Politechnika Łódzka

Wydział: EEIA
Grupa: 7D30
Dzień tygodnia: Niedziela

Rok akad.: 2017/2018
Semestr: VII

Sprawozdanie z ćwiczenia z
Podstawy Sztucznej Inteligencji

Temat ćwiczenia: **Problem Komiwojażra**

Data wykonania ćwiczenia	Podpis	Data oddania sprawozdania	Podpis
01.12.2017		01.12.2017	

Wykonał:
Radosław Subczyński
Nr inteksu 200137

Początek badań nad problemem komiwojażera nie jest jasny. Wspomina o nim podręcznik z 1832[a], który zawiera przykładową trasę po Niemczech i Szwajcarii, lecz nie zawiera żadnych matematycznych uzasadnień.

W 1859 irlandzki matematyk William Rowan Hamilton sformułował problem istnienia cyklu o długości *n* w grafie *n*-wierzchołkowym[4].

Za pierwszego autora, który sformalizował matematycznie problem komiwojażera uznaje się austriackiego matematyka Karla Mengera, który zdefiniował go w 1930 zwracając szczególną uwagę na trudność w obliczeniu rozwiązania. Niezależnie od niego ten sam problem poruszył w 1934 Hassler Witney na wykładzie w Princeton University. Natomiast pierwsza próba rozwiązania problemu miała miejsce w 1937, gdy Merrill Flood pracował nad rozwiązaniem wyznaczania tras dla autobusów szkolnych].

Z uwagi na bardzo prosty opis problemu oraz opinię o bardzo trudnym obliczeniowo procesie optymalizacji, problem komiwojażera stał się bardzo popularny[5]. Fascynacja ta trwa od lat pięćdziesiątych XX wieku do dziś, zarówno wśród amatorów jak i profesjonalistów.

Nazwa pochodzi od typowej ilustracji problemu, przedstawiającej go z punktu widzenia wędrownego sprzedawcy (komiwojażera): dane jest n miast, które komiwojażer ma odwiedzić, oraz odległość / cena podróży / czas podróży pomiędzy każdą parą miast. Celem jest znalezienie najkrótszej / najtańszej / najszybszej drogi łączącej wszystkie miasta, zaczynającej się i kończącej się w określonym punkcie

Miasta: Kutno, Warszawa, Poznań, Kraków

Odległości:

	Kutno	Warszawa	Poznań	Kraków
Kutno	0	130	180	300
Warszawa	130	0	320	350
Poznań	180	320	0	360
Kraków	300	350	360	0

Należy znaleźć najkrótszą trasę zaczynającą się np. z Kutna, przechodzącą jednokrotnie przez wszystkie pozostałe miasta i wracającą do Kutna.

Problem ten jest NP-trudny.

Początek badań nad problemem komiwojażera nie jest jasny. Wspomina o nim podręcznik z 1832[a], który zawiera przykładową trasę po Niemczech i Szwajcarii, lecz nie zawiera żadnych matematycznych uzasadnień.

W 1859 irlandzki matematyk William Rowan Hamilton sformułował problem istnienia cyklu o długości *n* w grafie *n*-wierzchołkowym[4].

Za pierwszego autora, który sformalizował matematycznie problem komiwojażera uznaje się austriackiego matematyka Karla Mengera, który zdefiniował go w 1930 zwracając szczególną uwagę na trudność w obliczeniu rozwiązania. Niezależnie od niego ten sam problem poruszył w 1934 Hassler Witney na wykładzie w Princeton University. Natomiast pierwsza próba rozwiązania problemu miała miejsce w 1937, gdy Merrill Flood pracował nad rozwiązaniem wyznaczania tras dla autobusów szkolnych].

Z uwagi na bardzo prosty opis problemu oraz opinię o bardzo trudnym obliczeniowo procesie optymalizacji, problem komiwojażera stał się bardzo popularny[5]. Fascynacja ta trwa od lat pięćdziesiątych XX wieku do dziś, zarówno wśród amatorów jak i profesjonalistów[5][2].

Nazwa pochodzi od typowej ilustracji problemu, przedstawiającej go z punktu widzenia wędrownego sprzedawcy (komiwojażera): dane jest n miast, które komiwojażer ma odwiedzić, oraz odległość / cena podróży / czas podróży pomiędzy każdą parą miast. Celem jest znalezienie najkrótszej / najtańszej / najszybszej drogi łączącej wszystkie miasta, zaczynającej się i kończącej się w określonym punkcie

Ja w swoim przykładzie wykonałem rozwiązałem swój przpadek na 2 rózne sposoby poprzez bruteforce oraz algorytm zachłanny.

Oba rozwiązania są dostępne pod linkiem :

https://github.com/rsubczynski/Problem-Voyagera

Zacznę od opisu Problemu od opisania wspólnych części obu programów:

W Klasie App config znajdziemy zmienną dla ilu miast chcemy rozwiązać problem jest to :

```
public final static int LIST_CITY_SIZE = 5;
```

Następnym miejscem wsólnym jest klasa City która jest encją (modelem miasta):

Która posiada :

```
private String cityName;
private int x;
private int y;
```

CityName – Nazwa Miasta;
x,y – wspolrzedne miast;

klasa posiada tez metode liczącą odległość miedzy 2 miastami:

```
public double distanceToCity(City city) {
    int x = Math.abs(getX() - city.getX());
     int y = Math.abs(getY() - city.getY());
     return Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2));
}
```

Następ

Politechnika Łódzka

Początek badań nad problemem komiwojażera nie jest jasny. Wspomina o nim podręcznik z 1832[a], który zawiera przykładową trasę po Niemczech i Szwajcarii, lecz nie zawiera żadnych matematycznych uzasadnień.

W 1859 irlandzki matematyk William Rowan Hamilton sformułował problem istnienia cyklu o długości *n* w grafie *n*-wierzchołkowym[4].

Za pierwszego autora, który sformalizował matematycznie problem komiwojażera uznaje się austriackiego matematyka Karla Mengera, który zdefiniował go w 1930 zwracając szczególną uwagę na trudność w obliczeniu rozwiązania. Niezależnie od niego ten sam problem poruszył w 1934 Hassler Witney na wykładzie w Princeton University. Natomiast pierwsza próba rozwiązania problemu miała miejsce w 1937, gdy Merrill Flood pracował nad rozwiązaniem wyznaczania tras dla autobusów szkolnych].

Z uwagi na bardzo prosty opis problemu oraz opinię o bardzo trudnym obliczeniowo procesie optymalizacji, problem komiwojażera stał się bardzo popularny[5]. Fascynacja ta trwa od lat pięćdziesiątych XX wieku do dziś, zarówno wśród amatorów jak i profesjonalistów[5][2].

Nazwa pochodzi od typowej ilustracji problemu, przedstawiającej go z punktu widzenia wędrownego sprzedawcy (komiwojażera): dane jest n miast, które komiwojażer ma odwiedzić, oraz odległość / cena podróży / czas podróży pomiędzy każdą parą miast. Celem jest znalezienie najkrótszej / najtańszej / najszybszej drogi łączącej wszystkie miasta, zaczynającej się i kończącej się w określonym punkcie

Metodą do odpalania Programów jest Klasa Main:

```
    public static void main(String[] args) {
//         Zachłanny zachłanny = new Zachłanny();
         BruteForce bruteForce = new BruteForce();
     }
```

Gdzie w zależności który algorytm chcemy odpalić powinniśmy go od komentować :

Metoda Bruteforce porównuje miaso każde z każdym :

```
private ArrayList<City> cityList = new ArrayList<>();

    public BruteForce() {
        initCityList();
        List<Integer> droga = new ArrayList<>();
        //droga.add(0);   jezeli zaczynamy od pierwszego miasta na liscie to
odkomentowujemy -
        // jezeli chcemy przetestowac wszystkie permutacje w ktorych pierwszy wezel
moze byc kazdym z miast zostawiamy zakomentowane
        List<List<Integer>> lists = permutacje(droga, cityList.size());
        Double minodleglosc = null;
        List<Integer> minList = null;
        for (List<Integer> list : lists) {
            double odleglosc = 0;
            for (int i = 0; i < list.size(); i++) {
                if (i != list.size() - 1) {
                    odleglosc +=
cityList.get(list.get(i)).distanceToCity(cityList.get(list.get(i + 1)));
                } else {
                    odleglosc +=
cityList.get(list.get(i)).distanceToCity(cityList.get(list.get(0)));
                }
            }
            if (minodleglosc == null || minodleglosc > odleglosc) {
                minodleglosc = odleglosc;
                minList = list;
            }
            System.out.print("Dlugosc dla drogi:");
            for (int i = 0; i < list.size(); i++) {
                System.out.print(list.get(i));
            }
            System.out.print(list.get(0));
            System.out.println(" to " + odleglosc);
        }
        System.out.println("Najlepszy wariant to " + minodleglosc);
        for (int i = 0; i < minList.size(); i++) {
            System.out.print(minList.get(i));
        }
        System.out.print(minList.get(0));
    }

    private List<List<Integer>> permutacje(List<Integer> way, int size) {
        List<List<Integer>> allWays = new ArrayList<>();
        List<Integer> copyOfWay = new ArrayList<>();
```

```
        copyOfWay.addAll(way);
        Integer numberOfMissingNodesInWay = size - way.size();
        if (numberOfMissingNodesInWay == 1) {
            for (int i = 0; i < cityList.size(); i++) {
                if (!way.contains(i)) {
                    way.add(i);
                    allWays.add(way);
                    return allWays;
                }
            }
        } else {
            for (int i = 0; i < cityList.size(); i++) {
                if (!way.contains(i)) {
                    List<Integer> tempWay = new ArrayList<>();
                    tempWay.addAll(way);
                    tempWay.add(i);
                    allWays.addAll(permutacje(tempWay, size));
                }
            }
        }
        return allWays;
    }

    private void initCityList() {
        Random generator = new Random();
        for (int i = 0; i < AppConfig.LIST_CITY_SIZE; i++) {
            cityList.add(new City(Utils.generateRandom(5), generator.nextInt(100) + 1,
generator.nextInt(100) + 1));
            Utils.ShowLog("CityName = " + cityList.get(i).getCityName() + " X = " +
cityList.get(i).getX() + " Y = " + cityList.get(i).getY());
        }
    }
}
```

Drugi Algorym który opracowałem jest to Algorytm zachłanny który oblicza drogę do najbliższego miasta później usuwa go z listy i liczy następny najbliższy punkt :

```
private ArrayList<City> cityList = new ArrayList<>();

public Zachłanny() {
    iniCityList();
    City startCity = cityList.get(0);
    cityList.remove(cityList.get(0));
    City newStatCity = findTheSmalledDistance(startCity);
    if (cityList.size() != 0) {
        do {
            newStatCity = findTheSmalledDistance(newStatCity);
        } while (cityList.size() != 0);
    }
}


City findTheSmalledDistance(City startCity) {
    double tmpWay = 1000000000;
    City tempCity = new City("", 0, 0);
    for (City city : cityList) {
        System.out.println(startCity.distanceToCity(city));
        if (startCity.distanceToCity(city) < tmpWay) {
            tmpWay = startCity.distanceToCity(city);
            tempCity = city;
            tempCity.setDistance(startCity.distanceToCity(city));
        }
    }
    Utils.ShowLog("Najmniejsza odległosc to :" + tempCity.getDistance() + " CityName :"
+ tempCity.getCityName() + " X :" + tempCity.getX() + " X :" + tempCity.getY());
    Utils.ShowLog("");
    cityList.remove(tempCity);
    return tempCity;
}


private void iniCityList() {
    Random generator = new Random();
    for (int i = 0; i < AppConfig.LIST_CITY_SIZE; i++) {
        cityList.add(new City(Utils.generateRandom(5), generator.nextInt(100) + 1,
generator.nextInt(100) + 1));
        Utils.ShowLog("CityName = " + cityList.get(i).getCityName() + " X = " +
cityList.get(i).getX() + " Y = " + cityList.get(i).getY());
        Utils.ShowLog("");
    }

}
```