

# Implizite Algorithmen für Graphprobleme

Daniel Sawitzki

daniel.sawitzki@uni-dortmund.de

<http://sourceforge.net/projects/thefigaro>

FB Informatik, LS 2  
Universität Dortmund  
D-44221 Dortmund

Betreuer der Arbeit: Prof. Dr. Ingo Wegener  
Art der Arbeit: Diplomarbeit  
Fachbereich der GI: 0

## Zusammenfassung

In vielen wichtigen Anwendungsgebieten sind heutzutage Probleme auf sehr großen Graphen zu lösen. Beispiele sind die Verifikation von Schaltkreisen und die Modellierung von Verkehrsnetzen und des WWWs. Die Arbeit mit der klassischen Adjazenzlistendarstellung von Graphen stößt hier an Speicher- und Laufzeitgrenzen. Implizite Graphdarstellungen fassen die Kantenmenge als boolesche Funktion auf und können kompakte, sublineare Größen für stark strukturierte Graphen besitzen. Die Diplomarbeit beschäftigt sich mit Algorithmen, die wichtige Graphprobleme auf implizit vorliegenden Graphen lösen.

## 1 Einführung

Graphalgorithmen auf Graphen  $G = (V, E)$  erhalten ihre Eingabe üblicherweise in Form von Adjazenzlisten, die für jeden Knoten  $v \in V$  die zu ihm adjazenten Knoten  $adj(v) \subseteq V$  enthalten. Diese Darstellung besitzt die Größe  $O(|V| + |E|)$  und wird *explizit* genannt.

Jedoch existieren Anwendungsgebiete, in denen Probleme auf so großen Graphen zu lösen sind, dass eine explizite Speicherung auf heutigen Rechnern nicht möglich ist. Ein Beispiel hierfür sind die Zustandsübergangsgraphen sequentieller Schaltkreise, deren Zustandsmenge exponentiell mit der Größe ihres Speichers wächst. So entstehen z. B. Graphen mit  $10^{27}$  Knoten und  $10^{36}$  Kanten.

In einem abgeschwächten Szenario möchten wir Probleme auf Graphen lösen, deren Größe eine explizite Darstellung noch erlaubt. Jedoch sind wichtige, als effizient geltende Algorithmen hier nicht mehr praktikabel. Der Flussmaximierungsalgorithmus von Malhotra, Pramodh Kumar und Maheshwary [4] benötigt z. B. kubische Laufzeit. Solche Graphen entstehen in Anwendungsgebieten wie der Modellierung des WWW, von Verkehrsnetzen oder von sozialen Netzwerken.

Im allgemeinen ist der lineare Platzbedarf der Adjazenzlisten optimal. Desweiteren können wir keinen Flussmaximierungsalgorithmus entwerfen, der immer mit linearer oder gar sublinearer Laufzeit auskommt. Wir erwarten jedoch, dass sehr große, in der Praxis auftretende Graphen auch starke Strukturen und Regelmäßigkeiten besitzen, die eine kompaktere Darstellung durch spezielle Datenstrukturen erlauben.

*Implizite* (oder auch *symbolische*) Algorithmen arbeiten auf Eingaben in *impliziter Darstellung*. Die Eingabe solcher Verfahren wird durch Teilmengen problemspezifischer Obermengen charakterisiert. Die Algorithmen erhalten so eine Teilmenge in Form einer booleschen Funktion  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , die binäre Kodierungen ihrer Elemente auf 1 abbildet und ansonsten eine 0 liefert und als *charakteristische Funktion* bezeichnet wird.

Entsprechende Graphalgorithmen arbeiten typischerweise auf impliziten Knoten- und Kantenmengen. Dazu werden die Knoten nummeriert:  $V = \{v_0, \dots, v_{|V|-1}\}$ . Seien  $x$  und  $y$  binäre Knotenkodierungen der Länge  $\lceil \log |V| \rceil$ . Die Kantenrelation  $E(x, y)$  ist nun die charakteristische Funktion der Menge  $E$  und bildet das Paar  $(x, y)$  genau dann auf 1 ab, wenn  $(v_{|x|}, v_{|y|}) \in E$  gilt.

*Ordered Binary Decision Diagrams (OBDDs)* sind graphbasierte Datenstrukturen für boolesche Funktionen, die günstige Eigenschaften für den Einsatz in impliziten Algorithmen bieten (siehe [5]). Auf der einen Seite gestatten sie eine kompakte Repräsentation vieler wichtiger Funktionen, auf der anderen Seite bieten sie effiziente Operationen an.

Die erhofften Strukturen und Regelmäßigkeiten der Funktionen führen zu kompakten OBDDs. Schlimmstenfalls jedoch besitzt ein OBDD für  $E(x, y)$  die Größe  $O(|V|^2)$  und ist damit nicht wesentlich größer als eine entsprechende Adjazenzmatrix. Die OBDD-Größe lässt sich ebenfalls durch  $O(|E| \cdot \log |V|)$  abschätzen und ist daher auch nicht wesentlich größer als eine Adjazenzlistendarstellung. In dieser Diplomarbeit werden deswegen OBDDs als Datenstruktur der charakteristischen Funktionen betrachtet.

OBDDs sollen im Folgenden als Black Box aufgefasst werden. Implizite Algorithmen sind meistens unabhängig von einer konkreten Datenstruktur beschrieben. Sie arbeiten mit Funktionsoperationen wie Konjunktion, Disjunktion, Negation und Quantifizierung. OBDDs bieten hierfür effiziente Operationen an. Als wesentlich und teuer gilt die binäre Synthese  $f \otimes g$  für  $\otimes \in B_2$  (z. B.  $f \vee g$ ). Sei  $|G_f|$  die Größe des OBDDs  $G_f$  für Funktion  $f$ . Eine binäre Synthese wird dann in Analysen typischerweise mit Laufzeit  $O(|G_f| \cdot |G_g|)$  bewertet.

Eine Quantifizierung  $(Qx_i) f(x)$  für  $Q \in \{\exists, \forall\}$  kann ebenfalls durch eine binäre Synthese realisiert werden. Alle anderen wichtigen Operationen sind in linearer oder gar konstanter Zeit bzgl. der OBDD-Größe möglich. Daher werden in Analysen oft nur die binären Synthesen und Quantifizierungen betrachtet.

Beim Entwurf impliziter Verfahren wird versucht, durch eine boolesche Operation möglichst viele Knoten bzw. Kanten parallel zu verarbeiten. Die an der Operation teilnehmenden charakteristischen Funktionen sollen viele Elemente repräsentieren und zugleich eine kleine OBDD-Darstellung besitzen. So kann eine geringere Laufzeit als bei der sequentiellen Verarbeitung der Elemente durch explizite Algorithmen erreicht werden.

Jedoch können implizite Algorithmen auch inhärent sequentielle Eigenschaften besitzen. Dies bedeutet, dass trotz einer kompakten Darstellung der Daten Elemente der Eingabe einzeln betrachtet werden müssen, und somit keine Parallelisierung möglich ist.

Motivation der Diplomarbeit war die Umsetzung allgemeiner, wichtiger Graphalgorithmen in implizite Form. Der implizite Flussmaximierungsalgorithmus in 0-1-Netzwerken von Hachtel und Somenzi (HS-Algorithmus) [3] stellt hier einen ersten Schritt dar.

## 2 Analyse des HS-Algorithmus

Der implizite Flussmaximierungsalgorithmus in 0-1-Netzwerken von Hachtel und Somenzi [3] erhält die Kantenrelation  $E(x, y)$  eines Netzwerks sowie die charakteristischen Funktionen  $s(x)$  und  $t(x)$  von Quelle und Senke. Es wird ein maximaler Fluss von  $s$  nach  $t$  ermittelt, dessen Kanten schließlich als implizite Menge  $F(x)$  vorliegen.

Obwohl der HS-Algorithmus ein implizites Verfahren ist, gehört er durch seine grundlegende Arbeitsweise zur Klasse der Niveaunetzwerkalgorithmen (siehe [1]). Er behandelt die Ebenen der Niveaunetze einzeln.

Ein Ziel der Diplomarbeit war, die Laufzeit des HS-Algorithmus zu analysieren. Dazu wurden zunächst Teile des HS-Algorithmus, die in [3] nur skizzenhaft beschrieben werden, konkretisiert. Von Interesse war die Anzahl der verschiedenen wesentlichen OBDD-Operationen, die bei einer Flussmaximierung durchgeführt werden.

Eine einfache, asymptotische worst-case Abschätzung führte zu einer Anzahl von  $O(|V|^3 \log |V|)$  Operationen. Falls alle teilnehmenden OBDDs worst-case Größe besitzen, erhält man eine Gesamtlaufzeit von  $O(|V|^9 / \log |V|)$ . Diese Ergebnisse sind jedoch für uns uninteressant – im worst-case sind die impliziten Algorithmen nicht mehr praktikabel. Wir hoffen im Gegenteil, dass günstige Struktureigenschaften großer Eingaben ermöglichen, den Platzbedarf und die Laufzeit expliziter Algorithmen zu schlagen.

Aus diesem Grund wurden einige Netzwerkeigenschaften und Iterationsanzahlen nicht abgeschätzt, sondern gehen als Variablen in die Analyse ein. Dies sind die Anzahl der Phasen  $P$ , die maximale Niveaunetzwerktiefe  $l_{max}$ , die Summe der Niveaunetzwerktiefen  $L$ , die Anzahl kodierender Bits  $n = \lceil \log |V| \rceil$  sowie die Iterationsanzahlen  $S$  und  $C$ , auf deren Bedeutung hier nicht näher eingegangen wird. Sie müssen für konkret betrachtete Netzwerke ersetzt werden, um eine bessere Schätzung der Operationen zu erhalten. Diese Vorgehensweise führte zu einer Anzahl von

$$9P + ((n + 5)l_{max} - n + 4)S + (n + 10)L + (5n + 10)C$$

wesentlichen OBDD-Operationen. In der Diplomarbeit wird desweiteren zwischen binären Synthesen und Quantifizierungen unterschieden, sowie der Anzahl von Knotenargumenten, die die charakteristischen Funktionen erhalten. Diese Unterscheidung wurde durch die Vermutung motiviert, dass eine höhere Anzahl von Knotenargumenten in vielen Fällen zu größeren OBDDs führt.

Da die Ebenen der Niveaunetze einzeln behandelt werden, geht  $L$  linear in die Abschätzung mit ein. Dies ist eine unangenehme, inhärent sequentielle Eigenschaft: Die Netzwerke können Tiefe  $\Theta(|V|)$  besitzen und somit zu linearer Laufzeit in  $|V|$  führen, obwohl die einzelnen OBDDs sehr kompakt, und die OBDD-Operationen sehr effizient auszuführen sind. Ein alternativer Algorithmus, der dieses Problem umgeht, wird in Abschnitt 4 vorgestellt.

### 3 Analyse des HS-Algorithmus auf quadratischen Gitternetzwerken

Um konkrete Operationsanzahlen für eine ausgewählte Netzwerkklasse zu erhalten, wurden *quadratische Gitternetzwerke* betrachtet. Diese  $N \times N$ -Knotenmatrizen enthalten Kanten zwischen vertikal und horizontal benachbarten Knoten.

$$\begin{aligned} V &= \{0, \dots, N-1\}^2 \\ E &= \{(i, j), (k, l)\} \in V^2 \mid |i - k| + |j - l| = 1 \} \end{aligned}$$

Als Quelle fungiert Knoten  $(1, 1)$ , als Senke  $(N-2, N-2)$ . Unabhängig von  $N$  beträgt der maximale Flusswert stets 4.

Es wurde bewiesen, dass unabhängig von den konstruierten flussverbessernden Pfaden zwei Phasen ausreichen, um einen maximalen Fluss zu berechnen ( $P = 2$ ). Ebenso wurden worst-case und best-case Werte der anderen Variablen ermittelt.

$$L = 4N - 8 \qquad l_{max} = 2N - 2 \qquad 2 \leq S \leq 4 \qquad 32N - 78 \leq C \leq 48N - 120$$

Einsetzen in die Formel aus Abschnitt 2 führt zu einer Operationsanzahl von  $(504N - 1240) \log N$  im worst-case und  $(336N - 804) \log N$  im best-case. Haben alle an den Berechnungen teilnehmenden OBDDs Größe  $o(N/\sqrt{N \log N})$ , benötigt der HS-Algorithmus auf quadratischen Gitternetzwerken asymptotisch weniger Rechenzeit als alle expliziten Flussalgorithmen.

Desweiteren wurden zwei Knotennummerierungen entwickelt. Während die eine optimal bzgl. der OBDD-Größe des Gitternetzwerks ist, führt die andere zu einer minimalen Anzahl OBDD-Operationen bei der Flussmaximierung.

### 4 Implizite Flussmaximierung durch iteratives Quadrieren

Die unangenehme sequentielle Eigenschaft des HS-Algorithmus motivierte die Entwicklung eines alternativen Ansatzes, der die Technik des *iterativen Quadrierens* nutzt. Die grundlegende Idee dabei ist, Pfadlängen iterativ zu verdoppeln und so im günstigen Fall mit logarithmisch-polynomiell vielen Operationen eine Flussmaximierung durchführen zu können – unabhängig von der Netzwerktiefe. Die nachfolgende Rekursionsgleichung zeigt die Berechnung einer Funktion  $PATH_{k+1}^{\leq}(x, y)$ , die eine 1 liefert, wenn ein Pfad von  $x$  nach  $y$  existiert, dessen Länge durch  $2^{k+1}$  beschränkt ist. Dies ist genau dann der Fall, wenn ein  $x$ - $z$ -Pfad und ein  $z$ - $y$ -Pfad mit jeweils durch  $2^k$  beschränkter Länge für einen Zwischenknoten  $z$  existiert.

$$PATH_{k+1}^{\leq}(x, y) = (\exists z) (PATH_k^{\leq}(x, z) \wedge PATH_k^{\leq}(z, y))$$

Die Weiterentwicklung dieser Idee führt zu einem Flussalgorithmus, der zunächst durch eine binäre Suche die Tiefe des Niveaunetzwerks feststellt. Letzteres wird in Niveaufolgen zerlegt, deren Länge jeweils Zweierpotenzen sind. Auf diesen Niveaufolgen werden getrennt kantendisjunkte, flussverbessernde Pfade berechnet. Schließlich werden die getrennt berechneten Pfade zusammengesetzt, um den Fluss zu verbessern und zur nächsten Phase überzugehen.

In allen Schritten wird iteratives Quadrieren verwendet, um die sequentielle Behandlung von Ebenen zu vermeiden. Insgesamt kann im günstigen Fall ein maximaler Fluss durch  $O(\log^3 |V|)$  OBDD-Operationen ermittelt werden. Besitzt die Eingabe z. B. große Netzwerktiefe, jedoch nur einen kleinen (möglicherweise konstanten) Flusswert, werden asymptotisch wesentlich weniger Operationen durchgeführt als beim HS-Algorithmus.

### 5 Heuristische Kantenselektion

Ein Modul des HS-Algorithmus selektiert implizit eine möglichst große Menge kantendisjunkter Pfade der Länge 2 zwischen Ebene  $m-1$  und  $m+1$  des jeweiligen Niveaunetzwerks. Im Rahmen der Diplomarbeit wurden alternative Verfahren entwickelt, die speziell für den Fall einfacher Ebenen  $m-1$ ,  $m$  und  $m+1$  geeignet sind. Die Knoten des Teilgraphs auf diesen drei Ebenen haben also einen durch 1 beschränkten Ingrad oder Outgrad. Die Heuristik erkennt einfache Ebenen und entscheidet, welches Verfahren zu verwenden ist. Dazu wird die Anzahl erfüllender Belegungen einiger charakteristischer Funktionen berechnet, um Knoten- und Kantenanzahlen zu ermitteln.

Sei  $S$  die Anzahl der Kanten zwischen den Ebenen  $m - 1$  und  $m$ ,  $Z$  die Anzahl der Knoten in Ebene  $m$  und  $E$  die Anzahl der Kanten zwischen Ebene  $m$  und  $m + 1$ .  $c \in \mathbf{R}_{\geq 1}$  sei eine Konstante. Gilt nun  $S/Z \leq c$  oder  $E/Z \leq c$ , so existieren höchstens  $c$ -mal so viele Startkanten bzw. Endkanten potenzieller kantendisjunkter Pfade zwischen Ebene  $m - 1$  und  $m$  wie Zwischenknoten in  $m$ . Für  $c = 1$  handelt es sich um eine einfache Ebene. Für größeres  $c$  werden auch noch „fast“ einfache Ebenen erkannt.

Experimente haben gezeigt, dass sich die Verwendung der Heuristik bei zufälligen Netzwerken weder positiv noch negativ auf die Laufzeit der Flussmaximierung auswirkt. Bei der Betrachtung nur bedingt zufälliger, im wesentlichen jedoch strukturierter Netzwerke konnte die Heuristik den HS-Algorithmus um ca. 60% beschleunigen.

## 6 Weitere Ergebnisse

Im Folgenden werden weitere Ergebnisse der Diplomarbeit lediglich angesprochen. Für einfache Netzwerke, in denen der Ingrad oder Outgrad jedes Knoten durch 1 beschränkt ist, wurde eine unangenehme Eigenschaft des HS-Algorithmus bewiesen: Seine Laufzeit ist nach unten durch den maximalen Flusswert beschränkt. Dieser kann jedoch lineare Größe in  $|V|$  besitzen. Es wurde eine Variante entwickelt, die dieses Problem umgeht und speziell auf die Struktur einfacher Netzwerke angepasst ist.

Bei der Reduktion des Maximum-Matching-Problems in bipartiten Graphen auf ein Flussproblem entstehen ebenfalls einfache Netzwerke. Daher führte eine weitere Anpassung zu einem impliziten Maximum-Matching-Algorithmus, der den Reduktionsschritt umgeht.

Schließlich wurde ein vollständig neuer Ansatz zur Flussmaximierung verfolgt: Die implizite Umsetzung eines Preflow-Push-Algorithmus [2]. Dabei wird versucht, die Push- und Relabel-Operationen implizit auf eine möglichst große Menge an wählbaren Kanten bzw. aktiven Knoten parallel anzuwenden. Jedoch taucht auch hier ein ähnliches Problem wie beim HS-Algorithmus auf: Es wird eine Breitensuche durchgeführt, die inhärent sequentiell ist. Auch hier ist eine Anwendung des iterativen Quadrierens denkbar.

Neben der theoretischen Analyse und Entwicklung von Algorithmen war eine Implementierung gewünscht. Zusätzlich zur Originalversion des HS-Algorithmus nach [3] wurden die in Abschnitt 5 vorgestellte Heuristik, die angepasste Variante für einfache Netzwerke sowie diverse kleinere Modifikationen implementiert. Diese Erweiterungen können getrennt aktiviert und verwendet werden.

## 7 Ausblick

In den Analysen wurde vor allem auf die Anzahl von OBDD-Operationen eingegangen. Erst eine genauere Abschätzung der Gesamtlaufzeit ermöglicht jedoch einen fairen Vergleich mit expliziten Algorithmen.

Zur weiteren Analyse bieten sich Netzwerkklassen wie z. B. die  $n$ -dimensionale Verallgemeinerung der quadratischen Gitternetzwerke an.

Nur ein Teil der im Rahmen der Diplomarbeit entwickelten Algorithmen ist bisher implementiert worden. Die Variante aus Abschnitt 4 sowie der implizite Preflow-Push-Algorithmus sind daher noch nicht experimentell untersucht worden. Dies kann Bestandteil zukünftiger Forschung sein.

Schließlich ist eine Umsetzung weiterer wichtiger Graphalgorithmen in implizite Form von Interesse. Beispiele sind die Flussmaximierung auf Netzwerken mit allgemeinen Kantenkapazitäten sowie das Kürzeste-Wege-Problem.

## Literatur

- [1] Even, S. (1979). *Graph algorithms*. Computer Science Press, Rockville, MD.
- [2] Goldberg, A.V. und Tarjan, R.E. (1988). A new approach to the maximum flow problem. *Journal of the ACM* 35, 921-940.
- [3] Hachtel, G.D. und Somenzi, F. (1997). A symbolic algorithm for maximum flow in 0-1 networks. *Formal Methods in System Design* 10, 207-219.
- [4] Malhotra, V.M., Pramodh Kumar, M. und Maheshwary, S.N. (1978). An  $O(|V|^3)$  algorithm for finding maximum flows in networks. *Information Processing Letters* 7, 277-278.
- [5] Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams*. SIAM, Philadelphia.