Kurzbeschreibung

-----------------

Gegeben sei ein gerichteter Graph. eine Kante hat des weiteren eine Kapazitätsfunktion f, die beschreibt, wie viele Objekte sich auf der Kante befinden und

maximal befinden können. Sie bildet also jede Kante auf ein Tupel aus zwei natürlichen Zahlen ab. Des Weiteren hat jede Kante ein Kantengewicht, welches

beschreibt, wie schnell ein Objekt durch die Kante vorraussichtlich traversieren kann. Jedes Objekt hat einen Startpunkt und einen Zielpunkt und dem-

zufolge einen Bewegungspfad. Jedes Objekt weiß, wie sich die anderen Objekte zu ihrem Ziel bewegen möchten und können ihren Pfad an jedem

Knoten ändern. Ziel ist es nun, dass alle Objekte möglichst schnell ihre Ziele erreichen. Dieses Problem gilt es zu parallelisieren.

Übertragen auf ein weniger abstraktes Szenario, könnte dies die Bewegung von Autos durch ein Straßennetz sein, die durch kommunikation

untereinander intelligent ihren Weg bestimmen können. Die Straßen sind hierbei die Kanten und die Knoten Ampeln. Autos können ihren Pfad

also an jeder Ampel ändern, was alle anderen Autos mitbekommen und ihren Pfad demzufolge auch anpassen können.

Insgesamt ist Lösung zwischen mikroskopisch und mesoskopische Sichtweise des Problems! http://vision-traffic.ptvgroup.com/en-uk/products/ptv-vissim/use-cases/mesoscopic-and-hybrid-simulation/

Lösungsansatz:

Der Algorithmus würde wie folgt funktionieren. Ein Ausführungsschritt des Algorithmus bezeichnen wir Step.

Jeder Step stellt die kleinste Zeiteinheit in der realen welt dar. Jede Kante hat eine Queue die speichert, welche Objekte sich momentan

auf der Kante befinden. Da diese Sortiert sind und die Fahrgeschwindigkeit der Objekte gleich ist, ist zu jedem Zeitpunkt klar

welches Objekt das Ende der Kante erreicht. Des Weiteren speichert die Queue für jedes Objekt wieviele Meter noch benötigt

werden bis das Objekt bei normaler Fahrt das Ende erreicht (rationale Zahlen). Da die Länge der Straße bekannt ist und die Geschwindigkeit des Autos auch ist

dies nur eine simple Division. Ein Auto kann dann die Kante verlassen, wenn die restliche Distanz 0 ist. Es kann nur ein Auto die gleiche restZahl

in der Queue haben und die Differnz der restZahl muss mind. der Mindestabstand sein. Jedes Auto versucht in einem Step eine global gesetzte Anzahl an Metern

zu fahren. Möchte ein Objekt über eine Kante in eine neue Kante übertreten, muss der Knoten(Ampel) dies zulassen. Der Knoten erlaubt jeweils zwei

gegenüberliegenen Kanten gleichzeitig die Durchfahrt. Ist die Durchfahrt nicht gewährleistet muss ein Fahrzeug was eine restdistanz von 0 in der Queue hat,

warten.

Ist ein auto an dem Zielknoten angekommen, wird das Auto gelöscht.

Des Weiteren werden abhängig vom Input neue Autos an Spawnpunkten generiert.

Danach werden die Pfade für die Objekte aktualisiert. Durch das löschen und erzeugen der Objekte kann es durchaus vorkommen, dass die optimalen Pfade sich

für die Objekte ändern, weshalb es wichtig ist für alle Objekte die eine restDistanz von 0 haben ihren Pfad überprüfen zu lassen. Dies geschieht

beispielsweise durch den D\* lite algorithmus, der dynamische änderungen der Karte verarbeiten kann, ohne komplett neu ausgeführt zu werden. Wurde ein optimaler

Pfad für ein Auto gefunden, werden alle Kantengewichte, abhängig von der Distanz des Objektes zur Kante um einen Faktor erhöht. So kann ein gewisser Fahr-

trend für die anderen Objekte festgestellt werden, sodass sie irgendwann eine alternative Route wählen, da die Kantengewichte zu hoch sind.

Der Algorithmus terminiert, sobald für alle Kanten gilt, dass die Anzahl der Objekte auf den Kanten 0 entspricht und die Eingabe abgearbeitet wurde.

Ein-Ausgabe: Anzahl der Steps bis Programm terminieren soll, Geschwindigkeit, Karte, Mindestabstand,

Schaltgeschwindigkeit der Ampel

Ausgabe: Jede N(am Anfang gleich 1) Steps gib für jedes Auto die Kante, die verbleibenen Meter bis ende aus.

(Eventuell auch für alle Kanten, die Kantengewichte ausgeben) (vielleicht am Anfang zum Testen auch konkrete Pfade von Autos).

Ampelstati

FALLS ZU KOMPLIZIERT: GIB FÜR JEDE KANTE DIE ANZAHL DER AUTOS AUS

Datenvisualisierung:

Position der Autos aus Daten berechnen. (Vielleicht auch schon epxlizit abspeichern) (Winkel der Kanten berechnen und somit bewegung von Auto)

X Beliebige Graphen generieren. Wichtig: Berechne durchschnittliche Anzahl der Autos auf Kante und stelle für alle Kanten (oder n zusammenfasssen) die

Durchschnittliche Anzahl an Autos graphisch dar.

Oder Anzahl Autos für Kanten über Steps.

Projektplan:

5-12.06 Konzept besprechen, verbessern und finalisieren. (insbesondere Parallelität) Umgebung und Sprache auswählen(Fragen!), Github repository öffnen

12-19.06 Informiere über Datentyp zum Speichern der Map(.dat) und erzeuge Testgraph. Erzeuge Klasse Car, Intersection(Trafficlight), Queue, Road, Spawner,

World, Testdurchlauf, dass Auto bei festgegebener Route zum Ziel findet und Queue und Kantengewichte nagepasst werden, Textuelle Ausgabe

19-26.06 Implementierung des Pathfinding-Algorithmus (ink. Kantengewicht update) und Visualisierung der Ergebnisse(java, libgdx). Einbinden von Open-Street map,

Sequenzielles Programm fertig!

26-03.07 Leistungsanalyse mit oprofile. Kleine Optimierungen. Beginne Parallelisierung durch Zerlegung der Karte und aufteilen durch MPI.

03-10.07 Fortsetzen von Implementierung MPI erste tests.

10-17.07 MPI und Klausuren

17-24.07 Klausuren

24-31.07 Klausuren

31-06.08 MPI wirklich feritg und OPenMP gut einbauen. Leistungsbewertung mit Vampire und Optimierung Kommunikationsmuster

06-13.08 PARALLELISIERUNG FERTIG! SPEEDUP DIAGRAMME UND SCHRIFTLICHE AUSARBEITUNG

Nachfragen, wie das mit Präsentation und Ausarbeitung aussieht. Müssen inhaltlich übereinstimmen, oder kann an schrift. und Code noch danach gearbeitet werden.

WICHTIG IMMER: IMMER ZU THEORETISCHEN AUSARBEITUNGEN UND WICHTIGEN ERKENNTNISSEN DIREKT AUFSCHREIBEN. (Beispielsweise irgendwas hat mega viel Kommunikationsaufwand

erzeugt, gefixxed, aufschreiben) für PROTOKOLLL!!!!!!!!

Parallelisierung:

Das Problem wird erstmal in kleine Komponenten per MPI zerlegt, die dann jeweils von einem Node

bearbeitet werden können, über den gleichen Speicherbereich.

Zerlege hierfür die Karte in quadratische Teile MPI Prozesse (2^n). Jeder Prozess weiß wie die gesamte Karte aussieht

zu jedem Zeitpunkt.

[TODO NOT SURE)

Parallelisierung zwischen den Prozessen wird kommuniziert, sobald ein Auto einen Bereich eines anderen Prozesses

betritt.

Parallels Wegfinden: Berechne Parallel für jedes Auto die Strecke. Evaluiere danach, ob in Straßen eine

Überfüllung stattgefunden hat und berechne für diese Autos die Strecke mit den neuen Kantengewichten. Überprüfe erneut

und akzeptiere wenn fertig.

Für fall dass alle Wege überfüllt sind überpüfe ob anzahl der Überfüllten autos gleich geblieben ist.

------------------------------------------------------------------

Modell:

alles in Steps(auch , Steps, also rationale Values >= 0), später für Simulation wird Zeit für einen Step definiert.

Autos entstehen an Spawnpunkten und werden entfernt sobald sie Ziel erreichen.

Ziel: Zu Beginn hat jedes Fahrzeug das Gleiche Ziel

Später: eine definierte anzahl an Zielpunkten konstruieren, die Wahrscheinlichkeiten haben als Ziel ausgewählt

zu werden.

Jeder Knoten ist eine Ampel und stellt übergang zwischen zwei Straßen da.

Kante ist Weg zwischen Ampeln

Jede Kante hat kapazität(queue), die angibt, wieviele Fahrzeuge noch auf den Weg passen.

Verhalten:

Jedes Auto hat festgelegte Geschwindigkeit. "Beschleunigung und Bremsen wird über Funktionsweise

der Ampeln und Knoten definiert.

Idealisierungen bzw. Vereinfachungen:

----------------------------------------

- Keine verschiedenen Fahrzeugtypen betrachten

-