résoudre le problème ‘Traveling salesman problem ‘ avec l'algorithme génétique

by :rahmoun oussama

Table des matières

1-Introduction :....3
1-TSP :....3
2-algorithemes génétiques :....3
2-Principe des algorithmes génétiques :....4
1-La mise en œuvre d’un algorithme génétique :....4
2-Vocabulaires des algorithmes génétiques et Fonctionnement:....4
2-1-La population initiale :....5
2-2-Fonction d'évaluation (fitness):....5
2-3- Le croissement :....5
2-5- La sélection :....7
3-Application des algorithmes génétiques au problème du ramassage scolaire :....7
3-1- problématique :....7
3-2-choix des outils :....7
3-3 module :....8
3-4 Algorithm:....9
4- Description de l’interface développée :....11
5-conclusion :....13

Tableau figures :

Figure 1: Organigramme de l'algorithme génétique 4
Figure 2:croisement 6
Figure 3: exemple mutation 6
Figure 4:intrface du application.....11
Figure 5: variance de cout 11
Figure 6:chemin plus court.....12
Figure 7:chemin dans maps.....12

1-Introduction :

1-TSP :

Le problème du voyageur de commerce (TSP) a été formulé en 1930, mais c'est encore aujourd'hui l'un des problèmes d'optimisation combinatoire les plus étudiés.

En 1972, 'Richard Karp' a prouvé que le problème du cycle hamiltonien était NP-complet qui est une classe de problèmes d'optimisation combinatoire. Cela signifie que le TSP était NP-difficile et que la complexité du calcul du meilleur itinéraire augmentera de façon exponentielle lorsque davantage de destinations s'ajouteront au problème.

Par exemple, il y a trois itinéraires possibles s'il y a quatre villes, mais il y a 360 itinéraires possibles s'il y a six villes. En effet, les scientifiques ne calculent pas seulement le chemin le plus efficace, mais aussi celui qui fonctionne. Ce calcul utilise l'approche de la force brute pour résoudre le problème en déterminant chaque possibilité, puis en choisissant la meilleure. En d'autres termes, recherchez chaque chemin unique que le vendeur pourrait emprunter.

Même si les difficultés de calcul augmentent avec chaque ville ajoutée à l'itinéraire, les informaticiens ont pu calculer la solution optimale à ce problème pour des milliers de villes depuis le début des années 90. Par exemple, de nombreuses villes d'un État américain pourraient faire partie de la zone de livraison du grand fournisseur de services logistiques . Déterminer l'itinéraire le plus court entre tous les arrêts que le véhicule doit effectuer au quotidien permettrait d'économiser beaucoup de temps et d'argent.

Donc pour résoudre ce problème nous allons utiliser l'algorithme génétique

2-algorithemes génétiques :

L'algorithme génétique est une algorithme d'optimisation, reposant le principe de la sélection naturelle, développé dans les années 70 par John H. Holland. Entrant dans la catégorie des algorithmes évolutionnistes, il fait appel à des algorithmes générés aléatoirement, dans un univers défini, qui interagissent entre eux et avec l'univers. Ils ont la capacité d'incorporer des informations présentes au sein de l'univers, et donc d'évoluer (de "muter") en fonction de ces inputs.

Il est important d'insister sur l'aspect aléatoire : ces algorithmes n'ont pas vocation à faire de tâche particulière. Ils se contentent d'exister et d'évoluer en fonction des éléments intégrés. Ces algorithmes sont ensuite confrontés à une sélection, en insérant un score à atteindre. A l'issue du test, seuls les algorithmes les mieux notés sont conservés, et confrontés à de nouveaux algorithmes générés aléatoirement. Le processus est répété de génération d'algorithmes en génération, pendant un certain temps. Avec ce double processus de mutation-sélection, des algorithmes de plus en plus performants finissent par émerger, de façon quasi "spontanée".

2-Principe des algorithmes génétiques :

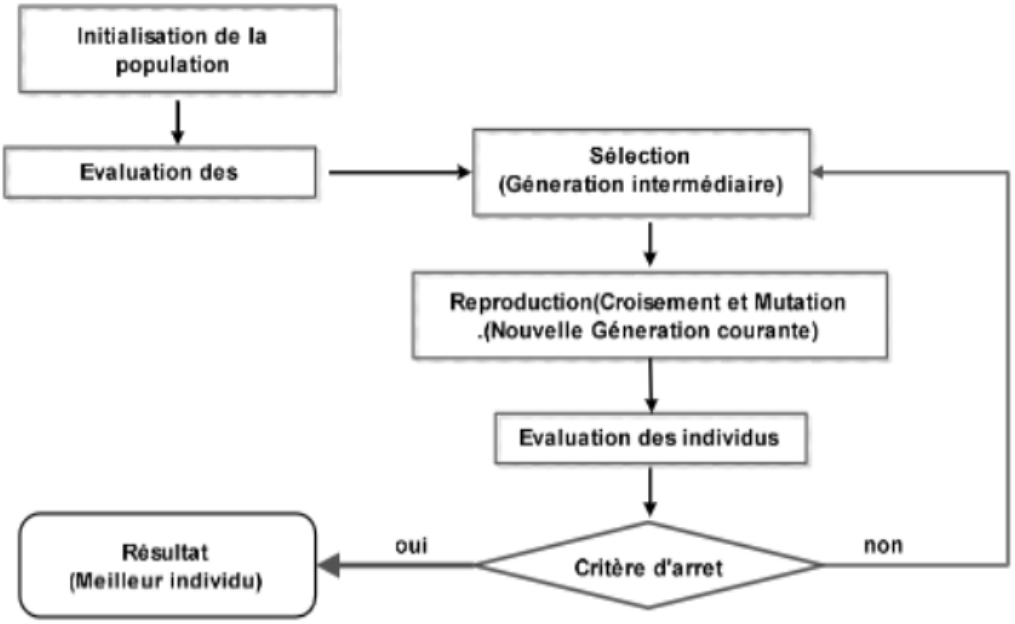
1-La mise en œuvre d'un algorithme génétique :

La mise en œuvre d'un algorithme génétique est réalisée suivant les étapes suivantes :

1. Création d'une population initiale.
2. Evaluation des individus de la population.
3. Sélection des meilleurs individus.
4. Reproduction (Croissement et mutation).
5. Formation d'une nouvelle génération.

La figure suivante montre l'organigramme de fonctionnement d'un algorithme génétique :

Figure 1:



Organigramme de l'algorithme génétique

2-Vocabulaires des algorithmes génétiques et Fonctionnement:

Dans cette section nous introduisons quelques vocabulaires utilisés dans la mise en œuvre des algorithmes génétiques.

- Individu : représenté par un chromosome (génome).
- Un chromosome est une chaîne de gènes.
- Génotype : l'ensemble des gènes représentés par un chromosome.
- Phénotype : l'ensemble des valeurs observables prises par chaque gène
- Fonction d'adaptation : fitness.
- Opération de reproduction :

Le croissement
La mutation

- Génération : l’ensemble de la population à un moment donné du processus.

2-1-La population initiale :

Le choix de la population initiale d'individus conditionne fortement la rapidité de l'algorithme. Pour générer la population initiale, nous avons, deux possibilités. Dans le cas où aucune information sur la position de la solution n'est disponible, le but est de recouvrir au mieux l'espace d'état. Une génération aléatoire est donc engendrée par des tirages uniformes sur chaque gène du chromosome. Dans le cas où l'utilisateur connaîtrait un sous-domaine de l'espace d'état où la solution se trouve, la population initiale est générée dans ce sous-espace.

Dans le cas d'une génération engendrée de façon aléatoire, il existe deux éventualités. Soit il est possible de savoir à l'avance si un point respecte les contraintes du problème d'optimisation, la population est alors générée aléatoirement dans le domaine admissible. Soit il n'est pas possible de le savoir, le respect des contraintes sera alors assuré via l'ajout d'une pénalisation dans la fonction- objectif et la population est uniformément distribuée sur tout l'espace d'état.

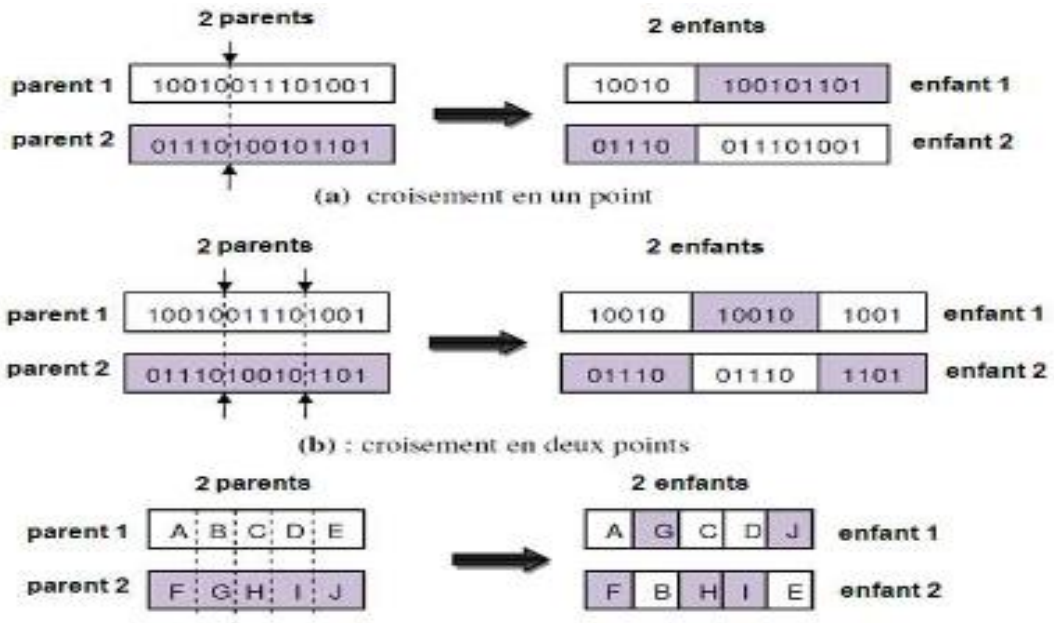
2-2-Fonction d'évaluation (fitness):

L'évaluation de l'adaptation de chaque individu à l'environnement est réalisée au moyen d'une fonction d'adaptation (fitness). Cette fonction attribue à chaque individu une valeur qui représente son niveau d'adaptation. La fonction d’adaptation peut affecter directement la qualité des résultats obtenus par l'AG ainsi que le temps d'exécution.

2-3- Le croissement :

La naissance d'un nouvel individu, nécessite la prise aléatoire d'une partie des gènes de chacun des deux parents. Ce phénomène, issu de la nature est appelé croisement (crossover). Il s'agit d'un processus essentiel pour explorer l'espace des solutions possibles. Une fois la sélection terminée, les individus sont aléatoirement répartis en couples. Les chromosomes parents sont alors copiés et recombinés afin de produire chacun deux descendants ayant des caractéristiques issues des deux parents. Dans le but de garder quelques individus parents dans la prochaine population, on associe à l'algorithme génétique une probabilité de croisement, qui permet de décider si les parents seront croisés entre eux ou s'ils seront tout simplement recopiés dans la population suivante. Il existe plusieurs types de croissement parmi lesquels on trouve : le croissement en 1 point, le croissement en deux points et le croissement en N points, ces types sont résumés dans la figure suivante :

Figure 2:croisement



2-4- La mutation :

L'opérateur de mutation, en générant de nouveaux gènes, a pour rôle de permettre d'explorer la totalité (en théorie) de l'espace d'état, ce qui correspond à la propriété d'ergodicité de parcours d'espace, essentielle aux AG pour leurs propriétés de convergence. En effet les preuves théoriques de convergence des AG peuvent fonctionner sans croisement, mais pas sans mutation.

L'opérateur de mutation fonctionne comme suit. Pour les problèmes discrets, un gène du chromosome est tiré aléatoirement et sa valeur est remplacée par une des autres valeurs possibles (tirée aléatoirement elle aussi). Dans le cas des problèmes continus, le gène est également tiré aléatoirement, et remplacé par une valeur aléatoire du domaine d'extension des gènes (espace d'état). La figure suivante montre un exemple de mutation.

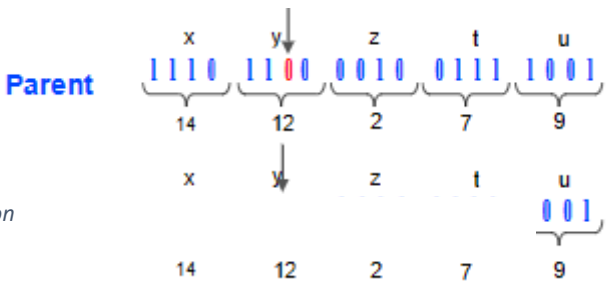


Figure 3: exemple mutation

2-5- La sélection :

L'opérateur de sélection est chargé de "favoriser les meilleurs individus. Plus formellement, l'opérateur de sélection va générer à partir de la population courante une nouvelle population par copie des individus choisis de la population courante. La copie des chaînes s'effectue en fonction des valeurs de la fonction d'adaptation. Ce procédé permet de donner aux meilleures chaînes, une probabilité élevée de contribuer à la génération suivante. Cet opérateur est bien entendu une version artificielle de la sélection naturelle, la survie darwinienne des chaînes les plus adaptées. Plusieurs stratégies sont possibles pour effectuer une telle sélection parmi lesquelles nous abordons :

✓ La sélection par classement : elle consiste à ranger les individus de la population dans un ordre croissant (ou décroissant selon l'objectif) et à retenir un nombre fixé de génotypes. Ainsi, seuls les individus les plus forts sont conservés. L'inconvénient majeur de cette méthode est la convergence prématurée de l'algorithme génétique.

✓ La sélection par tournoi : le tournoi le plus simple consiste à choisir aléatoirement un nombre k d'individus dans la population et à sélectionner celui qui a la meilleure performance. Les individus qui participent à un tournoi sont remis ou sont retirés de la population, selon le choix de l'utilisateur. Avec le tournoi binaire, sur deux individus en compétition, le meilleur gagne avec une probabilité p € [0, 5; 1];

3-Application des algorithmes génétiques au problème du ramassage scolaire :

3-1- problématique :

L'algorithme génétique peut être utilisé pour résoudre le "problème du voyageur de commerce" => « ramassage scolaire ». La problématique à résoudre ? Passer par un ensemble de points (ou d'adresses) en parcourant la distance globale la plus faible possible

3-2-choix des outils :

Nous avons choisi visual studio code comme espace de travail et lagunage python pour notre implémentation

Et pour les librairies :

```
geopy
math
random
matplotlib
numpy
tkinter
```

```
folium
webbrowser
```

3-3 module :

Node classe : Permet de stocker en mémoire un objet contenant les paramètres d'une location

```
class Node:  # Node = Location = Point
    def __init__(self, id, x, y):
        self.x = float(x)
        self.y = float(y)
        self.id = int(id)
```

Chromosome classe : Contient une liste du node qui présente un chemin

```
class Chromosome:
    def __init__(self, node_list):

        self.chromosome = node_list

        chr_representation = []
        for i in range(0, len(node_list)):
            chr_representation.append(self.chromosome[i].id)
        self.chr_representation = chr_representation

        distance = 0
        for j in range(1, len(self.chr_representation) - 1):  # get distances from the matrix
            distance += matrix[self.chr_representation[j] -
                               1][self.chr_representation[j + 1]-1]
        self.cost = distance
        if self.cost > 0:
            self.fitness_value = 1 / self.cost+0.001
        else:
            print(self.cost)
            print("the cost is going to here we have to stop The algorithm fails because each new
generation goes badly ")
            sys.exit()
```

Le chromosome calcule le coût pour lui-même à partir d'un tableau qui stocke la distance entre chaque Node et les autres Node

```
def create_distance_matrix(node_list, N):
    matrix = [[0 for _ in range(N)] for _ in range(N)]

    # classical matrix creation with two for loops
    for i in range(0, len(matrix)-1):
        # print(i, len(matrix))
        for j in range(0, len(matrix)-1):
            print(j, i, len(node_list), node_list[i], len(
                matrix), node_list[len(matrix)-1])
            matrix[node_list[i].id][node_list[j].id] = great_circle((float(node_list[i].x), float(
                node_list[i].y)), (float(node_list[j].x), float(node_list[j].y))).km
    return matrix
```

3-4 Algorithm:

Etap1: Cree population initial:

```
# initialization
def initialization(data, pop_size):
    initial_population = []
    for i in range(0, pop_size):  # creation of chromosomes as much as population size
        temp = create_random_list(data)
        print(temp)
        new_ch = ch.Chromosome(temp)
        initial_population.append(new_ch)
    return initial_population
```

```
def create_random_list(n_list):
    # the start and the end point should be the same, so we have to keep the first point
stored before shuffling
    start = n_list[0]

    temp = n_list[1:]
    temp = random.sample(temp, len(temp))  # shuffle the node list

    # add the start point to the beginning of the chromosome
    temp.insert(0, start)
    # add the start point to the end, because the route should be ended where it started
    temp.append(start)
    return temp
```

pour crée la population le programme initial va charger des données et choisir au hasard une Location présentée par id et latitude et longitude et il va charger les 3 paramètres dans un nouvel objet Node et l'insérer dans une liste ‘node_listet’ puis l'insérer dans la liste initiale de la population initiale

Etap2 : créé une nouvelle génération a partir du ancien génération

```
def create_new_generation(previous_generation, mutation_rate):
    # This is for elitism. Keeping the best of the previous generation.
    new_generation = [find_best(previous_generation)]

    # Using two chromosomes and creating two chromosomes. So, iteration size will be half
of the population size!
    for a in range(0, int(len(previous_generation)/2)):
        parent_1 = selection(previous_generation)
        parent_2 = selection(previous_generation)

        # This will create node lists, we need Chromosome objects
        child_1, child_2 = crossover_mix(parent_1, parent_2)
```

```
        child_1 = ch.Chromosome(child_1)
        child_2 = ch.Chromosome(child_2)

        if random.random() < mutation_rate:
            mutated = mutation(child_1.chromosome)
            child_1 = ch.Chromosome(mutated)

        new_generation.append(child_1)
        new_generation.append(child_2)

    return new_generation
```

Pour créer une nouvelle génération, nous utiliserons le croisement et la mutation nous choisirons 2 chromosomes au hasard puis nous appliquerons le croisement et la mutation et nous chargerons l'enfant dans une liste

puis on va refaire l'opération de création d'une nouvelle génération jusqu'au nombre d'itérations numbers_of_generations

Etape 3:

afficher le changement de coût

Et montrer le chemin aussi le plus court

La fonction find_thebest retournera le meilleur chemin en calculant le coût de chaque chromosome et en choisissant le plus petit coût

```
# Finding the best chromosome of the generation based on the cost
def find_best(generation):
    best = generation[0]
    for n in range(1, len(generation)):
        if generation[n].cost < best.cost:
            best = generation[n]
    return best
class Chromosome:
    def __init__(self, node_list):
        self.chromosome = node_list

        chr_representation = []
        for i in range(0, len(node_list)):
            chr_representation.append(self.chromosome[i].id)
        self.chr_representation = chr_representation

        distance = 0
        for j in range(1, len(self.chr_representation) - 1):  # get distances from the
matrix
            distance += matrix[self.chr_representation[j] -
                               1][self.chr_representation[j + 1]-1]
        self.cost = distance

        self.fitness_value = 1 / self.cost
```

4- Description de l’interface développée :

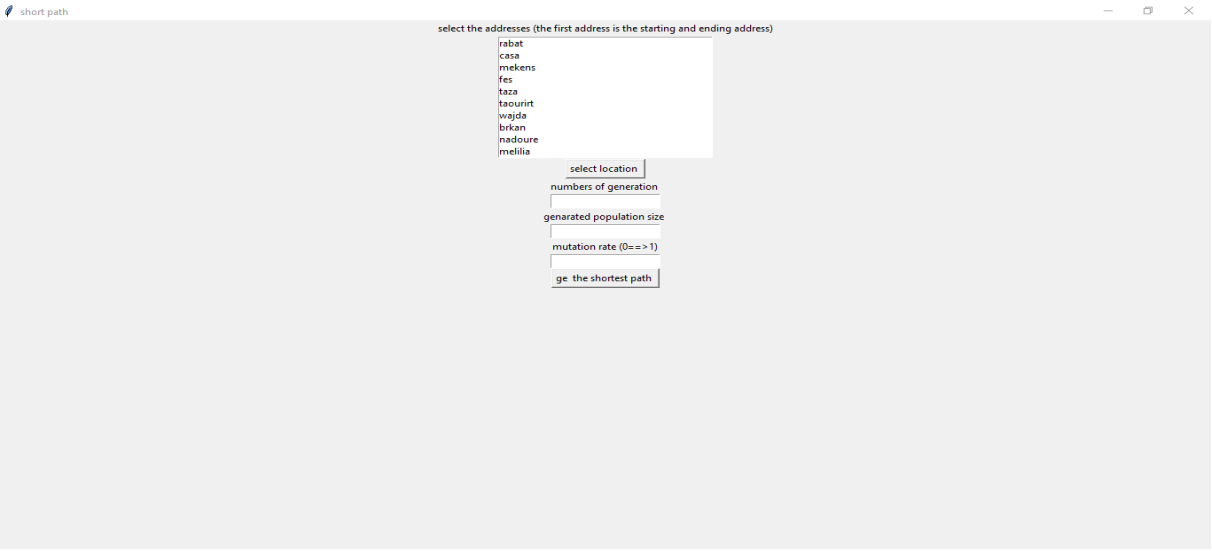


Figure 4:intrface du application

Cette interface affiche les adresse du locations puis utilisateur choisira les adresses pour trouver le chemin le plus court après il y a une entrée du nombre de la génération à générer et une autre de la taille de la population initiale à générer et à la fin une entrée pour l'opération de mutation

après avoir rempli les données initiales, cliquez sur le bouton obtenir le chemin le plus court

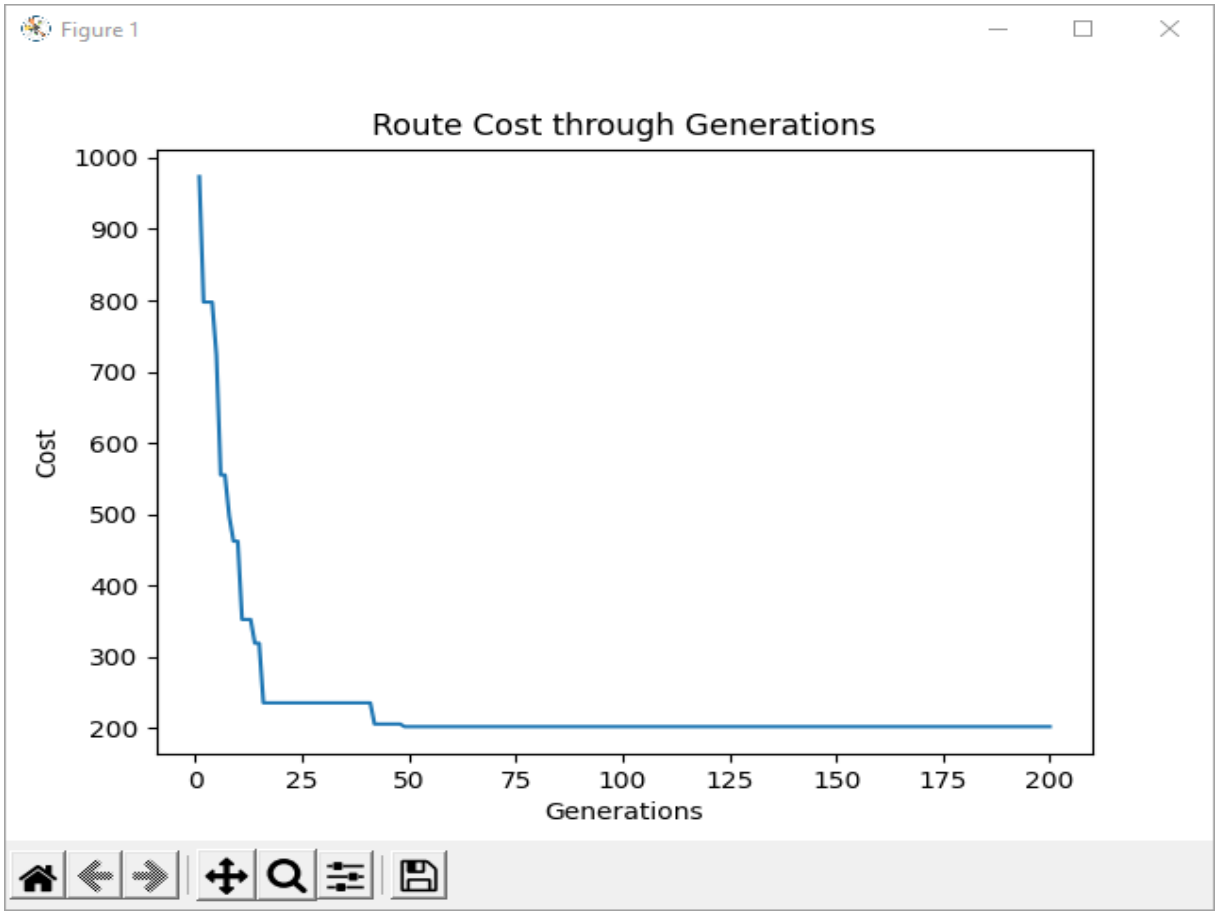


Figure 5: variance de cout

Ainsi, le programme affichera le changement de coût pour chaque nouvelle génération

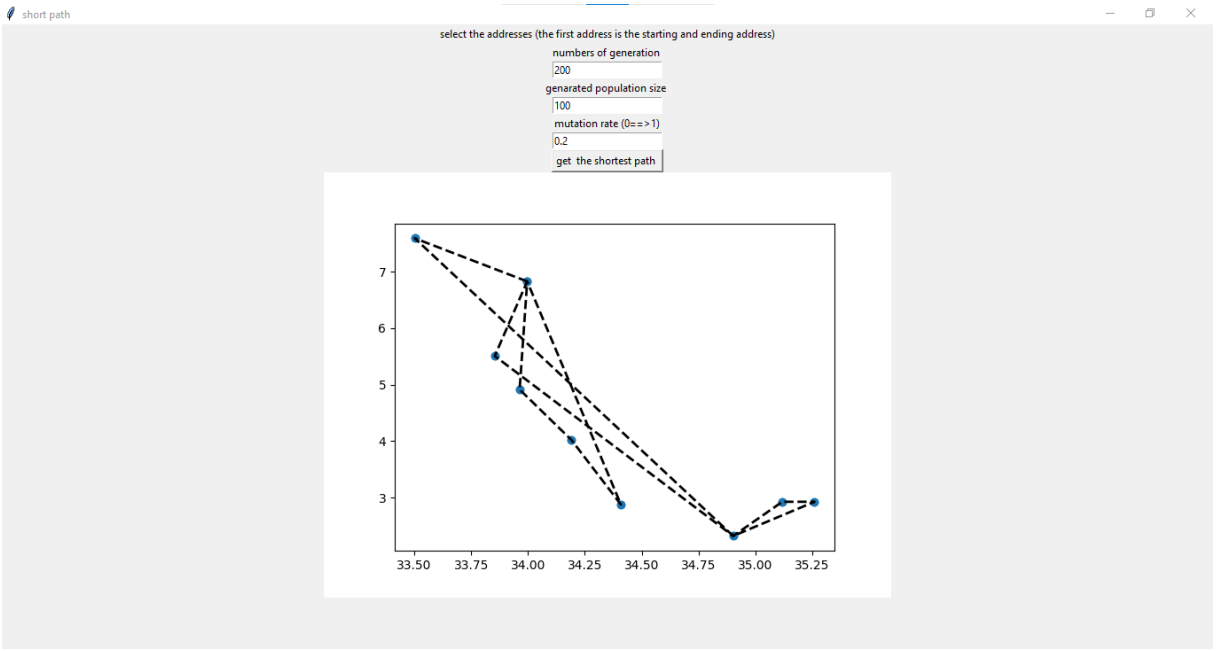


Figure 6:chemin plus court

Ensuite, pour chaque chromosome de la dernière génération, il affiche le chemin obtenu jusqu'au chemin le plus court donné par l'algorithme

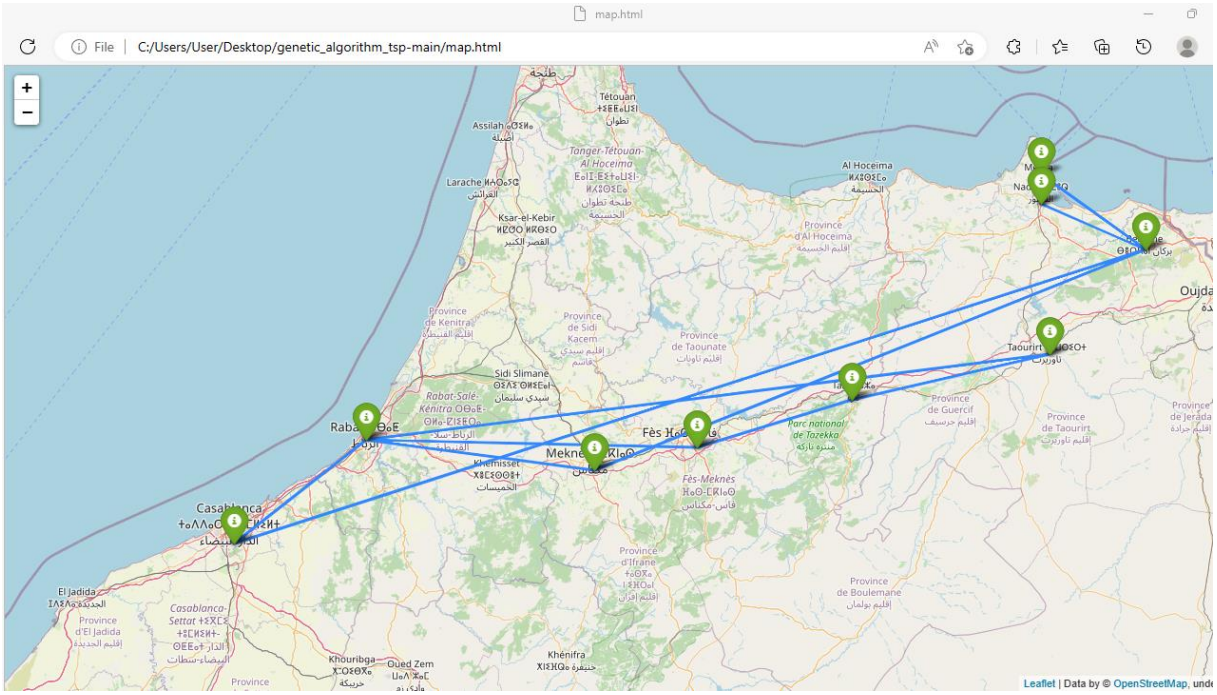


Figure 7:chemin dans maps

Et enfin, le programme ouvrira automatiquement le navigateur pour afficher le chemin sur ‘maps’

5-conclusion :

L'algorithme génétique, est un moyen de trouver une solution satisfaisante dans un délai raisonnable. Ce n'est pas la solution optimale qui elle mettrait des années a être calculées avec beaucoup d'individus en paramètre. Ce programme n'est pas vraiment optimisé, le but n'était pas non plus d'arriver à un résultat parfait. Mais tout de même, il fonctionne relativement bien.