Optimizing the T9 Keyboard

Selena Hu, Kyle Sayers, Yongqi Zhang

May 11, 2023

Abstract

In this paper, we attempt to optimize the layout of a T9 keyboard. We break the problem into two parts, a k-section problem to assign letters to blocks, and a quadratic assignment problem to optimize the positions of those keys on a 3x3 layout. We provide an improvement on existing algorithms, a (novel?) proof of generalization to higher dimension k-section problems, and simulation results of the new layout and an old layout to show that the new layout can help users save time when typing. Our implementation code can be found at `https://github.com/kylesayrs/keyboard_optimization_190`.

1 Introduction

The traditional design of the T9 keyboard is shown in Figure 1. If a user of this keyboard wants to type two letters 'e' and 'i' sequentially, the user will need to press block 3 two times, then press block 4 one time. However, if the user wants to type 'e' followed by 'd', then the user will need to press block 3 two times, wait for one second, and then press block 3 one time. This added one second delay is necessary to avoid letter ambiguity, but results in much slower typing. These "letter collisions" will be the subject of our group optimization task, while the distance between letters on different keys will be the subject of our position optimization task.

For our project, we propose a new method for rearranging letters in each block based on 1) how frequently they are used, and 2) how frequently they appear next to another letter. We will show that our method improves typing efficiency and reduces letter collisions.

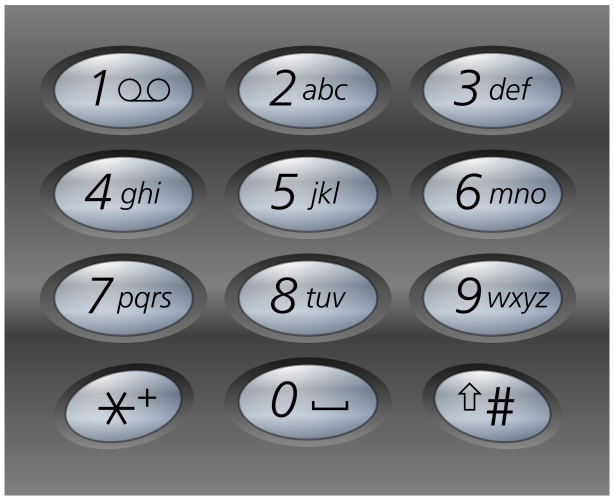


Figure 1: Traditional T9 keyboard layout [1]

2 Problem Statement

Our keyboard includes eight blocks for letters and one block for "Space". The 26 letters in the English alphabet are distributed across those eight letter blocks. Each block corresponds to either 3 or 4 letters.

We optimize the keyboard towards the following goals:

1. Avoid letter collisions: letter pairs that are frequently adjacent in English texts should not be assigned to the same block.
2. Prioritize letters most frequently used: letters most frequently used in English texts should be at the top of each block for the users to reach them easily.
3. Optimize block arrangement: if the letters on two blocks are used together frequently, then the two blocks should be close to each other.

3 Data Processing

The data we used is Financial Tweets data downloaded from Kaggle [2]. It includes more than 26000 Tweets from Twitter users about the stock markets. This data set is a good indicator of people's habit of using their phone's keyboard because 80% of the twitter users use twitter on mobile devices [3]. The size of the data set is also large enough to provide us with reliable data.
We cleaned the data from all punctuation, numbers, URLs and other characters. The data we eventually use only has lower case letters and spaces in it. We recorded the time each pair of letters appear next to each other and matched this index with the pair. We do not differentiate between the orders of them appearing. For example, the pair 'ij' will be considered the same as 'ji'. If there is a space between two letters, they are not considered as appearing together.
Then, from all 26 letters, we have $\binom{26}{2} = 325$ distinct pairs of letters and their corresponding index, which we call their bigram frequency.

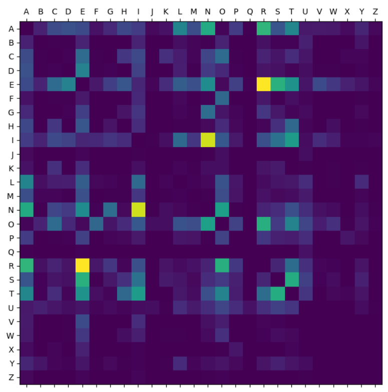


Figure 2: Bigram weights between letters

4 Method

4.1 K-Section Problem

4.1.1 Relaxed Convex Program

Our approach to finding the optimal groupings of letters onto keys is to first find a relaxed solution to an SDP for the k-section problem, project the relaxed partition guesses onto a set of k points, then perform a greedy swapping procedure to ensure each partition contains a (roughly) equal number of elements. This procedure is largely inspired by Freize and Jerrum[4], but we improve on their work by integrating modern clustering techniques and by providing proofs and implementations to generalize the algorithm to higher dimensional ks.

Given n points and $k = 2$, consider the following integer program:

$$\begin{aligned} \mathrm{IP_B}: \quad & \text{maximize } \frac{1}{2}\sum_{i<j} w_{ij}\left(1 - v_i \cdot v_j\right) \\ & \text{subject to } v_i \in \{-1, 1\}, \quad \forall i \leq n \end{aligned} \tag{1}$$

The optimal solution for this program will choose the sign of v_i and v_j such that if w_{ij} is large, v_i and v_j will have opposite sign, since

$$\begin{aligned} v_i \cdot v_j &= +1 \text{ if } v_i = v_j \\ v_i \cdot v_j &= -1 \text{ if } v_i \neq v_j \end{aligned} \tag{2}$$

Two vectors having the same sign can therefore be interpreted as being in the same group, while vectors with different signs are in separate groups.

In the next step, we generalize this concept to k groups and define $v_i \cdot v_j$ to be the ij^{th} entry of the gram matrix Y, thus converting the integer program into a relaxed, convex problem.

$$\begin{aligned} \mathrm{CP}: \quad & \text{maximize } \frac{k-1}{k}\sum_{i<j} w_{ij}\left(1 - Y_{ij}\right) \\ & \text{subject to } Y_{ii} = 1, \ \forall j \leq n \\ & Y \succcurlyeq 0 \end{aligned} \tag{3}$$

Note that the new constraints on Y are equivalent to the previous constraints on v_i and describe an ellipsoid shape. A more rigorous proof is provided by Felzenszwalb et al[5].

Finally, we add an extra constraint to limit the number of points that can be assigned to any given partition. Setting this limit to be equal to $\frac{n}{k}$ has the effect of promoting equally sized partitions in the final result.

$$
\begin{aligned}
\text{CP}: \quad & \text{maximize } \frac{k-1}{k}\sum_{i<j} w_{ij}\,(1-Y_{ij}) \\
& \text{subject to } Y_{ii}=1,\ \forall j \le n \\
& Y \succcurlyeq 0 \\
& \sum_{i<j} Y_{ij} < \frac{-n}{2}
\end{aligned} \tag{4}
$$

This version of this constraint for $k = 2$ was provided by Freize and Jerrum[4], however we provide an alternative proof that generalizes to higher k. This proof can be found in the appendix (figure 8).

Once an optimal gram matrix describing the dot products between vectors Y is found, we can perform the following gram matrix decomposition to reconstruct a set of points representing vectors for each letter.

$$
\begin{aligned}
& Y = U^T \Sigma U \\
& Y \approx X^T X \text{ for } X = \sqrt{\Sigma} U
\end{aligned} \tag{5}
$$

In this problem only the first $k-1$ rows of $\sqrt{\Sigma}U$ are useful, so the final row is discarded.

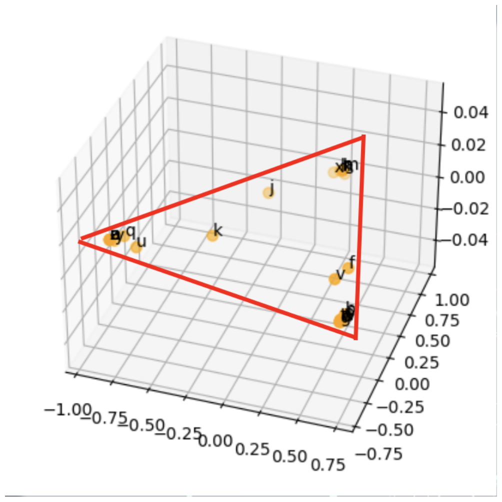


Figure 3: Visual of vector reconstructions constrained by a $k = 3$ simplex

4.1.2 Discretization via Clustering

Once vectors representing each letter are reconstructed, they need to be assigned to discrete partitions. This poses a challenge for those points which are not positioned directly at the vertex of the simplex (which in a practical setting includes all points). The original authors propose a scheme of projecting points to a set of randomly selected guide points, however this method provides inconsistent partitions, especially for higher dimensions of k. We instead assign points to partitions via kmeans clustering which leads to much higher quality results. We provide results comparing these two methods to random partitions as well as a method of partitioning using successive bisection of the letters.

We evaluate score as the sum of cuts across partitions and measure in terms of thousands (k).

Method (1000 iterations)	Best (k)	Mean (k)	Std (k)
Random partitions	2217	2083	60
Random point projection	2284	2233	32
KMeans clustering	**2285**	**2284**	**5**
Bisection w/ random point projection	2261		

Future work may involve determining an algorithm for gram matrix decomposition that preserves consistent vertices of the simplex so that points can be projected directly onto those vertices.

4.1.3 Greedy Swapping

Once a partition is assigned from the relaxed convex program, the number of elements in each partition much be balanced in order to be considered a valid k-section. We generalize Freize and Jerrum's[4] swapping method to k partitions as follows:

1. For each potential swap, compute the change in score that would result from that swap
2. Order swaps from highest score change to lowest score change
    (a) Select the swap with the highest score change
    (b) Check if the swap helps to balance the partitions (moves an element from a large partition to a smaller partition)
    (c) If the swap is helpful, perform the swap
    (d) Otherwise select the next highest scoring swap
3. After a swap is performed, repeat steps 1-2 until all partitions have equal size within a margin of one element.

4.1.4 Grouping Results

Using the above method we obtained the following optimized partition. This partition achieves a score of 2285k which is comparable to the 2304k upper bound[1] for these weights.

1. 'e', 'u', 'z'
2. 'c', 'g', 'w'
3. 's', 'm', 'f', 'v'
4. 'a', 'o', 'q'
5. 't', 'd', 'k'
6. 'i', 'y', 'x', 'j'
7. 'n', 'p', 'b'
8. 'r', 'l', 'h'

[1]An upper bound is computed by summing the highest m weights for each letter where m is the maximum number of weights that could be cut in an equal partition of the letters.

4.2 Prioritize Letters Most Frequently Used

Within each group, we put the letters that are more frequently used at the top, so a user has to press the buttons fewer times to type those letters.

4.3 Optimize Block Arrangement

After assigning the letters to each block, we want to put the block in optimized positions. More specifically, two blocks with higher inter-block weights should be assigned closer to each other. The purpose of this is to reduce the distance the user's thumb need to move when typing.
There are two factors affecting the positioning of blocks: the inter-block weight W and the distance matrix D. We define the inter-block weight W_{uv} as the sum of the weight between all pair of letters i, j, subject to $i \in u, j \in v$.

$$W = \begin{bmatrix} 0 & 34835 & 64063 & 27098 & 56596 & 17412 & 49490 & 81280 \\ 34835 & 0 & 8175 & 42226 & 14763 & 22591 & 27596 & 27092 \\ 64063 & 8175 & 0 & 63607 & 37088 & 39305 & 16019 & 20881 \\ 27098 & 42226 & 63607 & 0 & 59993 & 30964 & 76719 & 103049 \\ 56596 & 14763 & 37088 & 59993 & 0 & 38407 & 24241 & 42459 \\ 17412 & 22591 & 39305 & 30964 & 38407 & 0 & 56439 & 39507 \\ 49490 & 27596 & 16019 & 76719 & 24241 & 56439 & 0 & 20761 \\ 81280 & 27092 & 20881 & 103049 & 42459 & 39507 & 20761 & 0 \end{bmatrix}$$

The distance matrix D represents the physical distance between any of the two blocks. For our design, we put the letters on the first eight blocks of the T9 keyboard, while leaving the last one as the Space block. We assume the distance between two adjacent block to be 1.

$$D = \begin{bmatrix} 0.0000 & 1.0000 & 2.0000 & 1.0000 & 1.4142 & 2.2361 & 2.0000 & 2.2361 \\ 1.0000 & 0.0000 & 1.0000 & 1.4142 & 1.0000 & 1.4142 & 2.2361 & 2.0000 \\ 2.0000 & 1.0000 & 0.0000 & 2.2361 & 1.4142 & 1.0000 & 2.8284 & 2.2361 \\ 1.0000 & 1.4142 & 2.2361 & 0.0000 & 1.0000 & 2.0000 & 1.0000 & 1.4142 \\ 1.4142 & 1.0000 & 1.4142 & 1.0000 & 0.0000 & 1.0000 & 1.4142 & 1.0000 \\ 2.2361 & 1.4142 & 1.0000 & 2.0000 & 1.0000 & 0.0000 & 2.2361 & 1.4142 \\ 2.0000 & 2.2361 & 2.8284 & 1.0000 & 1.4142 & 2.2361 & 0.0000 & 1.0000 \\ 2.2361 & 2.0000 & 2.2361 & 1.4142 & 1.0000 & 1.4142 & 1.0000 & 0.0000 \end{bmatrix}$$

We model this problem as a Quadratic Assignment Problem:

$$\begin{array}{ll} \text{minimize} & \sum_{u,v\in[n]} W_{uv} D_{\pi(u)\pi(v)} \\ \text{subject to} & \pi \in \Pi \end{array} \tag{6}$$

where Π is the set of permutation functions of the set $\{1, 2, 3, 4, 5, 6, 7, 8\}$ [6].
We used brute force to calculate $\sum_{u,v\in[n]} W_{uv} D_{\pi(u)\pi(v)}$ for all possible π. The permutation that minimizes this problem is $\{0, 2, 6, 4, 3, 7, 5, 1\}$. The minimized sum is 3230525.41.

5 Result

We show the final layout for our T9 keyboard 4. We add a key for Space at the [3,3] position of the layout.
We did a simulation of a user typing on the old T9 keyboard and our new one. We recorded the total time that the user takes to type three testing texts: the original financial tweets dataset, one

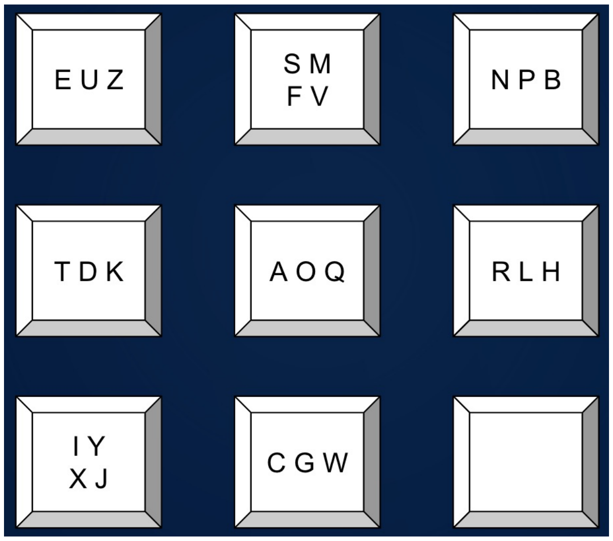
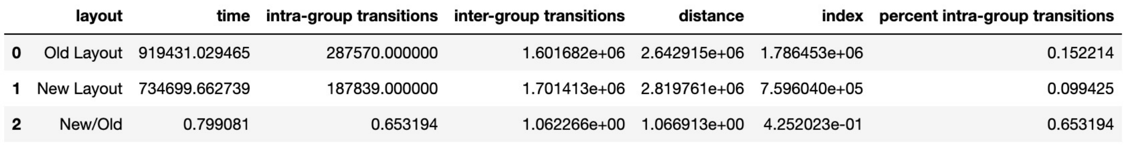


Figure 4: Final T9 Keyboard Design



	layout	time	intra-group transitions	inter-group transitions	distance	index	percent intra-group transitions
0	Old Layout	919431.029465	287570.000000	1.601682e+06	2.642915e+06	1.786453e+06	0.152214
1	New Layout	734699.662739	187839.000000	1.701413e+06	2.819761e+06	7.596040e+05	0.099425
2	New/Old	0.799081	0.653194	1.062266e+00	1.066913e+00	4.252023e-01	0.653194

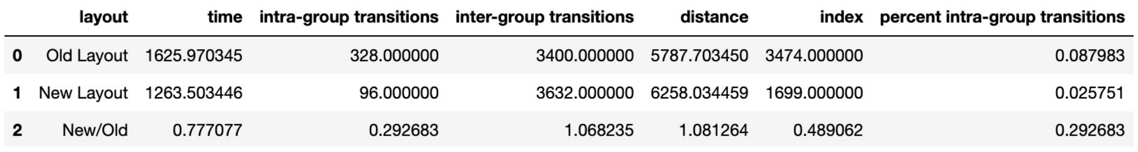
Figure 5: Simulation Results on Financial Tweets

email between one of the authors and her advisor, King Lear by William Shakespeare. The rules for the simulations are as the following:

1. If two contiguous letters are on the same block, the user need to wait for one second before typing the second letter.
2. To get the n^{th} letter on a block will take 0.1×n seconds.
3. The time for the user to switch to another block is Distance×0.1 (where the keys are distance 1 apart horizontally and vertically)

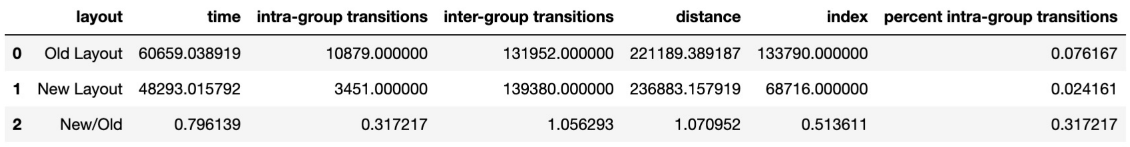
We also evaluated the results on three aspects each corresponding to one of our goals:

1. Number of the intra-group transitions. If this number decreases, then it means we reduced the number of letter collisions.
2. Number of total presses on each key. We evaluate this by calculating the sum of the group index for all letters in the testing text. For example, for the word 'math', it's sum of group



	layout	time	intra-group transitions	inter-group transitions	distance	index	percent intra-group transitions
0	Old Layout	1625.970345	328.000000	3400.000000	5787.703450	3474.000000	0.087983
1	New Layout	1263.503446	96.000000	3632.000000	6258.034459	1699.000000	0.025751
2	New/Old	0.777077	0.292683	1.068235	1.081264	0.489062	0.292683

Figure 6: Simulation Results on One Email



	layout	time	intra-group transitions	inter-group transitions	distance	index	percent intra-group transitions
0	Old Layout	60659.038919	10879.000000	131952.000000	221189.389187	133790.000000	0.076167
1	New Layout	48293.015792	3451.000000	139380.000000	236883.157919	68716.000000	0.024161
2	New/Old	0.796139	0.317217	1.056293	1.070952	0.513611	0.317217

Figure 7: Simulation Results on King Lear

index will be $2 + 1 + 1 + 3 = 7$ (because 'm' is the second letter in its group, 'a' is the first letter in its group...) on the new keyboard layout. If this number decreases, then this means we press the keys less times to type the texts.

3. Distance the user's thumb travels. This records the total distance the user's thumb travels to type the testing texts. We assume that the user only use one thumb.

We can see that for all three simulations, the total time decreased by more than 20%. This means our new layout can help the user to save time when typing. If we further decompose the results, for the Financial Tweets data, the number of intra-group transitions was 65% of the original one. For the other two, this number further decreased to around 30%. The sum of indices of the letters all decreased to around half of that of the old layout, meaning the user only need to press the buttons half as much as before. The distance the user's thumb moves, however, increased a little comparing to the old layout.

6 Discussion

The results are expected. It confirms that the new T9 keyboard layout we designed do shortens the time taken for users to type. However, we see that the distance the user's thumb moves is not reduced. This is probably because we did not optimize the position of the "Space" key. Also, the priority for keyboard positioning is lower than that for grouping. Essentially, for grouping, we want to put together letter pairs with low weights, but for positioning, we do the opposite. Comparing to the time the user spends on letter collision (one second each time), the time that the user's thumb moves is short (0.1 second for each unit distance). Therefore, this is only a minor problem.

7 Conclusion

In this project, we optimized the layout of the traditional T9 keyboard, so the user can save time by typing on our keyboard. We learned techniques for our translating intuitions into an optimization problem which could then be relaxed into a solvable convex problem.

8 Division of work

Selena: conducted literature review on how to cut a graph in a k blocks and maximize crossing edge; attempted to separate letters into 8 blocks using Max-k-cut, Bisection Cut and basic spectral clustering method with the distance matrix; and implemented each method. She also designed and implemented evaluation metric to compare the performances of different keyboards.
Yongqi: worked on cleaning up the data, optimizing block arrangement (using brute force to solve the quadratic assignment problem), and the final simulation of the three testing texts. Yongqi also wrote all the final report except for part of the Introduction, Section 4.1 and Appendix.

Kyle: proved that the equal partition constraint generalizes to higher dimensions, built on Selena's code to generalize to higher ks, visualized results, and performed projection experiments. He also wrote section 4 and and lightly edited the rest of the final report.

9 Bibliography

References

[1] Wikipedia contributors, "T9 (predictive text) — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=T9_(predictive_text)&oldid=1149210284, 2023, [Online; accessed 6-May-2023].

[2] D. Wallach, "Kaggle financial tweets," https://www.kaggle.com/datasets/davidwallach/financial-tweets?select=stockerbot-export.csv, accessed: 2010-09-30.

[3] N. Schaffer, "The top 21 twitter user statistics for 2023 to guide your marketing strategy," https://nealschaffer.com/twitter-user-statistics/, accessed: 2010-09-30.

[4] A. M. Frieze and M. Jerrum, "Improved approximation algorithms for max k-cut and max bisection," in *Conference on Integer Programming and Combinatorial Optimization*, 1995.

[5] P. Felzenszwalb, C. Klivans, and A. Paul, "Clustering with semidefinite programming and fixed point iteration," 2022.

[6] J. T. Vogelstein, J. M. Conroy, V. Lyzinski, L. J. Podrazik, S. G. Kratzer, E. T. Harley, D. E. Fishkind, R. J. Vogelstein, and C. E. Priebe, "Fast approximate quadratic programming for graph matching," *PLOS ONE*, vol. 10, no. 4, pp. 1–17, 04 2015. [Online]. Available: https://doi.org/10.1371/journal.pone.0121002

10 Appendix

want a constraint that ensures equal partitions

$$\sum_{i>j}^{n} \langle y_i, y_j \rangle = \sum_{y_i = y_j}^{n} \langle y_i, y_j \rangle + \sum_{y_i \neq y_j}^{n} \langle y_i, y_j \rangle \quad // \text{ same group + different group}$$

$$(\text{equal partition}) = (\#\,\text{intra}) \cdot 1 + (\#\,\text{inter}) \cdot \frac{-1}{k-1} = \frac{1}{2} k \cdot \binom{n/k}{2} \cdot 1 + \frac{1}{2} \cdot k \cdot \left(\frac{n}{k} \cdot \left(n - \frac{n}{k}\right)\right) \cdot \frac{-1}{k-1}$$

$$(\text{unequal partition}) = (\quad > \quad) \cdot 1 + (\quad < \quad) \cdot \frac{-1}{k-1}$$

$(\cdots$ calculator $\cdots)$

$-\frac{n}{2}$

(unequal partition) > (equal partition)

(relaxed partition) $\leq$ (equal partition) since vector magnitudes are ≤ 1

Constraint: $\sum_{i>j}^{n} \langle y_i, y_j \rangle \leq$ (equal partition) $= -\frac{n}{2}$

Proof of unequal partition lemma

Intra

$$\sum^{k} p_i = n$$

$$\sum^{k} \binom{p_i}{2} = \sum^{k} \frac{p_i!}{2!(p_i-2)!}$$

$$= \frac{1}{2!} \sum^{k} \frac{p_i!}{(p_i-2)!}$$

$$= \frac{1}{2!} \sum^{k} p_i \cdot (p_i - 1)$$

$$= \frac{1}{2!} \sum^{k} p_i^2 - p_i$$

$$= \frac{1}{2!} \left(\sum^{k} p_i^2 - n \right)$$

$$= \frac{1}{2!} \cdot \left(\sum^{k} (n \cdot \lambda)^2 - n \right)$$

$$= \frac{1}{2!} \cdot \left(n^2 \sum^{k} \lambda^2 - n \right)$$

$$= \frac{1}{2!} \cdot \left(n^2 \langle \lambda, \lambda \rangle^2 - n \right)$$

$\nearrow$

optimize norm of a simplex
max <=> most inequality
min <=> most equality

Inter

$$\sum^{k} p_i = n$$

$$\frac{1}{2} \sum^{k} p_i \cdot (n - p_i) = \frac{1}{2} \sum^{k} n p_i - p_i^2$$

$$= \frac{1}{2} \left(n^2 - \sum^{k} p_i^2 \right)$$

$$= \frac{1}{2} \left(n^2 - n^2 \sum^{k} \lambda^2 \right)$$

$$= \frac{1}{2} \left(n^2 - n^2 \langle \lambda, \lambda \rangle^2 \right)$$

$$= \frac{n^2}{2} \left(1 - \langle \lambda, \lambda \rangle^2 \right)$$

$\nearrow$

optimize $-1 \cdot$ norm of a simplex
min <=> most inequality
max <=> most equality

$$\sum^{k} p_i = n$$

$$\vec{p} = n \cdot \lambda \mid \lambda \in \Delta_k$$

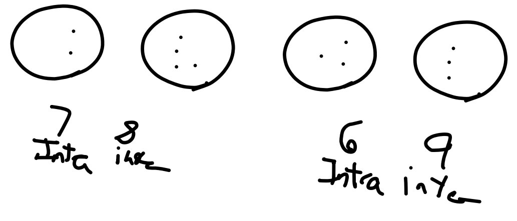


Figure 8