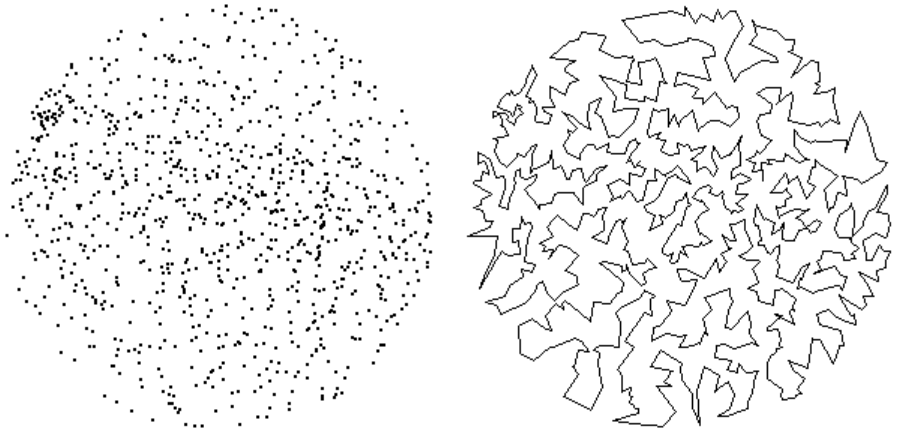
Data Structures and Algorithms: Assignment 4

Schrijf een programma dat een aantal heuristieken gebruikt om het handelsreizigersprobleem benaderend op te lossen[1].

1 Het handelsreizigersprobleem

Elk jaar tijdens de eindejaarsperiode gaan mensen op zoek naar kerstcadeaus voor de ganse familie. Dit jaar kiest Jan ervoor om naar het shopping center te gaan voor zijn aankopen. Hij heeft een lijst van 10 winkels die hij moet bezoeken. Door de enorme drukte heeft hij niet veel zin om lang door het shopping center te kuieren. Hoe kan Jan de kortste route vinden vanaf zijn auto die langs de 10 winkels loopt en terug?

Meer algemeen geformuleerd is er gegeven een verzameling van N punten. Het doel bestaat erin een zo kort mogelijke (gesloten) route te vinden die alle punten exact 1 keer bevat.



Het handelsreizigersprobleem is een beroemd optimisatieprobleem. In principe is het mogelijk om alle mogelijke routes uit te proberen. In de praktijk echter is het aantal mogelijkheden véél te groot (ongeveer N!), zodat deze aanpak nutteloos is. Voor grote waarden van N is er geen efficiënte methode bekend om de kortst mogelijke route te vinden.

[1] Dit practicum is gebaseerd op een practicum van Princeton University.

2 Heuristieken

Gelukkig bestaan er methodes die goed lijken te werken in de praktijk, hoewel ze niet noodzakelijk de kortste route vinden. Bij zulke methodes begint men vaak van een route bestaande uit 1 punt (van het eerste punt terug naar zichzelf) en voegt men iteratief, in een bepaalde volligorde, de andere punten toe aan die route tot ze volledig is. Om te bepalen waar in de route elk punt ingevoegd kan worden, wordt gebruik gemaakt van heuristieken. Enkele veelgebruikte heuristieken zijn:

- Dichtste-buur: voeg het nieuwe punt toe na het punt in de route waar het het dichtst bij ligt. Indien meerdere punten hiervoor in aanmerking komen, kies uit deze punten hetgeen eerst gevonden is.
- Minste verlenging: voeg het nieuwe punt toe zodanig dat de route zo weinig mogelijk langer wordt. Indien meerdere punten hiervoor in aanmerking komen, kies uit deze punten hetgeen eerst gevonden is.
- MST: Stel de Minimal Spanning Tree (MST) op van de grafe die elk van de punten met alle andere punten verbindt. Een goede route kan gevonden worden door deze MST in pre-order helemaal te doorlopen en terug te keren naar de startpositie.

3 Jouw taak

De klasse `World` stelt een rechthoeking gebied voor waarin een aantal punten liggen. De bedoeling van onze algoritmes is om een `Tour` op te stellen die alle punten bezoekt. Een tour bevat elk punt exact 1 keer. Het is echter wel mogelijk dat verbindingen elkaar kruisen voor met een verzameling van punten.

Implementeer de hierboven beschreven heuristieken in drie subklassen van de klasse `Tour`:`NearestNeighborTour`, `SmallestIncreaseTour` en tenslotte de klasse `MinimalSpanningTreeTour`. We raden je sterk aan om je oplossing te controleren door middel van JUnit testen.

Je mag extra klassen en methodes toevoegen. De bestaande API mag niet aangepast worden. Het is toegelaten om extra velden en methodes toe te voegen aan de klassen `Tour` en `IncrementallyConstructedTour`.

In- en uitvoer

De invoer bestaat uit de breedte w en hoogte h van de 2-dimensionale wereld waarin alle punten liggen, gevolgd door `N` coordinaten $x \in [0; w]$ en $y \in [0; h]$. Een invoerbestand met 4 punten zou de volgende gegevens kunnen bevatten:

```
600 600
532.6531 247.7551
93.0612 393.6735
565.5102 590.0000
```

```
10.0000 10.0000
```

In de map `inputs` hebben we enkele invoerbestanden voorzien. De uitvoer van je programma ziet er dan als volgt uit:

```
Tour Distance = 1822.186768
532.653076 247.755096
565.510193 590.000000
93.061203 393.673492
10.000000 10.000000
```

Voor elk van de heuristieken zijn main methodes voorzien (bijvoorbeeld NearestNeighborSolver.java). waarmee je invoerbestanden kunt inlezen, de gegeven heuristiek testen en de oplossing uitschrijven. Om deze uit te voeren, kun je volgende `ant` targets gebruiken:

```
ant nearestneighbor -Dinput=inputs/tsp10.txt

ant smallestincrease -Dinput=inputs/tsp10.txt

ant minimalspanningtree -Dinput=inputs/tsp10.txt
```

4 Vragen

Naast het implementeren van de heuristieken, vragen we om de uitvoeringstijd van je programma te testen als functie van N. Voer je programma hiervoor uit met elk van je heuristieken voor toenemende N zolang de uitvoeringstijd minder is dan 100 seconden. Start met N = 1000 en verdubbel N in elke stap. De posities van je punten kies je random. Voor timings kan je het bijgevoegde TSPTimer.java gebruiken. Toon de bekomen resultaten en bespreek de verschillen tussen de heuristieken.

Extra (niet verplicht, wel beloond met bonus) Implementeer je eigen heuristiek. Bespreek en vergelijk met de andere heuristieken. Bijvoorbeeld, elke tour waar verbindingen elkaar kruisen kan getransformeerd worden in een kortere tour zonder kruisende verbindingen.

5 Criteria

Je practicum wordt gequoteerd op basis van de hopelijk goede werking van je code en de antwoorden op bovenstaande vragen. Enkele zaken die belangrijk zijn om een goede score te kunnen halen zijn:

- De implementatie van elke methode die je moet implementeren voldoet in alle situaties aan de specificaties van die methode (die we in `/** commentaar */` in de .java file opgelegd hebben).

- De uitvoeringstijd van je programma moet redelijk zijn. Als je programma uren nodig heeft om kleine routes te construeren is er een probleem. Bij een standaard implementatie is er normaalgezien geen probleem.
- De vragen waarvan we vragen ze te beantwoorden in het verslag, zijn beantwoord.
- Wat je in je verslag schrijft, is waar.
- Wat je in je verslag concludeert op basis van metingen, volgt ook echt uit de metingen.

6 Indienen

De oplossing moet ten laatste vrijdag 24 mei om 14.00 uur ingediend worden via Toledo. Net als bij vorige opdrachten:

- plaats je je verslag als *.pdf in `report/`
- voer je

  ```
  ant release
  ```

  uit
- Hernoem je je bekomen *.zip-bestand naar `firstname_lastname.zip`

Je hoeft geen papieren versie van je verslag in te dienen.