Algorithmic Art

-Sriram Raju Dandu (sd9aj)

Description

Algorithmic Art is a creative blend of programming, design, and the arts. This project uses computer algorithms to process images and create compelling art. The two main algorithms incorporated in the project are Prim's Minimum Spanning Tree and Nearest-Neighbors along with 2-Opt Travelling Salesman Heuristic.

Prim's Minimum Spanning Tree

A minimum spanning tree (MST) is a subset of the edges of a connected, edge-weighted (un)directed graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible. Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

Nearest-Neighbors Travelling Salesman Heuristic

A famous problem in Computer Science, travelling salesman problem (TSP) essentially asks: given a set of points, commonly defined as cities on a map, find the shortest route that covers all the points exactly once. Many algorithms have been proposed as solutions to the TSP, two of which this project covers are the nearest neighbor and 2-Opt heuristics.

Algorithm

In creating the Algorithmic Art program, I employed an iterative process using stepwise refinement to gradually build the program. I used MATLAB to create the program. The algorithm takes an image as an input and outputs art based on the image. The algorithm has 2 stages:

Stage 1: Create a stippled image

The MST and TSP algorithms require points as inputs for the generating patterns. So, the image should be converted into dots such that the points still represent the image. The input images I converted into a black-white image. Stippled version of the image is created by constructing a grid from the image, where each cell in the grid is a certain dimension, such as 3x3, 4x4, 5x5, etc. Each of these cells holds a single pixel to represent all the pixels in that particular cell. If the number of dots within the grid are greater than a threshold, then a point is placed within the grid. Initially the point was placed in the center of the grid. However, the results were too uniform. So instead of placing each pixel in the center of a region, the pixel is placed in a random location in the region. This improvement resulted in a more stippled appearance as shown in the figure below.

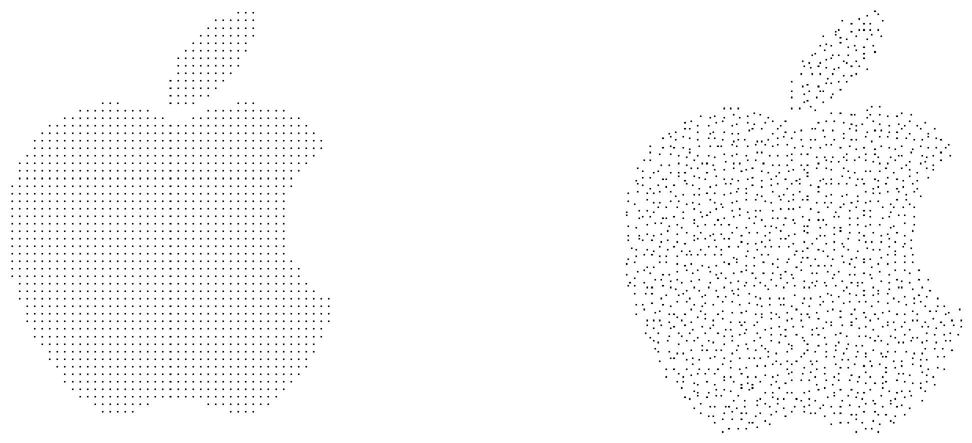


Figure 1: Left: Uniform points in stippled image. RIght: Random points in stippled image.

Stage 2: Connect the points

The points are connected to obtain two different patterns: One pattern (MST) is a graph that connects all the vertices with the least sum and second pattern (TSP) is a path connecting all points without revisiting a point more than once.

1. MST

The MST is computed based on a graph, so Delaunay triangulation was incorporated to construct a graph with points generated in the previous stage. The Delaunay triangulation for a given set P of discrete points in a plane is a triangulation such that no point in P is inside the circumcircle of any triangle. The Delaunay triangulation of the generated points is shown in the figure below.

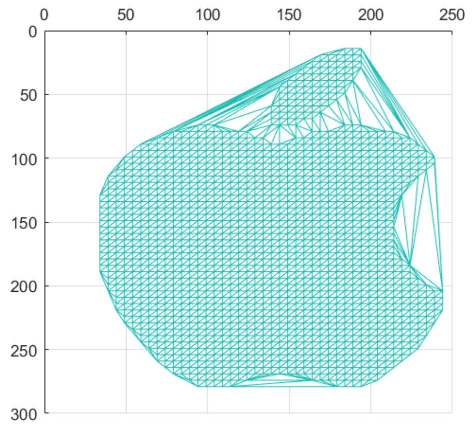


Figure 2: Delaunay Triangulation

The Prims MST algorithm was implemented on the graph generated using Delaunay triangulation. The algorithm may informally be described as performing the following steps:

- Initialize a tree with a single vertex, chosen arbitrarily from the graph.
- Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the shortest edge, and transfer it to the tree.
- Repeat the above step (until all vertices are in the tree).

The figure below shows the result obtained from the Prims MST algorithm.

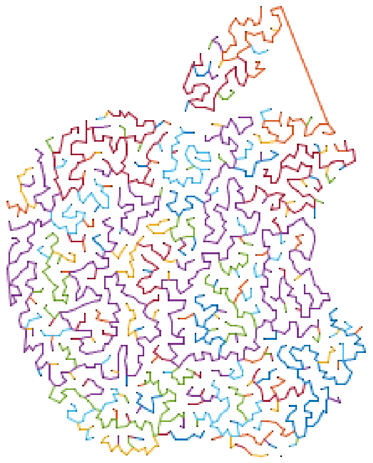


Figure 3: MST

The different colors in the figure are the different branches of the tree.

2. Nearest Neighbors + 2-Opt TSP heuristic

After creating the stippled image, the next step is to create the actual lines to connect the dots to build the path. A TSP heuristic known as the nearest neighbor's heuristic is implemented. It is a straightforward heuristic: for each point, choose the nearest point from all the other unvisited points, and mark the current point as "visited". Then the procedure is repeated until all the points are visited. Note that this "greedy" approach does not necessarily result in the shortest tour; nonetheless, it is an approximate and simple procedure. For example, images with a lot of fine-grained details or subtle color gradients don't translate well to TSP because many of the details are lost due to the stippling. The nearest neighbor's heuristic generates the result shown in the figure below.

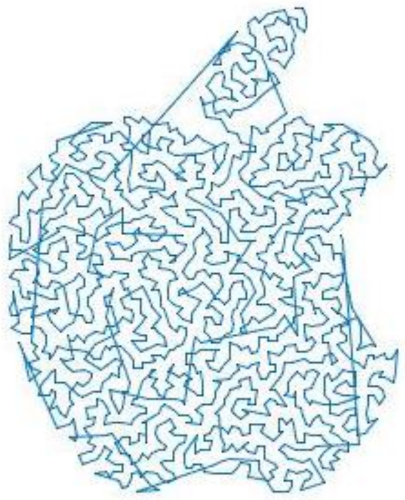


Figure 4: Nearest Neighbors TSP heuristic

As seen in the above figure, the nearest neighbors heuristic resulted in many long lines being drawn across the image. In order to handle this, 2-Opt heuristic is incorporated. 2-Opt improves an existing TSP tour by swapping certain edges if the resulting path creates a shorter tour. The heuristic works as follows: if the two lines in the TSP path intersect, swap the end points of the two lines. This results in a shorter overall tour, as well as vastly improved TSP art as shown below.

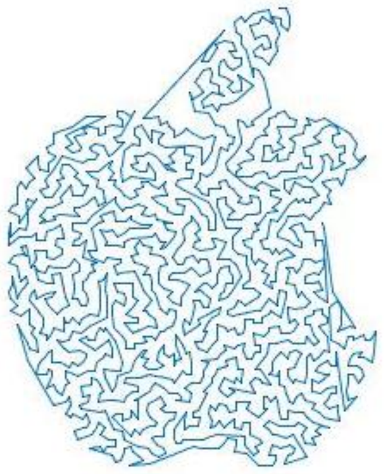


Figure 5: Result of TSP with 2-Opt heuristic

After I implemented the MST and TSP, I continued to experiment with a wide variety of images and improving the runtime as the 2-Opt heuristic took very long to run.

Implementation

All the algorithms were programmed and tested in MATLAB. More MST and TSP patterns can be found in Appendix.

Graphic User Interface

A GUI was built for easy use of the algorithms and see the results in real-time.

A snippet of the GUI is shown in the figure below.

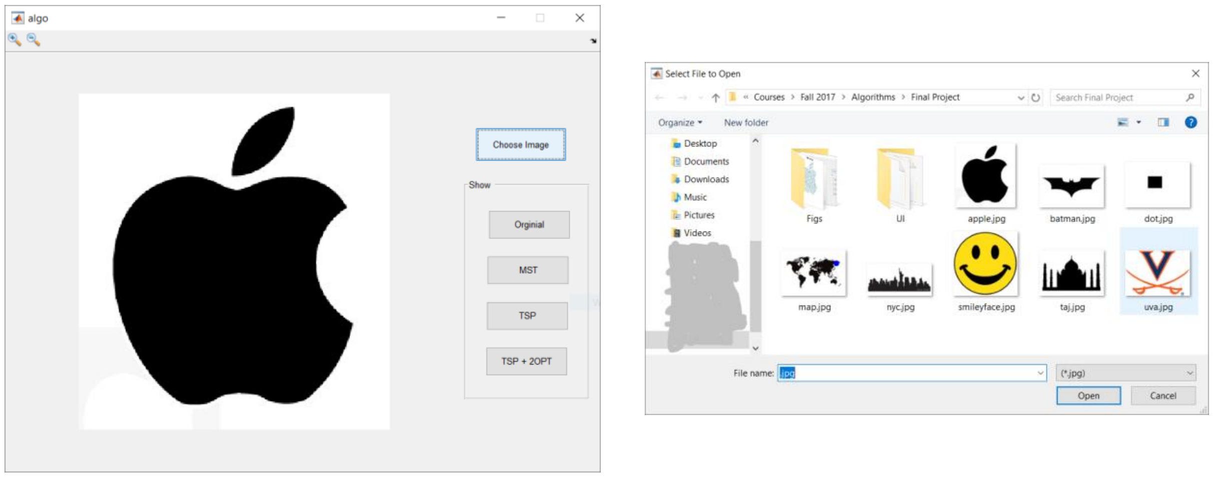


Figure 6: GUI for the Algorithmic Art

Hurdles

The first time, when running the 2 Opt heuristic, it took several points to untangle the TSP path consisting approximately 1500 points. Now it takes about 40 seconds to run. I incorporated multiple optimization techniques, such as using matrix operations to compute the Euclidean distances than using the traditional for loop. I also implemented the line intersection subroutine in a smart and simple which reduce time significantly. I constructed the line equation of 2 points and substituted the other 2 points in the equation. If the 2 points are on either side, then the sign of the values is different too. Both these techniques improved the runtime significantly.

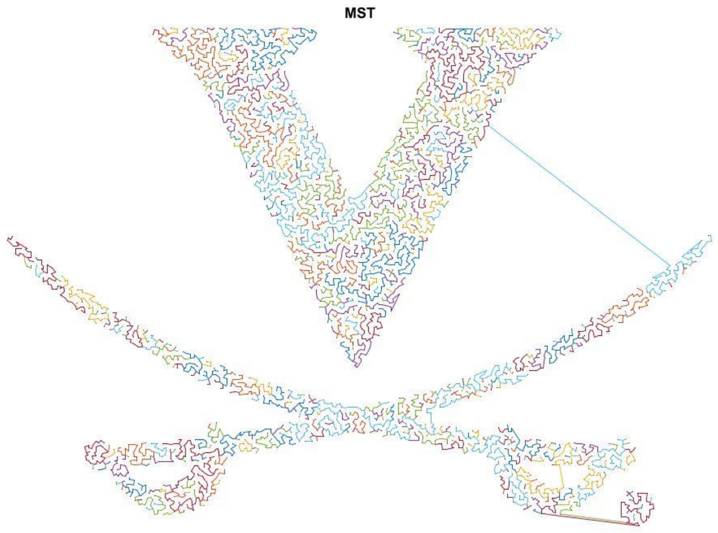
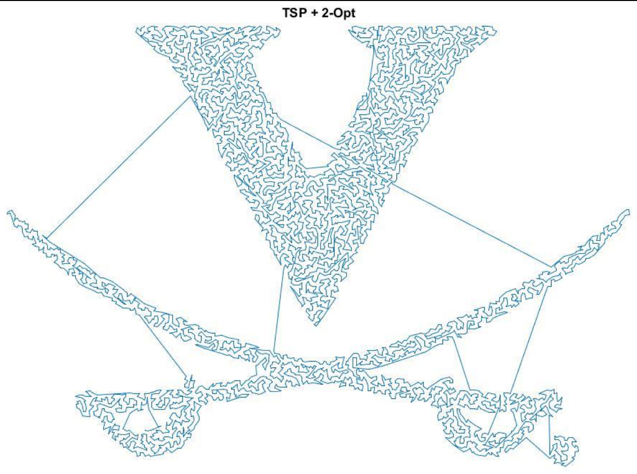
Runtime

For 1500 points,

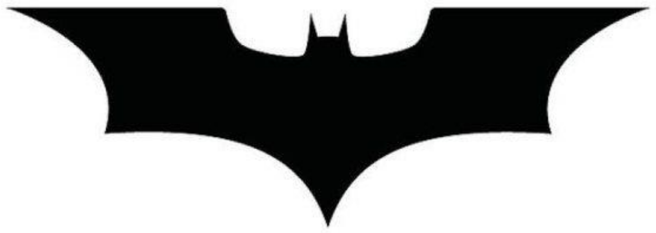
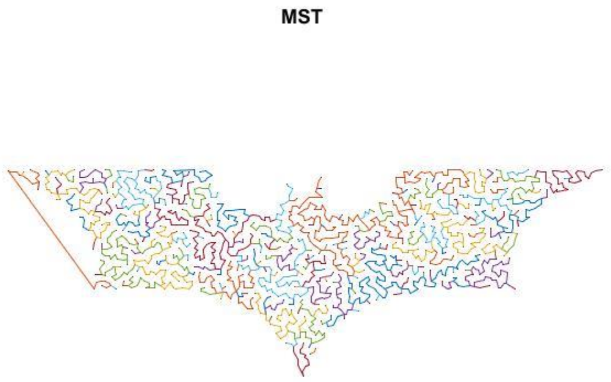
Algorithm	**Prims MST**	**NN-TSP**	**2-Opt**
Run Time (seconds)	0.79	2.23	40.46

Appendix

A. UVA Cavaliers

Original	
MST	
TSP + 2-Opt	

B. Batman

Original	
MST	
TSP + 2-Opt