher and sometimes lower, but there is little correlation between the number of nodes and the parameter values. Therefore I would assume that due to formulation of Alg-1-norm Gurobi is trying to branch over the binary variable which determines if a particular coordinate is positive or negative, and then immediately pruning infeasible branches. Therefore I don't think the number of nodes in formulations with the Alg-1-norm will be too reflective of the other results later on.
It is therefore hard to gauge on how this formulation is effected by the parameters (m, k, n), how-

ever it's clear that this formulation's performance was significantly better for smaller instances. Its worth stating at even large instances it is significantly better than the original algorithm, but as the data (from Inf-norm and Alg-1-norm) is suggesting it may be that branching on binary variables will be more effective for large instances, especially those with a large amount of clusters and dimensions.

We will now look forward now at this new formulation together with the infinity norm as a natural continuation; since infinity norm improved upon the results of the original algorithm substantially. Below is the table of results for the current best algorithm additionally (original algorithm and infinity norm formulation) with Alg-1-norm algorithm above ((1.2) ∩ (4.1) ∩ (5.1)):

Table 4: Computational results of algebraic 1-norm (5.1) and infinity norm (4.1)

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	751	0.13	0.268	0.268	0%
	16	3	3	Test2	1,832,866	96.5	0.0712	0.0712	0%
	20	3	3	Test3	4,504,017	300	1.86	0.0002	100%
	24	3	2	Test4	326,778	15.1	3.98	3.98	0%
	30	2	3	Test5	5,243,517	300	2.53	1.37	46.0%
Medium	500	2	2	Test6	1,876,523	1200	111.1	8.500	92.4%
	500	5	4	Test7	489,477	1200	193.9	0	100%
	500	8	4	Test8	302,170	1200	85.34	0	100%
	500	5	6	Test9	401,529	1200	151.3	0	100%
	500	8	6	Test10	187,721	1200	121.3	0	100%
Large	2000	5	4	Test11	81,208	2700	1198.2	0	100%
	2000	16	4	Test12	24,390	2700	2221.5	0	100%
	2000	5	8	Test13	6231	2700	1831.8	0	100%
	2000	16	8	Test14	5474	2700	2508.4	0	100%
	2000	16	12	Test15	6292	2700	3109.8	0	100%

Combining these formulations improved the results for small instances (other than 'Test3' which had slightly worse gap than it did with the Alg-1-norm) with the medium instances results splits with the smaller medium instances making improvements which Test8-10 bridging the middle ground between Alg-1-norm and Inf-norm results. Whereas the large instances have taken a step back with all the results being worse than Alg-1-norm's results and therefore also worse than Inf-norm.
The early results are promising though and importantly the lower bound on the instances did improve. This combined formulation did not immediately improve upon the results of its individual formulations however it is very likely that if given enough time this formulation would out-perform the individual formulations. What it isn't so clear is the effect this formulation is having on the lower bound, something we should consider is although for large instances it did not create lower bounds (LB) it's likely if given enough time this would produce better lower bounds than the individual formulations. This is an important consideration going forward as the true performance of a formulation could be in it improvements to lower bounds (our main objective) which may not be seen in the computation till later. Therefore we will continue to test different combination of these formulations.

5.2 Brute force formulation

This is the alternative linear formulation of the 1-norm, it is a more intuitive approach for representing the 1-norm linearly. The idea is just to create all possible $\pm$ combinations of the 1-norm instead of absolute values; $\|\underline{w_j}\|_1 = \pm w_1 \pm w_2 \pm \ldots \pm w_n$. Then imposing all these constraints such that only one $\pm$ combination is satisfied, which is equivalent to the 1-norm for that orthant.
The immediate problem with this approach is that the number of constraints for the 1-norm exponentially increases for the number of dimensions. For each $\|\underline{w_j}\|_1$ there is 2^n constraints, which means this formulation has exponential scaling.
Before stating the formulation we must define a vector $\underline{e}$ which consists 2^n n-dimensional vectors. This vector of vectors contains all possible combinations 1 and -1 over a n-dimensional vector. e.g. for 2-dimensions $\underline{e}$ would be:

$$\underline{e} = \{\, \{-1,-1\}\,,\ \{1,-1\}\,,\ \{-1,1\}\,,\ \{1,1\}\,\}$$

Using this newly defined vector $\underline{e}$ the formulation for this alternative 1-norm is the following:

$$\begin{array}{lll}
\sum_{r=1}^{n} (e_p)_q \underline{w_j}_q \geq 1 - (\sqrt{n}+1)(1-\varphi_{jp}) & j \in \{1,\ldots,k\} \quad ,p \in \{1,\ldots,2^n\} & (1) \\
\varphi_{jp} \in \{0,1\} & j \in \{1,\ldots,k\} \quad ,p \in \{1,\ldots,2^n\} & (2) \\
\sum_{p=1}^{2^n} \varphi_{jp} = 1 & j \in \{1,\ldots,k\} & (3)
\end{array} \tag{5.2}$$

Using vector $\underline{e}$ defined above we can obtain the one norm $\|\underline{w_j}\|_1 = \sum_{q=1}^{n} (e_p)_q \underline{w_j}_q$. Using binary variable φ_{jp} to make sure that at least one combination satisfies the constraint (≥ 1), similarly to previous algorithms $(\sqrt{n}+1)$ is our tightest constant such that if $\varphi_{jp} = 0$ then any possible $\underline{w_j}$ will satisfy (1).
Although this formulation is simple to state the computation difficulty with this formulation is that it adds a further $k * 2^n$ binary variables (φ_{jp}), as well as a further $k * 2^n$ constraints (of which only k are tight constraints). For this reason it does not make sense to add an upper bound on this algorithm, furthermore preliminary testing different instances with and without an upper bound of this one norm formulation supports this reasoning (test results not provided).

Below is table 5 containing the computational results for the improved formulation with this brute force 1-norm ((1.2) instead of algebraic 1-norm $\cap$ (4.1) $\cap$ (5.2)):

Table 5: Computational results of Brute force 1-norm (5.2)

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	719	0.13	0.268	0.268	0%
	16	3	3	Test2	8,817,025	300	0.0712	0	100%
	20	3	3	Test3	8,032,079	300	1.86	0	100%
	24	3	2	Test4	1,101,151	41.7	3.98	3.98	0%
	30	2	3	Test5	4,549,879	300	2.53	1.30	48.8%
Medium	500	2	2	Test6	1,800,370	1200	111.1	8.04	92.8%
	500	5	4	Test7	583,328	1200	196.7	0	100%
	500	8	4	Test8	315,397	1200	110.3	0	100%
	500	5	6	Test9	434,086	1200	195.1	0	100%
	500	8	6	Test10	230,476	1200	156.6	0	100%
Large	2000	5	4	Test11	133,342	2700	730.2	0	100%
	2000	16	4	Test12	6070	2700	2182.0	0	100%
	2000	5	8	Test13	2689	2700	2151.7	0	100%
	2000	16	8	Test14	2360	2700	6519.7	0	100%
	2000	16	12	Test15	5148	2700	3600.9	0.00	100%

As this is the alternate 1-norm formulation we will make a lot of comparisons to the algebraic 1-norm formulations. Over all instances the algebraic formulations seems to perform better; with a few exceptions however they are similar in these cases with brute force 1-norm having only slightly better results. Due to the tightness of this formulation we would of expected a significant improvement to at the lower bound value, however this has not been the case.

A reason why this formulation is not performing as well as we might of expected is maybe explained by the nodes. With the number of binary variables exponentially scaling we would expect the number of nodes to be significantly higher than the number of nodes for algebraic 1-norm, however in most cases the number of nodes is similar with cases of it being both higher and lower. Therefore it is likely Gurobi is branching over the feasible region with the use of envelopes rather than binary variables.

It's reasonable to conclude that this formulation has not performed as we expected is due to the time frame; due to its exponential nature for large samples it would require large time frames to reach a result. If given more time its possible this algorithm could out-perform other algorithms (which are also given the same extended time) which may reach a "dead end" in there computations; this is a common thing with the basic formulations.

This issue here is convergence, its not clear how appropriate/effective the value is because its convergence maybe be slow at first but perform much better after considerable time, similarly some algorithms may of perform well at first but results may slow and it may reach a dead end computationally where not much progress is made. This is likely to be a recurring issue and not just subject to this formulations, as a result I investigate test 3, test 8, and test 15 on longer time frame solves (over all formulations in this paper) to get an idea for their convergence. This formulation does still

make progress on longer time frames where others get stuck, however this formulation will find it hard to escape the exponential scaling drawback.

The natural continuation for this brute force 1-norm formulation is to add the infinity norm as we have done previously. The results for this formulation are summarised in table 6:

Table 6: Computational results of Brute force 1-norm (5.2) + infinity norm (4.1)

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	1038	0.10	0.268	0.268	0%
	16	3	3	Test2	5,689,503	300	0.0712	0.069	2.6%
	20	3	3	Test3	5,211,273	300	1.86	0	100%
	24	3	2	Test4	561,025	25.3	3.98	3.98	0%
	30	2	3	Test5	4,442,663	300	2.53	1.68	33.8%
Medium	500	2	2	Test6	1,733,970	1200	111.1	10.8	90.3%
	500	5	4	Test7	455,288	1200	209.7	0	100%
	500	8	4	Test8	295,997	1200	100.3	0	100%
	500	5	6	Test9	356,542	1200	192.1	0.00	100%
	500	8	6	Test10	241,302	1200	132.1	0	100%
Large	2000	5	4	Test11	93,492	2700	978.00	0	100%
	2000	16	4	Test12	19,973	2700	1424.0	0	100%
	2000	5	8	Test13	6888	2700	1567.7	0.00	100%
	2000	16	8	Test14	2199	2700	1511.7	0	100%
	2000	16	12	Test15	6587	2700	906.2	0.00	100%

Comparing these results with algebraic 1-norm + Infinity norm (table 4). After the initial computational results of the brute force 1-norm I was expected that these results would follow the same downward trend, however this was not the case. The results were more in-line/similar with the Alg + Inf formulation than would be thought, moreover for large instances this formulation had some of the best results we have seen so far (mostly better than the results of Alg + Inf).
The number of lower bounds has improved with it mostly reaching a very small LB for medium/large instances. Potentially suggesting this formulations (or complicated formulations in general) could be important for obtaining lower bounds.
So why have the results improved so much from are original brute-force testing? The reason of why these results have had a turn-around is of course due to the addition of the Inf-norm, however these results aren't just the middle ground between the Inf-norm and brute-force 1-norm formulation so they must be correlating (i.e. not working independently). As we discussed in the previous set of results the brute force formulation is likely very strong however likely has slow convergence, therefore the reason for this improvement in results is due to Inf-norm acting as a catalyst speeding up the convergence of the brute force formulation. This is an idea we can explore later, adding future formulations and seeing how brute-force's convergence is effected.

Lastly adding the algebraic formulations just to analyses the effects of this, however due to the both algebraic and brute force formulation being the same geometrically we should expect the results to be worse due to the extra computation with no feasible region benefit:

Table 7: Results of Brute force 1-norm (5.2) + infinity norm (4.1) + algebraic 1-norm (5.1)

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	706	0.10	0.268	0.268	0%
	16	3	3	Test2	5,343,529	300	0.0712	0.047	34.0%
	20	3	3	Test3	3,869,358	300	1.86	0	100%
	24	3	2	Test4	535,926	25.5	3.98	3.98	0%
	30	2	3	Test5	5,015,245	300	2.53	1.51	40.5%
Medium	500	2	2	Test6	1,726,788	1200	111.1	11.0	90.3%
	500	5	4	Test7	421,691	1200	220.3	0.00	100%
	500	8	4	Test8	287310	1200	106.8	0.00	100%
	500	5	6	Test9	319,497	1200	259.5	0.00	100%
	500	8	6	Test10	233,867	1200	131.1	0	100%
Large	2000	5	4	Test11	39,100	2700	1331.6	0	100%
	2000	16	4	Test12	15,466	2700	1138.5	0	100%
	2000	5	8	Test13	30,541	2700	3667.03	0.00	100%
	2000	16	8	Test14	2734	2700	12053.4	0	100%
	2000	16	12	Test15	9453	2700	5751.4	0.00	100%

These results are as we expected, the objective values have servilely deteriorated with the adding of this algebraic 1-norm; with the exception Test12 which improved its results. The reason for this anomaly is likely due to high ratio of clusters to dimensions and the other instances with similar ratios have similar effects on the objective values compared to previous results.
One interesting point is that although the formulation has caused a negative effect to most objective values it has however produced very small lower bounds on the large instances which previously hadn't been achieved. This could suggest that more formulations and extra constraints may be useful for obtaining lower bounds in the future, but have the drawback that objective value converge (much) slower. This is a idea which will be answered through further testing in our future chapters about rotations; which will be using lots of norm formulations.

6 Rotations

In this chapter we will look to create rotation matrix in n-dimensions, with the goal applying these rotations to out existing norms to create different linear formulations. We will present an algorithm which creates a rotation matrix for point to point rotations (disregarding orientating of an object). More complex rotation algorithms may be better in theory but creating such an algorithm might not only be computationally complicated but also mathematically complicated; however possible. It is worth pointing out the majority of difficulty of creating a rotation in n dimensions is that rotation matrices for dimensions $n > 3$ are not readily available. This is not to say they don't exist but there

is no given formulas. Therefore the rotation matrices constructed are based on this paper [7], which uses the composition of $2D$ rotations matrices (more on this later). Using this idea we will construct an algorithm later which produces a desired rotation matrix.

6.1 n dimensional rotation matrix

In this subsection we will construct the algorithm for producing a rotation matrix which rotates vector $\underline{a}$ to $\underline{b}$. Before defining the algorithm there are some concepts and ideas we must state and establish about rotation in n-Dimensions.
Since we know how to rotate objects in 2 and 3 dimensions ($n = 2, 3$) where rotations matrix are well (and easily) defined, it would make sense we start with 2 dimensions and build upon that. The difficulty with n-dimensional rotations is visualising/conceptualising a rotation in n-dimensions ($n > 3$) is not possible, therefore simplifying these rotations to a 2 dimensional case will allow us to formulate some of the calculations needed to rotate coordinates to the correct position.
The idea of this algorithm is to work with only 2 dimensional rotations (i.e. plane rotations) while keeping the other $(n-2)$ dimensions fixed. In 3 dimensions an example of this would be rotating around the y-axis, we are keeping y fixed while x and z are rotating.

Below is the standard 2-dimensional rotation matrix which is (x, y) are rotated by angle θ.

$$R(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

We will use this 2-dimensional form along with the standard identity matrix to apply to a matrix of any size, this will result in only rotating 2 coordinates while keeping the rest fixed. Going back to our 3-dimensional case we have below the matrix for rotations about the y-axis. The following matrix describes this rotation, and its clear from this y remains fixed while x and z are rotated.

$$R_{1,3}(\theta) = \begin{bmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\theta) \end{bmatrix}$$

We will continue to use this notation $R_{i,j}$ to denote that the rotation plane is x_i and x_j in n-dimensions, so for the example above $x_1 = x$ and $x_3 = z$. In 3-dimensions there are only 3 possible rotation planes however in 4-dimensions there are 6 possible planes, and in general there are $n(n-1)/2$. This is a consideration we will have to take into account.

Remark. Rotations by this method are **not unique** for $n > 2$.

Although rotating vectors $\underline{a}$ to $\underline{b}$ is not affected by this non-uniqueness, this is because vectors can be thought of as points or lines (i.e 0-dimensional or 1-dimensional objects). However objects with more dimensions ($n \geq 2$) will be effected by this non-uniqueness, because it will be able to rotate a point of this object to another point, but doesn't describe how the object/shape is rotated about that point. This is not something we will ignore (in fact we will try to minimise this effect) but it is more important the shape is rotated to the correct position than the rotation of the shape at that position.

Building on from our simple 2 and 3 dimensional rotation matrix, our n-dimensional matrix will be the identity matrix with 4 of its entries changed from the standard 2-dimensional rotation matrix. The following matrix is rotating by angle θ in the i , $i+1$ plane, while all other coordinates remain

fixed and invariant.

$$R_{i,i+1}(\theta) = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & cos(\theta) & -sin(\theta) & \dots & 0 \\ 0 & 0 & \dots & sin(\theta) & cos(\theta) & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 1 \end{bmatrix}$$

Similarly the more general case would if we could rotate between any i, j plane, the corresponding rotation matrix is as follows:

$$R_{i,j}(\theta) = \begin{bmatrix} 1 & 0 & \dots & 0 & \dots & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \dots & cos(\theta) & \dots & -sin(\theta) & \dots & 0 \\ \vdots & \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & \dots & sin(\theta) & \dots & cos(\theta) & \dots & 0 \\ \vdots & \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix}$$

This more general form of the rotation matrix (in the i, j plane) will be more effective at solving our rotations as we can pick i and j, the importance of this is because some i and j combinations will be mathematically impossible to make progress on the algorithms.
The idea is to rotate every coordinate of $\underline{a}$ to every coordinate of $\underline{b}$ sequentially (but not in a particular order). Each rotation matrix $R_{i,j}$ will rotate 1 coordinate at a time and therefore to fully complete the rotation we will need $n-1$ rotation matrices because the last coordinate is fixed due to preservation of length. In each case we will be using $R_{i,j}$ to rotate the i^{th} coordinate to the correct position, we will denote a rotated vector $\underline{a}$ as $\underline{a'}$ (i.e $R_{i,j}(\theta)\underline{a} = \underline{a'}$). With each rotation $R_{i,j}$ only changing 2 of the coordinates of $\underline{a}$, more importantly we will use these equations to determine a given θ for $R_{i,j}$ as we know we want our rotated $\underline{a})_i$ coordinate to be fully rotated to $\underline{b}_i$; i.e $(R_{i,j}(\theta)\ \underline{a})_i = \underline{a'}_i := b_i$:

$$\begin{aligned} a_i\ cos(\theta) - a_j\ sin(\theta) = a'_i := b_i \\ a_i\ sin(\theta) + a_j\ cos(\theta) = a'_j \end{aligned} \tag{6.1}$$

Our second equation is not needed for any direct calculations but we need to keep track of where we rotated a_j. Therefore we can string composite of rotations together with a series of $R_{i,j}$ to complete the full rotation.
Equation 6.1 will determine θ; however we need to convert this to a simpler form so that we can solve for θ analytically. Notice if we take out a factor of $\sqrt{p^2+q^2}$ then 6.1 becomes:

$$\sqrt{(a_i)^2+(a_j)^2}\left(\frac{a_i}{\sqrt{(a_i)^2+(a_j)^2}}\ cos(\theta)\ -\ \frac{a_j}{\sqrt{(a_i)^2+(a_j)^2}}\ sin(\theta)\right) = a'_i.$$

The coefficients of $cos(\theta)$ and $sin(\theta)$ can be described as there own cos and sin term, with the angle being $tan^{-1}(\frac{q}{p})$. We can then apply cosine angle formula ($cos(x \pm y) = cos(x)cos(y) \mp sin(x)(sin(y))$

to obtain:

$$p\,cos(\theta) \pm q\,sin(\theta) = \sqrt{p^2+q^2}\,cos\left(\theta \mp tan^{-1}\left(\frac{q}{p}\right)\right) \tag{6.2}$$

This form is clearly much easier to solve, furthermore q can be any real value here, however in this equation $p > 0$ but for $p < 0$ the form is the same but negative (i.e. $-\sqrt{p^2+q^2}\,cos(\theta \mp tan^{-1}(\frac{q}{p}))$).

Note: We are not interested in the actual value of individual θ_r (angle of each 2 dimensional rotation) as we only need the matrix, we will need θ_r to calculate $R_{i,j}$. This is important to simplify this process further, as previously its been softly implied that $i < j$ but this is not a constraint we need to maintain; in fact this would cause problems if this was the case. The reason both $i < j$ and $i > j$ (but $i \neq j$) are perfectly valid approaches is because with $i < j$ we just get equation 6.1, but with $i > j$ we would get the following equation:

$$a_j\,sin(\theta) + a_i\,cos(\theta) = a'_i$$

This is our rotation of $i - j$ coordinates by angle θ, but instead if we consider this as a rotation of angle $-\theta$ then the equation above just become 6.1 again. Using this we can simplify generating $R_{i,j}$; for the case where $i > j$ we can create the matrix same as we would for $i < j$, below is the corresponding matrix.

$$R_{i,j}(\theta) = \begin{bmatrix} 1 & 0 & \dots & 0 & \dots & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \dots & cos(\theta) & \dots & sin(\theta) & \dots & 0 \\ \vdots & \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & \dots & -sin(\theta) & \dots & cos(\theta) & \dots & 0 \\ \vdots & \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix}$$

The only change is the sign of each sine term; the same as rotating by $-\theta$.

Putting all this together we can state the algorithm for generating the rotation matrix for rotating $\underline{a}$ to $\underline{b}$:

Algorithm 1: Generating a n-Dimensional rotation matrix (n x n)

1 $\underline{a}, \underline{b} \in \mathbb{R}^n$, and $\underline{a}, \underline{b} \neq \underline{0}$
2 Initialise matrix $R = I_n \ \in M_{n\,x\,n}(\mathbb{R})$
3 **while** $\underline{a} \neq \underline{b}$ **do**
4 **for** i *in* $(1, 2, \ldots, n)$ **do**
5 **if** $|a_i| > |b_i|$ **then**
6 **for** j *in* $1, 2, \ldots, n$ **do**
7 **if** $i \neq j$ *and* $a_j \neq b_j$ **then**
8 $T = \sqrt{a_i^2 + a_j^2}$
9 **if** $a_i < 0$ **then**
10 $T = -T$
11 $\alpha = tan^{-1}\,(\frac{a_j}{a_i})$
12 $\theta = cos^{-1}\,(\frac{b_i}{T}) - \alpha$
13 **if** $sign(a_i\ sin(\ \theta) + a_j\ cos(\ \theta)) \neq sign(b_j)$ **then**
14 $\theta = -\theta - 2\alpha$
15 Generate $R_{ij}(\theta)$
16 $R = R_{ij}(\theta) * R$
17 $\underline{a} = R_{ij}(\theta) * \underline{a}$
18 **Break**
19 **return** R

Note this algorithm assumes that they are equal distance ($\|\underline{a}\|_2 = \|\underline{b}\|_2$). However the algorithm works in general with just the additional step of just scaling $\underline{b}$ by $\frac{\|\underline{a}\|}{\|\underline{b}\|}$; before step 3.
I will briefly explaining what this algorithm is doing and explain some steps which may not be immediately obvious. Lines 4 - 7 are loops and if statements which decide our i and j for our coordinates. Firstly they search over all possible i with the conditions $|a_i| > |b_i|$, this ensures that an element a_i can be rotated onto b_i; because if b_i was much larger than a_i the rotation wouldn't be possible. Additionally the strict inequality ensures that $a_i \neq 0$ and that it isn't already in the correct position (i.e. $a_i \neq b_i$). Lines 6 and 7 are doing a similar thing for j, checking a_j is not already in the correct position and that $i \neq j$. Lines 8 - 10 determine the value of T, making it negative if the initial *cos* term is negative. Lines 13 - 14 aren't explained previously, they are for the last iteration of the algorithm to ensure the last coordinate is not the $-b_j$ instead of b_j. Each

iteration of the algorithm will rotate 1 coordinate with the last iteration rotating 2 coordinates, now due to rotations preserving lengths/norms the last coordinate is fixed except its which is it's sign which could be either positive of negative. To get round this, line 13 checks the outcome of a'_j, if the sign is not what it should be it changes the value of θ to $-\theta - 2\alpha$. Which make look strange however its just the fact *cos* is an even function, therefore:

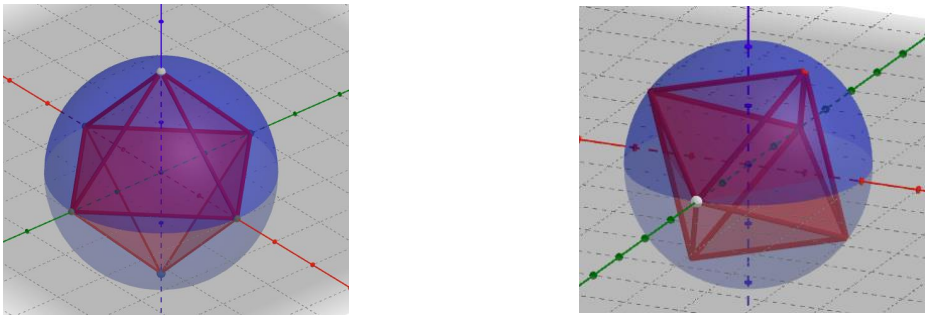
$$T\ cos(\ (-\theta - 2\alpha) + \alpha) = T\ cos(\ -(\theta + \alpha)) = T\ cos(\ \theta + \alpha)$$

This will mean that a'_i unaffected while a'_j will have the opposite sign than previously.

7 Rotations along the axis

This chapter is about rotations of the existing formulations (specifically the algebraic 1-norm). Rotating a single formulation is not likely to bring any added benefit as geometrically it is the same amount of volume, additionally the formulation is the same with different scalars for the rotations. With rotations we can create multiple rotations of the same formulations, slightly rotated from one another to create a larger linear geometric infeasible region (therefore reducing the linear feasible region). The benefit of rotating the same formulation is previously the linear feasible region increased with the dimension. Now with rotations we are able to improve this scaling so that the linear feasible region scales with the dimensions, however this comes at a quadratic complexity cost. Although we will be using rotations to scale our linear feasible region we will be adding more constraints and therefore calculations to the problem.

This rotation along the axis can be seen graphically in figure 4 below:



(a) 1-norm within 2-norm

(b) Rotated 1-norm within 2 norm

Figure 4: Rotation of $\frac{\pi}{6}$ along an axis

The octahedron is rotated $\frac{\pi}{6}$ along the $x - z$ plane. This is only one rotation of one octahedron the idea of this formulation is to create many rotation to reduce the linear feasible region.

An important new parameter in this chapter is going to be the angle of rotation between each formulation, too small an angle will create too many formulation. Which is not only bad because the additional equations but they will also create lots of overlap between their linear feasible regions, making each rotated formulation less effective.
There are infinite rotations which can be made, and the effectiveness of one rotation over another will vary from each instance, therefore creating a geometrical symmetric linear feasible region will

help avoid any bias to instances.
The simplest rotation idea is to rotate the 1-norm (hyperoctahedron) along each axis plane intercept. We know there are $(n(n-1))/2$ intercepts which means any rotations formulated along the axis will scale quadratically with the dimensions.

When figuring out by what angle to rotate the 1-norm by we need to consider the symmetric properties and avoid any formulation which is not symmetric, while also not creating too many formulations that Gurobi struggles with the number of equations. Due to the how the algebraic 1-norm is formulated we would only have to rotate in one Orthant (because of the symmetry of the hyperoctahedron), so we will limit our rotations to partition the set $[0, \frac{\pi}{2}]$. (It's convenient to use the term partition here because although its not immediately obvious, there rotation can be thought of partitioning the hypersphere into subsets based on there angles, this way of thinking will be more relevant in later chapters). Mathematically speaking we should partition the set $[0, \pi]$ because rotating a coordinate a_i is given by the formula $a_i\, cos(\theta) - a_j\, sin(\theta) = a'_i$. Applying this coordinate to the Algebraic 1-norm is then π periodic, however geometrically the set of $[\frac{\pi}{2}, \pi]$ has lots of overlap with $[0, \frac{\pi}{2}]$. As a result computationally it will more sense to just rotate over $[0, \frac{\pi}{2}]$ to produce better results; furthermore from preliminary testing this is also the case, with partitions of $[0, \frac{\pi}{2}]$ out performing the partition of $[0, \pi]$ on longer time frames and on large instances.

When deciding how to partition $[0, \frac{\pi}{2}]$ we should consider factors of $\frac{\pi}{2}$. The largest factor (and therefore rotation) would be by $\frac{\pi}{2}$ however geometrically this shape is same. Therefore starting by partitioning the set into 2 and 3 subsets and if the data improves for 3 subsets (compared to 2) we could experiment with 4 subsets. Our rotations angles will be $\frac{\pi}{6}$ and $\frac{\pi}{3}$ for our partition of 3, and just a rotation of $\frac{\pi}{4}$ for the partition of 2.
We have previously said that the number of axis intercepts is $(n(n-1))/2$, so partitioning each $[0, \frac{\pi}{2}]$ d times leads us to the following number of rotations $C = (d-1)\frac{(n(n-1)}{2}$, where $d = \{2, 3\}$. We now have all the information to start our rotation algorithm along the axis,

$$
\begin{array}{rcll}
R_r(\theta_r)(\underline{w_j}) & = & \underline{w'_j}_r & j \in \{1, \dots k\}, \quad r \in \{1, \dots C)\} \quad (1) \\
(\underline{w'_j}_{qr}) & = & \underline{\rho_j}_{qr} - \underline{\eta_j}_{qr} & q \in \{1, \dots n\}, \quad j \in \{1, \dots k\}, \quad r \in \{1, \dots C)\} \quad (2) \\
\underline{\rho_j}_{qr} & \leq & v_{jqr} & q \in \{1, \dots n\}, \quad j \in \{1, \dots k\}, \quad r \in \{1, \dots C)\} \quad (3) \\
\underline{\eta_j}_{qr} & \leq & 1 - v_{jqr} & q \in \{1, \dots n\}, \quad j \in \{1, \dots k\}, \quad r \in \{1, \dots C)\} \quad (4) \\
 & & \underline{\rho_{jr}} \in (\mathbb{R}^+)^n,\ \underline{\eta_{jr}} \in (\mathbb{R}^+)^n & j \in \{1, \dots k\}, \quad r \in \{1, \dots C)\} \quad (5) \\
 & & v_{jqr} \in \{0, 1\} & q \in \{1, \dots n\}, \quad j \in \{1, \dots k\}, \quad r \in \{1, \dots C)\} \quad (6) \\
1 & \leq & \sum_{q=1}^{n} (\underline{\rho_j}_{qr} + \underline{\eta_j}_{qr}) & r \in \{1, \dots C)\} \quad (7)
\end{array}
\tag{7.1}
$$

In this formulation every thing is very similar compared to algebraic 1-norm formulation (5.1), however because of the number of rotations and therefore number of algebraic 1-norms there is an additional index r. This additional index is used to denote which rotated algebraic 1-norm is being formulated. It's also worth noting that this will not include the original (non-rotated) algebraic 1-norm in this formulation.

7.1 Rotation angle: $\frac{\pi}{6}$

The rotation along the axis is $\frac{\pi}{6}$ which corresponds to partitioning the set $[0, \frac{\pi}{2}]$ 3 times. The computational results for the rotations of the algebraic 1-norm at $\frac{\pi}{6}$ and $\frac{\pi}{3}$ along each axis intercept are as follow:

Table 8: Computational results of rotation by $\frac{\pi}{6}$ (7.1)

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	792	0.11	0.268	0.268	0%
	16	3	3	Test2	1,771,461	199.7	0.0712	0.0712	0%
	20	3	3	Test3	2,391,763	300	1.86	0	100%
	24	3	2	Test4	693,314	35.1	3.98	3.98	0%
	30	2	3	Test5	2,359,541	168.2	2.53	2.53	0%
Medium	500	2	2	Test6	1,456,107	1200	111.1	12.5	88.8%
	500	5	4	Test7	519,528	1200	256.4	0	100%
	500	8	4	Test8	236,783	1200	99.8	0	100%
	500	5	6	Test9	229,058	1200	279.7	0	100%
	500	8	6	Test10	141,867	1200	420.5	0	100%
Large	2000	5	4	Test11	132,744	2700	1091.2	0	100%
	2000	16	4	Test12	7449	2700	1803.0	0	100%
	2000	5	8	Test13	96,548	2700	2269.5	0.00	100%
	2000	16	8	Test14	7302	2700	8201.7	0	100%
	2000	16	12	Test15	7071	2700	10734.4	0	100%

The results for this dual rotation are very varied, with some instances performing well and others performing very badly. What is clear is that as the dimension increases the result become worse, which is to be expected due to its quadratic number of formulations. Interestingly on a longer time frame the algorithm is continuously making improvements (not reaching a "dead end" like other algorithms) and even out performing other algorithms on that longer time frame. This is reinforcing our claim from the brute force 1-norm that on longer time frames more calculations/constraints will yield better results.

Importantly the combination of formulations from these rotation has worse results than the standard algebraic 1-norm, even on longer time frames, however the algebraic 1-norm does reach a "dead end" which the rotations didn't but even on 3 times the standard time frame the results didn't catch up with the standard algebraic 1-norm, however that's not to say on a longer time frame it wouldn't catch up.

Test 13 provides some interesting insight into how Gurobi is likely branching in this formulation. Test 13 has a larger number of nodes than we would expect. This must be due to the parameters, specifically it having a high dimensions to cluster ratio. We also see similar changes in the number of nodes with other instances when considering this ratio. This is because Gurobi is able to branch over possible sets of $\underline{w_j}$ and then branch over possible sub regions of each $\underline{w_j}$.

Adding the infinity norm formulation (4.1) to build upon this formulation. The idea is that hopefully this Inf-norm will act as a catalyst accelerating its progress in the algorithms similar to how it improved brute-force 1-norm. Additionally we won't be using the standard non-rotated algebraic 1-norm because it does not add much geometric volume compared to the rotated 1-norms, furthermore this is supported by preliminary testing with the non-rotated Alg-1-norm.

Table 9: Computational results of rotation by $\frac{\pi}{6}$ (7.1) + infinity norm (4.1)

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	1282	0.16	0.268	0.268	0%
	16	3	3	Test2	3,077,397	300	0.0712	0	100%
	20	3	3	Test3	3,082,750	300	1.86	0	100%
	24	3	2	Test4	406,429	22.2	3.98	3.98	0%
	30	2	3	Test5	1,847,151	148.6	2.53	2.53	0%
Medium	500	2	2	Test6	1,764,846	1200	111.1	11.0	90.1%
	500	5	4	Test7	428,421	1200	268.6	0	100%
	500	8	4	Test8	297,295	1200	105.6	0	100%
	500	5	6	Test9	258,534	1200	369.7	0	100%
	500	8	6	Test10	131,606	1200	184.9	0	100%
Large	2000	5	4	Test11	136,712	2700	1456.2	0	100%
	2000	16	4	Test12	9966	2700	1242.6	0	100%
	2000	5	8	Test13	27,615	2700	1599.0	0	100%
	2000	16	8	Test14	7258	2700	3014.6	0	100%
	2000	16	12	Test15	1319	2700	3220.1	0	100%

The addition of the infinity norm in this formulation has not had the catalyst effect that we have seen it had previously. The results have all slightly declined from the formulation without the Inf-norm, but on the large instances the results have improved; presumably this is due to the Inf-norm's effectiveness on large instances. Additionally over longer time frames the results are some of the best we have seen of all the formulations (notably Test 15 reach 570.7 over 8100 seconds). This leads back to our theory of how convergence works for this problem, specifically that large formulations may have slower convergence but produce better results on a longer time frame;‘ which is the case here.
The number of nodes is following similar patern to the previous formulations with the node count being higher on instances with high dimensions to cluster ratios (e.g. Test 13 and Test 9). Additionally Test 15 is has less than 5 times the amount of nodes than the formulation without the Inf-norm but the objective value is much better. Here the Inf-norm is clearly allowing Gurobi to be more critical on its calculations, and as a result acting as a catalyst effect but only for large instances.
Last it's worth noting that with previous large formulations and combinations of formulations we have got more progress on lower bound, however this formulation has taken a step back on the lower bounds front. Especially with Test 2 which finished previously however with the addition of the Inf-norm it failed to obtain any lower bound.

7.2 Rotation angle: $\frac{\pi}{4}$

The rotation along the axis is $\frac{\pi}{4}$ which corresponds to portioning the set $[0, \frac{\pi}{2}]$ 2 times. The computational results for the rotations of the algebraic 1-norm at $\frac{\pi}{4}$ along each axis intercept are as follow:

Table 10: Computational results of rotation by $\frac{\pi}{4}$ (7.1)

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	767	0.09	0.268	0.268	0%
	16	3	3	Test2	1,338,297	97.2	0.0712	0.0712	0%
	20	3	3	Test3	3,793,174	300	1.86	0.004	100%
	24	3	2	Test4	332,574	15.0	3.98	3.98	0%
	30	2	3	Test5	5,061,739	300	2.53	2.10	17.0%
Medium	500	2	2	Test6	1,611,058	1200	111.1	9.88	91.1%
	500	5	4	Test7	487,311	1200	229.2	0	100%
	500	8	4	Test8	307,233	1200	139.3	0	100%
	500	5	6	Test9	299,294	1200	328.6	0	100%
	500	8	6	Test10	218,296	1200	172.6	0	100%
Large	2000	5	4	Test11	105,922	2700	1125.5	0	100%
	2000	16	4	Test12	19,944	2700	3019.0	0	100%
	2000	5	8	Test13	82,082	2700	1862.9	0.00	100%
	2000	16	8	Test14	6483	2700	2505.8	0	100%
	2000	16	12	Test15	6735	2700	4779.4	0	100%

The idea with this formulation is it is half as many algebraic 1-norms and as a result we should expect it to scale better as the instances get larger. However the results seemed to have improved for most instances with only a few slightly behind the previous $\frac{\pi}{6}$ rotation. Additionally the preliminary long time frame results are good as well, besting most the long time frame results for the $\frac{\pi}{6}$ rotation. It seemed some results did hit a possible "dead end" with the progress slowing and stopping way before the time limit on some instances. Suggesting this kind of formulation has a longer convergence than simpler formulation but its shorter than our $\frac{\pi}{6}$ rotation formulation which is what we would expect.
The number of nodes again follows the recurring theme that it depends on the dimension to cluster ratio. Although the number of nodes has not been to reflective of the objective value for an instance across formulations.

Again adding the infinity norm formulation (4.1) to build upon this formulation and hopefully use this Inf-norm as a catalyst for this large formulation.

Table 11: Computational results of rotation by $\frac{\pi}{4}$ (7.1) + infinity norm (4.1)

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	1173	0.14	0.268	0.268	0%
	16	3	3	Test2	4,350,968	300	0.0712	0	100%
	20	3	3	Test3	3,646,360	300	1.86	0.04	97.8%
	24	3	2	Test4	1,474,132	70.3	3.98	3.98	0%
	30	2	3	Test5	3,522,516	223.1	2.53	2.53	0%
Medium	500	2	2	Test6	1,785,864	1200	111.1	8.24	92.6%
	500	5	4	Test7	463,096	1200	197.9	0	100%
	500	8	4	Test8	329,220	1200	104.1	0	100%
	500	5	6	Test9	224,731	1200	210.5	0	100%
	500	8	6	Test10	151,171	1200	129.4	0	100%
Large	2000	5	4	Test11	124,360	2700	1224.7	0	100%
	2000	16	4	Test12	12,620	2700	846.2	0	100%
	2000	5	8	Test13	22,478	2700	2748.0	0	100%
	2000	16	8	Test14	6201	2700	4245.6	0	100%
	2000	16	12	Test15	5726	2700	2869.5	0	100%

The Inf-norm has improved the a lot of the results with only a few declining. Additionally notice that Test 2 was not solved in this formulation but previously without the Inf-norm it was, this was the same situation we had with the rotation by $\frac{\pi}{6}$, this seems to be a recurring scenario for this rotated formulations; showing a potential drawback of this Inf-norm formulation. The addition of the Inf-norm also had a lot of repeated traits which we had seen with previous iterations, such as: change in nodes, improvement in objective values, and slight declines in lower bounds.
Test 3 has been a instance of interest throughout this investigation as although it is one of the small instances no formulation has been able to definitely solve it, furthermore all previous formulations have not even obtained a lower bound to this instances, even on 900 second time frames. This formulation is the only one to obtain a significant lower bound at 300 seconds and additionally it was able to solve it at 842 seconds. This individual result isn't significant but it is surprising that only one formulation has solved this problem and no other formulation has even come close.

In conclusion these formulations are large scale formulation which add huge amount of equations/constraints to the model. It's clear that the results over the default time period are not as good as simpler formulations like Inf-norm + Alg-1-norm. However this would be disregarding there convergence (which has not been fully analysed in this paper) as these formulations seem to provide endless areas of development for Gurobi to work on, and as a result these formulations don't reach "dead end" (in a short time period). The result of this conclusion is that on longer time frames and for larger problems is that large formulations will start off slow but give the best results over a long time period, although its important to note that some large formulations aren't good and the success of a formulations long term convergence can not be determined by how well it starts.

8 Inf-norm vs Algebraic 1-norm formulation

This chapter is comparing the Inf-norm and Algebraic 1-norm formulations, choosing to dismiss the brute force norm because of its exponential scaling therefore not being viable for large scale problems. With our previous rotation chapter (6) and now being able to rotate objects in n-dimensional space, we can now rotate the hyper-octahedron (algebraic 1-norm) to a corner of the hypercube associated to the Inf-norm.
Its important to notice that these objects are not the same shape, with the hyper-octahedron being significantly smaller volume than the hyper-cube of the Inf-norm. Therefore due to both these objects and formulations creating linear constraints it would be expected that the Inf-norm would produce better results, however the the 2 formulations are very different with the Inf-norm being more reliant on binary variables to create boundaries, where as the Alg 1-norm uses lots more variables however they are more fixed and determined by $\underline{w_j}$ (or $R(\theta)\underline{w_j}$) rather than actually being variables.
There is no need to state the formulation for this as it is the same as the algorithm for the algebraic 1-norm 5.1 except that $(\underline{w_j})$ would be rotated, i.e. $(\underline{w_j}) \rightarrow R(\underline{w_j})$. Where R is the rotation matrix that maps the point on the hyper-octahedron $(1, 0, \ldots, 0)$ to a point on the hyper-cube $(\frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}})$. As we stated previously rotation aren't unique (for dimensions $n > 2$) and as a result the rotation of the hyper-octahedron around the line $(0, \ldots, 0) \longleftrightarrow (\frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}})$ is undetermined. This is something we can't control with the current algorithm, creating a new algorithm which could determine/control this would be extremely hard and way above the scope of this paper. Furthermore the effect of this degree of freedom of rotation doesn't effect the shapes volume and therefore the linear feasible region is the same in size.

Table 12: Computational results of this rotated algebraic 1 norm

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	752	0.09	0.268	0.268	0%
	16	3	3	Test2	4,542,194	206.6	0.0712	0.0712	0%
	20	3	3	Test3	6,262,454	300	1.86	1.06	43.3%
	24	3	2	Test4	480,387	19.01	3.98	3.98	0%
	30	2	3	Test5	5,390,716	300	2.53	1.83	27.7%
Medium	500	2	2	Test6	1,680,057	1200	111.1	8.02	92.8%
	500	5	4	Test7	598,009	1200	177.4	0	100%
	500	8	4	Test8	369,840	1200	104.6	0	100%
	500	5	6	Test9	294,242	1200	173.5	0	100%
	500	8	6	Test10	232,198	1200	157.8	0	100%
Large	2000	5	4	Test11	32,088	2700	1004.9	0	100%
	2000	16	4	Test12	12,570	2700	1972.4	0	100%
	2000	5	8	Test13	36,076	2700	1977.3	0	100%
	2000	16	8	Test14	7230	2700	1580.0	0	100%
	2000	16	12	Test15	5917	2700	2096.5	0	100%

This formulation is just a rotated algebraic 1-norm, occupying the same amount of volume. Therefore we would expect the results to be very similar to the algebraic 1-norm, and comparing the computational results we can see this is some what true. However it appears that this rotated formulation is performing slightly better than the standard non-rotated algebraic 1-norm. Now its worth pointing out that some results are worse in some cases but with 9 out 15 instances performing better this rotated formulation, and only 4 out of the 15 instances performing better with the non-rotated formulation (2 of the instances performed very similar) it suggests the rotated formulation is slightly better. Now the reason for this is most likely due to how Gurobi sections the feasible region into smaller linear feasible regions. Gurobi will likely use the basis vectors $(0, \ldots, \pm 1, \ldots, 0)$ to create it's own boundaries for sectioning the feasible region, which is what the algebraic 1-norm was previously doing. Our formulation of the Algebraic 1-norm is stronger than how Gurobi is doing reducing the feasible region in to linear regions (which is seen by the jump in performance compared to the original algorithm), however there is clearly some overlap which leads us to question whether that this and any algorithm formulation performance will be better if its not tied to any basis vector, as these will be naturally done by Gurobi to reduce the feasible region.

Alternatively comparing this result with the Inf-norm, we suspect this rotated formulation is better than the non-rotated standard algebraic 1-norm, and the algebraic 1-norm out performed Inf-norm. Therefore even with the reduced volume compared to the Inf-norm this formulation is better in most cases, furthermore this rotated 1-norm has made the gap in performance smaller on instances which Alg 1-norm was worse than the Inf-norm.
An interesting case is when $n = 2$, this would mean that these two formulations create the same geometric shapes, but as we can see from the results the rotated algebraic 1-norm produces much better results. For Test4 Inf-norm finished in 133.8 seconds, compared to this time of 19.1 seconds, furthermore Test6 had a gap of 99.7% (Inf-norm) whereas this formulation had a gap of 92.8%.
This data suggests that the formulation of the algebraic 1-norm is much better than the Inf-norm formulation, even thought the Inf-norm geometrical occupies more volume. However this is not taking into consideration longer time frames; analysing the convergence of these algorithms. Additionally the Inf-norm is much stronger on larger scale instances which have higher number of dimensions, this is a trend I expect to continue so this analysis is situational to the size of the problem.

Next we look at the combination of this rotated 1-norm and the standard 1-norm. The idea behind this is that we can test if Gurobi benifits from having different formulation styles (Alg 1-norm + Inf norm) or if it prefers individually efficient formulations (Alg 1-norm + rotated Alg 1-norm). The computational results of this are summarised below: The results are as we would

Table 13: Computational results of this rotated algebraic 1 norm and the non-rotated algebraic 1-norm 5.1

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	883	0.12	0.268	0.268	0%
	16	3	3	Test2	5,023,487	300	0.0712	0.0624	12.4%
	20	3	3	Test3	4,537,937	300	1.86	0.08	95.7%
	24	3	2	Test4	424,519	20.0	3.98	3.98	0%
	30	2	3	Test5	5,240,327	300	2.53	2.05	19.2%
Medium	500	2	2	Test6	1,699,085	1200	111.1	13.5	87.9%
	500	5	4	Test7	477,822	1200	145.8	0	100%
	500	8	4	Test8	310,245	1200	118.1	0	100%
	500	5	6	Test9	327,018	1200	154.8	0	100%
	500	8	6	Test10	231,760	1200	146.4	0	100%
Large	2000	5	4	Test11	100,650	2700	860.9	0	100%
	2000	16	4	Test12	14,506	2700	3477.0	0	100%
	2000	5	8	Test13	20,790	2700	1772.3	0	100%
	2000	16	8	Test14	5376	2700	3461.5	0	100%
	2000	16	12	Test15	6775	2700	2421.4	0	100%

expect, this formulation produces some better results than it's Inf-norm alternative, however it falls short for large scale instances where the Inf-norm is strongest. Compared to it alternative these results have 7 out of 15 instance in which this formulation is better, and 6 out of 15 instances the Inf-norm + Alg 1-norm is better (with 2 out of 15 where the results were very similar).
Therefore concluding the Inf-norm has a very unique benefit that it is very good in high dimensional situations, although the formulation isn't very flexible and the formulation is not optimal for low dimensions. The best use of the Inf-norm formulation is in combination with Algebraic 1-norm or many rotated 1-norms, as the Inf-norm can serve as a catalyst for large dimensional problems.

9 Sectioning the hypersphere

This formulation is considerably different from our other formulations, and will require lots of prior work to show and prove the concept. Previously the problem has been dealt with without the sectioning of hypersphere, except the Inf-norm which splits the hypersphere in to sections depending on which facet of the hypercube $\underline{w_j}$ is above. This sectioning process will go much further, sectioning each orthant of the hypersphere $\underline{\underline{i}}$n to n sections, hence creating $n2^n$ sections. This exponential term may raise caution alarms because exponential constraints is not sustainable for large subsets. However by splitting the hypersphere into sections like this we are able to more precise calculations on the section which may off set the computation cost. More over Gurobi could easily prune off entire sections of the hypersphere with this method which may add a computational benefit for solving solutions definitely.

9.1 The idea

The idea is creating a linear line/planes between the vertex of the hypercube and the nearby basis vectors $e_i = (0, 0, \ldots, 1, \ldots, 0)$ (positive and negative basis vectors). This is best initially shown by a 2 dimensional example:

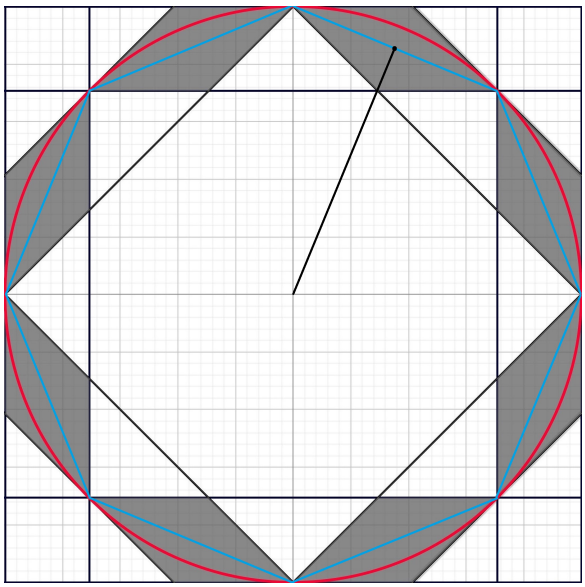


Figure 5: The blue lines show the new formulation compared to the previous 1-norm and inf-norm

This is illustrated on top of our previous 1-norm and inf-norm to motivate the improvements of a formulation like this. Previously when we covered the 1-norm + inf-norm we noted that linear feasible region cause geometric shapes which were all the same and could be considered as separate linear feasible regions, disjoint from one another (for all dimensions). Moreover due to the symmetrical nature of these shapes they could be split down the middle (i.e the blue line), where one half of the shape is all infeasible and therefore the other-side is the reduced feasible region. This concept holds true for all dimensions however the formulation is not that simple for dimensions $n > 2$.
Now the important factor to any one of those sections is the vector $\underline{a_p}$, this vector is the middle

of each of these sections and the middle of each of the blue lines in figure 5 (the figure above only includes one of these vectors). This $\underline{a_p}$ is the midpoint between the hypercube vertex and its associated basis vector, this formulation is consistent for all dimensions. One example of this vector is below; this is the midpoint between e_i and the hypercube vertex $(\frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}})$

$$\underline{a_p} = \left(\frac{1}{2\sqrt{n}}, \ldots, \frac{1}{2}(1 + \frac{1}{\sqrt{n}}), \ldots, \frac{1}{2\sqrt{n}} \right) \tag{9.1}$$

Note the i^{th} term is (corresponding to the same index of e_i) is the $\frac{1}{2}(1 + \frac{1}{\sqrt{n}})$ which will play a significant part in our calculations later on. Now we can define each sectioning as the perpendicular hyperplane to the vector $\underline{a_p}$ (the fact its perpendicular will play a significant role). All the points above the hyperplane are part of that given section. In 2 dimensions the hyperplanes create lines which partition the circle, in 3 dimensions the hyperplanes intersects the sphere which intersection is circle; which definitely is no longer a partition however it isn't clear if these hyperplanes even cover the hypersphere. Below is images of this intersection, its represented as 2 spheres intersecting which would be a circle on the sphere (the perpendicular hyperplane to $\underline{a_p}$). On the circle's circumference is the basis vector e_i and the vertex of the hypercube with the center of the circle being $\underline{a_p}$.

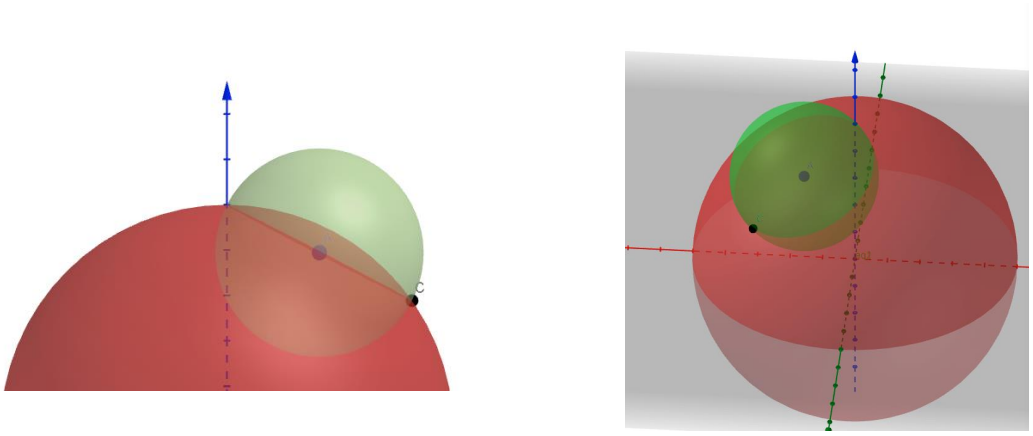


Figure 6: Representation of the perpendicular hyperplane to $\underline{a_p}$

In 4 dimensions and n dimensions the idea is harder to grasp and to hard to image, however the maths for this idea does still hold for larger dimensions (which we will see later on in this chapter). Now that we have our sections established, the idea is to rotate the points $\underline{a_p}$ to it's nearest hypercube vertex. The reason for doing this is at that point all the coordinates have the same absolute value, i.e. $|x_1| = |x_2| = \ldots = |x_n|$. These points are important because the vector $\underline{X_p} = (x_1, \ldots, x_n)$ is perpendicular to the geometry of the 1 norm, this is because each face of the hyper-octahedron can be described by there perpendicular vector which is our vector $\underline{X_p}$ (of some scalar of this vector). Hence when we rotate $\underline{a_p}$ to $\underline{X_p}$, $\underline{a_p}$ and its associated perpendicular hyperplane become some kind of one norm, with $\|\underline{R(a_p)}\|_1 \geq \underline{\Lambda}$ (where Λ is some constant). This operation is shown by a 2 dimensional example for one $\underline{a_p}$ described in figure 7.

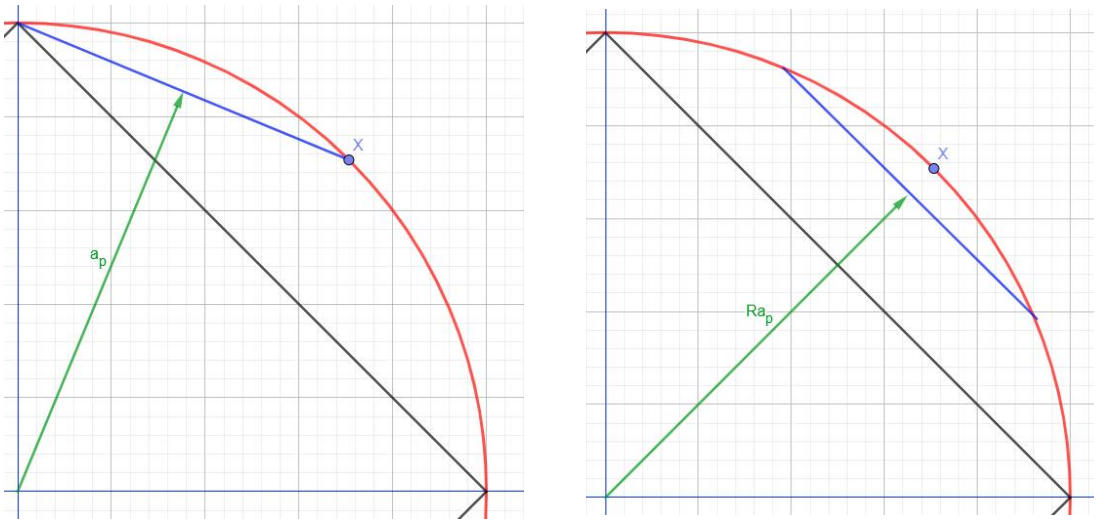


Figure 7: Rotation of $\underline{a_p}$ to $\underline{X}$

When rotating $\underline{a_p}$ to $\underline{X_p}$ it will also rotate the perpendicular hyperplane (as seen in figure 7). Therefore all the points above this hyperplane (these are the points part of the subset) will rotate with this rotation and therefore be above this perpendicular hyperplane at $R(a_p)$. This is the same now as the one norm and we can apply the proven algebraic 1-norm formulation to this subset with a tighter accuracy, i.e. $\| R(\underline{s_p}) \|_1 \geq \Lambda > 1$ with s_p being any point belonging to the subset associated to $\underline{a_p}$ (this inequality will be proven later). We can then apply 1-norm to this rotated subset for a tighter 1-norm lower bound.
Note: recall our rotation algorithm (Section 6) is limited as it has no control over the rotation of a shape about the point which it is rotated to. However in this circumstance this does not matter because the perpendicular hyperplane to $\underline{a_p}$ will always intersect the hypersphere of $\|\underline{w_j}\|_2 = 1$ with a smaller hypersphere (it is also 1 dimension smaller than the problem's dimensions at $n-1$) and therefore under rotation (to $\underline{X_p}$) all the points belonging to the subset s_p will still be above this new 1-norm; due to the infinite rotation symmetry of hyperspheres. Clearly for the 2 dimensional case this Λ is larger than 1 so therefore we have created a more precise 1-norm lower bound however its not clear how this formulation behaves in higher dimensions.

9.2 Does this method cover the whole hypersphere?

In short, no! For dimensions $n \geq 3$ then this formulation doesn't cover the whole hypersphere, however we can scale our center vector $\underline{a_p}$ by λ, so that the perpendicular hyperplane of $\underline{a_p}$ intersects the central points (points furthest from the every $\underline{a_p}$) for a given orthant. So what are the central points for a orthant? Lets consider the positive orthant (where all coordinates are positive $x_i \geq 0$, for all $i = 1, 2, \ldots, n$). Recall $\underline{a_p} = (\frac{1}{2\sqrt{n}}, \ldots, \frac{1}{2}(1 + \frac{1}{\sqrt{n}}), \ldots, \frac{1}{2\sqrt{n}})$, in each orthant there are n possible a_p, with the $\frac{1}{2}(1 + \frac{1}{\sqrt{n}})$ term being in every coordinate position. Note however the central points (concerned with a given $\underline{a_p}$) must have a non-zero entry in the same coordinate at which $\underline{a_p}$ has its $\frac{1}{2}(1 + \frac{1}{\sqrt{n}})$ term. This is because each $\underline{a_p}$ is the midpoint between our hypercube vertex and a basis vector e_i (which is what make the $\frac{1}{2}(1 + \frac{1}{\sqrt{n}})$ term in the i^{th} coordinate), it wouldn't make sense that this hyperplane perpendicular to $\underline{a_p}$ intersected a center point which had 0 component in the

i^{th} coordinate.
This is best explained by an 3 dimensional example: let $\underline{a_p} = (\frac{1}{2\sqrt{3}}, \frac{1}{2}(1+\frac{1}{\sqrt{3}}), \frac{1}{2\sqrt{3}})$ and central point $(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$, clearly these coordinates are too far apart. Figure 8 below illustrates this idea:

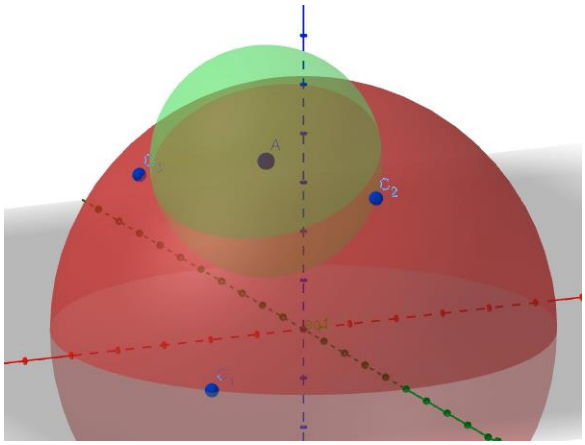


Figure 8: The 3 center point of the positive orthant in 3 dimensions

In 3 dimensions we see all the center points (in blue) for a given orthant. Where the spheres intersects is our perpendicular plane. As we can see from this figure it is clear that currently they don't intersect at the center points, however by scaling $\underline{a_p}$ by λ (with $\lambda < 1$) the perpendicular plane intersection will increase.
How do we determine a largest λ such that these constraints are as effective as possible and hyperplanes to $\underline{a_p}$ cover the whole hyper-sphere while keeping the hyperplane intersections as small as possible. Firstly what are the central points? In 3 dimensions the central points are $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)$ in any order and with each coordinate being positive or negative, in 4 dimension the central points are $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0)$ and $(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, 0)$, again these are both in any order with either each coordinate being positive or negative. So as the dimension increases there is an exponential increase the number of central points; the number of center points in a single orthant for a given dimension n is $\sum_{r=1}^{n-2} \frac{n!}{r!(n-r)!} = 2^n - n - 2$, the sum is correct because r represents the number of zeroes a center point can have (from r between 1 and $n-2$) and all the different positions it can take in the coordinate (proof idea is $(1+t)^n = \sum_{r=0}^{n} \frac{n!}{r!(n-r)!}\, t^n$ by the binomial theorem, and setting t=1, $\sum_{r=0}^{n} \frac{n!}{r!(n-r)!} = 2^n$).
The number of points isn't particularly important, because we just need to find which points are furthest away from $\underline{a_p}$. It is not clear which one is furthest away from $\underline{a_p}$; this is important to obtain the maximum λ such that this hyperplane covers all the points we need it to cover. Therefore the general equation of a center point is:

$$C_{\underline{a_p}} = \left(\pm \frac{1}{\sqrt{b}}, \pm \frac{1}{\sqrt{b}}, \ldots, 0 \right) \qquad b = (2, 3, \ldots, n-1) \tag{9.2}$$

Again this can be any order; as long as there is a non-zero term $(\frac{1}{\sqrt{b}})$ in the same position as $\underline{a_p}$'s $\frac{1}{2}(1+\frac{1}{\sqrt{n}})$ term (ignoring the sign of the coordinates). Any $C_{\underline{a_p}}$ has b non-zero entries, all of

which are $\pm\frac{1}{\sqrt{b}}$, with $b-1$ of the entries being permutable with no change to it's distance from $\underline{a_p}$, and 1 term being our fixed position term in relation to $\underline{a_p}$'s $\frac{1}{2}(1+\frac{1}{\sqrt{n}})$ term. Now as this is the perpendicular hyperplane at $\underline{a_p}$ (or $\lambda\underline{a_p}$) the center points furthest away will lie on this hyper plane (opposed to the other center points which will be above the hyperplane).

$$(C_{\underline{a_p}} - \lambda\underline{a_p}) \cdot \underline{a_p} = 0 \tag{9.3}$$

Rearranging this equation we get

$$\lambda = \frac{C_{\underline{a_p}} \cdot \underline{a_p}}{\underline{a_p} \cdot \underline{a_p}} \tag{9.4}$$

The denominator is just the 2 norm of $\underline{a_p}$ which is calculable and fixed for all $\underline{a_p}$:

$$\underline{a_p} \cdot \underline{a_p} = \frac{1+\frac{1}{\sqrt{n}}}{2} \tag{9.5}$$

Using equation 9.2 we are able to determine the scalar product of $C_{\underline{a_p}} \cdot \underline{a_p}$ for all center points:

$$C_{\underline{a_p}} \cdot \underline{a_p} = (b-1)\,\frac{1}{\sqrt{b}}\,\frac{1}{2\sqrt{n}} + \frac{1}{\sqrt{b}}\,\frac{1}{2}\left(1+\frac{1}{\sqrt{n}}\right)$$

Simplifying this leads to

$$C_{\underline{a_p}} \cdot \underline{a_p} = \frac{1}{2\sqrt{b}}\left(\frac{b-1}{\sqrt{n}} + (1+\frac{1}{\sqrt{n}})\right) \tag{9.6}$$

This expression is not fully simplified for convenience and keeping a common term with our $\underline{a_p}$ norm. Now substituting these equations back into equation 9.4:

$$\lambda = \frac{\frac{1}{2\sqrt{b}}\left(\frac{b-1}{\sqrt{n}} + (1+\frac{1}{\sqrt{n}})\right)}{\frac{1+\frac{1}{\sqrt{n}}}{2}}$$

And simplifying this leads to our final equation for λ:

$$\boxed{\lambda\,(b) = \frac{1}{\sqrt{b}} + \frac{b-1}{\sqrt{b}(\sqrt{n}+1)}} \tag{9.7}$$

Currently this doesn't tell us a value of b for which λ is minimum. n is fixed so we can just differentiate and solve the critical points,

$$\frac{d\lambda}{db} = \frac{-1}{2\,b^{3/2}} + \frac{1}{\sqrt{n}+1}\left(b^{-1/2} - \frac{b-1}{2}b^{-3/2}\right) \tag{9.8}$$

Setting $\frac{d\lambda}{db} = 0$ and rearranging:

$$\frac{1}{2\,b^{3/2}} = \frac{1}{\sqrt{n}+1}\left(b^{-1/2} - \frac{b-1}{2}b^{-3/2}\right)$$

$$\frac{1}{2} = (b - \frac{b-1}{2})\frac{1}{\sqrt{n}+1}$$

$$\frac{\sqrt{n}+1}{2} = \frac{b+1}{2}$$
$$b = \sqrt{n}$$

So $b = \sqrt{n}$ is a critical point, and it can be shown that this is a minimum point. So we have found our b for which is λ is minimum, however $\sqrt{n}$ is not always a integer where as b is always an integer. To keep λ as maximal as possible we can try the integers either side of $\sqrt{n}$ (if $\sqrt{n}$ is not an integer) and then take the minimum value of λ which we will denote λ^*:

$$\lambda^* = \min\big(\, \lambda(\, \lfloor\sqrt{n}\rfloor\,),\ \lambda(\, \lfloor\sqrt{n}\rfloor + 1)\,\big) \tag{9.9}$$

(Note: this equation will still holds even if $\sqrt{n}$ is an integer).

Analysing λ further for larger dimensions, consider its minimum value. Ignoring whether $\sqrt{n}$ is integer or not lets consider the minimum of $\lambda(b)$, so therefore

$$\lambda^* \geq \min(\,\lambda(b)\,) = \lambda(\sqrt{n}) = \frac{1}{\sqrt[4]{n}} + \frac{\sqrt{n}-1}{\sqrt[4]{n}(\sqrt{n}+1)}$$

Now this $\min(\,\lambda(b)\,)$ (and therefore λ^*) does converge to 0 as $n \to \infty$ however the convergence is really slow. For example $\lambda\,(5) = 0.924$, $\lambda\,(12) = 0.834$, $\lambda\,(50) = 0.659$, $\lambda\,(100) = 0.575$, (all to 3.s.f). Considering this problem is not likely to have dimensions above 30 this constant will still be close to one, however how will this effect the lower bound of our tighter 1-norm constraint.

9.3 Formulation analysis and Results

Using all the calculations and information from previous subsections we are able to create this formulation with proof that the formulation will cover the whole hypersphere (with the tightest possible constraints). Analysing the properties of the formulation which make this different from previous simpler formulations, as well as analysing the results and some of the formulation's drawbacks.

Previously we created an inequality relating the 2-norm and 1-norm (3.3) which holds in general for any point in n dimensions, however an exact transformation can be derived for a given point with set characteristics. This is useful tool for us because in this rotation we will be rotating $\underline{a_p}$ to its associated hypercube vertex $\underline{X_p}$ (for now we will ignore the fact $\underline{a_p}$ is scaled by λ^* for simplicity then rescale it later; because it is a scalar this will have no effect on the result). Therefore under rotation $\|\underline{a_p}\|_2 = \|\,R(\underline{a_p})\,\|_2$, this point is $R(\underline{a_p})$ is in line with a vertex of the hypercube $\underline{X_p}$ (see Figure 7). Currently we don't know the value of the lower bound of this 1-norm constraint (Λ), however given we know $\underline{a_p}$ and $\underline{X_p}$ we can work out Λ. To start with we begin with our 2-norm at $\underline{a_p}$:

$$\|\underline{a_p}\|_2 := A$$

Recall $\|\underline{a_p}\|_2 = \|\,R(\underline{a_p})\,\|_2$ because length is preserved under rotations. Now since $R(\underline{a_p})$ is rotated such $|y_1| = |y_2| = \ldots = |y_n|$, therefore we can say $R(\underline{a_p}) = (y, y, \ldots, y)$ (ignoring the sign of each coordinate as it does not effect the solution).

$$\sum_{p=1}^{n} y_i^2 = A^2$$

$$ny^2 = A \Rightarrow y = \frac{A}{\sqrt{n}}$$

$$\| R(\underline{a_p}) \|_1 = \sum_{p=1}^{n} |y_i| = ny = \sqrt{n}A$$

Therefore as all points $\underline{s_p}$ above the hyperplane of $\underline{a_p}$ satisfy $\|R(\underline{s_p})\|_2 \geq \|R(\underline{a_p})\|_2$, therefore $\|R(\underline{s_p})\|_1 \geq \|R(\underline{a_p})\|_1$.

$$\| R(\underline{q_p}) \|_1 \geq \sqrt{n}A$$

So given a vector $\underline{a_p}$ with $\|\underline{a_p}\|_2 = A \;\; (A < 1)$, once rotated $R(\underline{a_p})$ will have to satisfy $\|R(\underline{a_p})\|_1 \geq \sqrt{n}A$. This result is independent of the sign on each coordinate as norms are absolute and positive definite. However $\underline{a_p}$ is really $\lambda^* \underline{a_p}$ so the equation becomes:

$$\| R(\lambda^* \underline{a_p}) \|_1 \geq \sqrt{n}\, \lambda^*\, A \tag{9.10}$$

Previously we defined this lower bound as Λ, therefore we now have $\Lambda = \sqrt{n}\, \lambda^*\, A$, by the previous subsections we have a value for A and λ^*. So putting it all together to obtain:

$$\Lambda = \sqrt{n}\, \lambda^*\, A \geq \sqrt{n} \left(\frac{1}{\sqrt[4]{n}} + \frac{\sqrt{n} - 1}{\sqrt[4]{n}(\sqrt{n} + 1)} \right) \left(\sqrt{\frac{1 + \frac{1}{\sqrt{n}}}{2}} \right)$$

The $\geq$ is necessary as we are taking minimum of $\lambda(b)$ where as λ^* is defined slightly differently, the values are very close but not equal (important to maintain tight constraints). This equation simplifies very nicely:

$$\begin{aligned}
\Lambda &\geq \sqrt[4]{n} \left(1 + \frac{\sqrt{n} - 1}{\sqrt{n} + 1} \right) \left(\sqrt{\frac{1 + \frac{1}{\sqrt{n}}}{2}} \right) \\
&= \sqrt[4]{n} \left(\frac{2\sqrt{n}}{\sqrt{n} + 1} \right) \left(\sqrt{\frac{\sqrt{n} + 1}{2\sqrt{n}}} \right) \\
&= \sqrt[4]{n} \left(\frac{2\sqrt{n}}{\sqrt{n} + 1} \right) \left(\sqrt{\frac{\sqrt{n} + 1}{2}} \right) \frac{1}{\sqrt[4]{n}} \\
&= \left(\sqrt{2n} \right) \left(\frac{\sqrt{\sqrt{n} + 1}}{\sqrt{n} + 1} \right)
\end{aligned}$$

$$\Lambda \geq \left(\sqrt{\frac{2n}{\sqrt{n} + 1}} \right) > 1 \qquad , \forall\, n > 1 \tag{9.11}$$

Really $\Lambda \simeq \left(\sqrt{\frac{2n}{\sqrt{n}+1}} \right)$ but because we use the minimum of $\lambda(b)$ instead of λ^* this will be either equal or slightly less than Λ. Never the less we can see from this that $\Lambda > 1$, so therefore is always giving a tighter constraint than the standard 1-norm constraint: Therefore if we take any point $\underline{s_p}$ belonging to the subset associated with the point $\underline{a_p}$

$$\| R(\underline{s_p}) \|_1 \geq \Lambda \geq \left(\sqrt{\frac{2n}{\sqrt{n} + 1}} \right) > 1 \tag{9.12}$$

Now putting all the concepts and calculations together we can create an algorithm for this formulation.
Initially we pre-calculate all the center points of our subsets $\underline{a_p}$; we only have to calculate the upper-hemisphere points as all $\underline{a_p}$ shares a rotation symmetry with their direct reflection subset on the lower hemisphere, this means when $\underline{a_p}$ is rotated then its reflection is also rotated correctly. Here our set rotation index is p, with $p \in \{1, \dots, n\, 2^{n-1}\}$ because each rotation is rotating $\underline{a_p}$ to its nearest hypercube vertex $\underline{X_p}$. With each rotation matrix being denoted as R_p (the rotation matrix is determined by our rotation algorithm 19).

$$
\begin{array}{rcllll}
R_p(\underline{w_j}) & = & \underline{w'_{jp}} & & j \in \{1,\dots k\} & p \in \{1,\dots C\} \quad (1)\\
(\underline{w'_{jp}}_{\,q}) & = & \underline{\rho_{jp}}_{\,q} - \underline{\eta_{jp}}_{\,q} & q \in \{1,\dots n\} \quad j \in \{1,\dots k\} & p \in \{1,\dots C\} \quad (2)\\
\underline{\rho_{jp}}_{\,q} & \leq & v_{jqp} & q \in \{1,\dots n\} \quad j \in \{1,\dots k\} & p \in \{1,\dots C\} \quad (3)\\
\underline{\eta_{jp}}_{\,q} & \leq & 1 - v_{jqp} & q \in \{1,\dots n\} \quad j \in \{1,\dots k\} & p \in \{1,\dots C\} \quad (4)\\
 & & \underline{\rho_{jp}},\ \underline{\eta_{jp}} \in (\mathbb{R}^+)^n & & j \in \{1,\dots k\} & p \in \{1,\dots C\} \quad (5)\\
 & & v_{jqp} \in \{0,1\} & q \in \{1,\dots n\} \quad j \in \{1,\dots k\} & p \in \{1,\dots C\} \quad (6)\\
 & & z_{jp} \in \{0,1\} & & j \in \{1,\dots k\} & p \in \{1,\dots C\} \quad (7)\\
 & & \sum_{p=1}^{C} z_{jp} \geq 1 & & j \in \{1,\dots k\} & \quad (8)\\
\sum_{q=1}^{n} (\underline{\rho_{jp}}_{\,q} + \underline{\eta_{jp}}_{\,q}) & \geq & \Lambda - (\Lambda - 1)\,(1 - z_{jp}) & & j \in \{1,\dots k\} & p \in \{1,\dots C\} \quad (9)
\end{array}
\tag{9.13}
$$

A lot of this algorithm is built upon the algebraic 1-norm formulation. Line 1 is the rotation R_p, lines 2-6 are the same as a the algebraic 1 norm with an additional index p for each rotation.
Lines 7 and 8 are introducing a new binary variable z_{jp} which we use to ensure that $\underline{w_j}$ belongs to atleast 1 subset. In previous algorithms/formulations we have used similar binary variables constructions, e.g. the Infinity norm algorithm 4.1 uses a binary variable dictating $\underline{w_j}$ lies above to exactly 1 side of the hypercube. This is the same idea but the subsets overlap more than with the Inf-norm and are therefore not distinct, hence ≥ 1 allows for $\underline{w_j}$ to belong to multiple subsets of the hypersphere in the event $\underline{w_j}$ is a point in the region of 2 or more subsets. Line 9 is the standard summation formula we use for the 1 norm, the RHS uses λ^* as the lower bound with the binary variable working as a way to "turn-off" the constraint making it a standard 1-norm constraint with a lower bound of 1, (which any $R_p(\underline{w_j})$ will satisfy because it is on the hypersphere).

Table 14: Computational results of rotation of this algorithm 9.13

Size	m	k	n	Name	Nodes	Time	Obj value	LB	Gap
Small	10	2	2	Test1	1595	0.28	0.268	0.268	0%
	16	3	3	Test2	1,449,116	199.7	0.0712	0.0032	95.6%
	20	3	3	Test3	1,741,285	300	1.86	0	100%
	24	3	2	Test4	984,611	63.7	3.98	3.98	0%
	30	2	3	Test5	2,798,868	300	2.53	2.38	6.2%
Medium	500	2	2	Test6	1,735,970	1200	111.1	9.54	91.4%
	500	5	4	Test7	360,862	1200	323.7	0	100%
	500	8	4	Test8	183,982	1200	138.8	0	100%
	500	5	6	Test9	29,572	1200	475.9	0	100%
	500	8	6	Test10	23,282	1200	—	0	—

There are no large instances included and that is because: the results either never reached a sensible value, didn't reach an objective value, and in one case wouldn't start. Now even from these initial results it was clear this algorithm wasn't going to work well on large instances, however the fact it didn't even show reasonable results can also be accounted to the test system being RAM limited (8gb). In some cases this formulation performed worse than the original algorithm which can largely be put down to the exponential factor accounted to the dimensional aspect.
The results don't tell us a lot about this concept or idea but do give an indication about other important considerations about how Gurobi evaluates formulations. Firstly it was clear that this formulation has a large number of binary variables compared to previous formulations so we would expect the number of nodes to be much higher as Gurobi branches over different binary variables combinations. This wasn't the case, in fact the number of nodes was much lower for many instances. This could be due to Gurobi struggling with the more advanced calculations of this formulation, however the other possible cause for the low node counts is that Gurobi prefers to branch over its own regions (polyhedral envelopes) than over binary variables. From a software point of view this would make more sense to branch over regions of the feasible region rather than binary variables as Gurobi does not know that in this situation that the binary variables are directly linked to restricting the feasible region. More testing on this idea (with another similar formulation) would give a better indication of how Gurobi is evaluating these binary variables.

It's important to realise that this formulation 'mathematically' is slightly better than previous formulations, with the constraints being tighter and the hypersphere is divided up into subregions. The formulation is 3 algorithm combined into 1: sub-sectioning then rotations, and then apply algebraic 1-norm. Now the rotation algorithm won't add much to the complexity as the rotations are pre-calculated, but the sub-sectioning and algebraic 1-norm are complex and will be large computations for Gurobi to evaluate.
This problem with this formulation is the approach, the idea is "too mathematical" and neglects computational aspects which leads to its bad results. The idea might be a good analytical approach for a human to subset the sections of the hypersphere into these regions but it neglects the way spacial branch and bound solvers work. Gurobi will make its own subregions which it will branch over and these overlapping regions may be getting in the way of its usual approach. An efficient use

of binary variables and reducing the number of them is clearly essential for an optimal formulation, as even in small dimensions (when the exponential aspect is not as significant) the problem doesn't perform as well as the geometric volume (linear feasible region) suggests it should.
This concept of sectioning a sphere into smaller regions but with more effective constraints has got lots of potential even in large dimension problems. However this particular formulation has some obvious drawbacks (which were not initially realise until doing the calculations). The biggest drawback is the dimension scale-ability, with each $\underline{w_j}$ hypersphere having $n2^{n-1}$ sets of constraints. This issue with this formulation is that each subset of $\underline{a_p}$ has only 1 rotation symmetry (i.e. Rotating $\underline{a_p}$ to the correct position only rotate 1 other $\underline{a_{p'}}$ to the correct position) so therefore all the subset rotations in the upper hemisphere are independent of one another, and therefore each subset must be rotated separately.
So why has this idea still got potential even after bad results? This particular idea is not good, and originally the formulation was over-simplified for how this would work in higher dimensions which resulted in a exponential scaling. Never the less the concept of splitting the hypersphere into more than the natural orthants of n-dimensional space makes intuitive sense. We can create more effective and tighter boundary in these sections, which in turn will create a smaller difference between the linear feasible region and the actual feasible region. Additionally creating subsection of the hypersphere give Gurobi the potential to prune off entire sections, although this in practice is unlikely till very late in the calculations due to this problem being multiple clusters (i.e. $k > 1$).

9.4 Ideas for an improved sectioning formulation

Using the calculations and ground work established in this chapter other sectioning formulations can be constructed with better success. There is another formulation I have been working on which builds upon the success the Inf-norm + Alg-1-norm formulation by adding additional Alg-1-norms and extending them until they are limiting. The Inf-norm + Alg-1-norm is not limiting as you could slightly increase the lower bounds of both these constraints and still cover the whole hypersphere. Additionally using this idea with additional 1-norms along each of the axis intercepts (similar to section 7) we can further extend each of these constraints creating an larger linear feasible region. This method would again require binary variables to separate the sections however the key component of this formulation is there would be no rotations and as a result the symmetries would be the same for all orthants, resulting in only one formulation having to be constructed for 1 orthant. Therefore the number of subsections we would have to calculate for is $n + 1 + n(n-1)/2$; the n is for the number of sections created by the Inf-norm, 1 is for the standard Alg-1-norm, $n(n-1)/2$ (the number of axis-intercepts in n dimensions) is for the 'modified' Alg-1-norm (we can create a 'modified' 1 norm on these intercepts without rotations but the details aren't needed right now). However the prior calculations for this formulation are actually much more complicated than the last formulation even though the concept is arguably easier. Although the once set up the formulation is much simpler. This is because previously the Inf-norm, 1-norm and 'modified' 1-norm have have lower bounds of $\frac{1}{\sqrt{n}}$, 1 and 1 (respectively), however now there lower bounds are α, β, and γ (respectively). These new lower bounds are all larger (or equal) than what they were previously but it's not clear what each of the values are? What values would yield the best results is another optimisation problem in itself. It is not as simple as maximising the internal volume of the linear region as different formulations act differently with one another. Evaluating these variables (α, β, and γ) will come down to the effectiveness of each formulation individually, for example the main drawback is increasing the lower bounds of an effective constraint reduces its subsection coverage and therefore its effectiveness so finding the right balance is key.

10 Conclusion

This paper was focused on creating new formulations to improve the algorithms ability to solve any instance on modern branch and bound solvers. Now we did limit ourselves to one branch and bound solver - 'Gurobi'. However we did not create any formulations (or internal code) that would be particularly advantageous to improving the results for Gurobi and not other solvers. This approach was to make the formulations more general to all branch and bound solvers rather than optimising it specifically for Gurobi. Therefore it would be very possible to improve upon the ideas and formulation by tweaking more niche setting within the branch and bound solvers.
To start with we created our Infinity-norm (section 4) and two 1-norm formulations (section 5) which acted as our basic formulations throughout the paper. Although both of these norms are the only ones which can be mathematically formulated linearly, there formulations are very different with both of them using a large number of binary variables very differently. There is merit to both of these formulations with higher dimensions being slightly favoured towards Inf-norm formulation. We then built upon the algebraic 1-norm (due to its geometric properties and previous performance) to create multiple rotations of this formulation to combine together to get a large formulation. These rotation fell short in the results due to the amount equations while the linear feasible region was very similar with a large amount of overlap between the linear feasible regions (of each rotated 1-norm), however on longer time frames the formulation performed very well. Lastly we create the idea of sectioning the hypersphere into smaller regions, this idea was the most complicated and ambitious but falls short to a number of compromises we had to make; in order for the formulation covers the whole hypersphere. This concept of sectioning the hypersphere into smaller regions has still got potential due to the success of the Inf-norm in higher dimensions; which is essentially just sectioning areas of the hypersphere using binary variables.

Throughout the testing there are 3 parameters: number of points m, number of clusters k, and number of dimensions n. The difference in these parameters over instances have given us indication of where and why a formulation maybe particularly weak. Importantly though with all these different formulations we can now have a good idea of how the parameters effect the algorithms and complexity. Firstly the m doesn't vary hugely between instances which was by design to group them into categories, however from external testing the number of points behaves as we would expect with no sudden changes in computation speed. Number of clusters k is not so simple, doubling the number clusters would result in more than double the complexity and really this parameter's effects are unavoidable as there is no way to reduce or mitigate the effects, additionally Gurobi branched a lot more when the ratio of dimensions to clusters was higher. Gurobi is likely branching over its own possible sets of feasible regions (for all the clusters), from a software point of view this would make sense and we would expect other branch and bound solvers to work in a similar way. Hence suggesting restricting Gurobi with very complex formulations is slowing this process, however more testing on larger time frames would need to be done in order to concretely conclude this. Lastly the number of dimensions n is the most important parameter, this is because all our formulation's complexity will depend on n and reducing algorithms dependence on n will vastly improve the results. The increase in dimension increases the complexity of the problem exponentially however there are certain formulations we can create which are not exponential and as a result scale in a beneficial way (because exponentially scaling formulations performance will fall off eventually).

With only 2 norms (1-norm and Inf-norm) which can be expressed linearly the different types of formulations will depend heavily on how we formulate these norms. For the Inf-norm we only created 1 formulation (4.1) but it was based on using binary variables to essentially dictate where

the point is in relation to the internal hypercube. This formulation seemed weak initially however performs very well in high dimensions, and because of how the formulation works this gives sufficient evidence to support the idea of sectioning the hypersphere into smaller regions. The 1-norm had two formulations: Algebraic (5.1) and Brute force (5.2). Both of these are vastly different with the brute force creating all possible $\pm$ combinations for the 1 norms, which adds a large amount of equations and variables. Yet despite this the brute force formulation out performed the Algebraic version on the instances we checked; although this difference in performance difference does drop off as the dimension increases. Additionally chapter 8 we analysed the difference between Alg-1-norm and Inf-norm which suggested the Alg-1-norm is a better formulation however these result were not clear cut and further testing would be needed to conclusively claim this. The key part to take from this is that the computational results do not directly improve from increasing the geometric volume (linear feasible region) and instead more consideration should be put into making the core formulation as efficient as possible.
Combination of formulations is an idea which we kept exploring and a lot more combinations were tried and not recorded. Firstly is the geometric aspect is a considerable factor in this situation, formulations covering similar regions did not yield good results. When combining formulations its important to make sure their linear feasible regions have as little intersection with one another. Additionally combinations of a lot formulations or complex formulations did not do well on the default time frame even if they had tighter constraints, likely due to the amount of constraint equations Gurobi would be dealing with. However in some cases on longer time frames these formulations converged better in some cases suggesting that they have slower convergence than simpler formulations but ultimately wouldn't get stuck at a "dead-end". Additionally the gap/lower-bound was better on these types of formulations (even on the standard time frame) creating lower bounds for even large dimension problems previously not seen before (although these lower bounds were extremely small e.g. $1e-14$). It would be interesting to see if this trend continued on longer time periods and on more powerful systems, as this could change our approach to formulation creation to create more constraints in order to increase the lower bound. On these large scale/combination formulations the Inf-norm consistently worked well as a catalyst accelerating the progress made on the computations. Due to the Inf-norms different style of formulation it was much more successful when working with an additional formulation and as a result was almost necessary for large formulations in order to speed up their convergence.

In conclusion the ideas discussed in this paper along with the results obtained give an good estimate of how these different formulations would on a larger system (or super computer). More testing on larger systems with larger instances should be explored in order to conclude these result and analysis predictions are correct. However from the testing results obtained in this paper we can extrapolate the finding and analysis to give good predictions of how modern spatial branch and bound solvers would work on larger systems/instances. Firstly the size of the instance will effect the best optimal formulation for solving the problem, with small instances working best with simple formulations (e.g. Inf-norm + Alg-1-norm) while on larger instances simple formulations would reach "dead ends" making little to no progress after a certain time period. Large instances required more complex formulations to obtain a reasonable solutions, however these formulations tend to have very slow convergence, which is why large formulations should be paired with the Inf-norm as it acts as a catalyst, speeding up convergence. Lastly the idea of sectioning a hypersphere into subsections has not proven successful in this paper but its potential for success is given by the Inf-norm which is splitting the hypersphere into sections based on faceit's of the hypercube. If there is an optimal algorithm it will most likely build upon the Inf-norm formulation and sectioning the hypersphere into more effective subregions.

10.1 Further ideas

There are a few ideas which would be worth exploring to build upon the work done in this paper:

- Firstly creating a more efficient formulation for sectioning the hypersphere, building upon the final paragraph of section 9, which discusses an idea for a new formulation for sectioning the hypersphere using combinations of the Inf-norm and Alg-1-norm with more effective lower bounds but without exponential scaling.
- Secondly the idea of remodelling the problem as a large multi-dimensional problem to perform more effective analysis. For example if $k = 3$ and $n = 4$, this would become a 12 dimensional problem with the first 4 coordinates corresponding to $\underline{w_1}$, the next 4 corresponding to $\underline{w_2}$, and so on. This idea would involve reformulating the majority of the algorithm, but the benefit is that we can get rid of k and work purely with dimensions $k * n$ which we can be controlled with algorithms. Additionally k seems to add more to the complexity of the problem than n so there is evidence to support that this reformulation could potentially reduce the complexity of the problem.
- Lastly building upon section 8 it was noted that the Alg-1-norm performed better when it was rotated off the standard basis vectors, probably because Gurobi would start at the basis vectors when creating its own polyhedral envelopes in an attempt to create its own linear regions around the feasible region of the hypersphere. Therefore rotating the formulations away from these basis vectors might result in a slight increase in performance.

References

[1] E. Amaldi, S. Coniglio. *A distance-based point-reassignment heuristic for the k-hyperplane clustering problem.* [*Description*]. Page 2, Section 2

[2] Alberto Costa, Leo Liberti *Relaxations of multilinear convex envelopes.* `https://www.lix.polytechnique.fr/~liberti/sea12b.pdf`

[3] O. Mangasarian. *Arbitrary-norm separating plane.* [*Operations Research Letters*]. 24 (1–2) (1999) 15–23.

[4] P. Bradely, O. Mangasarian, *Journal of Global Optimization.* [*k-plane clustering*]. 16 (2000) 23–32

[5] Cristian F. Pasluosta, Prerna Dua, and Walter J. Lukiw. *Nearest Hyperplane Distance Neighbor Clustering algorithm Applied to Gene Co-Expression Analysis in Alzheimer's Disease.* [*Medical uses of k-HC algorithm for analysing DNA*]. https://europepmc.org/article/PMC/3703613

[6] He, Hongmei & Qin, Zengchang. *A k-hyperplane-based neural network for non-linear regression* [*(2010) Proceedings of the 9th IEEE International Conference on Cognitive Informatics*].

[7] Zhelezov, Ognyan *N-dimensional Rotation Matrix Generation Algorithm.*

`https://www.researchgate.net/publication/323995682_N-dimensional_Rotation_Matrix_Generation_Algorithm`

[8] Author. *Title of paper.* [*Description*]. Specific section

[9] Knuth: Computers and Typesetting.
`http://www-cs-faculty.stanford.edu/~uno/abcde.html`