Optimal transport and the pancake cut

Johanna Siemelink

May 2024

Introduction

The goal of this project was to write a program that compares the pancake cut and an optimal transport between two discrete point sets. The hope is that increasing the number of points to approximate a continuous set may provide insight into the general optimal transport of simple shapes in two dimensions.

Last semester I familiarized myself with the problem in general and coded Megiddo's algorithm to obtain the pancake cut, and find a minimal optimal matching between two pointsets. This semester using my previous code as a starting point I built a program to run these steps recursively and devised a method to compare the the pancake cuts and the matchings.

Optimal transport

The question of optimal transport arises any time something has to be moved from one place to another. This is the mathematical question Monge proposed: When given two density functions how can we minimize a given cost function when moving everything between them. Kantorovich relaxation involves solving for a transport plan or map that minimizes the total cost, which is typically expressed as the integral of the cost function over the transport plan.

This is a rich field of research with numerous applications and a variety of versions. In this project we confine ourselves to two dimensional rectangles and try to approximate the continuous problem by checking discrete cases of increasing size.

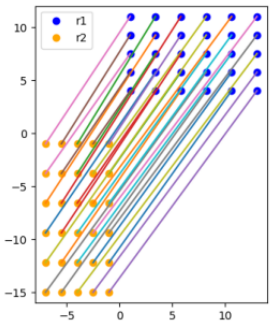
Pancake conjecture

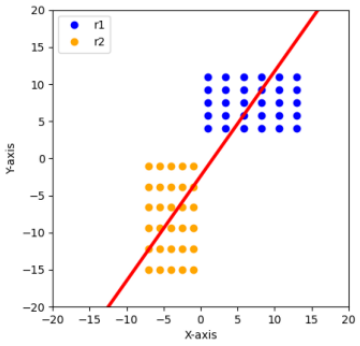
My thesis supervisor proposed the idea to investigate wether there are any links between the pancake cut and the optimal transport in 2 dimensions. The pancake cut is the two dimensional version of the ham and sandwich problem.

In the ham and sandwich problem two pieces of bread and a ham are somewhere in a 3d space. The goal is to share them equally for two: two equal pieces of top bun, bottom bun and ham. The famous theorem states that the sandwich can always be sliced into two equal parts with one plane. The two dimensional version is sometimes referred to as the pancake problem: slice two pieces of pancake in half with one line.

Conjecture: There exists an optimal transport map where the bisected halves of the pancake transfer to the other pancake's bisected halves without mixing. If this were true for all pancakes of any form, this could be applied recursively to the two halves, then the two quarters, then the eights and so forth, giving us the transfer map in some form.

Previous algorithms used now





Hungarian method

Used to find the optimal matching. A complete bipartite graph is weighted by the distances between the points on the plane. This way a minimum weight complete matching is an optimal transport for the discrete pointset.

Megiddo's algorithm

Megiddo's algorithm is a program that finds the pancake cut for two discrete sets that can be separated from each other by a line. It converts all the points into lines so the problem becomes finding the shared median point of two sets of lines. With smartly placed iterative queries the algorithm finds this point and thus the pancake cut in linear time.

Comparisons

Compare cuts and matches

To compare the pancake cut and the optimal matching is like comparing apples to oranges. But unlike fruits mathematical constructs can be transformed into comparable forms. Our options are to build a matching using the pancake cut, build a cut from the optimal matching, or transform both into a third object, I chose the option where I build a cut from the matching. I have to emphasize that this is a very heuristic approach, it does not give a result of any mathematical substance, but is merely a tool to better understand the problem.

My objective became to take an optimal matching and to use it to create a straight line cut that hopefully behaves like the pancake cut as much as possible. I'll outline the idea of my algorithm below:

Matching to "pancake cut"

1. Check all edges of the optimal matching and count the points that lie above and below the line the edge induces.
2. IF there is an edge where the difference between the number of points on each side of its induced line is 0, RETURN the line
3. ELSE take the two edges that result in the smallest differences and take their bisector that results in the smallest difference, RETURN the bisector

Compare cuts and cuts

Now we're comparing apples to apples, well even better it's cuts to cuts. We could compare the lines with each other, but that doesn't make sense. There are multiple valid pancake cuts, so the actual line we calculated with Megiddo's algorithm doesn't matter. Our other option is to test the cut we got from the matching on how well it bisects both sets. I just stored the size of the cut up sets and made a visualizing tool using a bar graph.

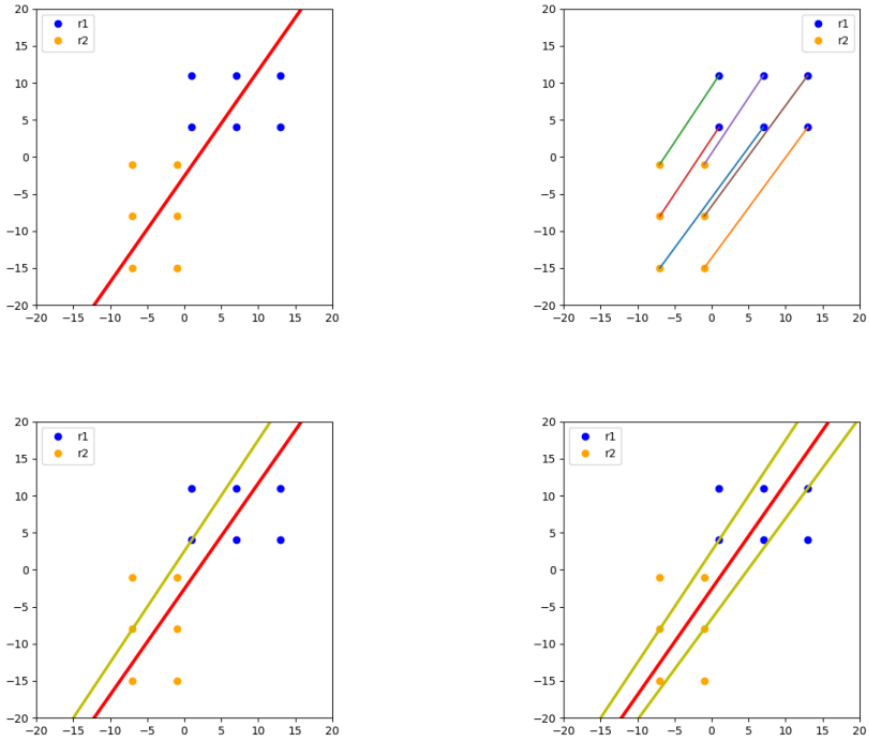
Recursion

The concept is to iteratively apply the pancake cut infinite times to learn where to send each point in the continuous version. So in the discrete version I cut and cut until I ran out of edges or points. As a point of reference, I implemented the same recursion with the real pancake cut as well as the cut built from the matching.

A small recursive example

Let's make good use of the visualization tools I implemented and follow step by step on a small example.

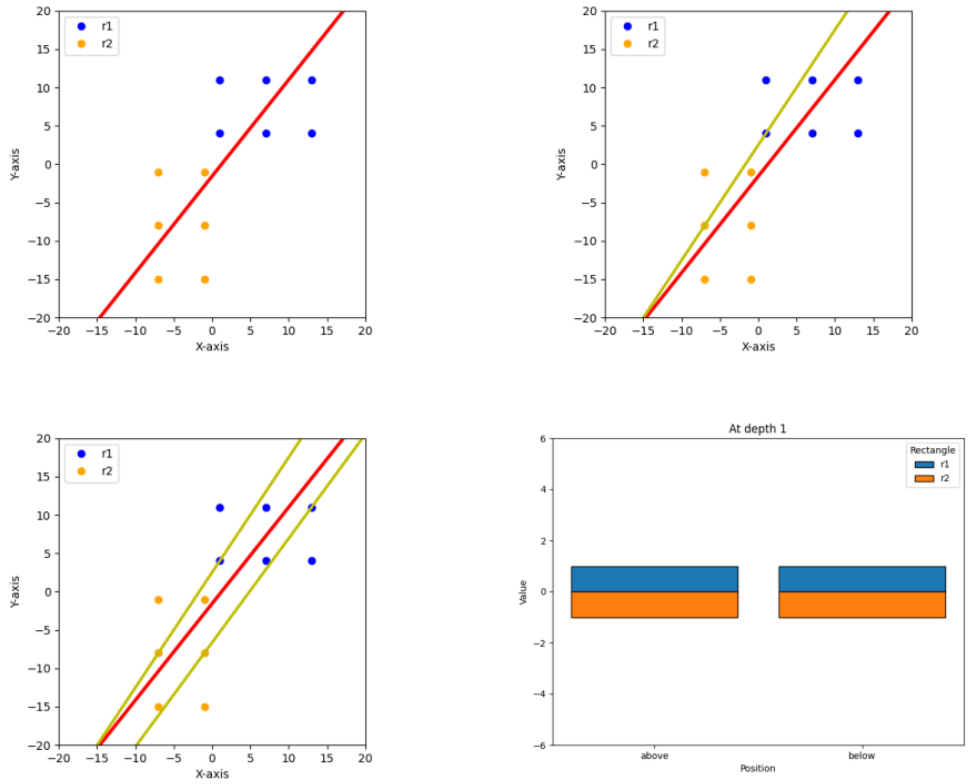
Cuts made from a Matching



The first cut is made taking the bisecting line of the purple and blue edges of the optimal matching that is in the upper right corner. It halves the pointsets perfectly.

At the next depth of the recursion there are three points in the upper pointset and three points in the lower. Fortunately the pink and brown edges already result in bisecting lines, so these are picked.

The real pancake cut



As seen on the figures, the first pancake cut given by Megiddo's algorithm is a different cut than the one obtained by the bisecting method, even though they both are perfect pancake cuts. When cutting three lines into two parts both methods find he same line, which is promising.

The lower right image is here to demonstrate the format of the bar graph where we can see the four parts that the two pointsets get cut into. In this instance we can see that when the green line on the right bisects the right half created by the first cut it does half the points, one on one side one on the other for both rectangles.

I like the way the unequal halves can be seen, but for bigger data in particular a table may be more useful

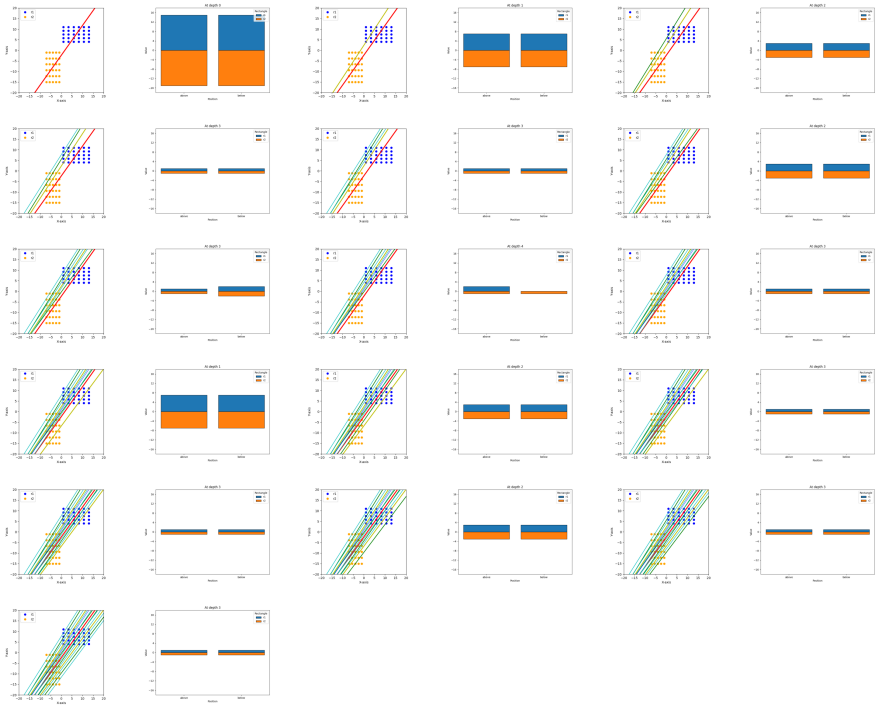
Matching cuts table

depth	line	r1_above	r2_above	difference	r2_below	r1_below	difference_2
0	(-2.600615561039831, 1.4261758199752403)	3	3	0	3	3	0
1	(2.5, 1.5)	1	1	0	1	1	0
1	(-6.642857142857142, 1.3571428571428572)	1	1	0	1	1	0

Results

For bigger instances it halving often appears not to happen perfectly at the deepest points of the recursion. This is not a problem, because at those deep instances there are a lot of possibilities for edge cases that result in these discrepancies.

The most important edge case is the one where only one edge of the optimal matching remains, which has to be chosen by the algorithm, but doesn't result in equal halves. But in theory we should always do it for denser pointsets which means this problem would disappear.



For this reason I propose to measure how similar a cutting is to count the number of discrepancies but take into account at what depth it happens.

Some discrepancies also happen with the Megiddo algorithm, but in that case the difference can only be at most 1. The first table is the matching generated cut differences, where I filtered out the differences at the deepest depth. The second one is the difference table generated by the good pancake cut theorem.

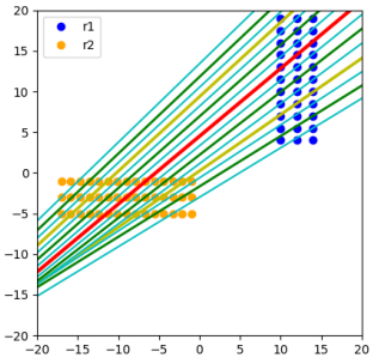
depth	difference	difference_2
2	1	1
3	1	2
3	1	0
3	1	1
3	1	1

depth	difference	difference_2
2	1	1
3	1	0
3	1	1
3	1	0

Different arrangements

Let us now look at some differently placed instances.

Rotated

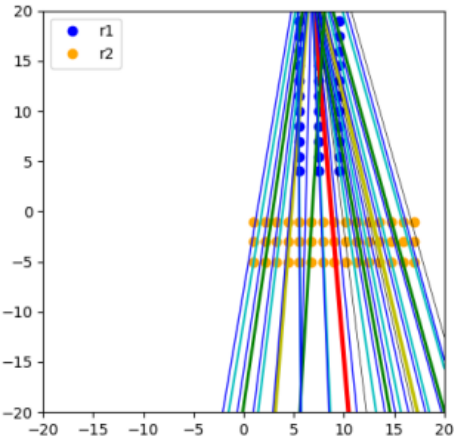


depth	difference	difference_2
4	1	1

depth difference difference_2

The first instance I tried was very nearly perfect, only one unequality at max depth, no others. But seeing as these instances have no mathematical proving power the only way something could be proven using these if we found a counterexample. In other words: when playing with different constructions the goal is to try and find one that works as badly as possible and hope we can learn something from it.

Above

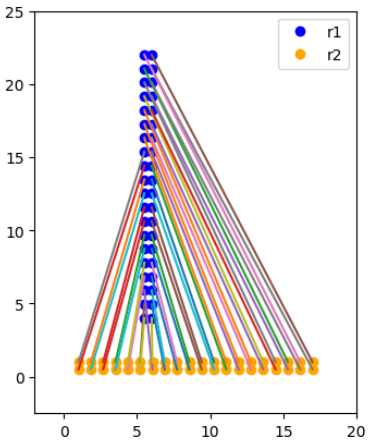


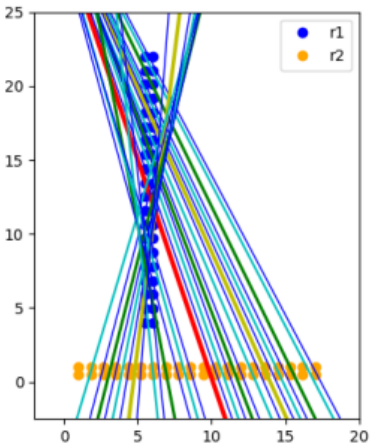
depth	difference	difference_2
3	1	1
2	1	1
3	0	1
5	0	1
3	0	1
5	0	1
2	1	1
3	2	1
3	1	2

depth difference difference_2

Here I did manage to create a lot of error. But this also looks like a sparsity issue. Because there are only three columns in the upper rectangle the left one gets sent left, the right one gets sent right and the one in the middle looks messy.

Slim rectangles

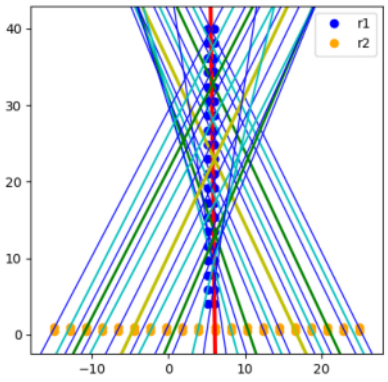


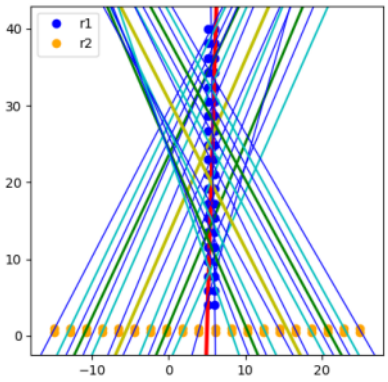


depth	difference	difference_2
0	2	2
1	0	2
3	1	1
2	0	2
3	0	2
1	0	2
2	2	2
3	2	0
3	1	1
2	2	0
3	2	0
depth	difference	difference_2

I tried to make the problem even worse, from these images it seems there is a pivoting point in the matching which might cause all the chaos.

Symmetric



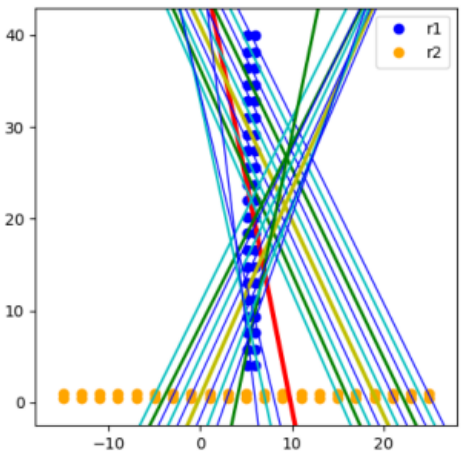


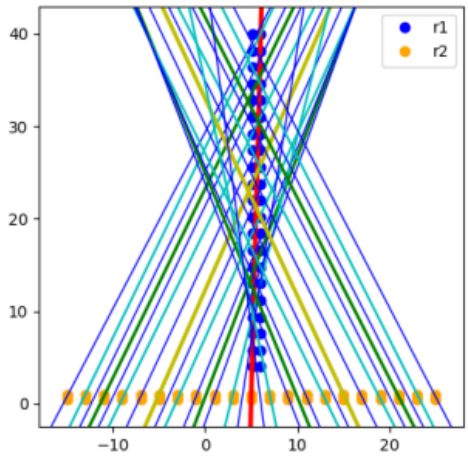
depth	difference	difference_2
depth	difference	difference_2
0	3	3
1	1	1
2	0	1
3	1	1
1	2	2
2	1	1
2	1	1

To my huge surprise I found an instance where the pancake cut preformed worse than the heuristic matching cut. I think this happened because of how Megiddo's algorithm actually works. Firstly we have to understand that we're looking for a median line so it should be no surprise that the last step of the algorithm is taking the mean (in some sense) of the upper middle line and the lower middle line (with odd size input these coincide, just like with the regular median). The two medial lines are calculated by starting out with all pointpairs between upper and lower pointset points. This means that -heuristically speaking- the Megiddo algorithm can't take options into account where the line doesn't run through a pointpair.

This means that even though the most logical way to proceed seems to be to start out with modifying the vertical cut bisecting the upper slim rectangle, this is not an option for

Megiddo's algorithm. It instead checks almost vertical lines whose slopes differ within the range of numerical error, leading to large differences further down the line.





depth	difference	difference_2
0	6	6
1	6	0
2	3	3
3	1	2
4	0	1
4	1	1
3	2	1
4	1	1
2	3	3
3	2	1
4	1	1
3	1	4
1	7	1
2	10	3
3	10	0
3	1	2
4	0	1
4	1	1
2	2	1
3	2	0
4	1	1
3	0	1
4	1	1
4	0	1

This is the same construction as the last one, but 2x21 instead of 2x20 points. In this case unfortunately the normal tendency persists

Conclusions

As is known, this is a very heuristic approach, thus it isn't ground for firm conclusions. Results appear imperfect, but the nature of approximating and numerical errors could be at fault here. Instead of trying to form concrete ideas I will sketch what we can learn and can use from these instances.

Numerical future

One route could be to try and mitigate the problems arising in the computer program. One could use the data of the concrete lines where it goes astray to form hypotheses as to why it happens, and try and eliminate these possibilities. Another heuristic idea to create a pancake cut from a matching instead of the bisectors of the best edges could also result in a better model.

Other heuristic approaches can be devised, it could well be that creating matchings from cuts result in a numerically more stable heuristic approach, or that there exists a third type of object that the matchings and the cuts can both be transformed into.

And of course for all options a faster computer could take more points resulting in a smoother picture and thus better results.

Mathematical future

If one would use mathematically rigourous tools on this problem, this project gives an excellent starting point: in the instance of one slim rectangle is almost but not quite symmetrically above a flat lying rectangle there is a pivot point where the optimal transport of the discrete case brings the points above this point to one side and the points below it to the other. I think investigating this point could be a promising approach to start work on the continuous case.

References

[1] Frank, András (revised 2010): Connections in Combinatorical Optimization. 3.3.1 The Hungarian method

[2] Megiddo, Nimrod (1985): Partitioning with two Lines in the Plane. – Journal of algorithms 6. 430-433