Computing 11, 125—136 (1973)
© by Springer-Verlag 1973

Zwei Verfahren zur Suche negativer Zyklen in bewerteten Digraphen

Von

W. Domschke, Karlsruhe

Mit 4 Abbildungen

(Eingegangen am 23. November 1972)

Zusammenfassung — Abstract

Zwei Verfahren zur Suche negativer Zyklen in bewerteten Digraphen. Das erste der angegebenen Verfahren basiert auf dem Dijkstra-Algorithmus [3], das zweite auf dem Algorithmus von Ford [9] (bzw. Moore [12]). Beiden wird eine mehrmals auszuführende „Prozedur" hinzugefügt. Dadurch wird für den Dijkstra-Algorithmus erreicht, daß er auf *beliebig reell bewertete Digraphen* anwendbar ist und daß vorhandene negative Zyklen entdeckt werden können; mit Hilfe des erweiterten Ford-Algorithmus können negative Zyklen mit geringerem Rechenaufwand als mit dem ursprünglichen Verfahren erkannt werden.

Two Algorithms to Detect Negative Cycles in a Valued Digraph. The first of the two methods developed is based on the Dijkstra-algorithm [3] the second on the algorithm of Ford [9] (respectively Moore [12]). To both of them a "procedure" is added which must be carried out several times. By this means one attains that the Dijkstra-algorithm can be applied to *any real valued digraph* and that existing negative cycles can be detected, by the aid of the extended Ford-algorithm negative cycles can be detected with less computing time than with the original method.

1. Einleitung und Definitionen

Zum Problem der Bestimmung kürzester Wege in Graphen ist seit 1956 eine Vielzahl von Arbeiten erschienen (vgl. z. B. die Bibliographie von Murchland [13]). Nicht alle der entwickelten Verfahren sind jedoch zur Ermittlung kürzester Wege in Graphen mit beliebigen reellen Pfeilbewertungen geeignet; insbesondere hat man sich noch wenig mit der Frage beschäftigt, mit welcher Methode mit möglichst geringem Rechenaufwand negativ bewertete Zyklen in einem Graphen ermittelt werden können.

Dantzig, Blattner und Rao haben auf der Grundlage des von Dantzig in [1] angegebenen Verfahrens zur Bestimmung kürzester Wege in Graphen mit ausschließlich nichtnegativer Pfeilbewertung einen Algorithmus entwickelt, der auch zur Anwendung auf Graphen mit beliebiger reeller Bewertung und insbesondere zur Suche negativer Zyklen geeignet ist (siehe [2]). Allerdings kann man sich leicht überlegen, daß das Verfahren hinsichtlich des benötigten Rechenaufwands noch verbessert werden kann, wenn man es auf der Grundlage

des Dijkstra-Algorithmus (vgl. [3]) entwickelt. Das dadurch entstehende Verfahren wird in Abschnitt 2.1. angegeben.

In Abschnitt 2.2. wird ein vom Verfasser auf der Grundlage des Ford-Algorithmus (der häufig auch als Moore-Algorithmus bezeichnet wird) entwickeltes Verfahren angegeben. Mit Hilfe dieses erweiterten Ford-Algorithmus ist es möglich, im gegebenen Graphen existierende negative Zyklen mit geringerem Rechenaufwand als mit dem ursprünglichen Verfahren zu finden, bei dem ja auch, falls nach Ausführung von $n-1$ ($n:=$Anzahl der Ecken des Graphen) Verfahrensschritten noch immer Entfernungsveränderungen möglich sind, auf das Vorhandensein negativer Zyklen geschlossen werden kann.

Zunächst sind einige Definitionen erforderlich:

Ausgegangen wird von einem *zusammenhängenden bewerteten Digraphen*[1] $G=(V, E, c)$ mit der *Eckenmenge* $V=\{1, 2, \ldots, n\} \subset \mathbb{N}$, der *Pfeilmenge* $E \subset V \times V$ und der *Kosten- oder Bewertungsfunktion* $c: E \to \mathbb{R}$.[2]

Ecke v heißt *Nachfolger* von Ecke u, u *Vorgänger* von v, falls $(u, v) \in E$. Die Menge aller Nachfolger einer Ecke u wird mit $\mathcal{N}(u)$, die aller Vorgänger mit $\mathcal{V}(u)$ bezeichnet.

Ein *Weg* in einem bewerteten Digraphen $G=(V, E, c)$ ist eine Folge von Ecken

$$(v_1, v_2, \ldots, v_i) \text{ mit} \begin{cases} v_1 \in V & \text{, falls } i=1 \text{ (}\textit{trivialer}\text{ Weg)} \\ (v_j, v_{j+1}) \in E \text{ für } j=1\,(1)\,i-1, & \text{falls } i>1. \end{cases}$$

v_1 heißt *Anfangsecke*, v_i *Endecke* des Weges.

Eine Ecke v_j heißt von einer Ecke v_i aus *erreichbar*, falls ein Weg von v_i nach v_j (d.h. mit v_i als Anfangs- und v_j als Endecke) existiert. Einen Weg, dessen Anfangs- und Endecke identisch sind, nennen wir *Zyklus*. Insbesondere nennen wir w einen *einfachen Zyklus*, falls er keinen Zyklus w' als Teilweg echt enthält.

Wir schreiben $(v_i, v_j) \in w$, falls (v_i, v_j) ein im Weg w enthaltener Pfeil ist.

$$c(w) := \begin{cases} \sum_{j=1}^{i-1} c(v_j, v_{j+1}), & \text{falls } i>1 \\ 0 & \text{, falls } i=1 \end{cases}$$

heißt die (*Kosten-*)*Länge* des Weges (oder, falls $v_i=v_1$, des Zyklus) $w=(v_1, v_2, \ldots, v_i)$. Falls $c(w)<0$, sagt man, der Weg w besitze negative Länge oder, falls $v_i=v_1$, w sei ein *Zyklus negativer Länge* oder ein *negativer Zyklus*.

Ein Weg $\bar{w}$, der unter allen von Ecke u nach Ecke v führenden Wegen die kleinste Kostenlänge besitzt, heißt *kürzester Weg* von u nach v. $c(\bar{w})$ bezeichnet man als *kürzeste Entfernung* von u nach v.

---

[1] Zur Definition dieses Begriffes siehe z. B. Harary, Norman, Cartwright [10] oder Domschke [4].

[2] $\mathbb{N}:=$Menge der natürlichen Zahlen; $\mathbb{R}:=$Menge der reellen Zahlen.

Die einen bewerteten Digraphen $G=(V, E, c)$ vollständig charakterisierende *Kosten- oder Bewertungsmatrix* $C(G)=(c_{uv})$ wird für alle $u, v \in V$ definiert durch

$$c_{uv} := \begin{cases} 0 & \text{für } u=v \\ c(u,v), & \text{falls } (u,v) \in E \\ \infty & \text{sonst.} \end{cases}$$

2. Die Verfahren

2.1. Ein auf der Grundlage des Dijkstra-Algorithmus entwickeltes Verfahren[3]

a) Der Dijkstra-Algorithmus[4]

Mit diesem Verfahren können kürzeste Entfernungen (und Wege[5]) von einer Ecke a (*Startecke*) zu allen Ecken eines Digraphen *mit nichtnegativer Pfeilbewertung* bestimmt werden.

Man bildet im k-ten Schritt ($k=1, 2, \ldots$) des Verfahrens eine Teilmenge A_k und eine zu A_k disjunkte Teilmenge B_k der Eckenmenge. Für die Folge (A_k) gilt: $|A_k|=k$[6] und $A_1 \subset A_2 \subset A_3 \subset \ldots$.

A_k umfaßt ausschließlich solche Ecken, deren kürzeste Entfernung von der Startecke bis zum Abschluß des k-ten Schrittes bereits bekannt ist, B_k enthält alle diejenigen Ecken u, für die bis zum Abschluß des k-ten Schrittes bereits ein — nicht notwendig kürzester — Weg von a nach u gefunden worden ist, die aber noch nicht Element von A_k sind. Bezeichnet man mit $d_k: V \to \mathbb{R} \cup \{\infty\}$ diejenige Abbildung, die am Ende des k-ten Schrittes ($k=1, 2, \ldots$) jeder Ecke v ihre bis dahin gefundene (= aktuelle) kürzeste Entfernung $d_k v$ von der Startecke a zuordnet, dann läßt sich das Verfahren wie folgt beschreiben:

Zu Beginn setzt man $A_1 := \{a\}$, $B_1 := \mathcal{N}(a)$, $d_1 a := 0$; ferner $d_1 v := c(a, v)$ für alle $v \in B_1$ sowie $d_1 u := \infty$ für alle sonstigen Ecken u. Damit ist Schritt 1 beendet.

Im k-ten Schritt ($k=2, 3, \ldots$) bestimmt man eine Ecke $v_k \in B_{k-1}$, für die gilt: $d_{k-1} v_k = \min_{v \in B_{k-1}} d_{k-1} v$.

Man bildet $A_k := A_{k-1} \cup \{v_k\}$, $B_k := B_{k-1} \cup \mathcal{N}(v_k) - A_k$ und prüft, ob für die Elemente $v \in \mathcal{N}(v_k) - A_k$ die bisher ermittelten kürzesten Entfernungen von a sich dadurch verkürzen lassen, daß man Wege von a nach v über v_k betrachtet. Für alle Ecken $v' \in \mathcal{N}(v_k) - A_k$, für die $d_{k-1} v' > s := d_{k-1} v_k + c(v_k, v')$ ist, setzt

---

[3] Vgl. auch das von Dantzig, Blattner und Rao auf der Basis des Dantzig-Algorithmus entwickelte Verfahren in [2] sowie in [4], S. 63—66.

[4] Siehe hierzu insbesondere Dijkstra [3], Domschke [4], S. 22—26, sowie Domschke [5].

[5] In Abschnitt 2 wird der Einfachheit halber nur die Bestimmung der kürzesten Entfernungen beschrieben. Die zugehörigen kürzesten Wege lassen sich daraus leicht ermitteln (vgl. hierzu z. B. Domschke [4], S. 10—33, oder die ALGOL-PROGRAMME in diesem Heft).

[6] $|M| :=$ Anzahl der Elemente der Menge M.

man $d_k v' := s$. Allen sonstigen Ecken u (einschließlich v_k) ordnet man die Werte $d_k u := d_{k-1} u$ zu. Erreicht man ein q, so daß nach Abschluß des q-ten Schrittes $B_q = \emptyset$ $(q = 1, 2, \ldots)$, so ist das Verfahren beendet.

b) Algorithmus zur Suche negativer Zyklen

Mit diesem Algorithmus können kürzeste Entfernungen (und Wege) von einer Ecke a zu allen Ecken eines Digraphen *mit beliebiger reeller Pfeilbewertung* bestimmt werden. Falls der betrachtete Graph einen (von der gegebenen Startecke a aus erreichbaren) negativen Zyklus enthält — und somit nicht zu allen Ecken kürzeste Entfernungen ermittelt werden können —, läßt sich dieser mit Hilfe des Verfahrens ermitteln.

Die Grundschritte des Verfahrens entsprechen denen des Dijkstra-Algorithmus. Im k-ten Grundschritt $(k = 1, 2, \ldots)$ werden also auch hier eine Teilmenge A_k und eine zu A_k disjunkte Teilmenge B_k der Eckenmenge mit $|A_k| = k$ und $A_1 \subset A_2 \subset A_3 \subset \ldots$ gebildet. Da jedoch in einem zugrunde liegenden Graphen Pfeile mit negativer Bewertung auftreten können, ist nach dem k-ten Grundschritt nicht gewährleistet, daß A_k nur solche Ecken enthält, deren kürzeste Entfernung von Ecke a bereits bekannt ist. Entfernungskorrekturen sind auch für diese Ecken weiterhin erforderlich. Sie erfolgen mittels einer Prozedur, die nach jedem Grundschritt eingefügt wird.

Die nach dem k-ten Grundschritt $(k = 2, 3, \ldots)$ auszuführende Prozedur verläuft wie folgt:

Im i-ten Prozedurschritt $(i = 1, 2, \ldots, k)$ bildet man eine Menge T_i mit den Eigenschaften: $|T_i| = i$ und $T_1 \subset T_2 \subset \ldots \subset A_k$. Nach Abschluß des i-ten Schrittes enthält T_i ausschließlich solche Ecken der Menge A_k, für die die Entfernung von a vor Ausführung des $(k+1)$-ten Grundschrittes nicht mehr verkürzt werden kann.

Im 1. Prozedurschritt setzt man $T_1 := \{v_k\}$ [7] sowie $d_k v_k := d_{k-1} v_k$ und für alle Ecken $v \in A_k$ mit $d_{k-1} v > d_k v_k + c(v_k, v)\ (=: m)$ die Entfernungsmarke $d_k v := m$; für alle sonstigen Ecken $u \in A_k$ bildet man $d_k v := d_{k-1} v$.

Ⓓ Im i-ten Prozedurschritt $(i = 2, 3, \ldots, k)$ sucht man unter allen Ecken aus $A_k - T_{i-1}$ eine Ecke u_i, für die gilt:

$$(m :=)\ d_{k-1} u_i - d_k u_i = \max_{u \in A_k - T_{i-1}} (d_{k-1} u - d_k u).$$

Ist $m = 0$, so kann die Prozedur abgebrochen und mit dem $(k+1)$-ten Grundschritt fortgefahren werden.

Ansonsten setzt man $T_i := T_{i-1} \cup \{u_i\}$.

Gilt nunmehr $d_k u_i + c(u_i, v_k) < d_k v_k$, so existiert in dem zugrunde liegenden Graphen ein negativer Zyklus, dem sowohl u_i als auch v_k angehören. Gilt diese Beziehung nicht, so prüft man für *alle* Nachfolger v von u_i (nur für Nachfolger von

[7] Zur Bedeutung von v_k vergleiche die Beschreibung des Dijkstra-Verfahrens!

u_i, die aus T_i sind, ist mit Sicherheit keine Entfernungsverringerung möglich), ob deren Entfernungsmarken $(d_k v)$ sich dadurch verkürzen lassen, daß man Wege von a nach v über u_i betrachtet $(d_k v > d_k u_i + c(u_i, v)?)$.

Ist $i<k$, so fährt man anschließend mit dem $(i+1)$-ten Prozedurschritt bei Ⓓ, ansonsten mit dem $(k+1)$-ten Grundschritt fort.

Der Beweis dafür, daß das beschriebene Verfahren stets zu richtigen Ergebnissen gelangt, kann leicht aus den in [2] sowie [4], Seite 24–26, gegebenen Beweisen hergeleitet werden.

2.2. Ein auf der Grundlage des Ford-Algorithmus entwickeltes Verfahren

a) Der Ford-Algorithmus[8]

Mit diesem Algorithmus können kürzeste Entfernungen (und Wege[9]) von einer Ecke a zu allen Ecken eines Digraphen *mit beliebiger reeller Pfeilbewertung* bestimmt werden. Allerdings können mit diesem Verfahren im Graphen vorhandene negative Zyklen nicht frühzeitig, d. h. vor Beendigung des gesamten Verfahrens[10], erkannt und lokalisiert werden.

Im Laufe der Ausführung des Algorithmus sind bestimmte Ecken des Graphen markiert und damit Element einer Menge A. Markiert bzw. Element von A sind stets solche Ecken u, deren aktuelle kürzeste Entfernung von der Startecke a im Laufe des Verfahrens verringert werden konnte, von denen aus man jedoch nach erfolgter Entfernungsverringerung noch nicht ihre Nachfolger $(v \in \mathcal{N}(u))$ aufgesucht hat, um zu prüfen, ob sich deren Entfernungsmarken ebenfalls vermindern lassen, indem man Wege von a über Ecke u betrachtet.

Bezeichnet man mit $d: V \to \mathbb{R} \cup \{\infty\}$ diejenige Abbildung, die in jedem Stadium des Verfahrens jeder Ecke v ihre bis dahin gefundene kürzeste Entfernung dv von der Startecke zuordnet, dann läßt sich das Verfahren wie folgt beschreiben:

Zu Beginn ist $A=\{a\}$, $da=0$, und man setzt $dv:=\infty$ für alle $v \in V-A$.

In jedem Schritt des Verfahrens wählt man ein beliebiges Element u aus A. Für jeden Nachfolger v der ausgewählten Ecke u prüft man, ob gilt: $dv > du + c(u, v)$. Ist dies der Fall, wird $dv := du + c(u, v)$ gesetzt, die Ecke v wird in die Menge A aufgenommen.

---

[8] Siehe hierzu insbesondere Ford [9], Moore [12], Domschke [5] sowie Domschke [4], S. 14—17.

[9] Siehe hierzu Fußnote 5.

[10] Hierzu ist zu bemerken: Wird der Ford-Algorithmus in der in 2.2. a) gewählten Weise beschrieben und programmiert, so läßt sich nur schwer angeben, nach wievielen Schritten das Verfahren abgebrochen werden soll, weil feststeht, daß für den gegebenen Graphen aufgrund vorhandener negativer Zyklen zu bestimmten Ecken überhaupt kein kürzester Weg gefunden werden kann. Beschreibt und programmiert man das Verfahren dagegen in der z. B. in [5] oder in [4], S. 14—17, gewählten Form, so weiß man, daß, falls nach $n-1$ $(n := |V|)$ Verfahrensschritten (des sogenannten Einzel- oder auch des Gesamtschrittverfahrens) noch markierte Ecken vorhanden sind, im gegebenen Graphen negative Zyklen existieren müssen. Lokalisiert sind sie damit allerdings noch nicht!

Ist für sämtliche Nachfolger der Ecke u diese Prüfung durchgeführt, wird u aus A ausgesondert, und man fährt mit einem neuen Element aus A fort.

Das Verfahren endet, sobald die Menge A leer ist.

b) Algorithmus zur Suche negativer Zyklen

Mit diesem Verfahren können kürzeste Entfernungen (und Wege) von einer Ecke a zu allen Ecken eines Digraphen *mit beliebiger reeller Pfeilbewertung* bestimmt werden. Darüber hinaus ist es möglich, im Graphen vorhandene (von der gegebenen Startecke a aus erreichbare) negative Zyklen mit geringerem Rechenaufwand als für den ursprünglichen Ford-Algorithmus erforderlich zu erkennen *und zu lokalisieren.*

Das Verfahren besteht aus einem *Grundprozeß* und einer — unter Umständen mehrmals auszuführenden — *Prozedur.* Die Vorgangsweise im Grundprozeß ist dieselbe wie beim Ford-Algorithmus. Man führt auch hier eine Menge A mit der gleichen Bedeutung wie oben ein. In jedem Schritt des Grundprozesses wählt man — wie oben — eine beliebige Ecke, die Element von A ist, aus und prüft, ob sich die aktuellen kürzesten Entfernungen für deren Nachfolger reduzieren lassen. Lassen sich jedoch für Nachfolger q_j einer Ecke $p \in A$ die aktuellen kürzesten Entfernungen reduzieren *und erfolgt dies über Pfeile mit negativer Bewertung* (gilt also

$$d q_j > d p + c(p, q_j) \quad \text{und} \tag{1}$$

$$c(p, q_j) < 0 \quad \text{für Ecken } q_j \in \mathcal{N}(p)), \tag{2}$$

so wird vor Ausführung des nächsten Grundschrittes eine Prozedur durchgeführt; d. h. sobald ein Pfeil mit negativer Bewertung auftritt, wird von seiner Anfangsecke aus systematisch weitergesucht, um eventuell vorhandene negative Zyklen aufzufinden. Nach Beendigung der Prozedur wird, falls kein negativer Zyklus gefunden wurde, der Grundprozeß mit einer neuen Ecke aus A fortgesetzt.

Die Prozedur ist ein etwas abgewandelter Ford-Algorithmus. Gesucht werden primär etwa vorhandene negative Zyklen sowie negative kürzeste Entfernungen (und Wege negativer Länge) von derjenigen Ecke $p \in A$, bei der der Grundprozeß unterbrochen wurde, zu solchen Ecken v des Graphen, deren aktuelle kürzeste Entfernungen von der Startecke a sich durch Betrachtung von über Ecke p führenden Wegen noch verringern lassen.

Auch im Laufe der Prozedur werden somit bestimmte Ecken markiert und entweder in eine nur während der Prozedur geführte Menge T oder in die im Grundprozeß betrachtete Menge A aufgenommen. Element von T ist jede Ecke v, für die gilt:

1. Ihre aktuelle kürzeste Entfernung von der Startecke a ist im Laufe der Prozedur reduziert worden, Nachfolger sind jedoch noch nicht aufgesucht worden, um eventuell deren Entfernungsmarken zu ändern.
2. Ihre aktuelle kürzeste Entfernung von Ecke p (sie wird im folgenden stets mit $d^p v$ bezeichnet) ist negativ.

Kann im Laufe der Prozedur für eine Ecke u zwar deren aktuelle kürzeste Entfernung von Startecke a verringert werden, ist aber ihre aktuelle kürzeste Entfernung von p nicht negativ, so wird sie nicht in die Menge T aufgenommen; man nimmt sie jedoch in die Menge A auf, damit im Rahmen des später fortzusetzenden Grundprozesses deren Nachfolger aufgesucht und eventuell Entfernungsänderungen für sie vorgenommen werden können.

Zu jeder Ecke $v \in T$ wird neben der Entfernung $d^p v$ von Ecke p ein Wert $l(v)$ gespeichert, der die Anzahl der Pfeile im bislang gefundenen kürzesten Weg von p nach v angibt.

Zu Beginn der Prozedur ist $T=\{q \mid dq > dp + c(p,q) \wedge c(p,q) < 0\}$, $d^p q = c(p,q)$ sowie $l(q)=1$ für alle $q \in T$ und $d^p v = \infty$ für alle sonstigen Ecken v.

(F) In jedem Schritt des Verfahrens wählt man aus der Menge T eine Ecke u aus, für die gilt: $l(u) = \max_{v \in T} l(v)$.

Für jeden Nachfolger z von u bildet man: (1) $m_1 := du + c(u,z)$;
(2) $m_2 := d^p u + c(u,z)$.

Falls $m_1 \geq dz$, geht man zum nächsten Nachfolger von u bzw. zum nächsten Schritt ((F)) über; ansonsten setzt man $dz := m_1$ und prüft:

1. Ist $m_2 < 0$ und $z = p$, so existiert im zugrunde liegenden bewerteten Digraphen ein negativer Zyklus, dem sowohl u als auch p als Ecken angehören (siehe Beispiel in Abb. 1).

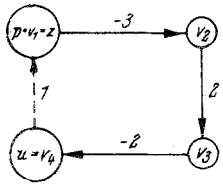

Abb. 1

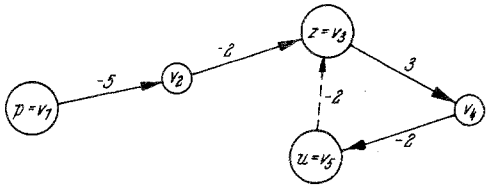

Abb. 2

2. Ist $m_2 < 0$, $z \neq p$ und $d^p z < 0$, so können zwei Fälle vorliegen:
   a) u und z sind Ecken eines (und desselben) Zyklus (siehe Beispiel in Abb. 2);
   b) Im Laufe der Prozedur wurden ein (bislang kürzester) Weg von p nach u und ein (bislang kürzester) Weg von p nach z gefunden, und beide Wege enthalten bis zu einer Ecke $\bar{p}$ identische Teilwege (siehe Beispiel in Abb. 3).

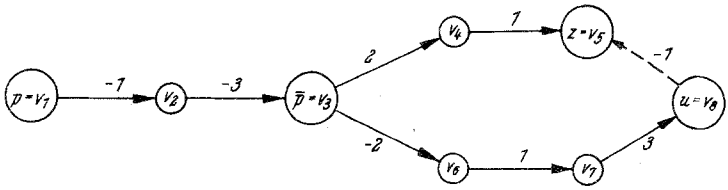

Abb 3. Hier wird angenommen, daß man aus $T=\{v_4, v_6\}$ zunächst die Ecke v_4 zur Fortsetzung des Prozesses auswählt

Ob Fall a) oder Fall b) vorliegt, kann leicht überprüft werden. Voraussetzung dafür ist, daß man während der Durchführung des gesamten Verfahrens (oder zumindest der Prozedur) zu jeder Ecke z deren unmittelbaren Vorgänger in dem bis zum jeweiligen Stadium gefundenen kürzesten Weg von a nach z (bzw. von p nach z) notiert[11].

Falls weder Fall 1. noch Fall 2. a) gegeben sind (wodurch das Verfahren zum Abschluß käme), prüft man ferner:

3. Ist $m_2 < 0$, so setzt man $d^p z := m_2$, $l(z) := l(u) + 1$, und z wird Element von T; ansonsten wird z Element von A (von z aus wird dann erst im Rahmen des Grundprozesses weitergegangen).

Sind alle Nachfolger von u auf diese Weise überprüft worden, wird u aus T entfernt und die Prozedur mit einem neuen Element von T bei (F) fortgesetzt.

Die Prozedur ist beendet, falls $T = \emptyset$.

Der Beweis dafür, daß das Verfahren stets zu richtigen Ergebnissen gelangt, ist einfach zu führen:

Falls im gegebenen bewerteten Digraphen kein negativer Zyklus vorhanden ist, kann der Beweis dafür, daß stets die kürzesten Entfernungen gefunden werden, sofort aus dem in [4], S. 16/17, angegebenen Beweis zum Ford-Algorithmus abgeleitet werden.

Enthält der gegebene Graph jedoch einen von der gegebenen Startecke a aus erreichbaren negativen Zyklus $\zeta = (v_1, v_2, \ldots v_i = v_1)$, dann gilt:

Es existieren j (≥ 1, $\leq i$) Ecken in ζ, so daß die kürzeste Entfernung von jeder dieser Ecken v_j zu jeder anderen Ecke des Zyklus kleiner 0 ist. Dann wird entweder

a) in irgendeinem Stadium des Grundprozesses der Fall eintreten, daß ein v_j erstmals markiert ($\in A$) ist und die aktuelle kürzeste Entfernung von der Startecke a für dessen Nachfolger q_j über v_j verringert werden kann — dies führt zur Unterbrechung des Grundprozesses und zur Ausführung der Prozedur, in deren Verlauf der Zyklus gefunden wird (Fall 1., vgl. Abb. 1) — oder

b) es existiert mindestens ein Weg $w = (p_1, p_2, \ldots, p_h = v_j)$ zu einer Ecke v_j mit der obigen Eigenschaft, für den gilt:

   α) p_1 ist von der Startecke a aus erreichbar;

   β) $c(p_1, \ldots, p_i) < 0$ für alle $i = 2, 3, \ldots, h$;

   γ) die Summe aus der kürzesten Entfernung von p_h zu jeder Ecke des Zyklus und $c(w)$ ist kleiner 0;

   und es tritt der Fall ein, daß im Grundprozeß p_1 markiert ist und die aktuelle kürzeste Entfernung von der Startecke a für dessen Nachfolger p_2 über p_1

[11] Der Einfachheit halber wurde hier die Vorgängernotierung bei der Beschreibung der Verfahren nicht berücksichtigt (vgl. Fußnote 5).

verringert werden kann. Dies führt ebenfalls zur Unterbrechung des Grundprozesses und zur Ausführung der Prozedur, in deren Verlauf der Zyklus gefunden wird (Fall 2.a, vgl. Abb. 2).

Sind mehrere negative Zyklen im gegebenen Graphen vorhanden, so wird durch das eben beschriebene Verfahren derjenige gefunden, für den zuerst einer der Fälle a) oder b) eintritt.

2.3. Ein Beispiel

Die Vorgehensweise beider Verfahren soll an einem sehr einfachen Beispiel nochmals veranschaulicht werden. Gegeben sei der bewertete Digraph in Abb. 4, gesucht seien die kürzesten Entfernungen (und Wege) von Ecke 3 zu allen Ecken des Graphen.

$$C(G)=\begin{pmatrix} 0 & \infty & \infty & \infty & -4 \\ \infty & 0 & \infty & -6 & \infty \\ 4 & \infty & 0 & -1 & 1 \\ \infty & \infty & \infty & 0 & 3 \\ \infty & 2 & \infty & -3 & 0 \end{pmatrix}$$

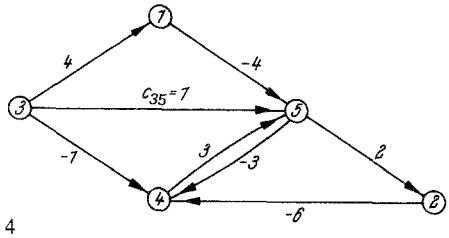

Abb. 4

a) Verfahren 2.1.

$G1$:[12] $A_1=\{3\}$; $B_1=\{1,4,5\}$; $d_1\,1=4$, $d_1\,2=\infty$, $d_1\,3=0$, $d_1\,4=-1$, $d_1\,5=1$.

$G2$: $v_2=4$; $A_2=\{3,4\}$; $B_2=\{1,5\}$; $d_2\,1=4$, $d_2\,2=\infty$, $d_2\,5=1$.

$P1$: $T_1=\{4\}$; $d_2\,4=-1$, $d_2\,3=0$.

$P2$: Abbruch, da $d_1\,3=d_2\,3$.

$G3$: $v_3=5$; $A_3=\{3,4,5\}$; $B_3=\{1,2\}$; $d_3\,1=4$, $d_3\,2=3$.

$P1$: $T_1=\{5\}$; $d_3\,3=0$, $d_3\,4=-2$, $d_3\,5=1$.

$P2$: $u_2=4$; $T_2=\{4,5\}$; kein Zyklus, keine Entfernungsverringerungen, Abbruch zu Beginn von $P3$.

$G4$: $v_4=2$; $A_4=\{2,3,4,5\}$; $B_4=\{1\}$; $d_4\,1=4$.

$P1$: $T_1=\{2\}$; $d_4\,2=3$, $d_4\,3=0$, $d_4\,4=-3$, $d_4\,5=1$.

$P2$: $u_2=4$; $T_2=\{2,4\}$; $d_4\,5:=d_4\,4+c(4,5)=0$.

$P3$: $u_3=5$; $T_3=\{2,4,5\}$; $d_4\,5+c(5,2)=0+2<3=d_4\,2\Rightarrow$ negativer Zyklus mit 5 und 2 als Ecken.

b) Verfahren 2.2.

$G0$: $A=\{3\}$; $d\,3=0$, $d\,1=d\,2=d\,4=d\,5=\infty$.

$G1$: $d\,1=4$, $d\,2=\infty$, $d\,3=0$, $d\,5=1$; $d\,3+c(3,4)=0+\underline{(-1)}<\infty=d\,4$;

[12] Auf „Gk:" folgen die Ergebnisse nach Grundschritt k; auf „Pi:" diejenigen nach Prozedurschritt i.

$d\,4:=-1$, Unterbrechung des Grundprozesses; $4\notin A, 3\notin A, A=\{1,5\}$;

$P0$: $p=3$; $T=\{4\}$; $d^p\,4=d^3\,4=-1, l(4)=1$.

$P1$: $l(4)=\max\limits_{v\in T} l(v)=1$; da $\mathcal{N}(4)=\{5\}$, nur zu bilden:

$m_1=d\,4+c(4,5)=2>1=d\,5$, $m_2=d^3\,4+c(4,5)=2>0$

Ecke 5 wird nicht Element von T; $4\notin T$.

$P2$: $T=\emptyset\Rightarrow$ Ende der Prozedur.

$G2$: Wahl von $1\in A$ als markierte Ecke; $d\,1=4$, $d\,2=\infty$, $d\,3=0$, $d\,4=-1$, $d\,1+c(1,5)=4+(-4)=0<1=d\,5\Rightarrow d\,5:=0$, Prozedur!

$P0$: $p=1$; $T=\{5\}$; $d^1\,5=-4, l(5)=1$.

$P1$: $l(5)=\max\limits_{v\in T} l(v)=1$; da $\mathcal{N}(5)=\{2,4\}$, ist zu bilden:

a) $m_1=d\,5+c(5,2)=2<\infty=d\,2$, $m_2=d^1\,5+c(5,2)=-2<0$.
$d\,2:=2$; Fall 1. und 2. nicht erfüllt. $d^1\,2=-2, l(2)=2$; $2\in T$.

b) $m_1=d\,5+c(5,4)=-3<-1=d\,4$, $m_2=d^1\,5+c(5,4)=-7<0$.
$d\,4:=-3$; Fall 1. und 2. nicht erfüllt. $d^1\,4=-7$; $l(4)=2$;
$4\in T$, $5\notin T$, damit $T=\{2,4\}$.

$P2$: Wahl von $2\in T$, $l(2)=\max\limits_{v\in T} l(v)=2$; da $\mathcal{N}(2)=\{4\}$, nur zu bilden:

$m_1=d\,2+c(2,4)=-4<-3=d\,4$, $m_2=d^1\,2+c(2,4)=-8<0$.
$d\,4:=-4$; nur Fall 2. b) erfüllt. $d^1\,4=-8$, $l(4)=3$; $4\in T$, $2\notin T$, damit $T=\{4\}$.

$P3$: $l(4)=\max\limits_{v\in T} l(v)=3$; da $\mathcal{N}(4)=\{5\}$, nur zu bilden:

$m_1=d\,4+c(4,5)=-1<0=d\,5$,
$m_2=d^1\,4+c(4,5)=-5<0$. $d\,5:=-1$; Fall 2. a) ist erfüllt, der vorliegende Graph enthält einen negativen Zyklus, dem sowohl Ecke 4 als auch Ecke 5 angehören.

Rechenerfahrungen

Zu beiden oben angegebenen Verfahren sind vom Verfasser ALGOL-Programme entwickelt worden (sie erscheinen unter der Rubrik „Algorithmen" in diesem Heft). Sie wurden anhand von mittels Pseudozufallszahlen gebildeten, beliebig reell bewerteten Digraphen mit (oder auch ohne) negativen Zyklen und maximal 150 Ecken bezüglich des benötigten Rechenaufwands miteinander verglichen.

Der benötigte Rechenaufwand hängt bei beiden Verfahren zunächst von der Art der Programmierung, insbesondere von der gewählten Weise, auf die der Graph in den Rechner eingelesen wird, ab. Wie bei den ursprünglichen Verfahren von Dijkstra und Ford[13] bringt auch hier die Eingabe mittels der einzelnen Pfeile gegenüber der Eingabe mittels Kostenmatrix für das Verfahren 2 (Zykl Ford) größere Vorteile als für Verfahren 1 (Zykl Dijkstra). Darüber hinaus ließ sich aber bei mehr als 100 ausgeführten Tests zum Vergleich zweier Prozeduren kein

[13] Vgl. Domschke [4], S. 84—88 und S. 90—95.

funktionaler Zusammenhang zwischen dem Quotienten aus den Rechenzeiten für beide Verfahren und einer leicht nachprüfbaren Eigenschaft des Graphen (wie Ecken- oder Pfeilzahl) feststellen, wie dies offensichtlich für die ursprünglichen Verfahren von Dijkstra und Ford der Fall ist (vgl. Domschke [4], S. 103—106). Der Rechenaufwand, der von jedem der beiden Verfahren verursacht wird, hängt ganz entscheidend von der Lage des zu suchenden Zyklus innerhalb des gegebenen Graphen ab. Es lassen sich leicht Beispiele angeben, in denen das Verfahren 2 den Zyklus mit geringerem Rechenaufwand findet als Verfahren 1 und umgekehrt.

Vergleichsergebnisse: Beim Vergleich der Matrixeingabeprozeduren ergab sich, daß in etwa zwei Drittel aller Fälle die Prozedur zu Verfahren 1 (Zykl Dijkstra) weniger Rechenzeit als die Prozedur zu Verfahren 2 benötigte, während in den restlichen Fällen das Umgekehrte galt. Bei Vergleich der Pfeileingabeprozeduren war Zykl Dijkstra dagegen in weniger als der Hälfte aller Fälle besser als Zykl Ford.

Zum Vergleich der hier angegebenen zu bisher existierenden Verfahren läßt sich sagen:

a) Da der ursprüngliche Dijkstra-Algorithmus dem Verfahren von Dantzig [1] bezüglich des benötigten Rechenaufwands überlegen ist, gilt mit Sicherheit, daß das vom Dijkstra-Algorithmus abgeleitete Verfahren in Abschnitt 2.1 dem vom Dantzig-Algorithmus abgeleiteten Verfahren in [2] überlegen ist.

b) Daß zur Bestimmung negativer Zyklen in Graphen das Verfahren Zykl Ford in fast allen Fällen dem ursprünglichen Ford-Algorithmus weitaus überlegen ist, überlegt man sich ebenfalls sofort. Nur für wenige Spezialfälle (wie z. B. demjenigen, daß der gesamte Graph nur aus einem Zyklus $(1, 2, \ldots, n)$ besteht) ist der Rechenaufwand für beide Verfahren gleich hoch oder für Zykl Ford geringfügig höher.

c) Ein Vergleich der in dieser Arbeit angegebenen Verfahren mit der „Direct Search Method" von Florian und Robert [8] wurde anhand von Rechentests, die den oben beschriebenen entsprachen, durchgeführt. In mehr als 80% der gerechneten Beispiele erwiesen sich sowohl Zykl Ford (Pfeileingabe) als auch Zykl Dijkstra (Matrixeingabe) einem zur Direct Search Method entwickelten Programm überlegen, wobei letzteres im Durchschnitt die 1,5fache Rechenzeit benötigte.

Bedeutung der Verfahren

Beide Verfahren erlangen insbesondere im Rahmen eines von Domschke in [6] und [7] angegebenen Algorithmus zur Bestimmung kostenminimaler Flüsse in Kapazitätendigraphen Bedeutung; denn die Suche negativer Zyklen stellt einen wesentlichen Teil des Verfahrens, das sich dem Out-of-Kilter-Algorithmus bezüglich des benötigten Rechenaufwands als weitaus überlegen erweist, dar.

Literatur

[1] Dantzig, G. B.: On the shortest route through a network. Man. Sci. **6,** 187 (1960).
[2] Dantzig, G. B., W. O. Blattner and M. R. Rao: All shortest routes from a fixed origin in a graph. In: Théorie des graphes, journées internationales d'étude, Rome, juillet 1966 (V. Rosenstiehl, Hrsg.), S. 85. Paris-New York: 1967.
[3] Dijkstra, E. W.: A note on two problems in connection with graphs. Num. Math. **1,** 269 (1959).
[4] Domschke, W.: Kürzeste Wege in Graphen: Algorithmen, Verfahrensvergleiche. (Mathematical Systems in Economics, Heft 2.) Meisenheim/Glan: Hain. 1972.
[5] Domschke, W.: Vergleich des Rechenaufwands der „Shortest-Route"-Algorithmen von Ford, Bellman und Dijkstra. Oper. Res.-Verfahren **11,** 26 (1972).
[6] Domschke, W.: Ein Algorithmus zur Bestimmung kostenminimaler Flüsse in Kapazitätendigraphen [erscheint in Oper. Res.-Verfahren **15** (1973)].
[7] Domschke, W.: Two new algorithms for minimal cost flow problems (erscheint in Computing).
[8] Florian, M., and P. Robert: A direct search method to locate negative cycles in a graph. Management Science **17,** 307 (1971).
[9] Ford, L. R. Jr.: Network flow theory. Rand Paper P-923. Santa Monica 1956.
[10] Harary, F., R. Z. Norman, and D. Cartwright: Structural models: An introduction to the theory of directed graphs. New York-London-Sydney: Wiley. 1965.
[11] Knödel, W.: Graphentheoretische Methoden und ihre Anwendungen, S. 26. Berlin-Heidelberg-New York: Springer. 1969.
[12] Moore, E. F.: The shortest path through a maze. The Annals of the Computation Laboratory of Harvard University **30,** 285 (1959).
[13] Murchland, J. D.: Bibliography of the shortest route problem. London Business School. Transportation Network Theory Unit **6.2** (1969).
[14] Yen, J. Y.: On the efficiency of a direct search method to locate negative cycles in a network. Man. Sci. **19,** 333 (1972).

Dr. Wolfgang Domschke
Institut für Wirtschaftstheorie und
Operations Research der Universität Karlsruhe
Kaiserstraße 12
D-7500 Karlsruhe
Bundesrepublik Deutschland