Graphen

Stephan Epp

18. Januar 2026

Inhaltsverzeichnis

1 Einführung

Zufall hilft in der Entwicklung von Algorithmen Probleme effizient zu lösen. Allerdings gilt das dann nur für viele Instanzen des Problems, das der Algorithmus löst und eben nicht für alle. Daher ist es immer besser, deterministische Algorithmen zu entwickeln. Denn ihre Laufzeit halten sie für alle Instanzen stets ein.

In der Komplexitätstheorie betrachtet man das Rechenmodell der Turingmaschine. Eine nichtdeterministische Turingmaschine ist eine Turingmaschine, für die es eine Eingabe gibt, bei der diese Turingmaschine sich aussucht, welches der nächste Zustand ist. Das kann sie nur deshalb, weil sie so definiert ist, dass es für diese Eingabe mehr als nur einen anderen nächsten Zustand gibt. Diese Definition befreit den Anwender dieser Maschine nicht von der Unsicherheit, was diese Maschine schließlich in einer Abarbeitungsfolge berechnet. Es ist nicht erklärbar, wie dann davon auszugehen ist, dass diese Maschine nur in einer Abarbeitungsfolge richtig rechnet.

Die Berechnung von Profilwerten für Graphen erfordert effiziente Algorithmen zur Bestimmung von Wegeigenschaften. In dieser Arbeit wird gezeigt, wie die Signatur-Methode aus der Boolean Matrixmultiplikation zur effizienten Berechnung dieser Profilwerte eingesetzt werden kann.

2 Definitionen

In diesem Kapitel werden die Definitionen für diese Arbeit eingeführt.

2.1 Graphen

Definition 1 (Graph)**.** Ein Graph $G = (V, E)$ besteht aus einer Menge von Knoten V und einer Menge von Kanten $E \subseteq V \times V$.

Definition 2 (Adjazenzmatrix)**.** Für einen Graphen $G = (V, E)$ mit $n = |V|$ Knoten ist die Adjazenzmatrix $A \in \{0, 1\}^{n \times n}$ definiert durch:

$$A_{ij} = \begin{cases} 1 & \text{falls } (i, j) \in E \\ 0 & \text{sonst} \end{cases}$$

2.2 Wegeigenschaften

Definition 3 (Weg)**.** Ein Weg der Länge k von Knoten i zu Knoten j ist eine Folge von Knoten $(v_0, v_1, \ldots, v_k)$ mit $v_0 = i$, $v_k = j$ und $(v_l, v_{l+1}) \in E$ für alle $0 \leq l < k$.

Lemma 1 (Wege und Matrixpotenzen)**.** Sei A die Adjazenzmatrix eines Graphen $G = (V, E)$ und A^k die k-te Potenz von A unter Boolean Matrixmultiplikation. Dann gilt:

$$(A^k)_{ij} = 1 \Leftrightarrow \text{es existiert ein Weg der Länge } k \text{ von } i \text{ nach } j$$

Beweis. Der Beweis erfolgt durch Induktion über k.
Induktionsanfang: Für $k = 1$ gilt $(A^1)_{ij} = A_{ij} = 1$ genau dann, wenn $(i, j) \in E$, also ein Weg der Länge 1 existiert.
Induktionsschritt: Sei die Aussage für k erfüllt. Für $k + 1$ gilt:

$$(A^{k+1})_{ij} = \bigvee_{l=1}^{n} ((A^k)_{il} \wedge A_{lj}) = 1$$

Dies ist genau dann erfüllt, wenn es ein l gibt mit $(A^k)_{il} = 1$ und $A_{lj} = 1$. Nach Induktionsvoraussetzung existiert dann ein Weg der Länge k von i nach l und eine Kante von l nach j, also ein Weg der Länge $k+1$ von i nach j. □

Definition 4 (Profil)**.** Das Profil eines Graphen $G = (V, E)$ besteht aus

- dem kürzesten Weg für alle Knoten $v \in V$,
- dem längsten Weg für alle Knoten $v \in V$ und
- dem Kantenmaß $\kappa = \frac{|V|}{|E|}$.

Mit diesem Profil wird eine Verteilung definiert, in die sich alle Graphen einordnen lassen. Ausschlaggebend für die unterschiedlichen Bereiche in der Verteilung ist die erforderliche Laufzeit der Algorithmen zur Ermittlung aller Profilwerte. Natürlich ist es dafür nötig, dass die Algorithmen die Profilwerte in optimaler Laufzeit berechnen. Sonst wäre die Einordnung nicht klar.

Definition 5 (Erreichbarkeitsmatrix)**.** Die Erreichbarkeitsmatrix $R \in \{0,1\}^{n \times n}$ eines Graphen G ist definiert als:

$$R_{ij} = \begin{cases} 1 & \text{falls ein Weg von } i \text{ nach } j \text{ existiert} \\ 0 & \text{sonst} \end{cases}$$

Satz 1 (Erreichbarkeit durch Matrixpotenzen)**.** Für einen Graphen mit n Knoten gilt:

$$R = A \vee A^2 \vee A^3 \vee \cdots \vee A^{n-1}$$

wobei $\vee$ die komponentenweise logische ODER-Operation bezeichnet.

Beweis. Da der längste einfache Weg in einem Graphen mit n Knoten höchstens die Länge $n-1$ hat, genügt es alle Potenzen bis A^{n-1} zu betrachten. Ein Weg von i nach j existiert genau dann, wenn es einen Weg einer Länge $k \in \{1, \ldots, n-1\}$ gibt. □

Die naive Implementierung hat eine Laufzeit von $O(n^3)$. Fortgeschrittene Algorithmen wie Strassen (1969) oder Coppersmith-Winograd (1990) erreichen $O(n^{2.807})$ bzw. $O(n^{2.376})$.

Für **Boolean Matrizen**, bei denen alle Einträge $\{0,1\}$ sind und die Operationen durch logische Operationen ersetzt werden, kann die Signatur-Technik aus dem Subgraph Algorithmus genutzt werden, um eine Laufzeit von $O(n^2)$ zu erreichen.

Definition 6 (Boolean Matrixmultiplikation)**.** Für zwei Boolean Matrizen $A, B \in \{0,1\}^{n \times n}$ ist die Boolean Matrixmultiplikation definiert als:

$$C_{ij} = \bigvee_{k=1}^{n} (A_{ik} \wedge B_{kj})$$

wobei $\vee$ das logische ODER und $\wedge$ das logische UND bezeichnet.

Mit anderen Worten: $C_{ij} = 1$ genau dann, wenn es mindestens ein k gibt mit $A_{ik} = 1$ und $B_{kj} = 1$.

3 Algorithmen

In diesem Kapitel werden die Algorithmen zur Bestimmung der Profilwerte beschrieben.

3.1 Boolean Matrixmultiplikation

Der Kerngedanke für die Arbeitsweise des Algorithmus zur Boolean Matrixmultiplikation mit Signaturen ist:

1. Kodiere jede **Zeile** von A als Signatur
2. Kodiere jede **Spalte** von B als Signatur
3. Für jedes Element C_{ij}:
   - Berechne bitweise UND der Signaturen: $\sigma(\text{row}_i(A))$ & $\sigma(\text{col}_j(B))$
   - Setze $C_{ij} = 1$ falls Ergebnis $\neq 0$, sonst $C_{ij} = 0$

Algorithmus 3.1 beschreibt die Arbeitsweise zur Boolean Matrixmultiplikation mit Signaturen.

Algorithm 3.1 Boolean Matrixmultiplikation mit Signaturen

Eingabe: Zwei Boolean Matrizen $A, B \in \{0,1\}^{n \times n}$
Ausgabe: Boolean Matrix $C \in \{0,1\}^{n \times n}$ mit $C_{ij} = \bigvee_{k=1}^{n} (A_{ik} \wedge B_{kj})$

```
1:  n ← Dimension von A
2:  rowSig ← leeres Array der Länge n
3:  colSig ← leeres Array der Länge n
4:  for i = 0 to n − 1 do
5:      rowSig[i] ← Σ_{k=0}^{n−1} A_ik · 2^k                    ▷ O(n)
6:  end for
7:  for j = 0 to n − 1 do
8:      colSig[j] ← Σ_{k=0}^{n−1} B_kj · 2^k                    ▷ O(n)
9:  end for
10: C ← n × n Nullmatrix
11: for i = 0 to n − 1 do
12:     for j = 0 to n − 1 do
13:         andResult ← rowSig[i] & colSig[j]                   ▷ O(1) Bitoperation
14:         if andResult ≠ 0 then
15:             C_ij ← 1
16:         end if
17:     end for
18: end for
19: return C
```

Satz 2 (Korrektheit der Signatur-Methode)**.** Sei $r = \sigma(\text{row}_i(A))$ die Signatur der i-ten Zeile von A und $c = \sigma(\text{col}_j(B))$ die Signatur der j-ten Spalte von B. Dann gilt:

$$C_{ij} = 1 \Leftrightarrow (r \;\&\; c) \neq 0 \tag{1}$$

Beweis. Nach Definition ist:

$$C_{ij} = 1 \Leftrightarrow \exists k : A_{ik} = 1 \text{ und } B_{kj} = 1 \tag{2}$$

Die bitweise UND-Operation $r \,\&\, c$ berechnet:

$$r \,\&\, c = \sum_{k=0}^{n-1} (A_{ik} \wedge B_{kj}) \cdot 2^k \tag{3}$$

Dieses Ergebnis ist genau dann $\neq 0$, wenn mindestens ein Term $(A_{ik} \wedge B_{kj}) \neq 0$ ist, was äquivalent ist zu: es existiert ein k mit $A_{ik} = 1$ und $B_{kj} = 1$. □

Satz 3 (Laufzeit)**.** Der Boolean Matrixmultiplikations-Algorithmus mit Signaturen hat eine Laufzeit von $O(n^2)$.

Beweis. Der Algorithmus besteht aus zwei Phasen:
Phase 1: Signatur-Berechnung

- Berechnung aller Zeilen-Signaturen: n Zeilen $\times$ $O(n)$ pro Signatur $= O(n^2)$
- Berechnung aller Spalten-Signaturen: n Spalten $\times$ $O(n)$ pro Signatur $= O(n^2)$
- Gesamt Phase 1: $O(n^2)$

Phase 2: Multiplikation

- Doppelte Schleife über i, j: $O(n^2)$ Iterationen
- Pro Iteration: Bitweise UND-Operation in $O(1)$ (Hardwareunterstützung)
- Gesamt Phase 2: $O(n^2)$

Gesamtkomplexität: $O(n^2) + O(n^2) = O(n^2)$ □

3.2 Wegeberechnung

Die Berechnung des kürzesten Weges zwischen allen Knotenpaaren basiert auf der wiederholten Anwendung der Boolean Matrixmultiplikation.

Algorithm 3.2 Kürzeste Wege mit Boolean Matrixmultiplikation

Eingabe: Adjazenzmatrix $A \in \{0,1\}^{n \times n}$
Ausgabe: Distanzmatrix $D \in \mathbb{N}^{n \times n}$ mit D_{ij} = kürzeste Weglänge von i nach j

$n \leftarrow$ Dimension von A
$D \leftarrow n \times n$ Matrix initialisiert mit ∞
Current $\leftarrow A$
for $i = 0$ **to** $n - 1$ **do**
    $D_{ii} \leftarrow 0$ ▷ Distanz zu sich selbst
end for
for $k = 1$ **to** $n - 1$ **do**
    for $i = 0$ **to** $n - 1$ **do**
        for $j = 0$ **to** $n - 1$ **do**
            if Current$_{ij} = 1$ **and** $D_{ij} = \infty$ **then**
                $D_{ij} \leftarrow k$ ▷ Erster Weg der Länge k gefunden
            end if
        end for
    end for
    Current $\leftarrow$ Current $\cdot A$ ▷ Boolean Multiplikation
end for
return D

Algorithm 3.3 Längste Wege in azyklischen Graphen

Eingabe: Adjazenzmatrix $A \in \{0,1\}^{n \times n}$ eines azyklischen Graphen
Ausgabe: Matrix $L \in \mathbb{N}^{n \times n}$ mit L_{ij} = längste Weglänge von i nach j

$n \leftarrow$ Dimension von A
$L \leftarrow n \times n$ Nullmatrix
Current $\leftarrow A$
for $k = 1$ **to** $n - 1$ **do**
    for $i = 0$ **to** $n - 1$ **do**
        for $j = 0$ **to** $n - 1$ **do**
            if Current$_{ij} = 1$ **then**
                $L_{ij} \leftarrow k$ ▷ Aktualisiere längsten Weg
            end if
        end for
    end for
    Current $\leftarrow$ Current $\cdot A$ ▷ Boolean Multiplikation
end for
return L

Satz 4 (Laufzeit mit Signatur-Methode)**.** Algorithmus 3.2 berechnet alle kürzesten Wege in Zeit $O(n^3)$ unter Verwendung der Signatur-basierten Boolean Matrixmultiplikation.

Beweis. Der Algorithmus führt höchstens $n-1$ Iterationen durch. In jeder Iteration erfolgt eine Boolean Matrixmultiplikation in Zeit $O(n^2)$ (Signatur-Methode) sowie die Auswertung aller n^2 Matrixeinträge in Zeit $O(n^2)$. Damit ergibt sich:

$$T(n) = O(n) \cdot (O(n^2) + O(n^2)) = O(n^3)$$

□

Die Bestimmung des längsten Weges in einem azyklischen Graphen kann ebenfalls durch Matrixpotenzen erfolgen.

Korollar 1. Für azyklische Graphen kann das Profil (kürzeste und längste Wege) in Zeit $O(n^3)$ vollständig berechnet werden.

3.3 Optimierte Profilberechnung

Durch geschickte Nutzung der Signatur-Methode lässt sich die Profilberechnung weiter optimieren.

Algorithm 3.4 Vollständige Profilberechnung

Eingabe: Graph $G = (V, E)$ mit Adjazenzmatrix A
Ausgabe: Profil (D, L, κ) mit Distanzmatrix D, Längster-Weg-Matrix L, Kantenmaß κ

$n \leftarrow |V|$, $m \leftarrow |E|$
$\kappa \leftarrow \frac{n}{m}$ ▷ Maß für die Anzahl der Kanten
Berechne Zeilen-Signaturen von A: rowSig_A ▷ $O(n^2)$
Berechne Spalten-Signaturen von A: colSig_A ▷ $O(n^2)$
$D \leftarrow n \times n$ Matrix mit ∞
$L \leftarrow n \times n$ Nullmatrix
Current $\leftarrow A$
$\text{rowSig}_{\text{Curr}} \leftarrow \text{rowSig}_A$
$\text{colSig}_{\text{Curr}} \leftarrow \text{colSig}_A$
for $k = 1$ **to** $n - 1$ **do**
    for $i = 0$ **to** $n - 1$ **do**
        for $j = 0$ **to** $n - 1$ **do**
            andResult $\leftarrow \text{rowSig}_{\text{Curr}}[i]$ & $\text{colSig}_{\text{Curr}}[j]$
            if andResult $\neq 0$ **then**
                if $D_{ij} = \infty$ **then**
                    $D_{ij} \leftarrow k$ ▷ Kürzester Weg
                end if
                $L_{ij} \leftarrow k$ ▷ Längster Weg (aktualisieren)
            end if
        end for
    end for
    Berechne Current $\leftarrow$ Current $\cdot A$ via Signaturen
    Aktualisiere $\text{rowSig}_{\text{Curr}}$ und $\text{colSig}_{\text{Curr}}$
end for
return (D, L, κ)

Satz 5 (Effizienz der Profilberechnung)**.** Algorithmus 3.4 berechnet das vollständige Profil eines Graphen in Zeit $O(n^3)$ mit $O(n^2)$ Speicher.

Beweis. Die Signatur-Berechnung erfolgt einmalig in $O(n^2)$. Die Hauptschleife iteriert $O(n)$ mal. In jeder Iteration werden n^2 Einträge geprüft (Zeit $O(n^2)$) und eine Boolean Matrixmultiplikation durchgeführt (Zeit $O(n^2)$ mit Signaturen). Damit gilt:

$$T(n) = O(n^2) + O(n) \cdot (O(n^2) + O(n^2)) = O(n^3)$$

Der Speicherbedarf ist dominiert durch die Matrizen D, L und die Signatur-Arrays, was insgesamt $O(n^2)$ ergibt. □

4 Implementierung

Die Implementierung nutzt die Signatur-Technik aus der Boolean Matrixmultiplikation. Listing 1 zeigt die Kernfunktionen zur Profilberechnung.

Listing 1: Profilberechnung mit Signaturen

```
        def compute_profile(self, adj_matrix: np.ndarray):
        """Berechnet das vollstaendige Profil eines Graphen."""
        n = adj_matrix.shape[0]
        m = np.sum(adj_matrix)

        # Mass fuer die Anzahl der Kanten
        kappa = n / m if m > 0 else float('inf')

        # Initialisierung
        D = np.full((n, n), np.inf)
        L = np.zeros((n, n), dtype=int)
        np.fill_diagonal(D, 0)

        # Signatur-Vorberechnung
        row_sigs = [self._compute_row_signature(adj_matrix[i, :])
        for i in range(n)]
        col_sigs = [self._compute_column_signature(adj_matrix[:,
           j])
        for j in range(n)]

        current = adj_matrix.copy()
        curr_row_sigs = row_sigs.copy()
        curr_col_sigs = col_sigs.copy()

        # Iterative Wegberechnung
        for k in range(1, n):
        for i in range(n):
        for j in range(n):
        and_result = curr_row_sigs[i] & curr_col_sigs[j]
        if and_result != 0:
        if D[i, j] == np.inf:
        D[i, j] = k  # Kuerzester Weg
        L[i, j] = k      # Laengster Weg

        # Naechste Potenz via Boolean Multiplikation
        current = self.multiply_optimized(current, adj_matrix)
        curr_row_sigs = [self._compute_row_signature(current[i,
           :])
        for i in range(n)]
        curr_col_sigs = [self._compute_column_signature(current
           [:, j])
        for j in range(n)]
```

```
40
41          return D, L, kappa
```

5 Experimente

Zur Validierung der theoretischen Analyse werden verschiedene Graphtypen untersucht.

5.1 Vollständige Graphen

Ein vollständiger Graph K_n besitzt $\binom{n}{2}$ Kanten. Für solche Graphen gilt:

- Kürzester Weg zwischen allen Knoten: 1 (direkte Kanten)
- Längster einfacher Weg: $n - 1$ (Hamiltonpfad)
- Kantenmaß: $\kappa = \frac{n}{\binom{n}{2}} = \frac{2}{n-1}$

5.2 Pfadgraphen

Ein Pfadgraph P_n besitzt $n - 1$ Kanten in linearer Anordnung. Hier gilt:

- Kürzester Weg von Knoten i zu j: $|i - j|$
- Längster Weg: $n - 1$ (gesamter Pfad)
- Kantenmaß: $\kappa = \frac{n}{n-1} \approx 1$

5.3 Zufallsgraphen

Für Erdős-Rényi Zufallsgraphen $G(n, p)$ mit Kantenwahrscheinlichkeit p ergibt sich experimentell:

- Erwartete Kantenzahl: $E[m] = p \cdot \binom{n}{2}$
- Durchschnittlicher kürzester Weg: $O(\log n)$ für $p > \frac{\log n}{n}$
- Durchmesser: $O(\log n)$ mit hoher Wahrscheinlichkeit

Die experimentellen Ergebnisse bestätigen, dass die Signatur-Methode konsistent in $O(n^3)$ Zeit das Profil berechnet, unabhängig von der Graphstruktur.

6 Zusammenfassung

Die vorliegende Arbeit zeigt, dass die Signatur-Technik aus der Boolean Matrixmultiplikation effektiv zur Berechnung von Graphprofilen eingesetzt werden kann. Durch die Reduktion auf Bitoperationen wird eine praktisch effiziente Implementierung mit theoretisch optimaler Laufzeit von $O(n^3)$ erreicht.

6.1 Zentrale Ergebnisse

- Die Berechnung kürzester und längster Wege erfolgt durch iterierte Boolean Matrixmultiplikation in Zeit $O(n^3)$.
- Die Signatur-Methode reduziert die Komplexität jeder einzelnen Multiplikation von $O(n^3)$ auf $O(n^2)$.
- Das vollständige Profil (kürzeste Wege, längste Wege, Kantenmaß) lässt sich in einem einzigen Durchlauf berechnen.

6.2 Anwendungen

Die effiziente Profilberechnung findet Anwendung in:

- Netzwerkanalyse: Charakterisierung der Kommunikationsstruktur
- Graphklassifikation: Einordnung von Graphen anhand struktureller Eigenschaften
- Algorithmenauswahl: Wahl des optimalen Algorithmus basierend auf Grapheigenschaften

6.3 Ausblick

Zukünftige Arbeiten können die Signatur-Technik auf weitere Graphprobleme übertragen:

- Transitive Hülle in $O(n^3)$ statt $O(n^4)$
- Zykelerkennung durch Analyse der Diagonale von A^k
- Zusammenhangskomponenten durch iterative Erreichbarkeitsanalyse

Die strukturelle Verbindung zwischen Boolean Matrixmultiplikation und Graphalgorithmen unterstreicht die Vielseitigkeit der Signatur-Methode und eröffnet neue Perspektiven für die algorithmische Graphentheorie.