Section H
Unit #4
TSP
Distance

Content

Section H Unit #4 Distance between cities

Sketch H4.1 index.html
Sketch H4.2 an array of cities
Sketch H4.3 join the cities
Sketch H4.4 swap function
Sketch H4.5 swapping cities
Sketch H4.6 calculate distance
Sketch H4.7 shortest distance
Sketch H4.8 alternatives
Sketch H4.9 draw the best one

Introduction to the Travelling SalesPerson Problem

The travelling Salesperson (TSP) is a famous mathematic challenge. If you are unfamiliar with it you may think what is all the fuss about.

The problem

Essentially the problem goes like this, a salesperson has to visit say 5 cities, they can start at any city but need to visit each city in the shortest distance.

The solution

The solution is to measure the distance between all the cities and try all the different permutations until you get the lowest or smallest distance.

The Challenge

Although the computer can do all theses calculations very fast indeed when the number of cities gets larger the number of possible solutions (even though there is still only one optimum) is a factor of that number.

5 cities = 120 possibilities
10 cities = 3,628,800 possibilities
12 cities = 479,001,600 possibilities
20 cities = 2.432902e+18 possibilities which means it would take millions of years for a computer to calculate the optimum answer

A better solution

This is another approach which although doesn't solve it by brute force it does offer a much improved close run thing and that is to use a Genetic Algorithm.

So let's get started

Introduction to Unit #4 (TSP) distance

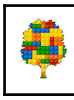
Measuring the distance between the cities. This is not optimal but just the mechanism of measuring.

Sketch H4.1 index.html

First of all we need to create another file called ga.js for the genetic algorithm part and add it to our index.html file

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/
1.4.0/p5.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/
1.4.0/addons/p5.sound.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8" />

  </head>
  <body>
    <script src="sketch.js"></script>
    <script src="ga.js"></script>
  </body>
</html>
```



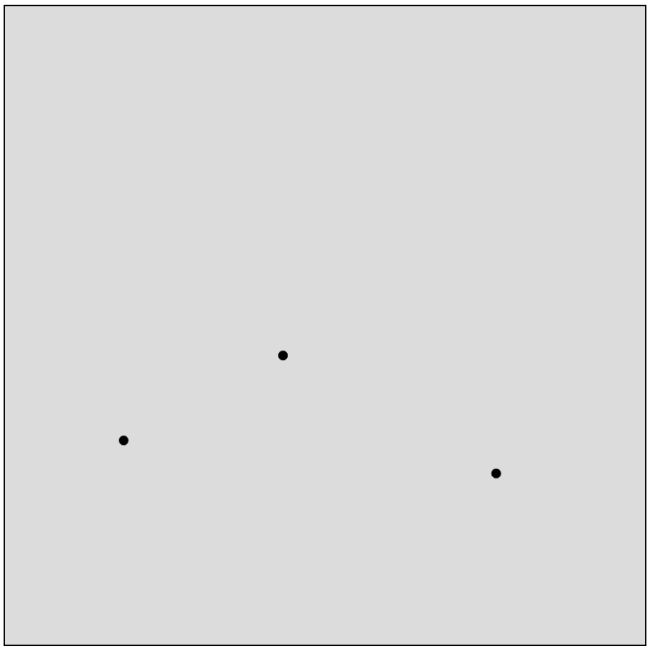
Sketch H4.2 an array of cities

Starting a new sketch. We create an empty array of cities, adding three random cities and draw them (you can have more if you want but three will be ample at this stage)

sketch.js

```
let cities = []
let totalCities = 3

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < totalCities; i++)
  {
    let v = createVector(random(width), random(height))
    cities[i] = v
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < cities.length; i++)
  {
    fill(0)
    circle(cities[i].x, cities[i].y, 5)
  }
}
```

We have our three random cities



Sketch H4.3 join the cities

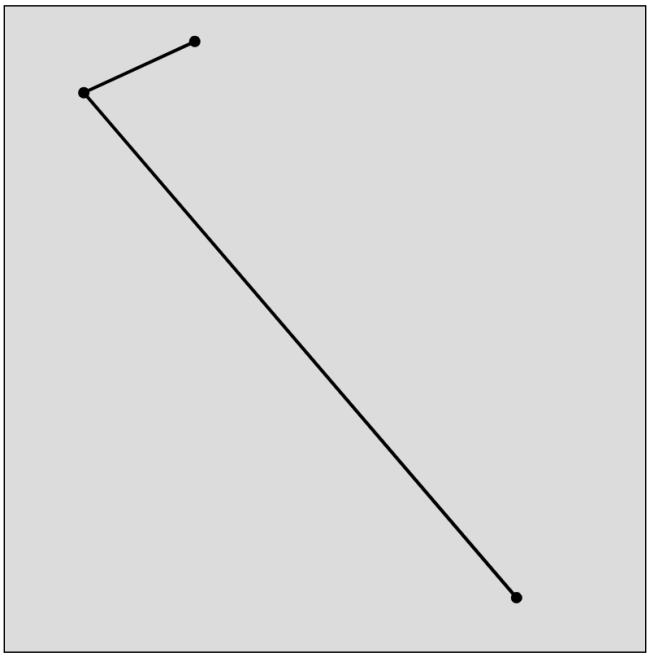
Now let us join them together with a line

sketch.js

```
let cities = []
let totalCities = 3

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < totalCities; i++)
  {
    let v = createVector(random(width), random(height))
    cities[i] = v
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < cities.length; i++)
  {
    fill(0)
    circle(cities[i].x, cities[i].y, 5)
  }
  beginShape()
  for (let i = 0; i < cities.length; i++)
  {
    noFill()
    strokeWeight(2)
    vertex(cities[i].x, cities[i].y)
```

```
    }
    endShape()
}
```

It will be different every time because
they are randomly generated



Sketch H4.4 swap function

We want to swap the order of the array of cities. We create a new function called swap(). In that function we can swap elements of the array around but we need to have a temporary one to hold the initial value otherwise they just become themselves again!

sketch.js

```
let cities = []
let totalCities = 3

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < totalCities; i++)
  {
    let v = createVector(random(width), random(height))
    cities[i] = v
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < cities.length; i++)
  {
    fill(0)
    circle(cities[i].x, cities[i].y, 5)
  }
  beginShape()
  for (let i = 0; i < cities.length; i++)
  {
    noFill()
```

```
    strokeWeight(2)
    vertex(cities[i].x, cities[i].y)
  }
  endShape()
}
```

```
function swap(a, i, j)
{
  let temp = a[i]
  a[i] = a[j]
  a[j] = temp
}
```

Sketch H4.5 swapping cities

Now we can test it out by calling the function in draw using the cities array and a random i and j.

sketch.js

```
let cities = []
let totalCities = 3

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < totalCities; i++)
  {
    let v = createVector(random(width), random(height))
    cities[i] = v
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < cities.length; i++)
  {
    fill(0)
    circle(cities[i].x, cities[i].y, 5)
  }
  beginShape()
  for (let i = 0; i < cities.length; i++)
  {
    noFill()
    strokeWeight(2)
    vertex(cities[i].x, cities[i].y)
```

```
  }
  endShape()
  let i = floor(random(cities.length))
  let j = floor(random(cities.length))
  swap(cities, i, j)
}

function swap(a, i, j)
{
  let temp = a[i]
  a[i] = a[j]
  a[j] = temp
}
```

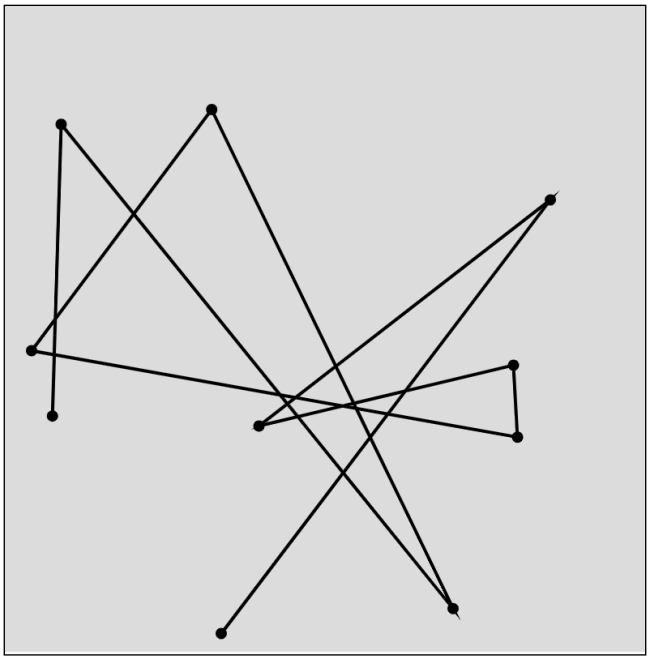
Notes

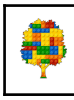
What you get are the lines being drawn differently as the sketch runs, where the order the lines are drawn are being swapped around

Challenge

Try more cities to get a better idea of what it looks like drawing different orders.

This is what 10 might look like and you can watch it swapping the order around and redrawing it





Sketch H4.6 calculate distance

We want to work out the total distance between cities for each iteration or possible order. We will create another function to do this called calcDistance() it will receive a list of points to calculate their total distance.

It is points.length - 1 because we are measuring to the last point in the array.

sketch.js

```
let cities = []
let totalCities = 3

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < totalCities; i++)
  {
    let v = createVector(random(width), random(height))
    cities[i] = v
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < cities.length; i++)
  {
    fill(0)
    circle(cities[i].x, cities[i].y, 5)
  }
  beginShape()
  for (let i = 0; i < cities.length; i++)
```

```
  {
    noFill()
    strokeWeight(2)
    vertex(cities[i].x, cities[i].y)
  }
  endShape()
  let i = floor(random(cities.length))
  let j = floor(random(cities.length))
  swap(cities, i, j)
}

function swap(a, i, j)
{
  let temp = a[i]
  a[i] = a[j]
  a[j] = temp
}
```

```
function calcDistance(points)
{
  let sum = 0
  for (let i = 0; i < points.length - 1; i++)
  {
    let d = dist(points[i].x, points[i].y, points[i + 1].x,
points[i + 1].y)
    sum += d
  }
  return sum
}
```

Notes

This won't work yet because we have not passed in the cities array to the points and have not called the calcDistance() function

Sketch H4.7 shortest distance

What we want is the best (shortest distance) and so we create a variable called recordDistance. At this point it is going to just generate a random set and order of points and give us our starting point. It might be the best it might not.

sketch.js

```
let cities = []
let totalCities = 3
let recordDistance

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < totalCities; i++)
  {
    let v = createVector(random(width), random(height))
    cities[i] = v
  }
  let d = calcDistance(cities)
  recordDistance = d
}

function draw()
{
  background(220)
  for (let i = 0; i < cities.length; i++)
  {
    fill(0)
    circle(cities[i].x, cities[i].y, 5)
  }
  beginShape()
```

```
  for (let i = 0; i < cities.length; i++)
  {
    noFill()
    strokeWeight(2)
    vertex(cities[i].x, cities[i].y)
  }
  endShape()
  let i = floor(random(cities.length))
  let j = floor(random(cities.length))
  swap(cities, i, j)
}

function swap(a, i, j)
{
  let temp = a[i]
  a[i] = a[j]
  a[j] = temp
}

function calcDistance(points)
{
  let sum = 0
  for (let i = 0; i < points.length - 1; i++)
  {
    let d = dist(points[i].x, points[i].y, points[i + 1].x,
points[i + 1].y)
    sum += d
  }
  return sum
}
```

Sketch H4.8 alternatives

We want to use the swap function to give us some alternatives to compare to see if we can find a better one. So we swap the cities around and then compare with our initial recordDistance. If it is less then that is the new recordDistance. I have added a console.log to see the results

sketch.js

```
let cities = []
let totalCities = 3
let recordDistance

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < totalCities; i++)
  {
    let v = createVector(random(width), random(height))
    cities[i] = v
  }
  let d = calcDistance(cities)
  recordDistance = d
}

function draw()
{
  background(220)
  for (let i = 0; i < cities.length; i++)
  {
    fill(0)
    circle(cities[i].x, cities[i].y, 5)
  }
  beginShape()
```

```
  for (let i = 0; i < cities.length; i++)
  {
    noFill()
    strokeWeight(2)
    vertex(cities[i].x, cities[i].y)
  }
  endShape()
  let i = floor(random(cities.length))
  let j = floor(random(cities.length))
  swap(cities, i, j)
  let d = calcDistance(cities)
  if (d < recordDistance)
  {
    recordDistance = d
  }
  console.log(recordDistance)
}

function swap(a, i, j)
{
  let temp = a[i]
  a[i] = a[j]
  a[j] = temp
}

function calcDistance(points)
{
  let sum = 0
  for (let i = 0; i < points.length - 1; i++)
  {
    let d = dist(points[i].x, points[i].y, points[i + 1].x,
points[i + 1].y)
    sum += d
```

```
  }
  return sum
}
```

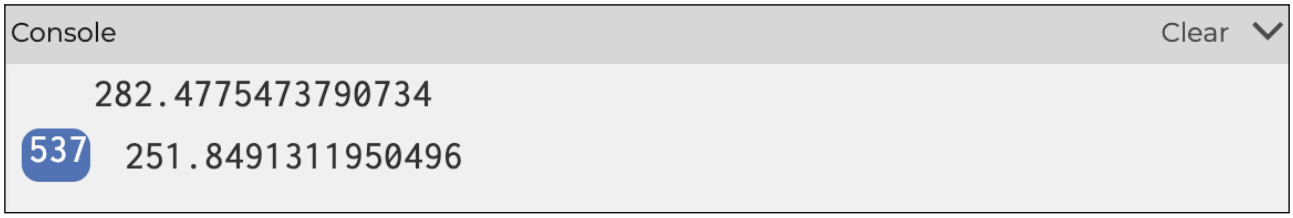
Notes

You may only get one number if the shortest is the first guess

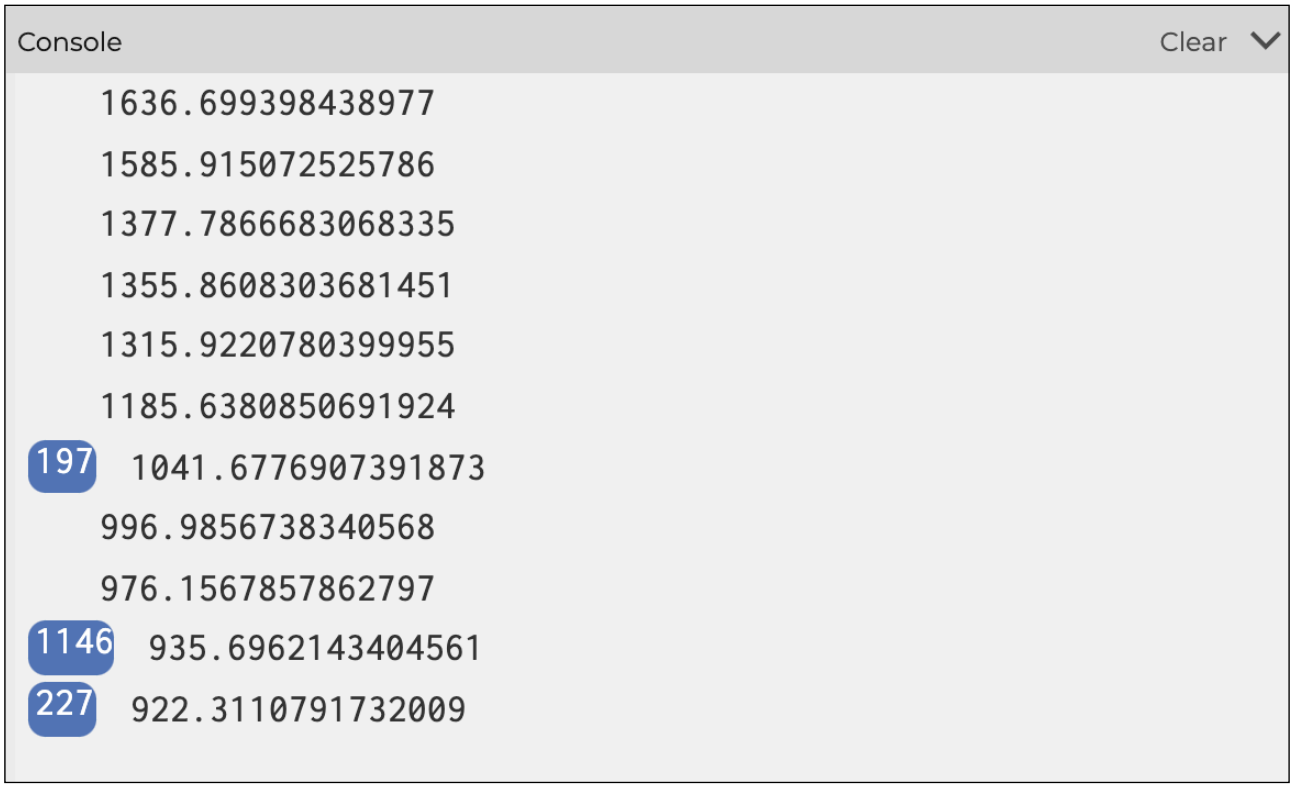
Challenge

Try 10 cities, you can see the improvements

I got an improvement from 282 to 251 with three cities



I tried it with 10 cities for comparison



Sketch H4.9 draw the best one

Back to three cities! We want to draw the current best (shortest) path as it iterates through them all. We create another variable called bestEver which will copy the best array so far that gives us the shortest distance. For this we use the slice() command. We also remove the console.log as we are going to draw the bestEver on screen.

Also some little tweaks so that it shows up nicely on the canvas.

! SAVE THIS SKETCH !

sketch.js

```
let cities = []
let totalCities = 3
let recordDistance
let bestEver

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < totalCities; i++)
  {
    let v = createVector(random(width), random(height))
    cities[i] = v
  }
  let d = calcDistance(cities)
  recordDistance = d
  bestEver = cities.slice()
}

function draw()
{
  background(220)
```

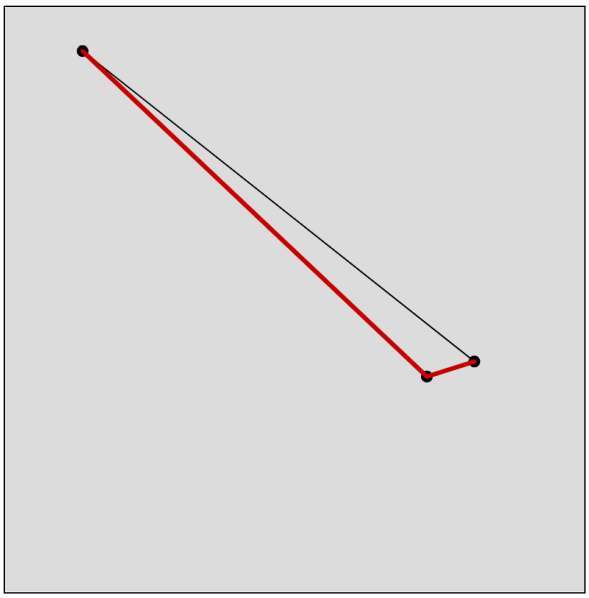
```
  for (let i = 0; i < cities.length; i++)
  {
    stroke(0)
    fill(0)
    circle(cities[i].x, cities[i].y, 5)
  }
  beginShape()
  for (let i = 0; i < cities.length; i++)
  {
    stroke(0)
    noFill()
    strokeWeight(1)
    vertex(cities[i].x, cities[i].y)
  }
  endShape()
  beginShape()
  for (let i = 0; i < cities.length; i++)
  {
    stroke(200, 0, 0)
    noFill()
    strokeWeight(3)
    vertex(bestEver[i].x, bestEver[i].y)
  }
  endShape()
  let i = floor(random(cities.length))
  let j = floor(random(cities.length))
  swap(cities, i, j)
  let d = calcDistance(cities)
  if (d < recordDistance)
  {
    recordDistance = d
    bestEver = cities.slice()
  }
```

```
}


function swap(a, i, j)
{
  let temp = a[i]
  a[i] = a[j]
  a[j] = temp
}


function calcDistance(points)
{
  let sum = 0
  for (let i = 0; i < points.length - 1; i++)
  {
    let d = dist(points[i].x, points[i].y, points[i + 1].x,
points[i + 1].y)
    sum += d
  }
  return sum
}
```

Notes

Important please save this sketch now, in the next section we will create a new sketch and return to this one afterwards.

Drawing the best one in red



This is with 10 cities and it has been running for a few minutes, this is clearly not optimal just by looking at it and it is why we need something like a genetic algorithm to help, otherwise this could take a very long time to brute force a solution.