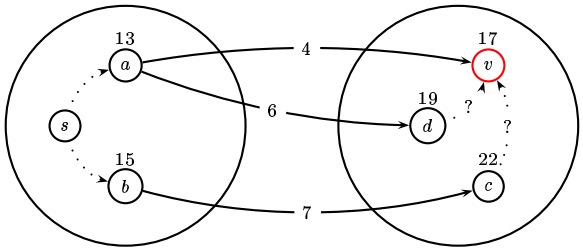
Algorithmen auf Graphen
kürzeste Wege

Markus Durzinsky*
Student der
Otto-von-Guericke-Universität Magdeburg

10. Mai 2004



Inhaltsverzeichnis

*eMail: Mardur@gmx.net Internet: www-e.uni-magdeburg.de/durzinsk

1 Einführung

Dies ist das ausgeschriebene Script zum Vortrag *kürzeste Wege*, gehalten am 11. Mai 2004. Der Vortrag entstand im Rahmen des Proseminars *Algorithmen auf Graphen*[1] im Sommersemester 2004 an der Otto-von-Guericke-Universität Magdeburg. Das Proseminar wurde betreut von Dr. Christian Borgelt[2].

2 Was sind Graphen?

Ein Graph $G = (V, E)$ besteht aus einer (endlichen) Menge von Knoten V und einer Menge von Kanten $E \subseteq V \times V$, die jeweils zwei Knoten verbinden. Die Knoten können dabei irgend welche Objekte sein, entsprechend haben die Kanten auch eine tiefere Bedeutung. Für das mathematische Modell spielt dies aber keine Rolle, weshalb man meisst annimmt, dass $V = \{1, 2, \ldots, n\}$, wobei $n = |V|$ die Anzahl der Knoten ist.

Wenn eine Kante nur in einer Richtung existiert, dann nennt man sie auch Bogen. In einem ungerichteten Graphen gibt es hingegen zu jeder Kante (u, v) die entsprechende Kante (v, u). Wenn der Graph keine mehrfachen Kanten enthält (was durch diese Definition als Menge schon impliziert wurde) und auch keine Schlingen (Kanten der Form (u, u)), bezeichnet man ihn als schlichten Graphen.

Ein Weg W in einem Graphen ist eine Folge von Kanten, wobei der Endknoten einer Kante auch der Startknoten der nächsten sein muss.

$$W = ((v_1, v_2), (v_2, v_3), \ldots, (v_{k-2}, v_{k-1}), (v_{k-1}, v_k)) \quad \text{mit } v_i \in V$$

Ausserdem darf keine Kante (und auch kein Knoten[3]) mehrfach in einem Weg enthalten sein. Damit ist das Problem, Wege mit bestimmten Eigenschaften zu finden, allgemein kombinatorischer Natur, mit exponentiell vielen Möglichkeiten.

Bei einer besonderen Art von Wegen stimmen Anfangs- und Endknoten überein. Solche Wege bezeichnet man als Kreise. Ein Kreis, der alle Knoten eines Graphen genau einmal enthält heisst Hamilton-Kreis.

Häufig wird auf den Kanten (eventuell auch auf den Knoten) eine Gewichtsfunktion $c_e \in \mathbb{R}\ \forall e \in E$ defieniert sein. Mit der Länge eines Weges bezeichnet man üblicherweise nicht die Anzahl seiner Kanten, sondern sein Gewicht. Das Gewicht berechnet sich aus der Summe der einzelnen Gewichte seiner Kanten.

$$c(W) = \sum_{e \in W} c_e$$

Bei der Suche nach kürzesten Wegen geht es also nicht um Wege aus wenigen Kanten, sondern mit möglichst geringem Gewicht.

[1] Internet: http://fuzzy.cs.uni-magdeburg.de/studium/lect.html

[2] eMail: borgelt@iws.cs.uni-magdeburg.de

[3] Ob ein Weg einen Knoten mehrmals durchlaufen darf ist reine Definitionsfrage.

3 Problem der kürzesten Wege

In diesem Script geht es nun darum, im Graphen $G = (V, E)$ den kürzesten Weg[4] von einem Startknoten zu einem Endknoten zu finden. Dies bezeichnet man als das Kürzeste-Wege-Problem.

Problematisch daran ist, dass dieses Problem bei beliebigen Kantengewichten $\mathcal{NP}$-schwer ist. Ein längster Weg kann als ein kürzester Weg mit negativen Gewichten aufgefasst werden. Damit lässt sich das Entscheidungsproblem

P: gibt es einen Hamilton-Kreis in G?

welches $\mathcal{NP}$-vollständig ist, mit Hilfe des Kürzeste-Wege-Problems lösen. Dazu betrachten wir einen speziellen Knoten v und seine Kanten im Graphen (Abbildung 1).

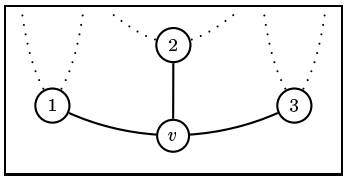


Abbildung 1: zu v inzidente Kanten in G

Ein Hamilton-Kreis in $G = (V, E)$ enthält jeden Knoten aus V, also insbesondere den Knoten v. Nun wird ein neuer Graph $G' = (V', E')$ konstruiert, in dem der Knoten v, wie in Abbildung 2 gezeigt, in zwei Knoten v_1 und v_2 aufgespalten wird. Ausserdem erhält jede Kante das Gewicht -1.

$$
\begin{aligned}
V' &= \{v_1, v_2\} \cup V \setminus \{v\} \\
E' &= \{(v_1, u) \mid (v, u) \in E\} \cup \{(v_2, u) \mid (v, u) \in E\} \cup E \setminus \{(v, u) \mid u \in V\} \\
c_e &= -1 \quad \forall e \in E'
\end{aligned}
$$

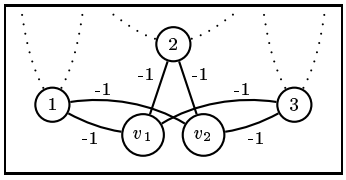


Abbildung 2: zu v_1 und v_2 inzidente Kanten in G'

Mit Hilfe eines Kürzeste-Wege-Algorithmus wird nun der kürzeste Weg von v_1 nach v_2 gesucht. Das entspricht einem Kreis maximaler Länge in G, der v enthält. Dieser Weg in G' hat genau dann das Gewicht $-n$, falls der Graph G einen Hamilton-Kreis enthält.

Damit ist das Kürzeste-Wege-Problem mindestens so schwer wie das Hamilton-Kreis-Problem.

[4]Im Allgemeinen ist so ein kürzester Weg nicht eindeutig. Es geht also um die Suche nach *einem* kürzesten Weg. Zu Gunsten einer einfacheren Formulierung werde ich im Weiteren immer von *dem* kürzesten Weg sprechen.

4 allgemeiner Kürzeste-Wege-Algorithmus

Kürzeste Wege haben eine Eigenschaft, die man zu deren Konstruktion verwenden kann. Ein kürzester Weg über mehrere Knoten besteht immer aus kürzesten Wegen zwischen diesen inneren Knoten.

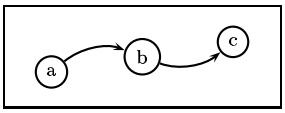


Abbildung 3: zusammengesetzte kürzeste Wege

Wenn der Weg von a nach b nicht optimal ist, dann kann auch der Weg von a nach c über b nicht optimal sein. Der Grundgedanke eines jeden Kürzeste-Wege-Algorithmus ist es nun, kürzere Wege immer wieder aus anderen (schon bekannten) kürzesten Wegen zusammenzusetzen.

```
solange nicht fertig
    wähle Knoten a,b,c
    wenn bisheriger Weg(a,c) länger als Weg(a,b) + Weg(b,c) dann
        setze Weg(a,c) auf Weg(a,b) + Weg(b,c)
```

Die Algorithmen unterscheiden sich nur in der Reihenfolge der Auswahl der einzelnen Knoten. Im Normalfall werden am Anfang nur Wege über eine Kante bekannt sein. Dann werden neue Wege durch Hinzufügen einer einzelnen Kante, etwa (b, c), konstruiert.

5 Dijkstra-Algorithmus

Da das allgemeine Problem $\mathcal{NP}$-schwer ist, gibt es keinen effizienten Algorithmus, der allgemeine kürzeste Wege berechnet (bzw. man kennt keinen). Dies ändert sich, wenn man weitere Bedingungen an den Graphen stellt. Etwa, dass alle Gewichte positiv sein sollen.

Wenn es keine negativen Kantengewichte gibt, dann kann sich ein Weg durch einen grösseren Umweg nicht weiter verkürzen. Ein Beispielszenario ist in Abbildung 4 zu sehen.

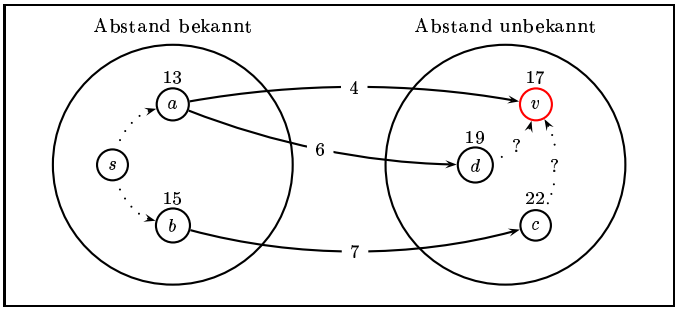


Abbildung 4: Minimaler Abstand von Knoten v?

Angenommen der minimale Abstand der Knoten a und b ist bereits (woher auch immer) bekannt. Nun gibt es einen Weg über a nach v mit der Länge 17.

Wenn es keine negativen Kantengewichte gibt, dann haben alle Wege nach v, die über c oder d verlaufen, mindestens das Gewicht 19 oder 22 plus dem Gewicht des Weges von dort nach v.

Knoten v kann in die Menge der bekannten Knoten aufgenommen werden. Der kürzeste Weg nach v besteht aus dem kürzesten Weg nach a und der Kante (a, v). Die Abstände der Knoten c und d könnten sich durchaus noch verkürzen, falls es einen besseren Weg dorthin über den Knoten v gibt.

Genau dieses Prinzip verwendet der Dijkstra-Algorithmus. Die Knoten sind in zwei Mengen unterteilt, wobei ein Knoten solange in der Menge V verbleibt, wie sich sein minimaler Abstand vom Startknoten noch ändern könnte.

```
Algorithmus Dijkstra
Input: G=(V,E), c(e) >= 0, Startknoten s ∈ V
Output: Aboreszenz (pre,dist)

pre[v]  := s   ∀ v ∈ V
dist[v] := ∞   ∀ v ∈ V
dist[s] := 0

while V ≠ ∅ do
    wähle v ∈ V mit c(v) minimal
    if c(v) = ∞ then
        return
    V := V \ {v}
    forall (v,u) ∈ E do
        if dist[u] > dist[v] + c(v,u) then
            dist[u] := dist[v] + c(v,u)
            pre[u] := v
```

Den Ablauf des Algorithmus kann man sich auch als Ausbreitung einer Suchfront vorstellen. Am Anfang befindet sich nur der Startknoten in der Front. Bei jeder Iteration wird gierig der kürzeste Knoten ausserhalb der Front gewählt und in diese mit aufgenommen.

Wenn der gierig gewählte Knoten den minimalen Abstand ∞ besitzt, dann sind alle Knoten die noch in V sind von s aus nicht erreichbar. Dann kann der Algorithmus stoppen.

Als Ergebnis des Algorithmus erhält man zu jedem Knoten den Abstand und den Vorgänger auf dem kürzesten Weg. Einen solchen gerichteten Graphen, bei dem jeder Knoten (bis auf den Startknoten) genau einen Vorgänger besitzt, nennt man auch Aboreszenz. Diese stellt einen gerichteten Baum dar, in dem es von der Wurzel zu jedem anderen Knoten genau einen Weg gibt.

Falls der Graph nicht den Vorraussetzungen entspricht, d. h. er enthält negative Kantengewichte, dann kann es vorkommen, dass die Vorgängerstruktur zwar eine Aboreszenz bildet, aber nicht die kürzesten Wege enthält.

Beispiel

Als Beispiel zum Verdeutlichen des Ablaufs dient der Graph aus Abbildung 5, wobei der Knoten 1 als Startknoten dient.

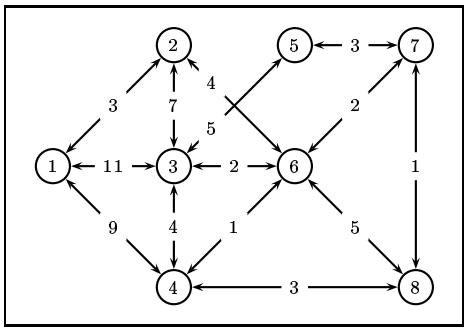


Abbildung 5: Beispielgraph

In jedem Schritt wird, von den bisher noch nicht gewählten, derjenige Knoten ausgewählt, dessen Abstand minimal ist. Alle von diesem Knoten fortlaufenden Kanten müssen dann überprüft werden, ob sich dadurch ein kürzerer Weg ergibt.

	1	2	3	4	5	6	7	8
1	$\boxed{0}$	∞	∞	∞	∞	∞	∞	∞
2		$\boxed{3^1}$	11^1	9^1	∞	∞	∞	∞
3			10^2	9^1	∞	$\boxed{7^2}$	∞	∞
4			9^6	$\boxed{8^6}$	∞		9^6	12^6
5			$\boxed{9^6}$		∞		9^6	11^4
6					14^3		$\boxed{9^6}$	11^4
7					12^7			$\boxed{10^7}$
8					$\boxed{12^7}$			
	0	3^1	9^6	8^6	12^7	7^2	9^6	10^7

Als Ergebnis erhält man den Abstand und den Vorgänger zu jedem Knoten auf einem kürzesten Weg. Nun lässt sich der entsprechende Aboreszenz-Baum wie in Abbildung 6 darstellen.

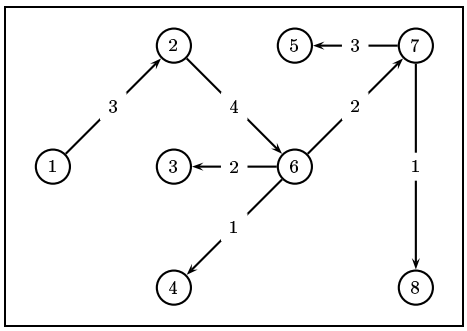


Abbildung 6: Aboreszenz-Baum

Aufwandsanalyse

Sei $n = |V|$ die Anzahl der Knoten und $m = |E|/n$ die durchschnittliche Anzahl Kanten pro Knoten. Die äussere Schleife wird genau n mal durchlaufen, da bei jeder Iteration ein Knoten v aus V entfernt wird. Der Test, ob sich über den Knoten v ein Weg verkürzt, wird dann m mal ausgeführt.

Als besonders zeitkritisch stellt sich die Suche nach dem Knoten v mit minimalem Abstand in der Restmenge heraus. Bei einer einfachen linearen Suche ergibt sich pro Schleifendurchlauf ein linearer Aufwand. Der Gesamtaufwand liegt damit in $O(nn + nm) = O(n^2)$. Unabhängig von der Anzahl der Kanten ergibt sich durch die lineare Suche ein quadratischer Aufwand.

Man kann für diese Suche auch eine Prioritäts-Warteschlange verwenden, die mit Hilfe eines Heap implementiert wird. Der Heap verwaltet die Knoten in einem Baum, wobei der Knoten an der Wurzel (in jedem Teilbaum) jeweils den kleinsten Abstand hat. Die Suche des Knotens v geht damit in $O(1)$. Leider muss der Knoten auch aus dem Heap gelöscht werden, was einen Aufwand von $O(\log n)$ bedeutet. Auch muss bei jedem Aufruf von `pre[u] := v` eventuell die Position des Knotens im Heap angepasset werden, was ebenfalls $O(\log n)$ kostet. Damit ergibt sich ein Gesamtaufwand in $O(n \log n + nm \log n) = O(nm \log n)$.

Bei einem vollständigen Graphen mit vielen Kanten ($m \approx n$) ist diese Variante mit $O(n^2 \log n)$ langsamer[5]. Wenn hingegen die Anzahl der Kanten pro Knoten beschrängt ist, fällt der Gesamtaufwand auf $O(n \log n)$, was eine deutliche Verbesserung gegenüber der linearen Suche bedeutet.

[5] Der worst-case der Heap-Variante ist bei einem vollständigen Graphen schlechter. Allerdings tritt dieser Fall nur ein, wenn sich bei jedem Knoten der Weg für alle anderen Knoten verkürzt, was tatsächlich selten der Fall ist. Praktisch ist die Heap-Variante fast immer besser als die lineare Suche.

6 Floyd-Warshall-Algorithmus

Eine ganz andere Art kürzeste Wege zu berechnen verfolgt der Floyd-Warschall-Algorithmus, auch Tripel-Algorithmus genannt. Dieser ermittelt kürzeste Wege von allen Startknoten zu allen Endknoten. Die Idee liegt darin, eine Indexschranke für die inneren Knoten eines Weges anzugeben.

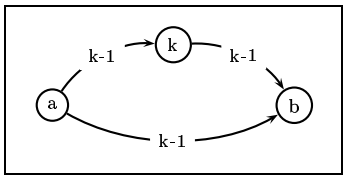


Abbildung 7: Konstruktionsprinzip

Im Iterationsschritt k werden nur Wege berücksichtigt, die über die Knoten 1 bis k verlaufen. Dieses Prinzip wird in Abbildung 7 verdeutlicht. Dabei gibt es zwei Möglichkeiten:

1. Der Weg enthält den Knoten k nicht. Damit ist es ein kürzester Weg, der nur die Knoten 1 bis $k-1$ enthält.

2. Der Weg enthält den Knoten k und setzt sich zusammen aus dem kürzesten Weg von a nach k und dem kürzesten Weg von k nach b. Diese beiden Wege enthalten wiederum nur die Knoten 1 bis $k-1$.

Das Problem wird also induktiv auf das Vorgängerproblem zurückgeführt. Im Induktionsanfang gibt es nur Wege ohne innere Knoten, also nur die Kanten selbst. Zum Schluss sind kürzeste Wege zugelassen, die alle beliebigen Knoten als innere Knoten enthalten.

Der Algorithmus hat keine Probleme mit negativen Kanten, allerdings darf der Graph keine Kreise mit negativem Gesamtgewicht enthalten.

```
Algorithmus Floyd-Warshall
Input: G=(V,E), c(u,v), keine negativen Kreise
       E=1,...,n
Output: Längenmatrix d, Knotenmatrix e

e[u,v] := 0      ∀ u,v ∈ V
d[u,v] := c(u,v) ∀ u,v ∈ V
d[v,v] := ∞      ∀ v ∈ V

for k := 1 to n do
  for u := 1 to n do
    for v := 1 to n do
      if u <> k and v <> k then
        if d[u,v] > d[u,k] + d[k,v] then
            d[u,v] := d[u,k] + d[k,v]
            e[u,v] := k
```

Am Ende des Algorithmus enthält `d` die Länge des kürzesten Weges und `e` den Index des höchsten Knotens auf diesem Weg. Der Weg muss dann rekursiv rekonstruiert werden.

Wenn der Graph nun doch einen Kreis negativer Länge enthält, sieht man dass an negativen Einträgen auf der Hauptdiagonalen von `d`. Die rekursive Konstruktion der Wege mit Hilfe von `e` ist dann nicht möglich, da sich Zyklen ergeben.

Beispiel

Auch der Floyd-Warschall-Algorithmus soll wieder an einem Beispielgraphen in Abbildung 8 veranschaulicht werden.

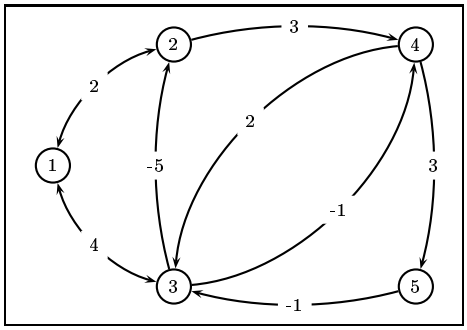


Abbildung 8: Beispielgraph

Als erstes muss die Matrix `d` mit den Kantengewichten initialisiert werden, wobei das Gewicht von nicht vorhandenen Kanten auf ∞ gesetzt wird.

`d`					`e`				
∞	2	$\boxed{4}$	∞	∞	0	0	0	0	0
2	∞	∞	3	∞	0	0	0	0	0
$\boxed{4}$	-5	$\boxed{\infty}$	-1	∞	0	0	0	0	0
∞	∞	2	∞	3	0	0	0	0	0
∞	∞	-1	∞	∞	0	0	0	0	0

Nun muss von jedem Eintrag auf die erste Zeile bzw. erste Spalte gelotet werden. Für den markierten Eintrag ergibt sich etwa ein besserer Weg mit Länge 8. Bei jeder Änderung muss dann auch die 1 als Knoten in der rechten Matrix `e` eingetragen werden.

∞	2	4	∞	∞	0	0	0	0	0
2	2	6	3	∞	0	1	1	0	0
4	-5	$\boxed{8}$	-1	∞	0	0	$\boxed{1}$	0	0
∞	∞	2	∞	3	0	0	0	0	0
∞	∞	-1	∞	∞	0	0	0	0	0

Dieses Verfahren wird dann für die anderen Knoten jeweils in der gleichen Zeile und Spalte wiederholt.

2	2	4	5	∞	2	0	0	2	0
2	2	6	3	∞	0	1	1	0	0
-3	-5	1	-2	∞	2	0	2	2	0
∞	∞	2	∞	3	0	0	0	0	0
∞	∞	-1	∞	∞	0	0	0	0	0
1	-1	4	2	∞	3	3	0	3	0
2	1	6	3	∞	0	3	1	0	0
-3	-5	1	-2	∞	2	0	2	2	0
-1	-3	2	0	3	3	3	0	3	0
-4	-6	-1	-3	∞	3	3	0	3	0
1	-1	4	2	5	3	3	0	3	4
2	0	5	3	6	0	4	4	0	4
-3	-5	0	-2	1	2	0	4	2	4
-1	-3	2	0	3	3	3	0	3	0
-4	-6	-1	-3	0	3	3	0	3	4

Im letzten Schritt ändert sich dann nichts mehr, da in diesem Beispiel kein kürzester Weg über den Knoten 5 verläuft.

Nun soll etwa der kürzeste Weg von 1 nach 5 konstruiert werden. Aus der Matrix `d` kann entnommen werden, dass dieser das Gewicht 5 besitzt. Und der Eintrag in `e` verrät, dass 4 der höchste Index eines inneren Knotens ist. Also besteht der Weg `1 - 5` aus `1 - 4 - 5`. Diese beiden Teilwege müssen wieder aufgespalten werden, so verläuft der Weg von 1 nach 4 über den Knoten 3. Es ergibt sich der Weg `1 - 3 - 4 - 5`. Auch der Weg von 3 nach 4 muss weiter aufgespalten werden in `1 - 3 - 2 - 4 - 5`. Diese ganzen Teilwege haben nun in der Matrix `e` den Eintrag 0, sind also einzelne Kanten.

7 Zusammenfassung

- Das allgemeine Problem der kürzesten Wege gehört zur Klasse der $\mathcal{NP}$-schweren Probleme und ist damit sehr schlecht lösbar.
- Für Spezialfälle gibt es effiziente Algorithmen.
- Der Dijkstra-Algorithmus berechnet kürzeste Wege in Graphen mit nichtnegativen Kantengewichten. Dazu wird eine Suchfront gierig um den jeweils nächsten Knoten erweitert.
- Für Graphen mit beliebigen Kantengewichten, aber ohne negative Kreise, eignet sich der Floyd-Warshall-Algorithmus. Dieser berechnet kürzeste Wege von jedem Startknoten zu jedem Endknoten, indem er Wege mit eingeschränkten Zwischenknoten konstruiert.

Literatur

[1] Peter Gritzmann, Rene Brandenburg: *Das Geheimnis des kürzesten Weges*, Springer-Verlag Berlin Heidelberg New York, 2002

[2] Kyle Loudon: *Mastering Algorithms with C*, O'Reilly & Associates, Inc. 1999