Labo over metaheuristieken

Implementeer op 3 verschillende manieren een TSP-probleem:

- met een zelfgemaakt framework, zie Toledo.
- met een bestaand Java framework OpenTS (https://www.coin-or.org/Ots/index.html) dat je helpt bij het implementeren van een applicatie die tabu search gebruikt.
- met het algemeen optimalisatieframework Optaplanner (https://www.optaplanner.org/) van RedHat.

Vertrek hiervoor van een klasse City, waarin je de x- en y-coördinaten van de stad bijhoudt. We zullen nl. rekenen met de afstand tussen steden in vogelvlucht. Zorg er voor dat er telkens een nieuwe stad aangemaakt wordt random een x- en y-coördinaat gegenereerd wordt. Om het testen en debuggen te vergemakkelijk zorg je er best voor dat de coördinaten van de steden elke keer ze opnieuw gegenereerd worden dezelfde zijn. Maak 20 steden aan en bereken de kortste route.
Bij de eerste 2 frameworks, die minder geavanceerd zijn dan Optaplanner, zal je zelf de zetten (“moves”), de kostfunctie e.d. moeten implementeren.
De kost van een oplossing kan op 2 manieren geëvalueerd worden:

- door de kost van de volledige oplossing te berekenen. Dit betekent dat als je een kleine verandering aan de oplossing gedaan hebt je ook de delen die niet veranderd zijn opnieuw berekent. Om de startoplossing te berekenen is dit de manier om het te doen, maar dit wordt beter niet gebruikt om tijdens het zoeken naar een betere oplossing de kost te berekenen. Aangezien er veel onnodig rekenwerk gebeurt (het grootste deel van de oplossing is nl. niet veranderd), vertraagt dit het zoeken veel. Hou in je achterhoofd dat er tijdens het zoeken heel veel zetten gedaan worden (afhankelijk van de zoektijd).
- telkens een zet gebeurt op een oplossing wordt de nieuwe kost berekend. Aangezien er maar een beperkt deel van de oplossing veranderd wordt, is het efficiënter om enkel het deel dat veranderd is te berekenen. Eigenlijk komt het er op neer dat je de “oude” en “nieuwe” kost van het veranderde deel berekent en het verschil van beide berekent. Indien er een verbetering is, zal de nieuwe kost kleiner zijn dan de oude kost en zal de totale kost dus verlagen. Dit noemt men **delta-evaluatie**.

De bedoeling van bovenstaande oefening is om met de 3 frameworks te leren werken. Je zal zien dat je bij Optaplanner minder zelf moet programmeren. In tegenstelling tot de eerste 2 frameworks moet je bijv. niet zelf de delta-evaluatie implementeren. Maar om van de interne delta-evaluatie te kunnen gebruik maken, zal je de harde en zachte beperkingen op een bepaalde manier moeten uitdrukken. De leercurve om dit te doen is vrij steil, maar gelukkig zijn er veel voorbeelden in de democode te vinden, waarop je kan verder werken. Het zoekalgoritme hoef je ook niet zelf te implementeren. Welk zoekalgoritme je wilt gebruiken, geef je aan in een configuratiebestand.

TIP: als je Optaplanner wilt gebruiken, maak dan een nieuwe Maven-project aan en kopieer het pom-bestand van een bestaand Optaplanner-voorbeeld om van te vertrekken.