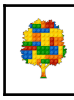
Section H
Unit #8
TSP
Crossover



Content

Section H Unit #8 Crossover

Introduction to Unit #8 Crossover

Crossover is when you take elements of one parent and elements of another parent and combine them in the child. Although in nature it is the dna of two individuals in a genetic algorithm you could take the dna from three or more and combine them into one.

Sketch H8.1 mutation rate

We need to make use of the mutation rate in the mutate() function because at present it is just mutating all the time which is not optimal. We only want some swapping to take place according to the mutation rate, this happens as it cycles through the total number of cities.

To make sure we don't pick the same one to swap we use this line of code:

let indexB = (indexA + 1) % totalCities

Although it seems like a very odd bit of code this ensures that whatever is selected randomly for indexA, indexB will never be the same. Just try to work it out by using a value for indexA and seeing what you get for indexB using numbers 0 - 10 as in the current number of totalCities.

ga.js

```
function calculateFitness()
{
  for (let i = 0; i < population.length; i++)
  {
    let d = calcDistance(cities, population[i])
    if (d < recordDistance)
    {
      recordDistance = d
      bestEver = population[i]
    }
    fitness[i] = 1 / (d+1)
  }
}

function normaliseFitness()
{
  let sum = 0
  for (let i = 0; i < fitness.length; i++)
```

```
  {
    sum += fitness[i]
  }
  for (let i = 0; i < fitness.length; i++)
  {
    fitness[i] = fitness[i] /sum
  }
}

function nextGeneration()
{
  let newPopulation = []
  for (let i = 0; i < population.length; i++)
  {
    let order = pickOne(population, fitness)
    mutate(order, 0.01)
    newPopulation[i] = order
  }
  population = newPopulation
}

function pickOne(list, prob)
{
  let index = 0
  let r = random(1)
  while (r > 0)
  {
    r = r - prob[index]
    index++
  }
  index--
  return list[index].slice()
}
```

```
function mutate(order, mutationRate)
{
  for (let i = 0; i < totalCities; i++)
  {
    if (random(1) < mutationRate)
    {
      let indexA = floor(random(order.length))
      let indexB = (indexA + 1) % totalCities
      swap(order, indexA, indexB)
    }
  }
}
```

Sketch H8.2 pick two

We now want to select two orders and do a crossover by creating a crossover function. We want to pick two rather than just pick one which is what we already do. We will do this in the nextGeneration() function. Instead of order we will have orderA and orderB as the parents of choice. From them we will do a crossover. We introduce a crossOver() function taking these two orders.

ga.js

```
function calculateFitness()
{
  for (let i = 0; i < population.length; i++)
  {
    let d = calcDistance(cities, population[i])
    if (d < recordDistance)
    {
      recordDistance = d
      bestEver = population[i]
    }
    fitness[i] = 1 / (d+1)
  }
}

function normaliseFitness()
{
  let sum = 0
  for (let i = 0; i < fitness.length; i++)
  {
    sum += fitness[i]
  }
  for (let i = 0; i < fitness.length; i++)
  {
```

```
    fitness[i] = fitness[i] /sum
  }
}

function nextGeneration()
{
  let newPopulation = []
  for (let i = 0; i < population.length; i++)
  {
    let orderA = pickOne(population, fitness)
    let orderB = pickOne(population, fitness)
    let order = crossOver(orderA, orderB)
    mutate(order, 0.01)
    newPopulation[i] = order
  }
  population = newPopulation
}

function pickOne(list, prob)
{
  let index = 0
  let r = random(1)
  while (r > 0)
  {
    r = r - prob[index]
    index++
  }
  index--
  return list[index].slice()
}

function mutate(order, mutationRate)
{
```

```
  for (let i = 0; i < totalCities; i++)
  {
    if (random(1) < mutationRate)
    {
      let indexA = floor(random(order.length))
      let indexB = (indexA + 1) % totalCities
      swap(order, indexA, indexB)
    }
  }
}
```

Notes

This won't run until we have created the crossover function. This is to demonstrate the principle. Whether it really enhances performance is unclear at this stage but in theory it should. The only way to really test is to run the same set of cities and compare their performance with and without crossover.

Sketch H8.3 crossover

Creating the crossOver() function. There are many ways of doing crossover eg we could take half of one and half of the other and combining those two halves. The problem here is that you could end up visiting the same city twice. The array for the dna must have in it different numbers you can't have two 3's for instance.

So we try this method. We sat by picking a random element in orderA

ga.js

```
function calculateFitness()
{
  for (let i = 0; i < population.length; i++)
  {
    let d = calcDistance(cities, population[i])
    if (d < recordDistance)
    {
      recordDistance = d
      bestEver = population[i]
    }
    fitness[i] = 1 / (d+1)
  }
}

function normaliseFitness()
{
  let sum = 0
  for (let i = 0; i < fitness.length; i++)
  {
    sum += fitness[i]
  }
  for (let i = 0; i < fitness.length; i++)
  {
```

```
    fitness[i] = fitness[i] /sum
  }
}

function nextGeneration()
{
  let newPopulation = []
  for (let i = 0; i < population.length; i++)
  {
    let orderA = pickOne(population, fitness)
    let orderB = pickOne(population, fitness)
    let order = crossOver(orderA, orderB)
    mutate(order, 0.01)
    newPopulation[i] = order
  }
  population = newPopulation
}

function pickOne(list, prob)
{
  let index = 0
  let r = random(1)
  while (r > 0)
  {
    r = r - prob[index]
    index++
  }
  index--
  return list[index].slice()
}

function mutate(order, mutationRate)
{
```

```
  for (let i = 0; i < totalCities; i++)
  {
    if (random(1) < mutationRate)
    {
      let indexA = floor(random(order.length))
      let indexB = (indexA + 1) % totalCities
      swap(order, indexA, indexB)
    }
  }
}

function crossOver(orderA, orderB)
{
  let start = floor(random(orderA.length))
}
```

Sketch H8.4 new order slice

There are two indices start and end, neworder takes a slice from the orderA array between those two indices.

ga.js

```
function calculateFitness()
{
  for (let i = 0; i < population.length; i++)
  {
    let d = calcDistance(cities, population[i])
    if (d < recordDistance)
    {
      recordDistance = d
      bestEver = population[i]
    }
    fitness[i] = 1 / (d+1)
  }
}

function normaliseFitness()
{
  let sum = 0
  for (let i = 0; i < fitness.length; i++)
  {
    sum += fitness[i]
  }
  for (let i = 0; i < fitness.length; i++)
  {
    fitness[i] = fitness[i] /sum
  }
}
```

```
function nextGeneration()
{
  let newPopulation = []
  for (let i = 0; i < population.length; i++)
  {
    let orderA = pickOne(population, fitness)
    let orderB = pickOne(population, fitness)
    let order = crossOver(orderA, orderB)
    mutate(order, 0.01)
    newPopulation[i] = order
  }
  population = newPopulation
}

function pickOne(list, prob)
{
  let index = 0
  let r = random(1)
  while (r > 0)
  {
    r = r - prob[index]
    index++
  }
  index--
  return list[index].slice()
}

function mutate(order, mutationRate)
{
  for (let i = 0; i < totalCities; i++)
  {
    if (random(1) < mutationRate)
    {
```

```
      let indexA = floor(random(order.length))
      let indexB = (indexA + 1) % totalCities
      swap(order, indexA, indexB)
    }
  }
}

function crossOver(orderA, orderB)
{
  let start = floor(random(orderA.length))
  let end = floor(random(start +1, orderA.length))
  let neworder = orderA.slice(start, end)
}
```

Sketch H8.6 new order array

We now loop through the orderB and change it with the neworder array

ga.js

```
function calculateFitness()
{
  for (let i = 0; i < population.length; i++)
  {
    let d = calcDistance(cities, population[i])
    if (d < recordDistance)
    {
      recordDistance = d
      bestEver = population[i]
    }
    fitness[i] = 1 / (d+1)
  }
}

function normaliseFitness()
{
  let sum = 0
  for (let i = 0; i < fitness.length; i++)
  {
    sum += fitness[i]
  }
  for (let i = 0; i < fitness.length; i++)
  {
    fitness[i] = fitness[i] /sum
  }
}
```

```
function nextGeneration()
{
  let newPopulation = []
  for (let i = 0; i < population.length; i++)
  {
    let orderA = pickOne(population, fitness)
    let orderB = pickOne(population, fitness)
    let order = crossOver(orderA, orderB)
    mutate(order, 0.01)
    newPopulation[i] = order
  }
  population = newPopulation
}

function pickOne(list, prob)
{
  let index = 0
  let r = random(1)
  while (r > 0)
  {
    r = r - prob[index]
    index++
  }
  index--
  return list[index].slice()
}

function mutate(order, mutationRate)
{
  for (let i = 0; i < totalCities; i++)
  {
    if (random(1) < mutationRate)
    {
```

```
      let indexA = floor(random(order.length))
      let indexB = (indexA + 1) % totalCities
      swap(order, indexA, indexB)
    }
  }
}

function crossOver(orderA, orderB)
{
  let start = floor(random(orderA.length))
  let end = floor(random(start +1, orderA.length))
  let neworder = orderA.slice(start, end)
  for (let i = 0; i < orderB.length; i++)
  {
    let city = orderB[i]
    if (!neworder.includes(city))
    {
      neworder.push(city)
      console.log(neworder)
    }
  }
  return neworder
}
```

Notes

Remember that it is trying to find the optimal where and when it can

Challenges

1. Change the number of cities
2. Change the population
3. Change the mutation rate
4. Make a game out of where someone has to join the cities manually and see if they can beat the algorithm.