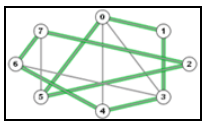
NP-vollständige Probleme in der GraphBench



Hamilton Kreis

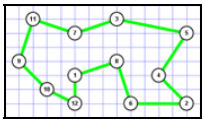
Ein Hamilton Kreis ist ein Pfad, der jeden Knoten genau einmal besucht. Existiert in einem Graphen ein Hamilton Kreis? Das Problem rund um diese Frage ist ein schwieriges Problem und NP-vollständig.

Eine Stadt kann einfach als Graph betrachtet werden, indem man sich die Kreuzungen als Knoten und die Strassenabschnitte als Kanten vorstellt. Als Beispiel für das Hamilton Kreis Problem in solch einem Graphen haben wir bereits in der Einführung die Suche nach einer Tour des Reisebusses kennengelernt.

Ein Hamilton Kreis lässt sich unter anderem mit Backtracking finden. Man beginnt bei einem Knoten und speichert die Pfade ausgehend von diesem Knoten zu unbesuchten Knoten. Für jeden Pfad speichert man neue Pfade ausgehend vom letzten Knoten zu unbesuchten Nachbarn. Diese Schritte führt man durch, bis ein Hamilton Kreis gefunden wird oder bis alle Pfade besucht wurden.

Wir können das Problem sowohl für ungerichtete als auch für gerichtete Graphen lösen, d.h. wenn die Kanten eine Richtung haben. Eine gerichtete Kante kann man mit einer Einbahnstrasse vergleichen. Ungerichtete Graphen lassen sich auf gerichtete reduzieren.

Eine wichtige Anwendung von Hamilton Kreisen treffen wir im nächsten Problem an.



Travelling Salesman (Problem des Handelsreisenden)

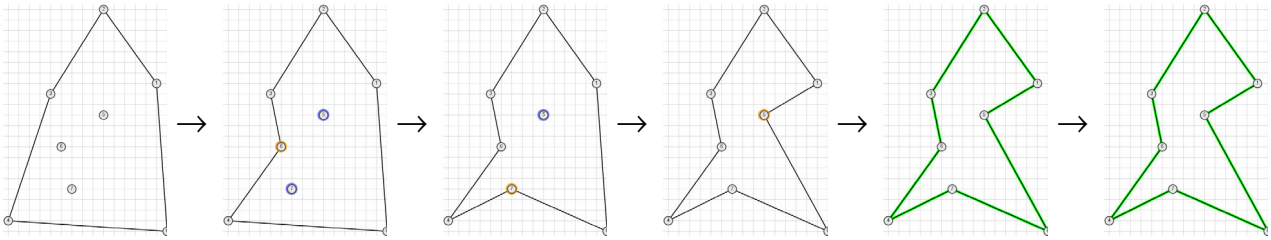
Das Travelling Salesman Problem ist verwandt mit dem Hamilton Kreis Problem. Den Kanten sind nun aber Gewichte zugewiesen. In der Entscheidungsvariante beantwortet man die Frage, ob in einem Graphen eine Tour existiert, die kürzer als *k* ist. Im Optimierungsproblem wird die kürzeste Travelling Salesman Tour gesucht. Eine Tour besucht jede Stadt genau einmal.

Wahrscheinlich kennen viele Leute dieses Problem aus dem Alltag. Es modelliert das Problem, mehrere Orte nacheinander auf dem kürzesten Weg zu besuchen und zum Anfangsort zurückzukehren. Handelsreisende, die

verschiedene Orte besuchen und den kürzesten Weg gehen möchten, müssen dieses Problem lösen können, daher der Name. Auch beim Postbeamten, der die Zeitungen austrägt, tritt dieses Problem auf. Jedoch ist dies nicht das beste Beispiel. Wieso sollten Handelsreisende eine Stadt oder Postbeamte ein Haus nicht mehrmals besuchen, wenn sie dadurch Zeit sparen?

Wir betrachten nun Algorithmen für das euklidische Travelling Salesman Problem. Im euklidischen TSP Problem entspricht das Gewicht einer Kante seiner Länge.
Mit Backtracking lässt sich dieses Problem natürlich lösen, jedoch sehr ineffzient. Der Algorithmus wählt die erste Kante und kreiert neue Pfade, indem er Kanten zu den unbesuchten Kanten hinzufügt. Dies macht er solange, bis entweder alle Pfade eine Tour sind oder bis alle Pfade länger als die bereits gefundene kürzeste Tour ist.

Dieses Problem fehlt in keinem Informatikunterricht und ist gut untersucht. Es existieren auch zahlreiche Heuristiken. Die Nearest Neighbour Heuristik beginnt beim ersten Knoten und wählt immer den nächsten (dessen Verbindungskante das kleinste Gewicht hat) unbesuchten Knoten bis kein Knoten mehr übrig bleibt. Das geht natürlich sehr schnell, ist aber dementsprechenend ungenau. Etwas besser schneidet die konvexe Hülle Heuristik ab. Sie erzeugt zuerst die konvexe Hülle aller Knoten. Dann wird der Knoten ausgewählt, welcher der Tour am nächsten liegt und hinzugefügt. Der Algorithmus terminiert, wenn alle Knoten Teil der Tour sind.

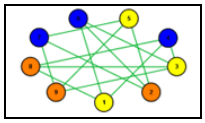


Die Greedy Heuristik wählt die Kante mit dem kleinsten Gewicht und fügt sie zur Tour hinzu, sofern die Kante keinen Zyklus erzeugt, bis kein Knoten mehr übrig bleibt. Two-Optimization generiert zuerst eine zufällige Tour. Danach sucht sie zwei Kanten und tauscht deren Endknoten, falls dies die Tour verkleinert. Dies macht der Algorithmus, bis keine zwei solchen Kanten mehr gefunden werden.

Nochmals zur Erinnerung: das Problem des Handlungsreisenden ist NP-vollständig. Das bedeutet, es ist zwar lösbar, jedoch mit erheblichem Aufwand. Alle bekannten korrekten Algorithmen sind so langsam, dass sie schon bei Graphen mit 300 Knoten nicht mehr in nützlicher Zeit ein Resultat generieren. Ein Algorithmus, der alle Pfade durchsucht, müsste in einem

Graphen mit 300 Knoten ungefähr 300! (mein Taschenrechner kann nicht einmal mehr 300! berechnen) Touren durchlaufen. Deshalb braucht man die Heuristiken, die als Abschätzung oftmals genügen.

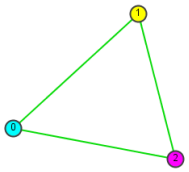
Das Sightseeing Bus Unternehmen könnte damit die kürzeste Tour in der Stadt berechnen, um Benzin und Zeit zu sparen.



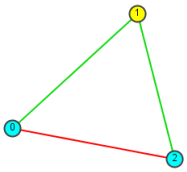
Färbung von Graphen (Colorability)

Eine beliebte Aufgabe im Geographieunterricht ist es, eine Landkarte mit verschiedenen Farbstiften so einzufärben, dass keine benachbarten Länder dieselbe Farbe aufweisen. Du fragst dich eventuell, wieviele verschiedene Buntstifte du dazu benötigst. Reichen 4 Farbstifte aus? Doch so einfach ist dieses Problem nicht zu lösen. Es handelt sich hierbei um einen Spezialfall des Graphfärbungsproblem.

Jedem Knoten wird eine Farbe zugewiesen. Eine Färbung ist gültig, wenn keine benachbarten Knoten mit derselben Farbe eingefärbt wurden. Knoten werden benachbart genannt, wenn sie durch eine Kante direkt verbunden sind.



gültige Färbung



ungültige Färbung

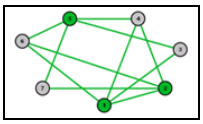
Ein Graph heisst *k*-färbbar, wenn er mit *k* Farben eingefärbt werden kann. Das kleinstmögliche *k* zu finden, ist ein NP-vollständiges Problem und daher aufwändig zu lösen.

Auch dieses Problem lässt sich mit Backtracking lösen. Backtracking testet für ein gegebenes *k* alle möglichen Färbungen bis entweder eine Lösung gefunden wird oder alle Möglichkeiten erschöpft sind. Das minimale *k* lässt sich so finden: Man lässt den Algorithmus für $k=1,2,...$ laufen, bis eine Färbung gefunden wird.

Die Anzahl Ausführungsschritte ist bei einer Heuristik viel geringer. Jedoch kann nicht garantiert werden, dass das Resultat korrekt ist. Die Greedy Heuristik sucht den Knoten mit den meisten nicht zugewiesenen Kanten. Eine Kante ist nicht zugewiesen, falls einer seiner Knoten nicht eingefärbt ist. Nach dem Färben dieses Knotens fährt er bei seinen Nachbarn gleichermassen weiter. Der Algorithmus terminiert, wenn er eine gültige Färbung gefunden hat oder falls ein Knoten nicht eingefärbt werden kann.

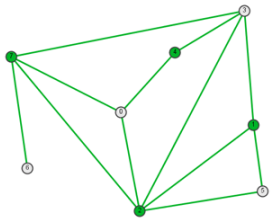
Um auf das Landkartenbeispiel zurückzukommen: Du kannst Dir die Landkarte als Graphen vorstellen. Länder werden zu Knoten. Die Knoten benachbarter Länder werden durch eine Kante verbunden.
Die gute Nachricht: Eine Landkarte ergibt einen planaren Graphen, das heisst die Kanten schneiden sich nur in den Knoten. Falls jedes Land aus nur einer zusammenhängenden Fläche besteht, benötigt die Färbung höchstens aus 4 Farben. Die schlechte Nachricht: Trotz dieser Einschränkung ist das Problem immer noch NP-vollständig .

Im Alltag existieren weitere prominente Beispiele der Graphenfärbbarkeit, denen wir täglich begegnen: Bei einer Ampelschaltung dürfen gewisse Ampeln nicht gleichzeitig auf grün geschaltet werden. Bei der Stundenplanerstellung sind die Stunden, die nicht gleichzeitig stattfinden dürfen (gleiche Lehrer, Klasse, Zimmer) miteinander in Konflikt. Beide Probleme lassen sich auf das Graphenfärbbarkeitsproblem abbilden.



Vertex Cover (Knotenüberdeckung)

Ein Vertex Cover ist eine Menge an Knoten, die alle Kanten abdecken. Eine Kante ist abgedeckt, wenn mindestens einer seiner Knoten im Vertex Cover liegt. Die Grösse eines Vertex Cover ist die Anzahl seiner Knoten.



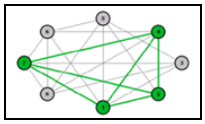
Vertex Cover der Grösse 4

Auch dieses Graphenproblem lässt sich als ein Entscheidungs- und Optimierungsproblem formulieren. In der ersten Variante fragt man sich, ob ein gegebener Graph ein Vertex Cover der Grösse *k* besitzt. Im Optimierungsfall sucht man das minimale Vertex Cover, also die minimale Anzahl Knoten, so dass alle Kanten abgedeckt sind. Wie alle Probleme in diesem Kapitel ist auch dieses NP-vollständig.

Wir betrachten einen Algorithmus für das Entscheidungsproblem. Gegeben ist ein Graph und man soll prüfen, ob alle Kanten mit 5 Knoten abgedeckt werden können. Backtracking findet in jedem Fall die richtige Lösung. Der Algorithmus erstellt eine Liste mit allen Knotenmengen der Grösse 5. Dann prüft er diese Knotenmengen darauf, ob sie ein gültiges Vertex Cover sind. Er terminiert, falls er eine Lösung gefunden hat oder alle Möglichkeiten getestet hat.

Für die Optimierungsvariante existiert eine Approximation, die ein Vertex Cover findet, das höchstens doppelt so gross wie das minimale ist. Der Algorithmus wählt zufällig eine Kante und fügt deren Knoten zum Vertex Cover hinzu. Dann werden die beiden Knoten und die angrenzenden Kanten gelöscht. Der Algorithmus beginnt von vorne bis die Knotenmenge ein gültiges Vertex Cover darstellt.

Ein vielleicht etwas gesuchtes Anwendungsbeispiel von Vertex Cover ist die Positionierung von Überwachungskameras in einem Gebäude. Man will überall dort Kameras installieren, wo mehrere Gänge aufeinandertreffen. Aus Kostengründen ist man natürlich daran interessiert, mit möglichst wenigen Kameras auszukommen.



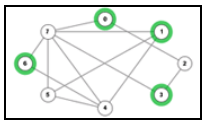
Clique

Unter einer Clique versteht ein Nichtinformatiker eine Gruppe Menschen, die miteinander befreundet sind. Also kennt in einer Clique jeder jeden.

Diese Erkenntnis projizieren wir nun auf einen Graphen. Die Knoten entsprechen den Menschen. Eine Kante zwischen zwei Menschen bedeutet, dass sie sich kennen. Eine Clique in einem Graphen ist eine Knotenmenge, in der jeder Knoten zu jedem anderen Knoten benachbart ist. Es ist also ein vollständiger Subgraph des ganzen Graphen. Wir können auch sagen: in einer Clique kennt jeder Knoten jeden. Das Problem, die maximale Clique in einem Graphen zu finden, ist NP vollständig. Es existiert also wahrscheinlich kein effizienter Algorithmus, der die maximale Clique berechnet.

Der Backtracking Algorithmus arbeitet ähnlich wie beim Vertex Cover Problem. Soll eine Clique der Grösse 3 gefunden werden, prüft er alle Knotenmengen der Grösse 3. Dies solange bis er eine Lösung findet oder alle Möglichkeiten erschöpft sind.

Man könnte den Algorithmus folgendermassen verbessern: sucht man nach einer Clique der Grösse *k*, können alle Knoten gelöscht werden, die weniger als *k*-1 Nachbarn besitzen. Denn solche Knoten können niemals zu einer Clique der Grösse *k* gehören. Ebenfalls können dann deren Kanten gelöscht werden und eventuell weitere Knoten ignoriert werden.



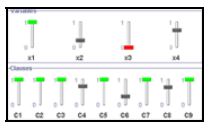
Independent Set (Unabhängige Menge)

In einer unabhängigen Menge sind keine zwei Knoten benachbart. In solch einer Menge gilt also für je zwei Knoten, dass sie nicht durch eine Kante verbunden sind.

Dieses Problem ähnelt dem vorangehenden Cliquenproblem. Eine Clique in einem Graphen entspricht einer unabhängigen Menge in dem Komplementgraphen. Den Komplementgraphen erhält man, wenn man alle Kanten des ursprünglichen Graphen löscht und die zuvor unverbundenen Knoten miteinander verbindet. Dies führt uns gleich zu einer Lösung. Wir können ein maximales Independent Set finden, indem wir die Clique im Komplementgraphen berechnen. Daraus lässt sich auch schliessen, dass das Independent Set ebenfalls NP vollständig sein muss.

Natürlich können wir die Lösung auch direkt finden. Soll eine unabhängige Menge der Grösse *k* berechnet werden, sucht Backtracking alle Untermengen der Grösse *k* nach einer unabhängigen Menge ab.

Es gibt Prominente, die mögen sich nicht besonders. Dies muss zum Beispiel bei den Oscarverleihungen beachtet werden. Wie platziere ich die Schauspieler, Regisseure und Produzenten im Saal so, dass keine zwei zerstrittene Promis neben- oder hintereinander sitzen müssen? Der Organisator kann die Stühle als Knoten darstellen und jene Stühle durch eine Kante verbinden, die neben- oder hintereinander liegen. Mit einer unabhängigen Menge findet er Stühle, die zu keinem anderen Stuhl aus der unabhängigen Menge benachbart sind.



Satisfiability (Erfüllbarkeit logischer Formeln)

Gegeben ist eine logische Formel und gesucht ist eine Belegung der Variablen, so dass die Formel erfüllt ist. Häufig wird 3-SAT, eine Vereinfachung dieses Problems betrachtet.

In 3-SAT sind die Formeln in konjunktiver Normalform mit höchstens 3 Literalen pro Klausel gegeben. Ein Beispiel einer solchen Formel ist $(x1 \vee x2 \vee x3) \wedge (\neg x1 \vee x2 \vee \neg x3) \wedge (\neg x2 \vee x3)$. Eine Belegung, die diese Formel erfüllt ist $x1=0, x2=1, x3=1$.

Cook hat bewiesen, dass das Erfüllbarkeitsproblem NP-vollständig ist. Diese Eigenschaft erleichtert die NP-Vollständigkeitsbeweise anderer Probleme. Durch die Reduktion auf Satisfiabilty in polynomialer Zeit kann man die NP-

Vollständigkeit eines Problems zeigen. Satisfiability ist also sozusagen die Mutter aller NP-vollständigen Probleme. Falls für Satisfiability ein effizienter Algorithmus gefunden würde, dann würden für alle NP-vollständigen Probleme effiziente Algorithmen existieren.

Eine Belegung, welche die Formel erfüllt, kann man mittels Backtracking finden. Man setzt alle Variablen auf false und probiert alle Möglichkeiten durch, bis eine Belegung gefunden wird, welche die Formel erfüllt; vorausgesetzt eine solche Belegung existiert.

Ein physikalisches Modell versucht die Formel mit physikalischen Kräften zu lösen. Dabei versucht jede Klausel ihre Literale in die Richtung zu ziehen, in der die Klausel erfüllt ist. Je näher die Literale am günstigen Wert liegen, desto stärker ist die ziehende Kraft. Beim 3-SAT Beispiel von oben versucht die zweite Klausel x2 nach 1 zu ziehen. X1 und x3 zieht er in Richtung 0.