Problem Set 2

Capacitated Vehicle Routing Problem (CVRP)

Johanna Sacher
221103072 | anvxt
johanna.sacher@student.uni-halle.de

Christian Paffrath
221103085 | anvxu
christian.paffrath@student.uni-halle.de

— Überblick —

In the following weeks, we are going to design and evaluate algorithms on the Capacitated Vehicle Routing Problem (CVRP). Please make yourself familiar with the details of the CVRP, in particular with the `vrp` *and* `sol` *file format, and how its objective function is calculated.*

I Problem 1: CVRP Solution Checker

I.1 Problemstellung

Implement a solution checker that tests whether a specified `sol`-file is valid to a certain `vrp`-file. Your solution checker gets as command line arguments the paths to an vrp and sol file and should output VALID if the solution is valid or a meaningful error message if not. Think about which aspects of a solution file must be checked and summarize them in a short report. Also give a short explanation how your error checker can be used.

I.2 Lösung

Ein Solution-Checker muss zwei grundlegende Dinge prüfen, und zwar zum einen, ob die `sol`-Datei das korrekte Format hat, und zum anderen, ob die Lösung an sich eine valide Lösung für das im `vrp`-File gegebene Problem ist. Zuerst muss überprüft werden, ob das Format der Datei die Vorgaben für einen `sol`-File erfüllt. Ist dies nicht der Fall, kann die Lösung gar nicht vom Algorithmus überprüft werden, da die Datei nicht korrekt eingelesen werden kann. Die `sol`-Datei besteht aus einer oder mehreren

Route-Zeilen und einer Cost-Zeile. Beide Zeilentypen haben ein genau spezifiziertes Format.
Sind alle formalen Vorgaben erfüllt, wird die Lösung an sich überprüft. Dafür wird getestet, ob die gegebenen Routen die Bedingungen des CVRP erfüllen:

1. Starten und enden alle Routen im Depot? (impliziert von der Lösungsdatei, wird nicht extra überprüft)
2. Wird jeder Kunde genau einmal besucht?
3. Kann der Bedarf aller Kunden auf einer Route mit der Kapazität eines Fahrzeugs erfüllt werden?

Die Lösung ist valide, wenn sich innerhalb der Routen keine Kunden wiederholen, alle in der vrp-Datei aufgelisteten Kunden besucht werden und wenn die Cost mit der Summe der Distanzen aller Routen übereinstimmt.

Unser Solution-Checker geht beide Dateien Zeile für Zeile durch: Aus der vrp-Datei werden die Capacity des Fahrzeuges sowie die Liste aller zu besuchenden Nodes mit ihren Coordinates und den Demands ausgelesen. Diese werden jeweils in einem LocationNode-Objekt gespeichert, das aus einer locationID, den x- und y- Koordinaten, einem locationDemand und einer vorerst leeren Visitation-Counter visited besteht. Alle LocationNodes werden in einem LocationsContainer gespeichert. Dieser hält einfach einen Vector mit den LocationNodes und bietet verschiedene Operationen darauf an. Aus der sol-Datei werden die Cost und alle Routen ausgelesen. Für jede Route wird markiert, welche LocationNodes besucht wurden und in einem Route-Objekt wird gespeichert, wie groß der Gesamtbedarf der Kunden dieser Route ist und wie lang die Gesamtdistanz der Wege ist. Wichtig dabei ist, jeweils das Depot als Start- und Endpunkt nicht zu vergessen, da dieses im sol-File nicht mit aufgeführt wird. Deswegen muss auch beachtet werden, dass Kunde x in der sol-Datei mit Node $x + 1$ in der vrp-Datei übereinstimmt. In der Node-Liste steht das Depot mit ID 1 an der ersten Position, während es, wie gesagt, in den Routen nicht mit aufgeführt wird. Nachdem alle Zeilen eingelesen wurden, werden die Bedingungen des CVRP abgecheckt:

- Zuerst testen wir, ob wir bei der Berechnung der Gesamtdistanz aller Routen auf das gleiche Ergebnis kommen, das im sol-File steht. Wir haben allerdings ziemlich sicher bei der Berechnung der Distanzen zwischen den Koordinaten einen

Fehler gemacht oder anders als die Ersteller der `sol`-Files gerundet, da unsere Berechnung der Gesamtdistanz nie mit den im `sol`-File aufgeführten Kosten übereinstimmt. Wir erklären in diesem Fall also eine Lösung **nicht** als falsch, wenn die Kosten nicht mit den von uns berechente Kosten übereinstimmen.

- Dann überprüfen wir, ob jeder Kunde genau einmal besucht wurde. Dies können wir am `visited`-Counter der einzelnen `LocationNodes` ablesen. Sobald ein Knoten (außer dem Depot) mehr oder weniger als einmal besucht wurde, wird die Lösung für falsch erklärt und eine Fehlermeldung ausgegeben.
- Außerdem testen wir, ob der Demand auf jeder Route höchstens so groß ist wie die Fahrzeugkapazität. Ist der Demand eine Route größer, wird die Lösung als *falsch* abgelehnt.

Wurden alle Bedingungen überprüft und keine Fehlermeldungen ausgegeben, ist die im `sol`-File präsentierte Lösung für den `vrp`-File valide und wird mit der Ausgabe »`Solution File is VALID.`« akzeptiert.

Wie der Solution Checker genutzt wird, erklären wir beispielhaft in der README.md Datei unseres GitHub Repositories.

II Problem 2: CVRP Greedy Heuristics

II.1 Problemstellung

Consider the following simple heuristic for the Capacitated Vehicle Routing Problem:

- As long as some location is unvisited, do the following:
  1. Start a new tour at the depot.
  2. As long as the vehicle capacity is not yet depleted, extend the tour by incorporating the nearest unvisited location and decrease the vehicle capacity by the associated demand.
  3. When the capacity is depleted, go back to the depot and start a new tour.

Implement this heuristic. Thoroughly test your implementation on a large set of instances.

II.2 Lösung

Für die Implementierung der Greedy-Heuristic haben wir uns zuerst anhand der Aufgabenstellung eine Code-Skizze überlegt. Diese Skizze haben wir dann versucht, in C++ zu implementieren. Leider lässt sich unsere Lösung aufgrund eines `segmentation faults` nicht compilieren und wir können den Fehler einfach nicht finden.

Das Main-Programm für den Greedy-Solver befindet sich im Ordner `main`, der hauptsächliche Code zum Lösen steht im `GreedySolver.cpp`. Wir haben uns bei der Implementierung ein bisschen an der Lösung von vss2sn auf GitHub orientiert.

$$\texttt{NodesContainerAllLocations}; \rightarrow l$$

$$\texttt{Vehicle Capacity}$$

```
if(anyOf(l.begin(), [](LocationNode node{return node.TimesVisited == 0})){
        Route r{};
        LocationNode current = AllLocations.GetDepot();
        r.AddLocation(current);
        LocationNode.closest = findClosest(r);
        if(r.Load() − closest.Demand() ≥ 0){
          r.Deliver(closest.Demand());
          r.IncreaseCose(|current-closest|);
          r.AddLocation(closest);
          closest.Visit();
        }
        else{
          r.AddLocation(Depot);
}
```

III Problem 3: CVRP Second Heuristic)

III.1 Problemstellung

Design a second heuristic for the Capacitated Vehicle Routing Problem. Your algorithm should produce solutions that significantly differ from that of the first heuristic. Describe your algorithm in words and with pseudo code. Implement and test the algorithm.

III.2 Lösung

Beim Greedy-Algorithmus kann es während einer Route passieren, dass wenn die eigene `Load` noch nicht erschöpft ist, sehr große Distanzen zurückgelegt werden um den `Demand` eines weit entfernten Kunden zu decken, wenn alle nah gelegenen Kunden einen höheren `Demand` haben. Dieses Problem versuchen wir zu beheben, indem wir eine Durchschnittsdistanz aller Routen ermitteln und diese in die Auswahl des nächsten zu besuchenden Kunden miteinbeziehen.
Falls ein Vehicle noch eine Rest-`Load` hat, die zu klein ist um den `Demand` des am nächsten gelegenen Kunden zu decken, aber ein weiter entfernter Kunde entdeckt wird, der einen passenden `Demand` hat, dann wird die Distanz zu diesem Kunden mit der Durchschnittsdistanz verglichen. Liegt die Distanz über dem Durchschnitt, dann fährt das Fahrzeug statt zum Kunden direkt zum Depot, füllt die `Load` wieder auf und startet die nächste Route. Der in diesem Fall nicht besuchte Kunde, kann dann in einer späteren Route, die näher im eigenen Umfeld verläuft, ohne große Umwege besucht werden. Im Worst Case kann es natürlich auch passieren, dass falls einzelne Kunden um überdurchschnittliche Distanzen von allen anderen Kunden entfernt liegen, diese in extra Routen einzeln vom Depot aus besucht werden müssen. Das Depot hat natürlich keine Distanzbeschränkung.

Durchschnittsdistanz Z_D berechnen:

$$Z_D = \frac{\text{Summe aller Distanzen}}{\text{Anzahl aller Verbindungen}}$$

Solange noch nicht alle Knoten besucht wurden:

- Starte eine neue Tour am Depot
- Füge den nearest unvisited Node zur Tour hinzu
- Decrease Load of Tour by Demand of that Node
- Solange die Kapazität des Fahrzeugs noch nicht erschöpft ist:
  - Füge den nearest unvisited node zur Tour hinzu, dessen Distanz $\leq Z_D$
  - Decrease Load of Tour by Demand of that Node
  - Wenn kein Knoten mehr hinzugefügt werden kann, kehre zurück zum Depot
  - Füge Depot als Endknoten hinzu

Diese Idee ließe sich basierend auf unserem `GreedySolver` relativ einfach implementieren, indem die Funktion `FindNearestUnvisited()` so angepasst wird, dass zusätzlich noch getestet wird, ob die Entfernung des potentiellen nächsten Knotens kleiner ist als die Durchschnittsdistanz.

IV Problem 4: CVRP Horse Race Experiment

IV.1 Problemstellung

Design and run an experiment to evaluate which algorithm produces better results.

a) Start by formulating suitable experimental goals and research questions.

b) Describe your experimental setup. In particular, discuss
   - which benchmark instances you chose (name, size, type, selection criteria)
   - how you compare the solution quality and efficiency of the algorithms

- your experimental environment (i.e. hardware, programming language, OS, etc)

c) Evaluate your experiment. Present and interpret your results as a report with the help of suitable charts and tables. At least, discuss the following questions:

- Does your experiment clearly demonstrates the superiority of one algorithm to the other? If so, do you think your observations are valid in a general sense?
- Are there instance classes where one algorithm is clearly better than the other?
- How does the running time of the algorithms relate to their solution quality?
- Does your experiments concludingly answer your research questions? Are there possible follow-up experiments?

IV.2 Lösung