Universität Augsburg
Institut für Informatik
Lehrstuhl für Organic Computing
Prof. Dr. Jörg Hähner

Ansprechpartner
David Pätzel
david.paetzel@informatik.uni-augsburg.de
Eichleitnerstr. 30, Raum 502

Sommersemester 2019

Organic Computing II

Aufgabenblatt 1

Dieses Übungsblatt ist Teil der Bonusregelung. Schicken Sie Ihre Lösung in der für diese Veranstaltung festgelegten Form **bis Montag, den 13. Mai 2019, 9:00 Uhr** an obenstehende E-Mail-Adresse.

1 Erstellen eines Repositorys für die Entwicklung

Erstellen Sie im GitLab des Rechenzentrums (`https://git.rz.uni-augsburg.de`) ein neues Projekt, zu dem Sie Ihre Team-Mitglieder und den Betreuer der Übung hinzufügen (RZ-Kennung: hoffmada, mindestens mit *Reporter*-Berechtigung). Verwenden Sie dieses Projekt zur Entwicklung Ihrer Abgaben für dieses und die folgenden Übungblätter! Befolgen Sie dazu folgende Richtlinien:

- Verwenden Sie eine *Gitignore*-Datei und committen Sie insbesondere keine Compiler-Artefakte! Im Gitignore-Repository[1] finden Sie gute Standard-Gitignore-Dateien für viele Programmiersprachen.
- Lesen Sie den Artikel *How to Write a Git Commit Message*[2]! Befolgen Sie auf diesem und allen folgenden Übungsblättern die dort genannten Regeln für Commit-Messages.
- Taggen Sie immer die abgabefertige Version Ihres Codes (siehe z. B. `https://git-scm.com/book/en/v2/Git-Basics-Tagging`)! Die Tags sollen das Format „abgabeX“ haben, wobei X für die Nummer des Übungsblattes steht (für das erste Übungsblatt sollte Ihr Tag also „abgabe1“ heißen).
- Laden Sie für die Abgabe Ihren Code immer als *tar.bz2*-Archiv herunter und senden Sie dieses per E-Mail an den Betreuer der Übung!

[1] `https://github.com/github/gitignore`
[2] `https://chris.beams.io/posts/git-commit/`

2 Travelling Salesman Problem

Auf diesem und dem nächsten Übungsblatt geht es um das *Travelling Salesman Problem* (TSP), welches wir wie folgt definieren:

- Gegeben eine Menge von *Städten* V.
- Diese sind vollständig verbunden durch $\frac{|V|(|V|-1)}{2}$ bidirektionale *Straßen*, wobei $e_{i,j}$ die Länge der Straße zwischen Stadt v_i und Stadt v_j ist und eine Stadt nicht mit sich selbst verbunden ist. Da die Straßen bidirektional sind, gilt $e_{i,j} = e_{j,i}$ für alle $v_i, v_j \in V$.
- Eine *Route* ist eine Sequenz von Städten, in der jede Stadt aus V genau einmal vorkommt. Die Menge aller Routen ist R. Die Menge aller in einer Route $r = (v_1, v_2, \ldots, v_n)$ verwendeten Straßen sei $s(r) = \{e_{1,2}, e_{2,3}, \ldots, e_{n-1,n}, e_{n,1}\}$.
- Die Länge einer Route r ist definiert als $l(r) = \sum s(r)$.
- Eine Lösung für das TSP ist eine Route r_0, so dass für alle $r \in R$ gilt: $l(r_0) \leq l(r)$.

Sie dürfen sich frei eine Programmiersprache aussuchen, um die folgenden Aufgaben zu lösen (es wäre jedoch empfehlenswert, eher eine High-Level-Sprache wie Haskell, Scala, Python, Java o. ä. zu wählen). Achten Sie auf eine *saubere* Programmierweise!

2.1 Generator für TSP-Probleme

1. Entscheiden Sie sich für eine geeignete Darstellung einer Probleminstanz (z. B. eine Abstandsmatrix) und dokumentieren Sie diese präzise!
2. Implementieren Sie einen Generator für TSP-Probleme! Dabei sollen
   - die Anzahl der Städte $|V|$ konfigurierbar und
   - die Straßenlängen
     - Ganzzahlen zwischen 1 und 1000 sein, die
     - gleichverteilt pseudo-zufällig aus einem konfigurierbaren *Random seed* generiert werden.

2.2 Erschöpfende Suche

Implementieren Sie einen Lösungsalgorithmus für von Ihrem Generator (Aufgabe 2.1) generierte Probleme, der den Lösungsraum *erschöpfend* nach der besten Lösung absucht! Ihr Programm soll zu Beginn die Probleminstanz (z. B. als Abstandsmatrix) und schließlich die optimale Route sowie deren Länge ausgeben (um die Messungen nicht zu verfälschen, soll dies die einzige Ausgabe sein).

2.3 Evaluation

Verwenden Sie im Folgenden als Random seeds die Zahlen von 1 bis 11. Sie müssen also jeweils zehn Durchläufe machen und für Ihre Antworten die Ergebnisse mitteln.

1. Welche Hardware (wieviel RAM und welche CPU) verwenden Sie für Ihre Experimente?
2. Wie lange dauert es auf Ihrer Hardware durchschnittlich in Sekunden, eine Lösung für $n = 10$ zu finden? Wie viele Routen mussten dafür durchsucht werden?
3. Wie lange dauert es auf Ihrer Hardware durchschnittlich in Sekunden, eine Lösung für $n = 15$ zu finden? Wie viele Routen mussten dafür durchsucht werden?
4. Gelingt es Ihrem Algorithmus auf Ihrer Hardware, eine Lösung für $n = 15$ oder sogar für $n = 20$ zu finden? Wenn ja, wie lange dauert es? Wenn nein, wie lange würde es vermutlich dauern?

Abgabe

Ihre Abgabe hat alle in Aufgabe 1 aufgeführten Anforderungen zu erfüllen. Sie besteht aus dem *tar.bz2*-Export des Tags *abgabe1* Ihres Git-Repositorys und hat folgenden Inhalt:

- Ihren *vollständigen Code* inklusive eines *Bash-* oder *Fish-Skripts* zum Kompilieren und Starten (oder einer kurzen Anleitung, wie er kompiliert und das resultierende Programm gestartet wird).
- Eine *saubere PDF-Datei in Präsentationsform* mit Ihren Antworten auf die gestellten Fragen. Diese sollen Sie im Rahmen des nächsten Übungstermins zur Vorstellung Ihrer Lösung hernehmen können.
- Ein Zuständigkeitsprotokoll in einer PDF-Datei mit Namen *Protokoll.pdf*. In diesem muss notiert sein, welches Teammitglied welchen Teil der Lösung beigesteuert hat.