APPENDIX

CODE

GeneticRoutines.py - C:\Users\kumar\Desktop\project\Genetic-Algorithm-TSP-Picking\libs\GeneticRoutines.py (3.6.2)

File Edit Format Run Options Window Help

```
from random import random, randint
from TSP import Route as Chromosome

def selection(pop):
    i = -1
    rand = random()
    while rand > 0:
        i += 1
        rand -= pop[i].chance
    return pop[i]


def mutation(population, rate):
    for chromosome in population:
        if random() < rate:
                r1 = randint(0, len(chromosome.genes) - 1)
                r2 = randint(0, len(chromosome.genes) - 1)
                chromosome.genes[r1], chromosome.genes[r2] = chromosome.genes[r2], chromosome.genes[r1]


def crossover(chromosome1, chromosome2):
    end = randint(0, len(chromosome1.genes))
    start = randint(0, end)
    section = chromosome1.genes[start:end]
    offspring_genes = list(gene if gene not in section else None for gene in chromosome2.genes)
    g = (x for x in section)
    for i, x in enumerate(offspring_genes):
        if x is None:
            offspring_genes[i] = next(g)
    offspring = Chromosome(offspring_genes, shuffled=True)

    return offspring
```

GeneticRoutines.py

```
class Route:
    def __init__(self, items, shuffled=False):
        self._genes = items[:]
        if not shuffled:
            shuffle(self._genes)
        self._fitness = None
        self.chance = None

    @property
    def fitness(self):
        distances = (Itemloc.distance(self._genes[i], self._genes[i + 1]) for i in range(-1, len(self._genes) - 1))
        self._fitness = 10 / sum(distances)
        return self._fitness

    @property
    def raw_fitness(self):
        return self._fitness

    @property
    def genes(self):
        return self._genes

    def copy(self):
        clone = Route(self._genes, shuffled=True)
        clone._fitness = self.raw_fitness
        clone.chance = None
        return clone

    def __repr__(self):
        return f'<Route ({self._fitness})>'

    def __str__(self):
        return '->'.join(repr(item) for item in self._genes)
```

TSP.py - C:\Users\kumar\Desktop\project\Genetic-Algorithm-TSP-Picking\TSP.py (3.6.2)

File Edit Format Run Options Window Help

```
from math import hypot
from random import shuffle

class Itemloc:
    def __init__(self, name: str, coords: tuple):
        self._name = name
        self._x, self._y = coords

    @property
    def name(self):
        return self._name

    @property
    def x(self):
        return self._x

    @property
    def y(self):
        return self._y

    @staticmethod
    def distance(item1, item2):
        return hypot(item2.x - item1.x, item2.y - item1.y)

    def __repr__(self):
        return f'{self._name}'


class Item:
    def __init__(self):
        self._items = []

    def add(self, x):
        if isinstance(x, list):
            self.items.extend(x)
        elif isinstance(x, Itemloc):
            self.items.append(x)
        else:
            assert False, "Wrong type"

    @property
    def items(self):
        return self._items
```

TSP.py

GeneticAlgorithm.py - C:\Users\kumar\Desktop\project\Genetic-Algorithm-TSP-Picking\libs\GeneticAlgorithm.py (3.6.2)

File Edit Format Run Options Window Help

```
from libs.GeneticRoutines import selection, mutation, crossover
import time

class GeneticAlgorithm:
    def __init__(self, size, mutation_rate=0.01, ptype=None, args=tuple()):
        assert ptype is not None, 'Population type cannot be None'
        assert type(args) == tuple, 'Arguments must be a tuple instead of ' + str(type(args))
        self._population = [ptype(*args) for _ in range(size)]
        self._mutation_rate = mutation_rate
        self._generation = 0
        self._fittest = self._population[0]
        self.evaluation()

    def individuals(self):
        for chromosome in self._population:
            yield chromosome

    def evaluation(self):
        fitness_sum = sum(chromosome.fitness for chromosome in self._population)
        for chromosome in self._population:
            chromosome.chance = chromosome.fitness / fitness_sum

    def best(self):
        return max(self._population, key=lambda k: k.fitness)

    @property
    def alltime_best(self):
        return self._fittest

    @property
    def generation(self):
        return self._generation

    def next_generation(self):
        new_population = []
        for _ in range(len(self._population)):
            chromosome1 = selection(self._population)
            chromosome2 = selection(self._population)
            new_population.append(crossover(chromosome1, chromosome2))
            mutation(new_population, self._mutation_rate)
        self._population = new_population
        self.evaluation()

    def run(self, seconds=5, reps=None):
        if reps is not None:
            assert isinstance(reps, int), 'Argument `reps` must be of integer type'

            for _ in range(reps - 1):
                pretender = self.best()
                if pretender.fitness > self._fittest.raw_fitness:
                    self._fittest = pretender.copy()

                self._generation += 1
                self.next_generation()
            pretender = self.best()
            if pretender.fitness > self._fittest.raw_fitness:
                self._fittest = pretender.copy()
            self._generation += 1
        else:
            t0 = time.time()
            while True:
                pretender = self.best()
                if pretender.fitness > self._fittest.raw_fitness:
                    self._fittest = pretender.copy()

                self._generation += 1
                if time.time() - t0 >= seconds:
                    break
                self.next_generation()
```

GeneticAlgorithm.py

mod-Graphic.py - C:\Users\kumar\Desktop\project\modified Project\Genetic-Algorithm-TSP-Picking\mod-Graphic.py (3.6.2)

File Edit Format Run Options Window Help

```
from tkinter import *
from TSP import *
from libs.GeneticAlgorithm import GeneticAlgorithm
import pygame
import matplotlib.pyplot as plt
import numpy as np

gen = []
fitn = []
List = []
varList = []
myApp = Tk()
myApp.title("Items-picking list")
myApp.geometry("1000x800")
d = {'Watch':(50,35),'Television':(37,39),'Bag':(26,35),'Book':(14,6),'Mobile':(7,18),'Laptop':(27,15),
     'Refrigerator':(9,12),'Air Conditioner':(42,36),'Pendrives':(16,8),'HDD':(31,37),'Flash card':(45,20),
     'Bottles':(35,8),'Fan':(9,24),'Ear phones':(5,24),'Electric Bulbs':(8,8),'Deodrants':(33,26),'Vaccum Cleaner':(32,40),
     'Sandals':(43,25),'Cameras':(29,24),'Power Banks':(46,27),'Blankets':(25,37),'Hair Dryer':(18,26),'Trimmer':(22,24),
     'Soaps':(18,20),'Washing Machine':(29,35),'Shoes':(20,41),'Goggles':(30,42),'Kerchiefs':(35,42),'Cosmetics':(44,21),
     'Shirts':(15,17),'Pants':(12,23),'Chairs':(23,32)}
p=[]

#myApp.configure(background='')


def map_items_onto_screen(items):
    for item in items:
        y = -int(15 * (item.x - 54))
        x = int(20 * (item.y - 3.5))
        yield (x, y)


def text_labels(items, population_size, mutation_rate):
    global arial_norm, arial_small
    arial_norm = pygame.font.SysFont('arial', 25)
    arial_small = pygame.font.SysFont('arial', 16)
    labels = []
    for item, (posx, posy) in zip(items, map_items_onto_screen(items)):
        labels.append((arial_norm.render(item.name, 1, (255, 255, 255)), (posx - 45, posy - 15)))
    labels.append((arial_small.render('Population size: {}'.format(population_size), 1, (255, 255, 255)), (1100, 10)))
    labels.append((arial_small.render('item count: {}'.format(len(items)), 1, (255, 255, 255)), (1100, 25)))
    labels.append((arial_small.render('Mutation rate: {}'.format(mutation_rate), 1, (255, 255, 255)), (1100, 40)))
    return labels
```

```
    ''' Main loop '''
    while refresh:
        ''' Event handling '''
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                refresh = False
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:
                    pause = True
                    while pause:
                        event2 = pygame.event.wait()
                        if event2.type == pygame.KEYDOWN and event2.key == pygame.K_SPACE:
                            pause = False
                        elif event2.type == pygame.QUIT:
                            pause = False
                            refresh = False

        ''' Genetic Algorithm process + statistics '''
        ga.run(reps=skipped_frames)
        current_best = ga.best()
        alltime_fittest = ga.alltime_best
        alltime_fitness = alltime_fittest.raw_fitness


        ''' Drawing part '''
        screen.fill((0, 0, 0))
        for point in map_items_onto_screen(WareHouse.items):
            pygame.draw.circle(screen, (255, 255, 255), point, 3)
        pygame.draw.aalines(screen, (100, 100, 25), True, list(map_items_onto_screen(current_best.genes)))
        pygame.draw.aalines(screen, (255, 255, 255), True, list(map_items_onto_screen(alltime_fittest.genes)))
                #gen.append(ga.generation)
                #fit.append(current_best.raw_fitness)
        x=ga.generation
        y=current_best.raw_fitness
        gen.append(x)
        fitn.append(y)
        screen.blit(arial_small.render('Generation: {}'.format(x), 1, (255, 255, 255)), (1100, 55))
        screen.blit(arial_small.render('Fitness: {:.4f}'.format(alltime_fitness), 1, (255, 255, 255)), (1100, 70))
        screen.blit(arial_small.render('Current Fitness: {:.4f}'.format(y), 1, (255, 255, 0)), (1100, 85))
        for label in stat_labels:
            screen.blit(label[0], label[1])
        pygame.display.flip()
```

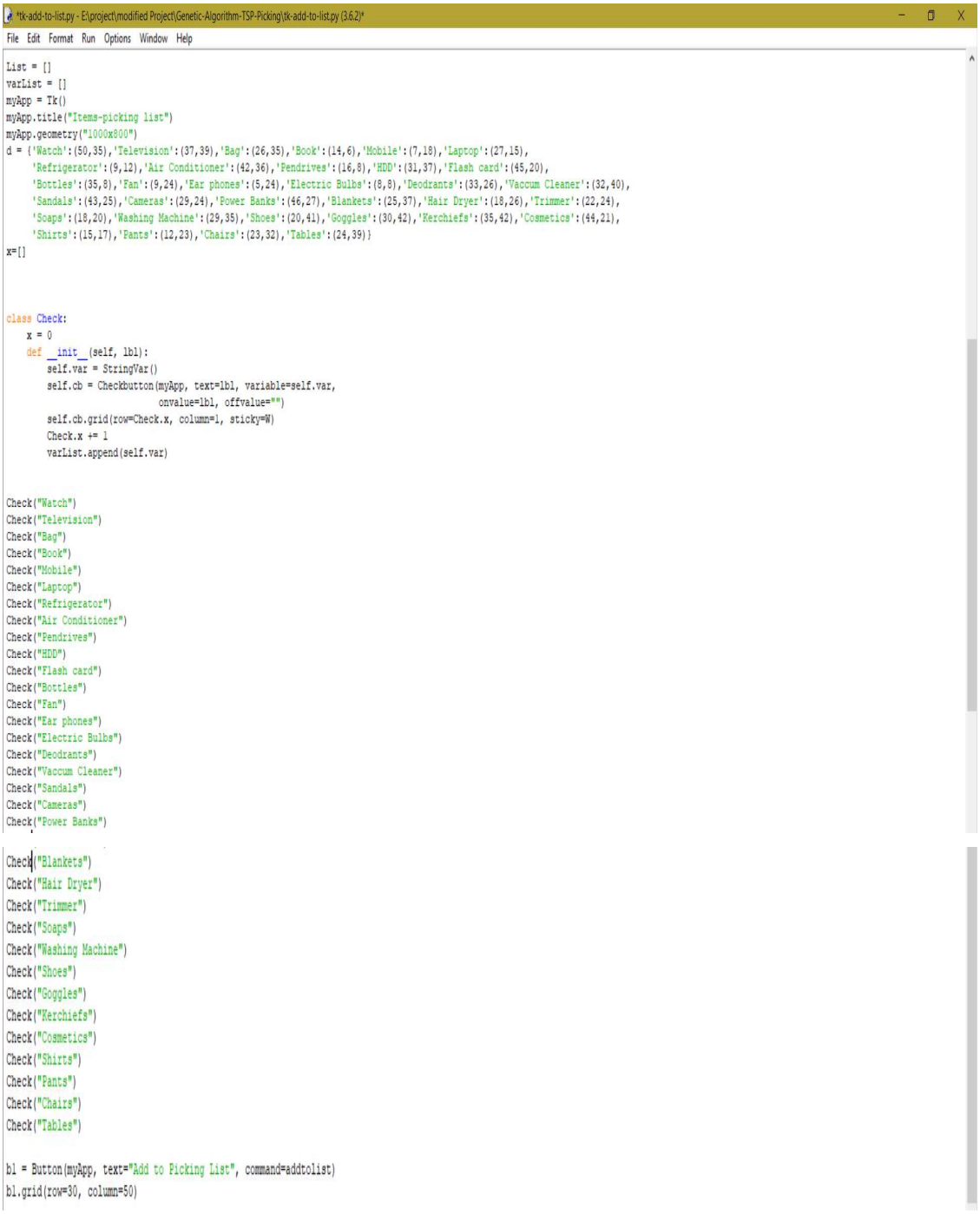
```
def addtolist():
    global List

    List = []
    for item in varList:
        if item.get() != "":
            List.append(item.get())

    for i in List:
        p.append(d[i])

    s=[]
    l=List
    for i in range(len(l)):
        s.append(Itemloc(l[i],p[i]))
    WareHouse = Item()

    WareHouse.add(s)

    population_size = 20
    mutation_rate = 0.01
    skipped_frames = 108

    pygame.init()
    pygame.display.set_caption('Picking Routes in warehouse')
    screen = pygame.display.set_mode((1300, 780))
    stat_labels = text_labels(WareHouse.items, population_size, mutation_rate)
    ga = GeneticAlgorithm(population_size, mutation_rate, ptype=Route, args=(WareHouse.items,))
    alltime_fittest = ga.alltime_best
    alltime_fitness = 0
    refresh = True
```

```
    print('Selected Items:', end=' ')
    print(*(item for item in WareHouse.items), sep=', ')
    print('Best route:', alltime_fittest)
    print('Best fitness:', alltime_fitness)
    plt.plot(gen,fitn)
    plt.xlabel('Generations')
    plt.ylabel('Fitness')
    plt.show()

class Check:
    x = 0
    def __init__(self, lbl):
        self.var = StringVar()
        self.cb = Checkbutton(myApp, text=lbl, variable=self.var,
                              onvalue=lbl, offvalue="")
        self.cb.grid(row=Check.x, column=1, sticky=W)
        Check.x += 1
        varList.append(self.var)


Check("Watch")
Check("Television")
Check("Bag")
Check("Book")
Check("Mobile")
Check("Laptop")
Check("Refrigerator")
Check("Air Conditioner")
Check("Pendrives")
Check("HDD")
Check("Flash card")
Check("Bottles")
Check("Fan")
Check("Ear phones")
Check("Electric Bulbs")
Check("Deodrants")
Check("Vaccum Cleaner")
Check("Sandals")
Check("Cameras")
Check("Power Banks")
Check("Blankets")
Check("Hair Dryer")
Check("Trimmer")
Check("Soaps")
Check("Washing Machine")
Check("Shoes")
```

```
Check("Goggles")
Check("Kerchiefs")
Check("Cosmetics")
Check("Shirts")
Check("Pants")
Check("Chairs")


bl = Button(myApp, text="Add to Picking List", command=addtolist)
bl.grid(row=30, column=50)
```

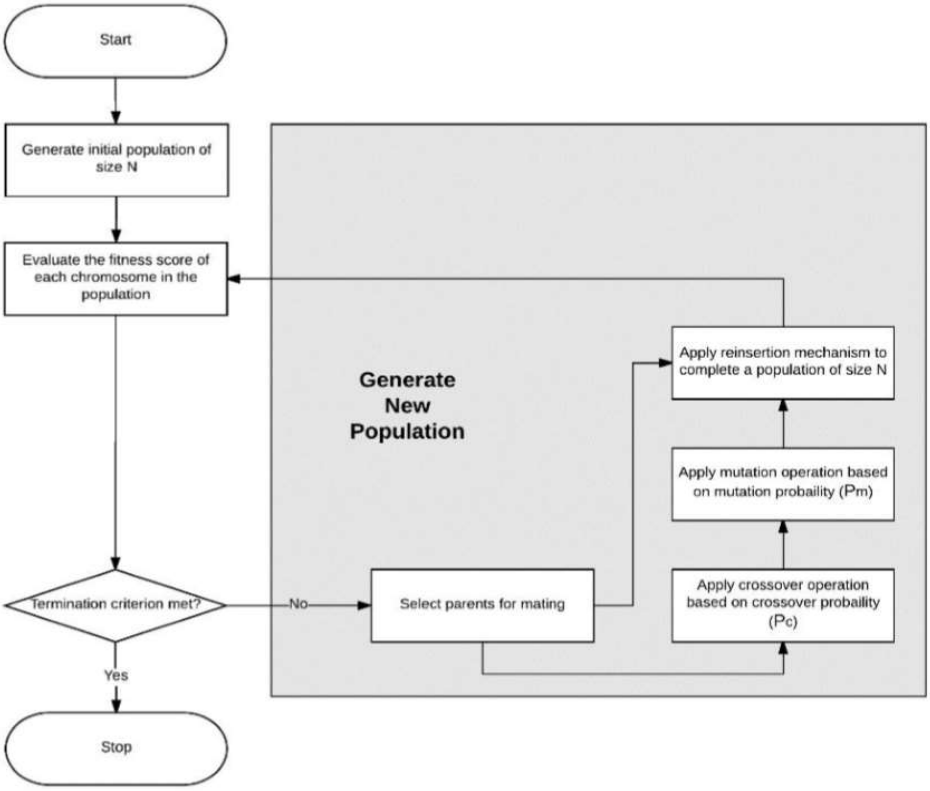
mod-Graphic.py


tk-add-to-list.py - E:\project\modified Project\Genetic-Algorithm-TSP-Picking\tk-add-to-list.py (3.6.2)
File Edit Format Run Options Window Help
from tkinter import *
from TSP import *
from libs.GeneticAlgorithm import GeneticAlgorithm
def addtolist():
global List
List = []
for item in varList:
if item.get() != "":
List.append(item.get())
for i in List:
x.append(d[i])
s=[]
l=List
for i in range(len(l)):
s.append(Itemloc(l[i],x[i]))
WareHouse = Item()
WareHouse.add(s)
print('Selected Items:', end=' ')
print(*(item for item in WareHouse.items), sep=', ')
ga = GeneticAlgorithm(100, mutation_rate=0.5, ptype=Route, args=(WareHouse.items,))
ga.run(seconds=40)
fittest = ga.alltime_best
best_fitness = fittest.fitness
print('Best route:', fittest)
print('Best fitness:', best_fitness)
print('Generations:', ga.generation)
#print(s)

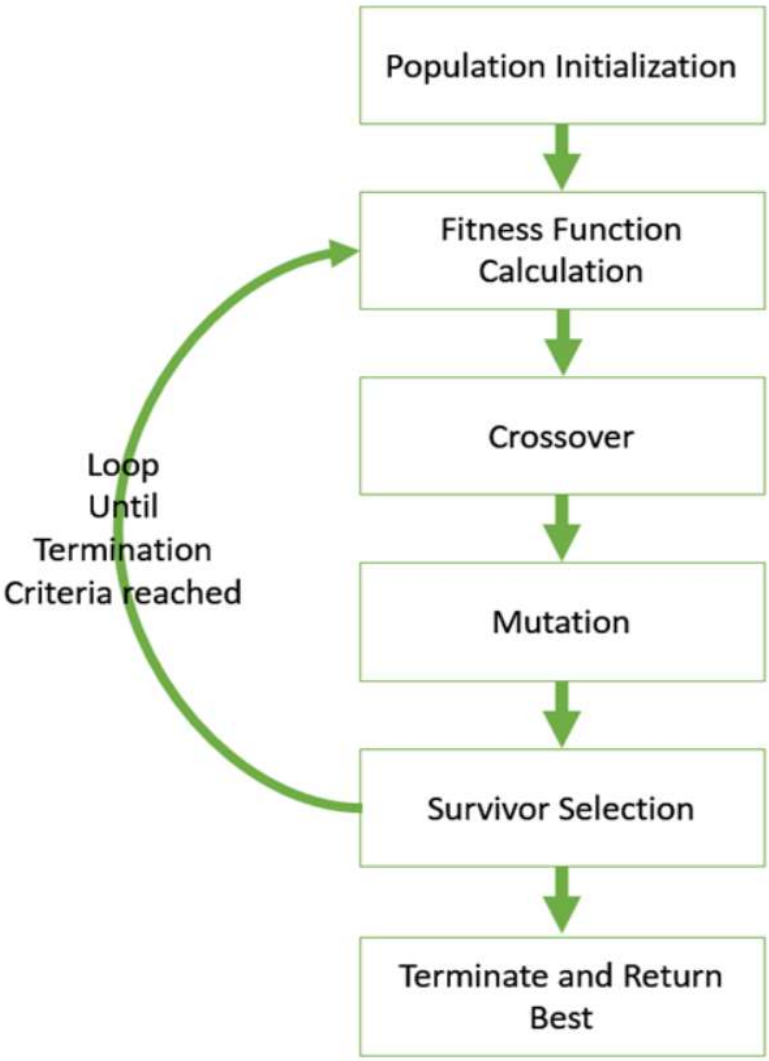


tk-add-to-list.py

LIST OF FIGURES



Algorithm Flow graph



Generalization of Algorithm