Themen für das Praktikum Algorithmen

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 2023

- Bewerbungen auf Praktikumsthemen werden bis zum **17.04.2023** um **08:00** unter der E-Mail Adresse **lab@algo.informatik.tu-darmstadt.de** angenommen. Danach ist keine Bewerbung mehr möglich. Bewerbungen unter anderen E-Mail Adressen werden ignoriert. Jede Bewerbung muss drei Themen mit Prioritäten (Erst-, Zweit- und Drittwunsch) enthalten. Zur Bewerbung auf ein Thema schreiben Sie uns bitte eine kurze Motivation, warum Sie genau dieses Thema bearbeiten möchten sowie Ihre Vorkenntnisse.
- Das Kick-Off Treffen findet nach Vereinbarung für jedes Thema separat statt.
- Um die Integration der Praktikumsergebnisse in die quelloffenen Projekte SORO-S/O bzw. MOTIS zu ermöglichen, müssen alle Teilnehmer Ihre Abgaben unter die MIT / Apache2 Lizenz stellen. Nach dem Kick-Off senden Sie dem Betreuer des Themas eine unterschriebene Lizenzvereinbarung (MIT / Apache2) und falls nötig eine unterschriebene Vertraulichkeitserkärung für die für das Praktikum bereitgestellten Daten. Dies wird gleichzeitig als verbindliche Zusage zur Bearbeitung des Themas angesehen.
- In einem zweiwöchigen Rythmus finden regelmäßige verpflichtende Treffen statt. Die genauen Termine werden beim Kick-Off Treffen vereinbart. Bei jedem Termin stellt jeder Teilnehmer einen Foliensatz mit 3 Folien vor (max. 5 min):
  - Welche Fortschritte / Erkenntnisse wurden seit dem letzten Treffen erreicht?
  - Welche nächsten Schritte sind geplant?
  - Liste mit Fragen - bitte achten Sie darauf, dass die Bewertungskriterien unter anderem selbstständiges Arbeiten berücksichtigen. Hier sollten Sie folglich nur Fragen stellen, deren Beantwortung nicht durch Google / bzw. das Lesen von bereitgestelltem Material möglich ist.
- Die endgültige Abgabe besteht aus:
  - Code, der für dieses Praktikum geschrieben wurde (beinhaltet Code, der Auswertungen, Tests oder ähnliches durchführt).
  - Einen Praktikumsbericht, in dem Sie beschreiben, welche Probleme Sie wie gelöst haben und wieso Sie diesen konkreten Ansatz gewählt haben.

Es gelten folgende Markierungen:

- **A**: Das Thema kann als Praktikum Algorithmen (6CP) oder Vertiefungspraktikum Algorithmen (6CP) bearbeitet werden.
- **P**: Das Thema kann als Projektpraktikum (9CP) bearbeitet werden.
- **E**: Das Thema kann einzeln bearbeitet werden.
- **G**: Das Thema kann auch als Gruppe bearbeitet werden.

Der Abgabetermin des Praktikums ist der 30.9.2023.

Hinweise:

- Planen Sie bitte zusätzlichen Aufwand ein, falls die genannten Programmiersprachen (C++, Typescript, Javascript, CUDA) oder verwendeten Technologien (z.B. Git, CMake, React, etc.) neu für Sie sind.
- Betrifft alle C++ Praktika: Als Referenz für die Komplexität des Projektes, das im Praktikum erweitert wird, können Sie MOTIS[1] anschauen. Das SORO Projekt aus den Themen 2.x ist ähnlich komplex.
  Bitte überlegen Sie sich genau, ob Sie mit einer großen "real-world" Modern-C++ Code Basis klarkommen. Bisherige Erfahrungen haben gezeigt, dass Studierende den Einarbeitungsaufwand und die Komplexität der Programmiersprache unterschätzen.
  Einen Crash-Kurs in die MOTIS/SORO-typische C++-Programmierung gibt es hier: `https://www.algo.informatik.tu-darmstadt.de/dl/cpp.pdf`.
- Falls Themen in Gruppen bearbeitet werden, steigt der Aufwand linear mit der Summe der CPs der Gruppenmitglieder. Die unten genannten Themen sind auf Einzelpersonen ausgelegt. Bei einer Bearbeitung in der Gruppe (nur bei Themen mit **G** möglich!) wird der Umfang entsprechend erweitert.

[1] `https://github.com/motis-project/motis`

Thema 1.1 Maximal Clique Enumeration [A, P, E, G]

Ein vollständiger Subgraph C eines Graphen G wird als Clique des Graphen G bezeichnet. Kann die Clique C nicht um weitere Knoten des Graphen G erweitert werden, spricht man von einer maximalen Clique. Nicht zu Verwechseln mit einer der größten Clique des Graphen, die maximale Anzahl Knoten über alle Cliquen hinweg besitzt. Beim Problem der Maximal Clique Enumeration (MCE) wird eine Aufzählung aller maximalen Cliquen gesucht. Bei MCE handelt es sich um ein NP-schweres Problem, das bereits vielfach in der Literatur untersucht wurde.

Aufgabe des Praktikums ist Literaturrecherche zum Vergleichen von bereits existiernden Algorithmen und Implementieren eines Algorithmus inklusive ausführlichem Testen. Bei der Implementierung soll die Performanz ausführlich ausgewertet werden.

Programmiersprache: C++20

Thema 1.2: Maintaining a Topological Order after Batch Insertion [A, P, E, G]

Zu einem gerichteten, azyklischen Graphen (DAG) G kann eine topologische Sortierung der Knoten ermittelt werden. Bei einer topologischen Sortierung handelt es sich um eine Reihenfolge der Knoten, sodass für jede Kante (u, v) aus gilt: u erscheint vor v in der Sortierung. Offensichtlich bleibt eine gegebene topologische Sortierung gültig, falls man Kanten aus dem Graphen entfernt. Anders verhält es sich beim Einfügen von neuen Kanten. Durch Zykelschluss kann das Ermitteln einer topologischen Sortierung unmöglich werden oder eine bestehende topologische Sortierung ungültig werden. Konkrete Problemstellung des Praktikums ist das möglicherweise notwendige Reparieren einer topologischen Sortierung nach Einfügen einer Menge neuer Kanten (Batch) in einen DAG.

Aufgabe des Praktikums ist Literaturrecherche zum Vergleichen von bereits existierenden Algorithmen und Implementieren eines Algorithmus inklusive ausführlichem Testen. Bei der Implementierung soll die Performanz ausführlich ausgewertet werden.

Programmiersprache: C++20

Thema 1.3: Graph-based Mapping for a Railway Infrastructure [A, P, E, G]

Bei OpenStreetMap (OSM) handelt es sich um ein offenes Kartenprojekt. In den OSM-Daten sind Kartendaten für das deutsche Schienennetz enthalten. Diese bestehen aus ’Nodes’ mit GPS-Koordinaten und weiteren Informationen, wobei die ’Nodes’ über

'Ways' und 'Relations' kombiniert werden. Zusätzlich liegen am Fachgebiet Algorithmik Infrastrukturdaten für das deutsche Schienennetz vor. Allerdings fehlen hier die GPS-Koordinaten. Ziel des Praktikums ist es diese beiden Datensätze miteinander zu verbinden. Daraufhin sollen die in den FGAlgo-Daten vorhandendenen, aber in den OSM-Daten nicht vorhandendenen, fehlenden Positionsdaten ermittelt werden. Da dabei die Schienentopologie ausschlaggebend ist müssen beide Datensätze als Graph modelliert und vergleichen werden.

Programmiersprache: C++20; frei, nach Absprache

Thema 2.1: Serialisierbarer Quad-Tree/R-Tree mit `cista` [A, P, E]

Die `cista`[2] Library erlaubt die effiziente Serialisierung von C++ Datenstrukturen. Grundlegende Datenstrukturen wie `string`, `vector`, etc. werden von `cista` bereits reimplementiert. Aufgabe in diesem Praktikumsthema ist es, eine effiziente Geo-Lookup Datenstruktur in `cista` zu implementieren und gegen gängige Implementierungen (wie z.B. Boost Geomertry R-Tree) bezüglich Zeit zum Aufbauen, Lookup-Dauer sowie der benötigten Zeit für schreiben/lesen von der Festplatte zu benchmarken (z.B. mit Google benchmark[3]).

Programmiersprache: C++20

Thema 3.1: Effiziente Optimierung einer Taxi-On-Demand-Flotte [A, P, G, E]

In diesem Projekt sollen verschiedene Scheduling-Algorithmen für ein On-Demand-Transport System[4] entworfen werden. Das System soll die Fahrten bis zu zwei Wochen in die Zukunft planen können. Zu Beginn wird eine vollständige Literaturübersicht zu bisherigen Ansätzen in diesem oder verwandten Bereichen (Ride Hailing, Demand Responsive Transport/DRT, etc.) erstellt.

Als Eingabe bekommt das System das Straßennetz (OpenStreetMap, fertige Routing Engines wie z.B. OSRM können genutzt werden), die Verfügbarkeit von Fahrern sowie eine sortierte Liste von Kunden-Anfragen (Abfahrtszeit/Ankunftszeit, von wo nach wo), die zu verschiedenen Zeitpunkten (z.B. kurz vor der Fahrt, zwei Wochen vor der Fahrt) eingehen. Jede Kundenanfrage wird von dem System akzeptiert oder abgelehnt. Der zu entwickelnde Algorithmus berechnet eine praktisch durchführbare Zuweisung, wann welcher Fahrer welchen Kunden mitnimmt und absetzt. Ziel ist es, die Anzahl der gefahrenen Kilometer zu minimieren, z.B. indem die Fahrzeugkapaztität maximal ausgenutzt wird.

[2] https://github.com/felixguendling/cista
[3] https://github.com/google/benchmark
[4] https://www.verkehrswendebuero.de/on-demand-verkehr/

Es müssen mindestens zwei verschiedene Ansätze (Greedy Algorithmus vs. Optimierung mit Scheduling Algorithmen) erarbeitet, beschrieben, implementiert und in einer aussagekräftigen Auswertung bzgl. Performanz und Qualität verglichen werden. Zum Einsatz kommen sollten auch Beschleunigungstechniken, wie z.B. die Verwendung von unteren Schranken (Luftlinien), um Lösungen frühzeitig ohne aufwändige Routings auf der Straße verwerfen zu können. Es ist möglich, Kunden eine andere Ankunfts/Abfahrtszeit zu kommunizieren als die gewünschte - dies kann insbesondere dafür genutzt werden, um einen extra Puffer in Fahrten einzuplanen, der es ermöglicht weitere Kunden mitzunehmen und abzusetzen ohne die kommunizierte Zeit zu verletzen.

Beschreibung einer möglichen Greedy-Lösung zum Einfügen einer neuen Fahrt in den bisherigen Fahrplan:

Es soll auf einfache Weise versucht werden, die Zahl der gefahrenen Kilometer zu minimieren. D.h. es übernimmt immer der Fahrer die Fahrt, der im Vergleich zur vorherigen Lösung den geringsten Zusatzaufwand (zusätzlich zurückgelegte Kilometer) hat. Es kommen nur Fahrer in Frage, bei denen die Fahrt eingebaut werden kann, ohne andere kommunizierte Ankunftszeiten zu verletzen. Pickup und Dropoff können beliebig aufeinander folgen, solange alle kommunizierten Ankunftszeiten eingehalten werden. Z.B. bisherige Fahrt P1 - P2 - D2 - D1 (P=Pickup, P1=Pickup User 1, D=Dropoff, D1 = Dropoff User 1). Mögliche Kombinationen:

```
PX - P1 - P2 - D2 - D1 - DX
P1 - PX - P2 - D2 - D1 - DX
P1 - P2 - PX - D2 - D1 - DX
P1 - P2 - D2 - PX - D1 - DX
P1 - P2 - D2 - D1 - PX - DX

PX - P1 - P2 - D2 - DX - D1
P1 - PX - P2 - D2 - DX - D1
P1 - P2 - PX - D2 - DX - D1
P1 - P2 - D2 - PX - DX - D1

PX - P1 - P2 - DX - D2 - D1
P1 - PX - P2 - DX - D2 - D1
P1 - P2 - PX - DX - D2 - D1

PX - P1 - DX - P2 - D2 - D1
P1 - PX - DX - P2 - D2 - D1

PX - DX - P1 - P2 - D2 - D1
```

D.h. für ein Zeitfenster um Pickup + Dropoff werden - alle sinnvollen Kombinationen für alle Fahrer generiert - per Routing (OSRM) durchgerechnet - ungültige Kombinationen eliminiert (versprochene Ankunftszeiten nicht eingehalten)

Aus den verbleibenden Lösungen wird die Lösung gewählt, die den geringsten zusätzlichen Kilometer-Aufwand erzeugt.

Programmiersprache: C++20