Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

Funktionsprinzipien und Anwendungen von Algorithmen zur Pfadplanung

Bearbeiter 1: Mohammed Salih Mezraoui
Bearbeiter 2: David Gruber
Bearbeiter 3: Marius Müller

Gruppe: WissArb22/Thema 2/Gruppe-7

Ausarbeitung zur Vorlesung Wissenschaftliches Arbeiten

Trier, 15.07.2022

Kurzfassung

In der Kurzfassung soll in kurzer und prägnanter Weise der wesentliche Inhalt der Arbeit beschrieben werden. Dazu zählen vor allem eine kurze Aufgabenbeschreibung, der Lösungsansatz sowie die wesentlichen Ergebnisse der Arbeit. Ein häufiger Fehler für die Kurzfassung ist, dass lediglich die Aufgabenbeschreibung (d.h. das Problem) in Kurzform vorgelegt wird. Die Kurzfassung soll aber die gesamte Arbeit widerspiegeln. Deshalb sind vor allem die erzielten Ergebnisse darzustellen. Die Kurzfassung soll etwa eine halbe bis ganze DIN-A4-Seite umfassen.

Hinweis: Schreiben Sie die Kurzfassung am Ende der Arbeit, denn eventuell ist Ihnen beim Schreiben erst vollends klar geworden, was das Wesentliche der Arbeit ist bzw. welche Schwerpunkte Sie bei der Arbeit gesetzt haben. Andernfalls laufen Sie Gefahr, dass die Kurzfassung nicht zum Rest der Arbeit passt.

The same in english.

Inhaltsverzeichnis

1

Einleitung und Problemstellung

Unter Pathfinding bzw. Wegfindung versteht man in der Informatik die algorithmengestütze Suche nach dem optimalen Weg(en) von einem Startpunkt zu einem oder mehreren Zielpunkten. In diesem Paper werden wir verschiedene Pfadplanungsalgorithmen wie den Dijkstra-Algorithmus und seine Variante, die häufig in Verkehrleitsystemen wie etwa Google Maps verwendet werden.

Um die Geschwindigkeit des Dijkstra-Algorithmus bei Blindsuchen zu optimieren werden A* und seine Varianten als Stand der Technik-Algorithmen für den Einsatz in statischen Umgebungen vorgestellt [Salah:2]

Das Thema Funktionsprinzipien und Anwendungen von Algorithmen zur Pfadplanung hat damals, wie heute einen wichtigen Stellenwert. Ob im Bereich der Netzwerkroutenplanung oder bei KI-Spielern in Computerspielen Pfadsuchalgorithmen sind so relevant wie nie. Weitere Beispiele für die Anwendung von Pfadsuchalgroithmen sind Robotik (z.B. Pakete im Logistikbereich), Planung von öffentlichen Verkehrsmitteln und Routenplanung von Navigationssystemen. Viele Bereiche im Alltag verwenden im Hintergrund Pfadsuchalgorithmen um den (kosten)günstigsten Weg zu finden, dabei ist es wichtig, dass diese effizient, akkurat und schnell sein müssen, damit die Hauptsysteme noch genügen Ressourcen übrig haben um gewünscht zu funktionieren. [FGK+21]

Wir werden uns in diesem Paper die Funktionsweise der wichtigsten Pfadsuchalgorithmen, angefangen mit den „einfachen“ uninformierten Suchalgorithmen wie der Breiten- oder Tiefensuche, welche einen ersten Einblick in die Thematik geben sollen und die Rahmenbedingungen und Problemumgebungen veranschaulichen sollen; Bis hin zu durch Heuristiken optimierte und informierte Pfadsuchalgorithmen wie Dijsktra oder A*, die häufig in Verkehrleitsystemen wie etwa Google Maps verwendet werden, und wie sie in heutiger Zeit sonst eingesetzt werden können, veranschaulichen und jeweils (Pseudo-)Code- und Anwendungsbeispiele geben.[RN10]

2

Algorithmen zur Pfadplanung

In diesem Kapitel wird beschrieben, warum es unterschiedliche Konsistenzmodelle gibt. Außerdem werden die Unterschiede zwischen strengen Konsistenzmodellen (Linearisierbarkeit, sequentielle Konsistenz) und schwachen Konsistenzmodellen (schwache Konsistenz, Freigabekonsistenz) erläutert. Es wird geklärt, was Strenge und Kosten (billig, teuer) in Zusammenhang mit Konsistenzmodellen bedeuten.

2.1 Warum existieren unterschiedliche Konsistenzmodelle?

Nach [Mos93] kann die Performanzsteigerung der schwächeren Konsistenzmodelle wegen der Optimierung (Pufferung, Code-Scheduling, Pipelines) 10-40 Prozent betragen. Wenn man bedenkt, dass mit der Nutzung der vorhandenen Synchronisierungsmechanismen schwächere Konsistenzmodelle den Anforderungen der strengen Konsistenz genügen, stellt sich der höhere programmiertechnischer Aufwand bei der Implementierung der schwächeren Konsistenzmodelle als ihr einziges Manko dar.

3

Optimierungsstrategien

In diesem Kapitel wird beschrieben, warum es unterschiedliche Konsistenzmodelle gibt. Außerdem werden die Unterschiede zwischen strengen Konsistenzmodellen (Linearisierbarkeit, sequentielle Konsistenz) und schwachen Konsistenzmodellen (schwache Konsistenz, Freigabekonsistenz) erläutert. Es wird geklärt, was Strenge und Kosten (billig, teuer) in Zusammenhang mit Konsistenzmodellen bedeuten.

3.1 Warum existieren unterschiedliche Konsistenzmodelle?

Laut [Mal97] sind mit der Replikation von Daten immer zwei gegensätzliche Ziele verbunden: die Erhöhung der Verfügbarkeit und die Sicherung der Konsistenz der Daten. Die Form der Konsistenzsicherung bestimmt dabei, inwiefern das eine Kriterium erfüllt und das andere dementsprechend nicht erfüllt ist (Trade-off zwischen Verfügbarkeit und der Konsistenz der Daten). Stark konsistente Daten sind stabil, das heißt, falls mehrere Kopien der Daten existieren, dürfen keine Abweichungen auftreten. Die Verfügbarkeit der Daten ist hier jedoch stark eingeschränkt. Je schwächer die Konsistenz wird, desto mehr Abweichungen können zwischen verschiedenen Kopien einer Datei auftreten, wobei die Konsistenz nur an bestimmten Synchronisationspunkten gewährleistet wird. Dafür steigt aber die Verfügbarkeit der Daten, weil sie sich leichter replizieren lassen.

4

Anwendungen

In diesem Kapitel wird beschrieben, warum es unterschiedliche Konsistenzmodelle gibt. Außerdem werden die Unterschiede zwischen strengen Konsistenzmodellen (Linearisierbarkeit, sequentielle Konsistenz) und schwachen Konsistenzmodellen (schwache Konsistenz, Freigabekonsistenz) erläutert. Es wird geklärt, was Strenge und Kosten (billig, teuer) in Zusammenhang mit Konsistenzmodellen bedeuten.

4.1 Warum existieren unterschiedliche Konsistenzmodelle?

Laut [Mal97] sind mit der Replikation von Daten immer zwei gegensätzliche Ziele verbunden: die Erhöhung der Verfügbarkeit und die Sicherung der Konsistenz der Daten. Die Form der Konsistenzsicherung bestimmt dabei, inwiefern das eine Kriterium erfüllt und das andere dementsprechend nicht erfüllt ist (Trade-off zwischen Verfügbarkeit und der Konsistenz der Daten). Stark konsistente Daten sind stabil, das heißt, falls mehrere Kopien der Daten existieren, dürfen keine Abweichungen auftreten. Die Verfügbarkeit der Daten ist hier jedoch stark eingeschränkt. Je schwächer die Konsistenz wird, desto mehr Abweichungen können zwischen verschiedenen Kopien einer Datei auftreten, wobei die Konsistenz nur an bestimmten Synchronisationspunkten gewährleistet wird. Dafür steigt aber die Verfügbarkeit der Daten, weil sie sich leichter replizieren lassen.

5

Zusammenfassung und Ausblick

In diesem Paper wurden Methoden vorgestellt, die ein Akteur zur Auswahl von Aktionen in Umgebungen wie z.B Graphen verwenden kann, die deterministisch, beobachtbar, statisch und vollständig bekannt sind. Mit diesen Methoden kann der Akteur durch eine Sequenz von Aktionen sein Ziel erreichen. Diesen Prozess nennt man Suche.

- Bevor ein Akteur mit der Suche nach Lösungen beginnen kann, muss ein Ziel identifiziert und ein Problem genau definiert Problem werden.
- Ein Problem besteht aus fünf Teilen: - dem Ausgangszustand, - einer Reihe von Aktionen, - einem Übergangsmodell, das die Ergebnisse dieser Aktionen beschreibt, - einer Zielüberprüfungsfunktion - und einer Pfadkostenfunktion. Das Problem wird durch eine Menge an Zuständen dargestellt. Ein Pfad der aus der Zustandsmenge besteht und vom Start zum Ziel führt ist eine Lösung.
- Suchalgorithmen behandeln Zustände und Aktionen atomar: Sie berücksichtigen keine interne Struktur, die sie besitzen könnten.
- Ein allgemeiner TREE-SEARCH-Algorithmus berücksichtigt alle möglichen Wege, um eine Lösung zu finden, während ein GRAPH-SEARCH-Algorithmus redundante Wege vermeidet.
- Suchalgorithmen werden auf der Grundlage von Vollständigkeit, Optimalität, Zeit- und Raumkomplexität beurteilt. Die Komplexität hängt von b, dem Verzweigungsfaktor im Zustandsraum, und d, der Tiefe der tiefsten Lösung, ab.
- Uninformierte Suchmethoden haben nur Zugriff auf die Problemdefinition. Die grundlegenden Algorithmen sind wie folgt:
  - Die Breadth-first-Suche expandiert zuerst die flachsten Knoten; sie ist vollständig, optimal für einheitliche Pfadkosten, hat aber eine exponentielle Raumkomplexität.
  - Die Uniform-Cost-Suche expandiert den Knoten mit den niedrigsten Pfadkosten, g(n), und ist optimal für allgemeine Pfadkosten.
  - Die Tiefensuche expandiert zuerst den tiefsten nicht expandierten Knoten. Sie ist weder vollständig noch optimal, hat aber eine lineare Raumkomplexität.
  - Die iterative Vertiefungssuche ist eine Wiederholung der Tiefensuche mit zunehmender Tiefenbegrenzung, bis ein Ziel gefunden wird. Sie ist vollständig,

optimal für die Kosten pro Schritt, hat eine vergleichbare Zeitkomplexität wie die Breitensuche und eine lineare Raumkomplexität.
  - Die bidirektionale Suche kann die Zeitkomplexität enorm reduzieren, ist aber nicht immer anwendbar und kann zu viel Speicherplatz beanspruchen.
- Informierte Suchmethoden können auf eine heuristische Funktion h(n) zurückgreifen, die die Kosten einer Lösung aus n schätzt.
  - Der generische Best-First-Suchalgorithmus wählt einen Knoten für die Expansion gemäß einer Bewertungsfunktion aus.
  - Der Greedy best-first search expandiert Knoten mit minimalem h(n). Er ist nicht optimal, aber oft effizient.
  - A*-Suche expandiert Knoten mit minimalem f (n) = g(n) + h(n). A* ist vollständig und optimal, vorausgesetzt, h(n) ist zulässig (für TREE-SEARCH) oder konsistent (für GRAPH-SEARCH).
  - RBFS (rekursive Best-First-Suche) und SMA* (vereinfachtes speicherbegrenztes A*) sind robuste, optimale Suchalgorithmen, die nur begrenzte Mengen an Speicher verwenden; mit genügend Zeit können sie Probleme lösen, die A* nicht lösen kann, weil ihm der Speicher ausgeht.
  - Die Leistung von heuristischen Suchalgorithmen hängt von der Qualität der heuristischen Funktion ab. Manchmal kann man gute Heuristiken konstruieren, indem man die Problemdefinition lockert und vorberechnete Lösungskosten für Teilprobleme in einer Musterdatenbank speichert.

[RN10]

Literaturverzeichnis

FGK+21. Foeada, Daniel, Alifio Ghifaria, Marchel Budi Kusumaa, Novita Hanafiahb und Eric Gunawanb: *A Systematic Literature Review of A* Pathfinding.* Elsevier B.V, 2021.

Mal97. Malte, Peter: *Replikation in Mobil Computing.* Seminar No 31/1997, Institut für Telematik der Universität Karlsruhe, Karsruhe, 1997. `http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=/ira/1997/31`.

Mos93. Mosberger, David: *Memory Consistency Models.* Technical Report 93/11, University of Arizona, November 1993.

RN10. Russell, Stuart J. und Peter Norvig: *Artificial Intelligence: A Modern Approach (Third Edition).* PearsonEducation,Inc., 2010.

Sachverzeichnis

A

Glossar

DisASTer	DisASTer (Distributed Algorithms Simulation Terrain), A platform for the Implementation of Distributed Algorithms
DSM	Distributed Shared Memory
AC	Linearisierbarkeit (atomic consistency)
SC	Sequentielle Konsistenz (sequential consistency)
WC	Schwache Konsistenz (weak consistency)
RC	Freigabekonsistenz (release consistency)

Arbeitsverteilung

Bitte tragen Sie auf dieser Seite ein, welches Mitglied Ihrer Gruppe welche Inhalte formuliert hat, da die Notengebung für jeden Studenten individuell vorgenommen wird.

Teilnehmer 1:

Inhalte:

Teilnehmer 2:

Inhalte:

Teilnehmer 3:

Inhalte: