GAs to Solve the TSP

Jacob House // Nabil Miri
Omar Mohamed // Hassan El-Khatib

Computer Science 3201
Fall 2018

The Team

Omar Mohamed	Project management Programmer
Nabil Miri	Algorithm implementation Debugging
Jacob House	Technical management Code quality control
Hassan El-Khatib	Programmer

Outline

- Our Approach
  - Population Size
  - Mating Pool Size
- Fitness Scoring
  - Euclidean Distance in $\mathbb{R}^2$
  - Individual Fitness
- Crossover
  - The Inver-Over Crossover Operator
- Mutation
  - The Scramble Mutation Operator
- Demonstration
- Any Questions?

MEMORIAL UNIVERSITY

Our Approach

Population Size

- For a route with n cities, we have

$$R := n \cdot (n-1) \cdots 3 \cdot 2 \cdot 1 = n!$$

  possible routes that cover all cities
- As n grows, so does R
- Population size P should also grow with n
- We define $P := 2n$ and choose P (not necessarily distinct) permutations of the set $\{0, 1, \ldots, n-1\}$ as the population

MEMORIAL UNIVERSITY

Our Approach

Population Size

- For a route with n cities, we have

$$R := n \cdot (n-1) \cdots 3 \cdot 2 \cdot 1 = n!$$

  possible routes that cover all cities
- As n grows, so does R
- Population size P should also grow with n
- We define $P := 2n$ and choose P (not necessarily distinct) permutations of the set $\{0, 1, \ldots, n-1\}$ as the population

MEMORIAL UNIVERSITY

Our Approach

Population Size

- For a route with n cities, we have

$$R := n \cdot (n-1) \cdots 3 \cdot 2 \cdot 1 = n!$$

possible routes that cover all cities

- As n grows, so does R
- Population size P should also grow with n
- We define $P := 2n$ and choose P (not necessarily distinct) permutations of the set $\{0, 1, \ldots, n-1\}$ as the population

MEMORIAL UNIVERSITY

- For a route with n cities, we have

$$R := n \cdot (n-1) \cdots 3 \cdot 2 \cdot 1 = n!$$

  possible routes that cover all cities
- As n grows, so does R
- Population size P should also grow with n
- We define $P := 2n$ and choose P (not necessarily distinct) permutations of the set $\{0, 1, \ldots, n-1\}$ as the population

MEMORIAL UNIVERSITY

Our Approach

Mating Pool Size

- Due to the large number of permutations of cities $c_1, c_2, c_3, \ldots c_n$, many of our candidate solutions are likely very low in fitness (*i.e.*, their total distance is very high)
- Define the mating pool size M to be

$$M := \left\lfloor \frac{1}{2} \cdot P \right\rfloor$$

- Due to the large number of permutations of cities $c_1, c_2, c_3, \ldots c_n$, many of our candidate solutions are likely very low in fitness (*i.e.,* their total distance is very high)
- Define the mating pool size M to be

$$M := \left\lfloor \frac{1}{2} \cdot P \right\rfloor$$

Fitness Scoring

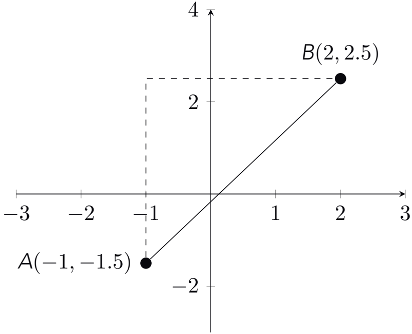
Euclidean Distance in $\mathbb{R}^2$

- Euclidean distance is computed using the formula

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

- Line $\overrightarrow{AB}$ measures

$$\begin{aligned} \left\|\overrightarrow{AB}\right\| &= \sqrt{(2+1)^2 + (2.5+1.5)^2} \\ &= \sqrt{9+16} \\ &= 5 \end{aligned}$$



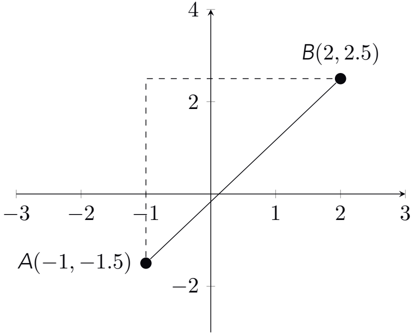
Fitness Scoring
Euclidean Distance in $\mathbb{R}^2$

- Euclidean distance is computed using the formula

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

- Line $\overrightarrow{AB}$ measures

$$\left\|\overrightarrow{AB}\right\| = \sqrt{(2+1)^2 + (2.5+1.5)^2}$$
$$= \sqrt{9+16}$$
$$= 5$$



Fitness Scoring
Individual Fitness

- Let I_m with $0 \leqslant m < P$ be a candidate solution of the form

$$I_m = \left(C_{c_m(\bar{1})}, C_{c_m(\bar{2})}, C_{c_m(\bar{3})}, \ldots, C_{c_m(\bar{n})}\right),$$

where $c_m \colon \mathbb{Z}_n \to \{0, 1, 2, \ldots, n-1\}$ is a bijection between congruence classes of indices of the n-tuple I_m and the indices of cities.

- For example, if $n = 7$ and c_1 is defined by

$$c_1 : \bar{0} \mapsto 4 \qquad c_1 : \bar{1} \mapsto 6 \qquad c_1 : \bar{2} \mapsto 2 \qquad c_1 : 3 \mapsto 1$$
$$c_1 : \bar{4} \mapsto 3 \qquad c_1 : \bar{5} \mapsto 5 \qquad c_1 : \bar{6} \mapsto 0$$

then I_1 looks like

$$I_1 = (C_4, C_6, C_2, C_1, C_3, C_5, C_0)\,.$$

MEMORIAL UNIVERSITY

- Let I_m with $0 \leqslant m < P$ be a candidate solution of the form

$$I_m = \left(C_{c_m(\bar{1})}, C_{c_m(\bar{2})}, C_{c_m(\bar{3})}, \ldots, C_{c_m(\bar{n})}\right),$$

  where $c_m \colon \mathbb{Z}_n \to \{0, 1, 2, \ldots, n-1\}$ is a bijection between congruence classes of indices of the n-tuple I_m and the indices of cities.

- For example, if $n = 7$ and c_1 is defined by

$$c_1 : \bar{0} \mapsto 4 \qquad c_1 : \bar{1} \mapsto 6 \qquad c_1 : \bar{2} \mapsto 2 \qquad c_1 : 3 \mapsto 1$$
$$c_1 : \bar{4} \mapsto 3 \qquad c_1 : \bar{5} \mapsto 5 \qquad c_1 : \bar{6} \mapsto 0$$

  then I_1 looks like

$$I_1 = (C_4, C_6, C_2, C_1, C_3, C_5, C_0)\,.$$

- Then define I_m's overall fitness score $F(I_m)$, to be the summation

$$F(I_m) := \sum_{j=0}^{n-1} \left\| \overrightarrow{C_{c_m(\bar{j})} C_{c_m(\overline{j+1})}} \right\|,$$

  where $\left\| \overrightarrow{C_{c_m(\bar{j})} C_{c_m(\overline{j+1})}} \right\|$ is the Euclidean distance between city $C_{c_m(\bar{j})}$ and the following city on route m, $C_{c_m(\overline{j+1})}$.
- Hence, the fittest individuals have the *lowest* score.

Fitness Scoring

Individual Fitness

- Then define I_m's overall fitness score $F(I_m)$, to be the summation

$$F(I_m) := \sum_{j=0}^{n-1} \left\| \overrightarrow{C_{c_m(\bar{j})} C_{c_m(\overline{j+1})}} \right\|,$$

where $\left\| \overrightarrow{C_{c_m(\bar{j})} C_{c_m(\overline{j+1})}} \right\|$ is the Euclidean distance between city $C_{c_m(\bar{j})}$ and the following city on route m, $C_{c_m(\overline{j+1})}$.

- Hence, the fittest individuals have the *lowest* score.

MEMORIAL UNIVERSITY

For our advanced technique, we have chosen to implement the *inver-over* crossover operator which functions according to the following algorithm.

1. Pick an individual $parent_1$ and copy it to *child*
2. Then pick two loci from $parent_1$ that depend on another individual $parent_2$ from the population
3. Invert everything between these loci in *child*
4. Repeat this process with the resulting offspring and another individual $parent_i$ until a stopping condition is reached

So *inver-over* is a multi-parent crossover operator.

MEMORIAL UNIVERSITY

For our advanced technique, we have chosen to implement the *inver-over* crossover operator which functions according to the following algorithm.

1. Pick an individual $parent_1$ and copy it to *child*
2. Then pick two loci from $parent_1$ that depend on another individual $parent_2$ from the population
3. Invert everything between these loci in *child*
4. Repeat this process with the resulting offspring and another individual $parent_i$ until a stopping condition is reached

So *inver-over* is a multi-parent crossover operator.

For our advanced technique, we have chosen to implement the *inver-over* crossover operator which functions according to the following algorithm.

1. Pick an individual $parent_1$ and copy it to *child*
2. Then pick two loci from $parent_1$ that depend on another individual $parent_2$ from the population
3. Invert everything between these loci in *child*
4. Repeat this process with the resulting offspring and another individual $parent_i$ until a stopping condition is reached

So *inver-over* is a multi-parent crossover operator.

MEMORIAL UNIVERSITY

For our advanced technique, we have chosen to implement the *inver-over* crossover operator which functions according to the following algorithm.

1. Pick an individual $parent_1$ and copy it to *child*
2. Then pick two loci from $parent_1$ that depend on another individual $parent_2$ from the population
3. Invert everything between these loci in *child*
4. Repeat this process with the resulting offspring and another individual $parent_i$ until a stopping condition is reached

So *inver-over* is a multi-parent crossover operator.

For our advanced technique, we have chosen to implement the *inver-over* crossover operator which functions according to the following algorithm.

1. Pick an individual $parent_1$ and copy it to *child*
2. Then pick two loci from $parent_1$ that depend on another individual $parent_2$ from the population
3. Invert everything between these loci in *child*
4. Repeat this process with the resulting offspring and another individual $parent_i$ until a stopping condition is reached

So *inver-over* is a multi-parent crossover operator.

Mutation

The Scramble Mutation Operator

The scramble mutation operator, given an individual represented as a sequence of integers, typically performs the following.

1. Picks two loci to form a segment
2. Randomly shuffles all information within the selected segments
3. Returns the mutated individual

However, we have slightly modified this operator…

Mutation

The Scramble Mutation Operator

The scramble mutation operator, given an individual represented as a sequence of integers, typically performs the following.

1. Picks two loci to form a segment
2. Randomly shuffles all information within the selected segments
3. Returns the mutated individual

However, we have slightly modified this operator…

The scramble mutation operator, given an individual represented as a sequence of integers, typically performs the following.

1. Picks two loci to form a segment
2. Randomly shuffles all information within the selected segments
3. Returns the mutated individual

However, we have slightly modified this operator…

We have added another condition to the operator…

> *Define a mutation factor $m \in (0, 1)$. Then the distance between the two chosen loci (i.e., the size of the mutation) can be no less than m multiplied by the length of the individual n.*

In other words, we have enforced that the product $m \cdot n$ is the infimum of the possible *severities* of the mutation.

We have added another condition to the operator:

> *Define a mutation factor $m \in (0, 1)$. Then the distance between the two chosen loci* (i.e., *the size of the mutation) can be no less than m multiplied by the length of the individual n.*

In other words, we have enforced that the product $m \cdot n$ is the infimum of the possible *severities* of the mutation.

MEMORIAL UNIVERSITY

We have added another condition to the operator:

> *Define a mutation factor m* $\in (0, 1)$. *Then the distance between the two chosen loci* (i.e., *the size of the mutation) can be no less than m multiplied by the length of the individual n.*

In other words, we have enforced that the product $m \cdot n$ is the infimum of the possible *severities* of the mutation.

Demonstration

Any Questions?

MEMORIAL UNIVERSITY