Identifizierung negativer Zyklen in gerichteten Graphen

Benjamin Schulz

4. August 2008

Inhaltsverzeichnis

1 Einführung

Dieses Script befasst sich mit der Theorie sowie angewandten Algorithmen, welche die Frage, wie man in gerichteten Graphen negative Zyklen erkennen kann, oder einen Beweis für deren Nichtexistenz erbringen kann, behandeln.

Dabei werden Erkenntnisse aus [1] verwendet.

1.1 Gerichtete gewichtete Graphen

Ein Graph G ist eine Menge von Knoten V und Kanten E.
$G = (V, E)$

Jeder Kante (u,v) die sich von Knoten u nach Knoten v läuft, hat eine Richtung und ein Gewicht l(u,v).
$V \rightarrow (-\infty, \infty)$

1.2 Finden von Negativen Zyklen

Ziel ist es, entweder einen negativen Zyklus zu finden, oder den Beweis aufzustellen, dass dieser Graph keinen solchen Zyklus enthält. Als Beweis gilt eine zulässige Potentialfunktion.

Eine Potentialfunktion weist jedem Knoten V einen Wert d zu.
$d : V \rightarrow (\mathbb{R}, \infty)$

Mit Hilfe der Potentiale lassen sich für jede Kante reduzierte Kosten $l_d(u,v)$ festlegen, die sich wie folgt definieren: $l_d(u,v) = l(u,v) + d(u) - d(v)$

Falls die reduzierten Kosten für alle Kanten nicht-negativ sind, wird die Potentialfunktion zulässig genannt.

1.2.1 ein negativer Zyklus => keine zulässige Potentialfunktion

Falls ein Graph einen negativen Zyklus enthält, existiert keine Potentialfunktion, bei der die reduzierten Kosten aller Kanten positiv oder 0 sind.
Es existiere ein Zyklus, bestehend aus den drei Knoten u, v und w (o.B.d.A). Dann gilt:
$l_d(u,v) + l_d(v,w) + l_d(w,u)$
$= l(u,v) + d(u) - d(v) + l(v,w) + d(v) - d(w) + l(w,u) + d(w) - d(u)$
$= l(u,v) + l(v,w) + l(w,u)$
Offensichtlich ist die Summe der Kantengewichte gleich der Summe der reduzierten Kosten der entsprechenden Kanten in einem Zyklus.
Bei einem negativen Zyklus ist die Summe negativ, womit die reduzierten Kosten von mindestens einer Kante negativ sein müssen.
Die Potentialfunktion ist somit nicht erfüllbar.

Eine zulässige Potentialfunktion ist nur möglich, wenn kein negativer Zyklus im Graph enthalten ist.

1.2.2 kein negativer Zyklus => eine zulässige Potentialfunktion

Falls kein negativer Zyklus im Graph enthalten ist, ist es immer möglich, eine zulässige Potentialfunktion zu finden.

Fall 1: alle Kantengewichte sind positiv (oder 0):
Jeder Knoten habe das Potential 0. Wegen $l_d(u,v) = l(u,v) + d(u) - d(v)$ sind alle reduzierten Kosten gleich den Kantengewichten, und damit positiv (oder 0).
=> Potentialfunktion zulässig.

Fall 2: es existieren negative Kantengewichte:
Wir fügen dem Graphen einen neuen Knoten s mit Potential 0 hinzu. Sowie ausgehende Kanten von diesem Knoten zu allen anderen Knoten des Graphen mit Gewicht 0.

Die Potentiale der Knoten werden interpretiert als die Länge eines Pfades von s zum jeweiligen Knoten.

Wenn Knoten v das Potential x hat, es also einen Pfad der Länge x von s zu diesem Knoten gibt, und außerdem eine Kante von v zum Knoten w existiert, mit der Länge y, gibt es offensichtlich einen Pfad von s zu w (über v), der Länge x+y. Wenn dies nun das Potential des Knotens w ist, besitzt die Kante v,w nicht-negative reduzierte Kosten:
$l_d(v,w) = l(v,w) + d(v) - d(w)$
$l_d(v,w) = y + x - (y + x) = 0$

Die Interpretation der Potentiale als Pfadlängen verletzt die „reduzierte Kosten“-Regel nicht.

Alle Knoten sind von s aus erreichbar, somit existiert von s aus zu jedem Knoten im Ursprungsgraph ein Pfad. Da keine negativen Zyklen vorhanden sind, existiert außerdem zu jedem Knoten ein kürzester Pfad.
Wenn das Potential eines Knotens exakt der Länge des kürzesten Pfades von s zu diesem Knoten entspricht, nennt man das Potential exakt.
Wie leicht einzusehen, ist in einem Graph, bei dem allen Potentiale exakt sind, die Potentialfunktion zulässig. Jede Kante hat nicht-negative reduzierte Kosten, da falls sie doch negativ wären, die Formel $l_d(u,v) = l(u,v) + d(u) - d(v)$ ein Beweis dafür wäre, dass die Potentiale nicht den Längen der kürzesten Pfade entsprechen. Das Potential von v wäre dann größer als das Potential von u plus der Pfadlänge l(u,v). Damit gäbe es einen kürzeren Pfad nach v, nämlich über u, statt dem bisherigen Pfad.

=> exakte Potentialfunktion möglich, wenn kein negativer Zyklus vorhanden ist

1.3 Scanning Algorithmus

Der Scanning Algorithmus hat das Ziel, in einem gerichteten Graph eine zulässige Potentialfunktion zu finden (falls vorhanden).

Der Kernschritt des Algorithmus überprüft dabei einen Knoten von dem Kanten mit negativen reduzierten Kosten ausgehen. Das Potential der über diese Kanten erreichbaren Knoten wird gesenkt, sodass die reduzierten Kosten nicht-negativ werden. Dieser Schritt wird solange ausgeführt, bis eine zulässige Potentialfunktion gefunden ist.
Im Falle eines negativen Zyklus im Graph terminiert der Algorithmus offensichtlich nicht.

Des weiteren wird zu jedem Knoten ein Status gespeichert: scanned, labeled, unreached.
scanned sind alle Knoten markiert, die in der Vergangeheit gescannt wurden, und deren Potential vermutlich exakt ist.
labeled sind alle Knoten markiert, die noch gescannt werden müssen.
unreached sind alle Knoten markiert, deren Potential vermutlich nicht exakt ist, ein Scannen zum momentanen Zeitpunkt also nicht sinnvoll ist.

Mit Hilfe des Status wird ein Überblick darüber behalten, welche Knoten gescannt werden sollen, und welche nicht, weil sie bereits gescannt wurden oder weil es momentan nicht sinnvoll ist sie zu scannen.

1.3.1 O(n)-Pass

Die Durchläufe des Algorithmus werden zu passes zusammengefasst. Ein pass ist dabei wie folgt definiert:
In jedem pass werden alle Knoten gescannt, die zu Beginn des pass labeled markiert waren.
Pro pass wird ein Knoten maximal einmal gescannt.

Da der Scan eines Knotens alle ausgehenden Kanten überprüft, und pro pass maximal alle Knoten gescannt werden, ist somit die Laufzeit eines pass bei m Kanten in O(m).

Falls kein negativer Zyklus im Graph vorhanden ist, sind nach maximal n passes, wobei n die Anzahl der Knoten ist, alle Potentiale exakt.
Dies lässt sich einfach veranschaulichen, indem man sich bewusst macht, dass ein kürzester Pfad von s zu einem beliebigen Knoten im Graph maximal über n-1 andere Knoten laufen kann.
In jedem pass werden alle Kanten gescannt, die von Knoten mit exaktem Potential zu Knoten, bei denen das Potential eventuell nicht exakt ist verlaufen, und die Potentiale entsprechend angepasst. Nach jedem pass existiert dadurch mindestens ein Knoten mehr, dessen Potential exakt ist.

Mit jedem pass nimmt die Länge der kürzesten Pfade zu Knoten, deren Potential nun exakt ist, zu. Der erste pass sorgt dafür, dass alle Knoten, deren kürzester Pfad über eine Kante verläuft, exakt sind. Der zweite pass verlängert diese Pfade nun, sodass die Potentiale aller Knoten, deren kürzester Pfad über zwei Kanten verläuft exakt werden. Und so weiter. Da ein kürzester Pfad in einem Graph aus n Knoten maximal alle n Knoten enthalten kann, ist man nach n passes fertig. Immer vorausgesetzt, dass kein negativer Zyklus im Graph enthalten ist.

Die auf dem Scanning Algorithmus aufbauenden Algorithmen nennt man daher auch O(n)-pass Algorithmen.

2 Negative Zyklen finden

Mit Hilfe des Scanning Algorithmus ist es also möglich zu einem Graph eine zulässige Potentialfunktion zu finden, falls es eine gibt. Im weiteren ist es sogar bereits möglich, negative Zyklen zu erkennen, falls man einen pass Zähler hinzufügt, der automatisch nach mehr als n passes abbricht. Wie gezeigt, findet man die Potentiale eines Graph mit n Knoten in maximal n passes, womit man schließen kann, wenn der Algorithmus mehr passes benötigt, ein negativer Zyklus im Graph enthalten sein muss.

Wünschenswert wäre es jedoch, wenn man im Falle eines negativen Zyklus nicht die maximale Laufzeit abwarten muss, sondern wenn möglich, einen derarigen Zyklus schon während der normalen Laufzeit erkennt.

Dafür speichert man erst einmal zu jedem Knoten, nachdem man das Potential im Scan Vorgang angepasst hat, einen Elternzeiger, der jeweils auf den direkten Vorgänger im kürzesten Pfad weist.
Falls der durch diese Zeiger gebildete Baum einen Zyklus enthält, also ein Knoten sein eigener Vorgänger ist, muss der Ursprungsgraph einen negativen Zyklus enthalten.
Der durch die Elternzeiger gebildete Baum wird im weiteren als Elternbaum bezeichnet.

2.1 Naive Methode

Die naive Methode folgt immer, wenn man eine Kante überprüft, und das Potential des über die Kante erreichbaren Knoten anpasst, den Elternzeigern den Pfad hinauf, ob der Knoten, dessen Potential man gerade geändert hat, sich bereits über dem momentan gescannten Knoten im Elternbaum befinden. Wenn dem so ist, hat man einen negativen Zyklus gefunden, sonst wird weiter gescannt.
Nachteil dieser Methode ist, dass das andauernde Durchlaufen des Elternbaumes in Laufzeit $\Theta(n)$ liegt.

2.2 subtree disassembly

Beim Subtree disassembly wird der Elternbaum erweitert und nicht nur der Eltern zu jedem Knoten gespeichert, sondern auch die Kinder eines Knotens. Dabei werden die Kinder eines Knotens in einer doppelt verketteten Liste gespeichert, mit einem Zeiger des Elternknotens auf das erste Kind der Liste.
Dem entsprechend wird nun nicht bei der Überprüfung einer Kante und der Änderung des Potentials des so erreichten Knotens v, nicht den Elternzeigern den Baum hinauf gefolgt, sondern man bewegt sich abwärts im Baum, und überprüft alle Nachfahren von v, ob der Elternknoten ein Nachkomme ist.
Falls tatsächlich der Elternknoten gleichzeitig ein Nachkomme ist, ist ein negativer Zyklus gefunden, und es kann abgebrochen werden. Falls nicht, werden alle Nachfahren mit „unreached“ markiert und aus dem Elternbaum entfernt, mit Ausnahme von v, dieser bleibt im Baum und wird „labeled“ markiert.
Das Abtrennen und als „unreached“ markieren hat den Sinn, dass da die Pfade von s zu diesen Knoten über v liefen und sich das Potential von v geändert hat, es sicher ist, dass auch die Potentiale der Nachkommen geändert werden müssen. Ein Scannen dieser Knoten wäre momentan nicht sinnvoll, also werden sie als „unreached“ markiert.

Dieses Überprüfen der Nachfahren mit Entfernen aus dem Elternbaum lässt sich ohne Kompexitätserhöhung während dem normalen Scannen ausführen, indem man die Kosten des Abtrennens gleich beim bilden des Baums veranschlagt (siehe auch Bankkontoparadigma).

2.3 subtree disassembly mit Updates

Subtree disassembly mit Updates geht ähnlich vor wie subtree disassembly, nur wird im Schritt, in dem der Teilbaum entfernt wird, und alle Knoten als unreached markiert werden, außerdem noch

das Potential aller Knoten angepasst auf einen „richtigeren“ Wert.
Das Potential des Knotens v sei um x gesenkt worden im Zuge des Scanning Schrittes. Damit kann man sich sicher sein, dass auch die Nachkommen ein mindestens um x niedrigeres Potential als momentan haben. Damit die reduzierten Kosten der Kanten zu diesen Knoten aber nachwievor negativ bleiben, um ein späteres Scannen zu ermöglichen, werden die Potentiale um x-1 angepasst, statt um x.

2.4 Admissible Graph Search

Der Admissible Graph enthält alle Knoten des Ursprungsgraphen, sowie alle Kanten deren reduzierte Kosten negativ oder 0 sind. Dies entspricht allen Kanten des Elternbaums, sowie allen Kanten über die man billiger zu Knoten kommt, als bisher.
Auf diesem werden nun von s aus per Tiefensuche Scans ausgeführt. Zu jedem gescannten Knoten wird seine Tiefe gespeichert. Falls man bei der Tiefensuche nun auf einen Knoten trifft, der eine Tiefe kleiner der momentanen hat, wurde ein Backarc gefunden. Der Graph zusammen mit dem Backarc bildet einen Zyklus, falls dieser eine negative Kante enthält, hat man einen negativen Zyklus gefunden, und es kann abgebrochen werden.

3 Die Algorithmen im Detail

3.1 BFCT (Bellman-Ford-Cherkassky-Tarjan)

BFCT benutzt „Subtree disassembly with updates“. Zu Beginn werden alle Knoten als „labeled“ markiert. „Labeled“ markierte Knoten werden in eine Fifo Warteschlange eingefügt, und so sukzessive abgearbeitet, bis entweder die Warteschlange leer ist oder ein negativer Zyklus gefunden wurde. Die Laufzeit des Algorithmus ist in O(nm).

3.2 MBFCT (Multi-BFCT)

MBFCT ist eine Variante von BFCT. Im Unterschied zu BFCT ist s zu Beginn nur mit einem Knoten verbunden. Alle Knoten bis auf den einen sind als „unreached“ markiert. Dann wird normal BFCT ausgeführt.
Falls so kein negativer Zyklus gefunden wird und es noch mit „unreached“ markierte Knoten gibt, wird einer dieser Knoten auf „labeled“ gesetzt, und eine Kante von s zu diesem Knoten eingefügt. Dann wird BFCT wiederum ausgeführt. Die in vorherigen Durchläufen gefundenen Potentiale bleiben erhalten. Dies wird so lange wiederholt, bis keine Knoten mehr mit unreached markiert sind, oder ein negativer Zyklus gefunden wurde.
Damit wird bis zu n mal BFCT ausgeführt, was diesem Algorithmus eine Worst-Case Laufzeit von $O(n^2m)$ beschehrt. Somit gehört er nicht zur Klasse der O(n)-pass Algorithmen.
In der Praxis kann dieser Algorithmus aber durchaus schneller als BFCT einen negativen Zyklus finden, wie man im Teil „Experimente“ sieht.

3.2.1 Unterschied zwischen MBFCT und BFCT

BFCT geht sofort in die Breite, da im ersten Pass direkt alle Knoten gescannt werden.
MBFCT dagegen scannt nur kleine Bereiche des Graphen. Nur der von jeweils einem Knoten erreichbare Subgraph.

3.3 GOR (Goldberg, Radzik)

GOR basiert auf „Admissible Graph Search“. In Form von Tiefensuche wird durch den Admissible Graph gescannt. Parallel zum Scan wird nach Zyklen gesucht.
Falls ein backarc gefunden wird, wird überprüft, ob der durch den backarc gebildete Zyklus eine negative Kante hat. Falls ja, ist ein negativer Zyklus gefunden, falls nicht, wird der backarc aus dem Admissible Graph entfernt.

3.4 RD (robust Dijkstra)

RD benutzt Ideen von Dijkstras Algorithmus in Bezug auf die Auswahl, welcher Knoten als nächstes gescannt werden soll. Im Gegensatz zu BFCT scannt RD nicht einfach den nächstbesten Knoten wie er gerade in der Warteschlange ist, sondern wählt aus, welcher Knoten am vielversprechendsten ist um fortzufahren. Dafür wird jedem Knoten ein Schlüssel zugeordnet. Der Schlüssel hat den Wert der letzten Potentialänderung des Knotens.

RD verwaltet 2 Warteschlangen. Beide enthalten mit „labeled“ markierte Knoten.
Warteschlange Q enthält alle „labeled“ Knoten, die im momentanen pass noch nicht gescannt wurden.
Warteschlange S enthält die Knoten, die labeled markiert sind, aber bereits in diesem pass gescannt wurden, also erst im nächsten pass wieder gescannt werden können.

Q ist dabei nach den Schlüsseln sortiert.
Es wird jeweils der Knoten mit dem maximalen Schlüssel aus Q genommen und gescannt. Neu „labeled“ markierte Knoten werden entweder in Q eingefügt, wenn sie in diesem pass noch nicht gescannt wurden, sonst in S. Bei bereits labeled markierten Knoten, deren Potential sich ändert, wird der Schlüssel aktualisiert und falls nötig, Q entsprechend sortiert.
Das wird so lange gemacht, bis Q leer ist, dann werden alle Knoten von S nach Q verschoben und ein neuer pass beginnt, solange bis beide Warteschlangen leer sind.

4 Der Random Graph Generator

In diesem Kapitel wird kurz auf den Random Graph Generator eingegangen, der benutzt wurde, um die Graphen für die im nächsten Kapitel gemachten Tests zu erzeugen.

Als Eingabe muss man angeben, wieviel Knoten und wieviele Kanten der Graph enthalten soll. Außerdem wird noch eine obere und untere Schranke für die Kantengewichte benötigt. In diesem Fall war die Schranke (0,1000).

Zu Beginn wird ein Hamiltonischer Kreis der Länge n gebildet. Dann werden m-n weitere Kanten zufällig eingefügt und jeder Kante ein zufälliges Gewicht aus den Schranken zugewiesen.

Nun werden eine Anzahl Knoten zufällig ausgewählt, die einen negativen Zyklus bilden sollen. Zwischen diesen werden nun Kanten mit Gewicht 0 und eine Kante mit Gewicht -1 eingefügt, sodass ein Zyklus entsteht.
Als nächstes wird versucht, den negativen Zyklus zu verstecken. Dafür wird für jeden Knoten eine zufällige Zahl zwischen 0 und 1000 gewählt. Diese wird zu eingehenden Kanten addiert und von ausgehenden Kanten sutrahiert.
Durch das Verstecken entstehen weitere negative Kanten, sodass der negative Zyklus nichtmehr die einzig negative Kante enthält. Außerdem enthält der negative Zyklus nun nicht nur Kanten mit Gewicht 0 oder -1 sondern Gewichte verteilt über den gesamten Bereich zwischen -1000 und 1000.

5 Experimente

Bei den Experimenten wird die Anzahl an Scans pro Knoten gemessen.
Die Algorithmen werden auf 5 verschiedenen Problemklassen angewendet. Graphen ohne negative Zyklen (1), Graphen mit einem kleinen negativen Zyklus (2), Graphen mit vielen kleinen negative Zyklen (3), Graphen mit größeren negativen Zyklen (4) sowie Graphen mit einem hamiltonischen negativen Zyklus (5).

5.1 Ergebnisse

Für dieses Experiment wurden Graphen mit 2.097.152 Knoten und 10.485.760 Kanten verwendet. Als Werte wurden jeweils der Durchschnitt aus 10 verschiedenen Graphen je Klasse verwendet.

Klasse	BFCT	MBFCT	GOR	RD
1	1,0105	1,0106	1,1019	1,0106
2	1,0078	0,5313	1,0938	0,7200
3	0,0319	$< 10^{-4}$	0,9882	$< 10^{-4}$
4	2,1388	0,3724	3,5164	1,9872
5	4,9904	2,5356	10,9011	4,5558

MBFCT hat in vielen Fällen die besste Performanz, trotz der Tatsache, dass er eine ganze Komplexitätsklasse schlechter ist als die anderen Algorithmen. Allerdings sind die Ergebnisse mit Vorsicht zu genießen. Da MBFCT sich auf kleine Bereiche des Graphen konzentriert ist deutlich schneller, falls ein negativer Zyklus in diesem Teil des Graphen ist. Wenn der Algorithmus allerdings Pech hat müssen viele BFCT Durchläufe gemacht werden, bevor der richtige Teilgraph gescannt wird. Die anderen Algorithmen dagegen suchen breit über den ganzen Graph, sind somit weniger anfällig für zufällig schlechte Auswahl der Scanreihenfolge.

Auch RD zeigt gute Performanz. Die gezielte Auswahl der Knoten nach ihrem Schlüssel scheint sinnvoll.
GOR dagegen kann nicht überzeugen und ist bei allen fünf Problemklassen der schlechteste.

5.2 Bad Case Fälle

Zum Vergleich gibt es noch die Laufzeiten auf besonders ungünstigen Graphen für die jeweiligen Algorithmen.

Übersicht über die Anzahl an Kanten und Knoten der verschiedenen Bad-Case-Graphen:

- Bad-BFCT: n = 6399, n = 7997
- Bad-MBFCT: n = 4799, m = 5597
- Bad-GOR: n = 3201, m = 4799
- Bad-RD: n = 4801, m = 7998

BFCT	**MBFCT**	**GOR**	**RD**
401	107356	803	528
O(n*m)	$O(n^2 * m)$	O(n*m)	O(n*m)

Hier zeigt sich nun die deutlich schlechtere Worst-Case Laufzeit von MBFCT. Durch das unter Umständen sehr oft wiederholte Ausführen von BFCT auf den Graph kann es zu geradezu astronomischen Anzahl Scanns pro Knoten kommen.
Aber auch die anderen Algorithmen brauchen sehr viele Scanns, wenn man die Zahlen mit der vorherigen Tabelle vergleicht, und nicht mit dem aus dem Rahmen fallenden MBFCT.

6 Schluss

Es war auf jeden Fall ein interessantes Thema. Wie man sieht, gibt es einige Ansätze, wie man negative Zyklen in Graphen finden kann, die auch noch auf zufälligen Graphen eine sehr hohe Geschwindigkeit zu haben scheinen. Wobei es keine Tests mit dem naiven Ansatz gab, somit ist ein Vergleich, wie schlecht es hätte sein können, nicht gegeben.

Für meinen Geschmack ging das Script aber zu wenig bzw. eigentlich garnicht auf die tatsächlichen Anwendungsgebiete ein, in denen eine Suche nach negativen Zyklen verwendet wird. Was auch die Frage aufwirft, sind die vom Random Graph Generator erzeugten Graphen typisch für die Art Graphen, in denen man negative Zyklen sucht? Oder kommen vielleicht nur spezielle Graphen in dem wie auch immer gearteten Anwendungsfeld der Suche nach negativen Zyklen vor?
Die einzigen Graphen, die ich gefunden habe, mit negativen Kantengewichten (also Graphen in denen zumindest theoretisch negative Zyklen vorhanden sein können) waren Straßenpläne für (momentan nicht existierende) Energiesparautos, die beim Berg ab fahren Strom produzieren. Ich bezweifle eher, dass die Suche nach negativen Zyklen alleinig mit der Automobilbranche und Energiesparautos verknüpft ist.

Literatur

[1] Boris V. Cherkassky, Loukas Geordiadis, Andrew V. Goldberg, Robert E. Tarjan, and Renato F. Werneck. Shortest path feasibility algorithms: An experimental evaluation. http://www.siam.org/proceedings/alenex/2008/alx08_012cherkasskyb.pdf, 2008.

[2] T.H. Corman, C.E Leiserson, and R.L.Rivest. *Introduction to Algorithms*. USA, 1999.

[3] C-H. Wong and Y-C Tam. *13th Annual European Symposium on Algorithms*, chapter Negative Cycle Detection Problem, pages 652–663. 2005.