Travelling salesman hill climbing

Github:

https://github.com/pjavier98/AI/tree/master/travelling_salesman_hill_climbing

travelling_salesman_hill_climbing.py

```
from util import *
from graph import *
from state import *
from random import randint

flag = 1

def search_hamiltonian_cycle(graph, initial_state, begin, distances_list):
  global flag
  graph.visited_states[initial_state.city] = 1

  if ((initial_state.depth == 9) and (distances_list[initial_state.city -
1][begin - 1] != -1) and flag):
    graph.solution = [initial_state]
    while initial_state.dad:
      if (initial_state == initial_state.dad):
        break

      graph.solution.insert(0, initial_state.dad)
      initial_state = initial_state.dad
    flag = 0
    graph.update_cost(distances_list, begin)
    return

  for state in graph.adj_list[initial_state.city]:
    if (not graph.visited_states[state.city]):
      state.update_depth(initial_state.depth + 1)
      state.dad = initial_state
      search_hamiltonian_cycle(graph, state, begin, distances_list)
      graph.visited_states[state.city] = 0

def hill_climbing(graph, begin, distances_list, amount_permutations):
  adj_list_begin = graph.adj_list[begin]

  solution = []
  solution.extend(graph.solution)

  new_solution = []
  for num in adj_list_begin:
    if (num.city != graph.solution[9]):
      for i in range(amount_permutations):
        x = randint(1, 8)
```

```
        y = randint(1, 8)

        new_solution = solution

        aux_state = new_solution[x]
        new_solution[x] = new_solution[y]
        new_solution[y] = aux_state

        if (graph.check_hamiltonian_cycle(new_solution, distances_list,
begin)):
          graph.print_best_way(0)

      new_solution.clear()
      solution.clear()
      solution.extend(graph.solution)

      for i in range(10):
        state = solution[i]
        if (state.city == num.city):
          solution[i] = solution[9]
          solution[9] = state
          break


def main():
  print('Select the start city: [1-10]: ', end='')
  begin = read_input(0, 10)
  print()

  distances_list = read_files('files/distances.txt')

  graph = Graph()
  graph.adj_list = graph.generate_graph(distances_list)

  initial_state = State(int(begin))
  initial_state.dad = initial_state

  search_hamiltonian_cycle(graph, initial_state, begin, distances_list)

  graph.print_best_way(True)
  hill_climbing(graph, begin, distances_list, 30000)
main()
```

graph.py

```
from state import *

class Graph:

  def __init__(self):
    self.adj_list = None
```

```
    self.visited_states = [0] * 11
    self.solution = []
    self.total_cost = 0

  def generate_graph(self, distances_list):
    graph = []
    graph.append([])
    input_adj_list = open('files/adj_list.txt', 'r')

    for i in range(10):
      adj_list = []
      for j in input_adj_list.readline().split():
        index = int(j) - 1

        state = State(int(j))
        adj_list.append(state)

      graph.append(adj_list)

    input_adj_list.close()
    return graph

  def check_hamiltonian_cycle(self, possible_solution, distances_list,
begin):
    cost = 0

    for i in range(9):
      previous_state = possible_solution[i].city
      current_state = possible_solution[i + 1].city

      dist = distances_list[previous_state - 1][current_state - 1]
      if (dist == - 1):
        return 0
      else:
        cost += dist

    # Cost from the last to the begin

    cost += distances_list[current_state - 1][begin - 1]
    if (cost < self.total_cost):
      self.total_cost = cost
      self.solution.clear()
      self.solution.extend(possible_solution)
      return 1
    return 0

  def update_cost(self, distances_list, begin):
    cost = 0
    for i in range(9):
      previous_state = self.solution[i].city
      current_state = self.solution[i + 1].city
      cost += distances_list[previous_state - 1][current_state - 1]

    cost += distances_list[begin - 1][current_state - 1]
```

```
    self.total_cost = cost

  def print_graph(self):
    for i in range(11):
      city = str(i)
      print(city + " -> ", end="")
      for j in self.adj_list[i]:
        print(str(j.city), end=" -> ")
      print('//')

  def print_best_way(self, flag):
    if flag:
      print('Hamiltonian Cycle Initial State: cost:
{}'.format(self.total_cost))
    else:
      print('Hamiltonian Cycle: cost: {}'.format(self.total_cost))
    for i in self.solution:
      print(i.city, end=" -> ")
    print('//', end='\n')
```

state.py

```
class State:

    def __init__(self, city):
        self.city = city
        self.depth = 0
        self.dad = None

    def __str__(self):
      # childrens_id = str_children()
      return ('city: {}\ndepth: {}\ndad: {}\n'
            .format(self.city, self.depth, self.dad.city
      ))

    def create_state():
      return State(city, depth, dad)

    def update_depth(self, depth):
      self.depth = depth

    def final_state(self, begin):
      return self.station == begin
```

util.py

```
def read_input(lower, upper):
  while True:
    try:
```

```
      number = int(input())
      if (number >= lower and number <= upper):
        return number
    except:
      pass
    print("Invalid input, please choose again from [" + str(lower) + "-" +
str(upper) + "]")

def read_files(path):
  fileDistances = open(path, 'r')

  inputFile = []
  for i in range(10):
      inputFile.append(list(map(int, fileDistances.readline().split())))

  fileDistances.close()
  return inputFile

def print_files(file):
  for i in file:
    print(i)
```