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 2

Dieses Übungsblatt ist Teil der Bonusregelung. Schicken Sie Ihre Lösung in der für diese Veranstaltung festgelegten Form **bis Montag, den 20. Mai 2019, 9:00 Uhr** an obenstehende E-Mail-Adresse.

1 Ein ameisenbasiertes Lösungsverfahren für das TSP

Dieses Übungsblatt baut auf dem Generator für TSP-Instanzen auf, den Sie auf dem vorangehenden Blatt implementiert haben.

Erweitern Sie bei der Lösung Ihre bisherige Implementierung. Teilen Sie dabei Ihren Code in drei Module auf:

- TSP (der TSP-Generator sowie grundlegende Datentypen für das TSP, sollten Sie welche eingeführt haben)
- Exhaustive (die erschöpfende Suche; benutzt das TSP-Modul)
- Ants (die Lösung für das vorliegende Übungsblatt; benutzt das TSP-Modul)

Achten Sie auf eine *saubere* Programmierweise!

1.1 Grundlagen

Verfahren zum Finden der optimalen Lösung des TSPs benötigen bereits bei kleinen Problemgrößen (Anzahl an Städten) auf handelsüblicher Hardware sehr viel Zeit. Da oftmals statt einer optimalen eine gute Lösung ausreicht, bieten sich Heuristiken als Alternative an. Ein Beispiel für eine naturinspirierte Heuristik ist das *Ant System* von Dorigo et al., welches später zum *Ant Colony System* (ACS) erweitert wurde (sowie einer leicht abgewandelten Form namens *Ant-Q*). Alle drei Systeme werden in einer Publikation mit dem Titel *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem* beschrieben, welche Sie unter folgender URL finden (von innerhalb des Uni-Netzwerks bzw. von innerhalb des Uni-VPNs kostenlos): `https://ieeexplore.ieee.org/document/585892`.

Im Folgenden geht es immer um den weiterentwickelten Ansatz (ACS) und nicht um das etwas einfachere Ant System.

Als Erstes sollen Sie sich einen Überblick über den Ansatz der Autoren verschaffen. Beantworten Sie dazu auf Basis der Veröffentlichung folgende Fragen:

1. ACS hat 6 Hyperparameter. Geben Sie für jeden an:
   - das Symbol (z. B. m),
   - die Domäne (z. B. $\mathbb{N}$) sowie
   - eine kurze Beschreibung was er konfiguriert (z. B. „Anzahl der Ameisen“)!

   Hinweis: Im Paper findet sich ein Tippfehler. Der Hyperparameter r heißt eigentlich (wie beim Ant System) ρ (mit diesem Symbol tritt er auch in der Formel direkt vorher auf).
2. Ant-Q fügt einen weiteren Parameter zum System hinzu. Geben Sie dessen Symbol an!
3. Welchen Wert für τ_0 empfehlen die Autoren? Wie schwierig ist es, an diesen Wert zu kommen? Argumentieren Sie gegebenenfalls mit O-Notation.

1.2 Implementierung

Implementieren Sie ACS[1], so dass es von Ihrem TSP-Generator erzeugte Instanzen des TSPs lösen kann! Dabei müssen Sie sich *nicht* strikt an die im Anhang des Papers vorgegebene algorithmische Struktur halten; gerade bei imperativen Implementierungen ist diese aber sicherlich hilfreich.

Ihr Programm bzw. Ihr Quellcode soll folgende zusätzliche Anforderungen erfüllen:

- Die Problemgröße, das Random Seed, die Anzahl an durchzuführenden Iterationen und die Hyperparameter sollen beim Starten des Programms über Kommandozeilenargumente konfigurierbar sein.

[1] *ohne* die in Kapitel VI des Papers vorgestellte Erweiterung durch lokale Suche

- Für die zufälligen Entscheidungen in ACS soll ebenfalls ein deterministischer (geseedeter) Pseudozufallszahlengenerator verwendet werden.
- Im Quellcode soll an jeder Stelle, an der eine Formel des Papers implementiert ist, ein Kommentar mit der entsprechenden Gleichungsnummer und einer kurzen Beschreibung stehen (z. B. „(4) globales Update der Pheromone“).
- Zu Beginn soll die Probleminstanz (z. B. als Abstandsmatrix) ausgegeben werden.
- Wenn die als Kommandozeilenargument übergebene Iterationsanzahl erreicht ist, soll Ihr Programm abbrechen und die Route mit der höchsten Pheromonkonzentration sowie deren Länge ausgeben.

1.3 Evaluation

Verwenden Sie im Folgenden als Random seeds die Zahlen von 1 bis 10. Sie müssen also jeweils zehn Durchläufe machen und für Ihre Antworten die Ergebnisse mitteln. Finden Sie für jede der Teilaufgaben gute Werte für die Hyperparameter und *geben Sie diese an*! Beziehen Sie Ausgaben nicht in ihre Messung mit ein (d. h. starten Sie Ihre Messung *nach* der initialen Ausgabe der Probleminstanz und beenden Sie sie *vor* der Ausgabe der besten gefundenen Route).

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 $n = 20$ zu finden? Wie viele Routen mussten dafür durchsucht werden?
4. Wie lange dauert es auf Ihrer Hardware durchschnittlich in Sekunden, eine Lösung für $n = 30$ zu finden? Wie viele Routen mussten dafür durchsucht werden?

Abgabe

Ihre Abgabe hat alle in Aufgabe 1 von Blatt 1 aufgeführten Anforderungen zu erfüllen. Sie besteht aus dem *tar.bz2*-Export des Tags *abgabe2* Ihres Git-Repositorys und hat folgenden Inhalt (verschicken Sie also *nicht* mehrere Dateien):

- 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 an welchem Teil der Lösung mitgearbeitet hat.