Graphen mit Knoten und Kanten

Stephan Epp

19. 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, jede **Zeile** von A als Signatur zu kodieren und jede **Spalte** von B als Signatur zu kodieren und für jedes Element 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 $C_{ij} = \bigvee_{k=1}^{n} (A_{ik} \wedge B_{kj})$

```
n ← Dimension von A
rowSig ← leeres Array der Länge n
colSig ← leeres Array der Länge n
for i = 0 to n − 1 do
    rowSig[i] ← Σ_{k=0}^{n−1} A_ik · 2^k                ▷ O(n)
end for
for j = 0 to n − 1 do
    colSig[j] ← Σ_{k=0}^{n−1} B_kj · 2^k                ▷ O(n)
end for
C ← n × n Nullmatrix
for i = 0 to n − 1 do
    for j = 0 to n − 1 do
        andResult ← rowSig[i] & colSig[j]          ▷ O(1) Bitoperation
        if andResult ≠ 0 then
            C_ij ← 1
        end if
    end for
end for
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 $r \mathbin{\&} c$ berechnet:

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

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

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.

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

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.

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:

$$T(n) = O(n^2) + O(n) \cdot (O(n^2) + O(n^2)) = O(n^3) \tag{4}$$

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

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)

3.4 Implementierung

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

Listing 1: Profilberechnung mit Signaturen

```python
def compute_profile(self, adj_matrix: np.ndarray):
        """Berechnet das vollstaendige Profil eines Graphen."""
        n = adj_matrix.shape[0]
        m = int(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)  # Distanz zu sich selbst ist 0

        # WICHTIG: Starte mit k=1 und Current = A (nicht A^0)
```

```python
        current = adj_matrix.copy()

        # Iterative Wegberechnung - O(n) Iterationen
        for k in range(1, n):
            for i in range(n):
                for j in range(n):
                    if current[i, j] == 1:
                        # Kuerzester Weg
                        if D[i, j] == np.inf:
                            D[i, j] = k

                        # Laengster Weg
                        L[i, j] = k

            # Naechste Potenz via Boolean Multiplikation
            current = self.multiplier.multiply_optimized(
                current, adj_matrix)

        return D, L, kappa
```

3.5 Graphtypen

Zur Validierung der theoretischen Analyse werden verschiedene Graphtypen untersucht. 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: $\kappa = \frac{n}{\binom{n}{2}} = \frac{2}{n-1}$

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: $\kappa = \frac{n}{n-1} \approx 1$

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

- 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

Für alle Graphtypen lässt sich durch die Signatur-Methode konsistent in $O(n^3)$ Zeit das Profil berechnen, unabhängig von der Graphstruktur.

4 Optimale Graphprofilverteilung

Die in dieser Arbeit entwickelten Algorithmen ermöglichen nicht nur die effiziente Berechnung von Graphprofilen, sondern garantieren darüber hinaus eine **optimale Einordnung** jedes Graphen in die Graphprofilverteilung. Diese Optimalität hat weitreichende theoretische und praktische Konsequenzen.

4.1 Optimalität der Einordnung

Satz 6 (Optimale Charakterisierung)**.** Die durch Algorithmus 3.4 berechnete Einordnung eines Graphen G in die Graphprofilverteilung mittels des Tripels (D, L, κ) ist optimal und nicht verbesserbar.

Beweis. Die Optimalität folgt aus drei Eigenschaften:

1. **Vollständigkeit**: Jeder Knoten $v \in V$ und jede Kante $e \in E$ wird in der Berechnung berücksichtigt. Es existiert keine versteckte Struktur, die nicht erfasst wird.

2. **Exaktheit**: Die Distanzmatrix D enthält für jedes Knotenpaar (i, j) die exakte kürzeste Weglänge. Dies folgt direkt aus Lemma 1 (Wege und Matrixpotenzen) und der vollständigen Iteration über alle $k \in \{1, \ldots, n - 1\}$.

3. **Determinismus**: Der Algorithmus liefert für jeden Graphen G stets das gleiche Profil. Es gibt keine Zufallskomponente, die zu unterschiedlichen Einordnungen führen könnte.

Daraus folgt: Jede andere Methode zur Berechnung des Graphprofils muss entweder das gleiche Ergebnis liefern (und ist damit äquivalent) oder liefert ein approximatives, suboptimales Ergebnis. □

Korollar 2 (Sicherheit von Aussagen)**.** Jede auf dem Graphprofil (D, L, κ) basierende Aussage über strukturelle Eigenschaften des Graphen G ist maximal informiert und nicht verbesserbar.

Dies hat eine fundamentale Konsequenz: Wenn eine strukturelle Eigenschaft eines Graphen aus seinem Profil ableitbar ist, dann ist diese Ableitung **garantiert korrekt** und kann durch keine andere Methode verbessert werden.

4.2 Hierarchische Graphprofilverteilungen

In der Praxis liegen oft komplexe Systeme vor, die auf verschiedenen Ebenen als Graphen modelliert werden können. Die optimale Profilberechnung ermöglicht die Analyse solcher **mehrstufiger Graphhierarchien**.

Definition 7 (Hierarchischer Graph)**.** Ein hierarchischer Graph $\mathcal{G} = (G_1, G_2, \ldots, G_h)$ besteht aus h Ebenen, wobei jede Ebene i durch einen Graphen $G_i = (V_i, E_i)$ repräsentiert wird. Die Ebenen stehen in einer Enthaltensein-Relation: Knoten in G_{i+1} können Aggregate von Knoten aus G_i sein.

Beispiel 1 (Rechenzentrum)**.** Ein Rechenzentrum lässt sich als hierarchischer Graph mit drei Ebenen modellieren:

- Ebene 1 (Rack-Ebene): Knoten sind Racks, Kanten sind physische Netzwerkverbindungen zwischen Racks

- Ebene 2 (Server-Ebene): Knoten sind Server, Kanten sind Verbindungen innerhalb und zwischen Racks

- Ebene 3 (VM-Ebene): Knoten sind virtuelle Maschinen, Kanten sind logische Kommunikationsbeziehungen

Für jede Ebene i kann das Profil (D_i, L_i, κ_i) berechnet werden.

Satz 7 (Laufzeit hierarchischer Profilberechnung)**.** Für einen hierarchischen Graphen $\mathcal{G} = (G_1, \ldots, G_h)$ mit $|V_i| = n_i$ beträgt die Gesamtlaufzeit zur Berechnung aller Profile:

$$T_{\text{gesamt}} = \sum_{i=1}^{h} O(n_i^3) = O\left(\sum_{i=1}^{h} n_i^3\right)$$

Die hierarchische Analyse ermöglicht die Detektion von **Anomalien auf verschiedenen Abstraktionsebenen**. Beispielsweise könnte ein plötzlicher Anstieg von κ_2 (Server-Ebene) bei konstantem κ_1 (Rack-Ebene) auf eine Netzwerkpartitionierung innerhalb eines Racks hinweisen.

4.3 Kanonische Einordnung in die Verteilung

Das Kantenmaß $\kappa = \frac{|V|}{|E|}$ teilt die Menge aller Graphen in charakteristische Bereiche ein:

Definition 8 (Graphdichteklassen)**.** Sei $G = (V, E)$ ein Graph mit $n = |V|$ Knoten. Dann lässt sich G wie folgt klassifizieren:

- **Sehr dicht**: $\kappa < 1$, d.h. $|E| > |V|$ (viele Kanten)

- **Ausgewogen**: $\kappa \approx 1$, d.h. $|E| \approx |V|$ (Bäume, Pfade)

- **Dünn**: $\kappa > 2$, d.h. $|E| < |V|/2$ (wenige Kanten)

Lemma 2 (Extremfälle)**.** Für spezielle Graphklassen gilt:

- Vollständiger Graph K_n: $\kappa = \frac{2}{n-1} \to 0$ für $n \to \infty$

- Pfadgraph P_n: $\kappa = \frac{n}{n-1} \to 1$ für $n \to \infty$

- Sterngraph S_n: $\kappa = \frac{n}{n-1} \to 1$ für $n \to \infty$

- Isolierte Knoten: $\kappa = \infty$ (keine Kanten)

Die Kombination aus κ, dem Durchmesser $\text{diam}(G) = \max_{i,j} D_{ij}$ und der maximalen längsten Weglänge $\max_{i,j} L_{ij}$ ermöglicht eine **mehrdimensionale Charakterisierung** von Graphen.

5 Anwendungen in der Praxis

Die optimale Berechnung von Graphprofilen eröffnet vielfältige Anwendungsmöglichkeiten in verschiedenen Domänen.

5.1 Neurowissenschaften und Gehirnforschung

5.1.1 Konnektomanalyse

Das menschliche Gehirn besteht aus ca. $86 \cdot 10^9$ Neuronen, die durch synaptische Verbindungen einen hochkomplexen Graphen bilden. Die optimale Profilberechnung ermöglicht:

- **Strukturelle Charakterisierung**: Berechnung von (D, L, κ) für neuronale Netzwerke unterschiedlicher Hirnregionen

- **Vergleichende Analyse**: Deterministische Gegenüberstellung von Konnektomen verschiedener Individuen

- **Entwicklungsanalyse**: Verfolgung der zeitlichen Entwicklung neuronaler Verbindungsmuster

Beispiel 2 (Kognitive Leistungsprofile)**.** Hypothese: Unterschiedliche kognitive Fähigkeiten manifestieren sich in unterschiedlichen Graphprofilverteilungen neuronaler Netzwerke.

Ein Individuum mit hoher räumlicher Intelligenz könnte in visuellen Kortexregionen ein Profil mit niedrigem κ (hohe Konnektivität) und geringem Durchmesser (schnelle Informationsverbreitung) aufweisen.

Die Optimalität der Profilberechnung garantiert, dass solche Unterschiede **deterministisch nachweisbar** sind, falls sie existieren.

5.1.2 Pathologie-Detektion

Neurologische Erkrankungen wie Alzheimer oder Schizophrenie gehen mit strukturellen Veränderungen im Gehirn einher. Diese Veränderungen sollten sich im Graphprofil widerspiegeln:

- **Baseline-Profil**: Berechne $(D_{\text{gesund}}, L_{\text{gesund}}, \kappa_{\text{gesund}})$ für gesunde Kontrollgruppen

- **Abweichungsdetektion**: Identifiziere Individuen mit signifikant abweichendem Profil

- **Früherkennung**: Longitudinale Beobachtung von Profiländerungen über Zeit

5.2 Künstliche Intelligenz

5.2.1 Optimale Netzwerkarchitektur-Suche

Tiefe neuronale Netze lassen sich als gerichtete Graphen modellieren, wobei Neuronen Knoten und Gewichtsverbindungen Kanten darstellen.

Algorithm 5.1 Profilbasierte Neural Architecture Search

Eingabe: Menge von Kandidaten-Architekturen $\mathcal{A} = \{A_1, \ldots, A_k\}$, Ziel-Profil $(D_{\text{target}}, L_{\text{target}}, \kappa_{\text{target}})$

Ausgabe: Optimale Architektur A^*

```
for jede Architektur A_i ∈ 𝒜 do
    Konstruiere Graphrepräsentation G_i von A_i
    (D_i, L_i, κ_i) ← ComputeProfile(G_i)
    score_i ← Distanz((D_i, L_i, κ_i), (D_target, L_target, κ_target))
end for
A* ← arg min_{A_i} score_i
return A*
```

Vorteil: Im Gegensatz zu stochastischen Suchverfahren (z.B. evolutionäre Algorithmen) ist die Bewertung jeder Architektur **deterministisch und reproduzierbar**.

5.2.2 Transferierbarkeit und Modellvergleich

Satz 8 (Strukturelle Äquivalenz)**.** Seien M_1 und M_2 zwei neuronale Netze mit Graphrepräsentationen G_1 und G_2. Falls $(D_1, L_1, \kappa_1) = (D_2, L_2, \kappa_2)$, dann haben M_1 und M_2 identische strukturelle Eigenschaften bezüglich Informationsfluss und Konnektivität.

Dies ermöglicht:

- **Modellselektion**: Wähle aus einer Menge vortrainierter Modelle dasjenige mit dem zum aktuellen Task passenden Profil
- **Pruning**: Entferne Verbindungen so, dass das Profil innerhalb akzeptabler Grenzen bleibt
- **Knowledge Distillation**: Übertrage Wissen zwischen Modellen mit ähnlichem Profil

5.2.3 KI-Sicherheit

Die deterministische Berechnung von Graphprofilen kann zur Überwachung unerwarteten Verhaltens eingesetzt werden:

- **Training Monitoring**: Berechne Profil nach jeder Epoche. Sprunghafte Änderungen in κ oder Durchmesser deuten auf *emergent behavior* hin
- **Adversarial Detection**: Adversariale Angriffe könnten sich in Änderungen des Aktivierungsgraphen manifestieren
- **Verifikation**: Zwei Versionen eines Modells sollten identisches Profil haben (deterministischer Integritätscheck)

5.3 Rechenzentren und Cloud Computing

5.3.1 Optimales Datacenter-Design

Die Topologie eines Rechenzentrums bestimmt dessen Leistungsfähigkeit. Die Profilberechnung ermöglicht systematisches Design:

Beispiel 3 (Topologie-Optimierung)**.** Gegeben seien Anforderungen:

- Maximale Latenz zwischen beliebigen Servern: ≤ 5 Hops
- Fehlertoleranz: Bei Ausfall eines Switches darf Durchmesser höchstens um Faktor 2 wachsen
- Kosteneffizienz: Minimiere Anzahl Switches (maximiere κ)

Algorithmus:

1. Generiere Kandidaten-Topologien $T_1, \ldots, T_m$
2. Für jedes T_i: Berechne (D_i, L_i, κ_i)
3. Filter: Behalte nur T_i mit $\max_{j,k} D_i[j,k] \leq 5$
4. Wähle $T^* = \arg\max_{T_i} \kappa_i$ (maximale Kosteneffizienz)

5.3.2 Load Balancing und Ressourcenallokation

Das aktuelle Profil der Kommunikationsstruktur kann zur dynamischen Lastverteilung genutzt werden:

Algorithm 5.2 Profilbasiertes Load Balancing

Eingabe: Aktuelle Kommunikationsmatrix $C \in \{0,1\}^{n \times n}$, neue Anfrage für Server s
Ausgabe: Ziel-Server t für Migration

```
(D, L, κ) ← ComputeProfile(C)
candidates ← {t : D[s,t] < threshold}                 ▷ Nahe Server
for jeder Kandidat t ∈ candidates do
    Simuliere Migration: C′ ← C mit zusätzlicher Kante (s,t)
    κ′ ← ComputeKappa(C′)
    impact_t ← |κ′ − κ|                               ▷ Strukturelle Änderung
end for
return arg min_t impact_t                             ▷ Minimale Störung
```

5.4 Soziale Netzwerke

5.4.1 Influencer-Identifikation

In sozialen Netzwerken sind Knoten mit spezifischen Profil-Eigenschaften besonders einflussreich:

- **Zentrale Knoten**: Knoten v mit $\sum_j D[v,j] = \min$ (geringe durchschnittliche Distanz zu allen anderen)
- **Brückenknoten**: Knoten, deren Entfernung κ signifikant erhöht (verbinden Komponenten)
- **Reichweiten-Knoten**: Knoten mit $\max_j L[v,j] = \text{hoch}$ (können lange Informationsketten initiieren)

5.4.2 Desinformations-Ausbreitung

Die Geschwindigkeit, mit der sich Falschinformationen verbreiten, hängt direkt vom Graphprofil ab:

Satz 9 (Maximale Verbreitungszeit)**.** Sei G das soziale Netzwerk und v_0 die Quelle einer Desinformation. Die maximale Zeit, bis alle erreichbaren Knoten die Information erhalten haben, ist begrenzt durch:

$$T_{\max} \leq \max_{j: D[v_0, j] < \infty} D[v_0, j]$$

Dies ermöglicht **präventive Maßnahmen**: Identifiziere Knoten mit hohem $\max_j D[v, j]$ und priorisiere dort Fact-Checking.

5.5 Biologie und Molekularbiologie

5.5.1 Protein-Interaktionsnetzwerke

Proteine interagieren in komplexen Netzwerken. Das Profil eines Protein-Interaktionsnetzwerks charakterisiert funktionale Eigenschaften:

Beispiel 4 (Drug Target Identification)**.** Problem: Finde Proteine, deren Inhibition eine Krankheit bekämpft.

Ansatz:

1. Konstruiere Protein-Interaktionsnetzwerk G_{disease} für Krankheit
2. Berechne (D, L, κ)
3. Identifiziere Proteine p mit hoher Zentralität: $\sum_j D[p, j]$ minimal
4. Simuliere Entfernung von p: Berechne neues Profil (D', L', κ')
5. Falls $\kappa' \gg \kappa$ (Netzwerk zerfällt), ist p ein guter Target

5.5.2 Evolutionäre Analyse

Vergleich von Protein-Netzwerken über Spezies hinweg:

- Berechne Profile (D_s, L_s, κ_s) für Spezies s
- Phylogenetischer Abstand korreliert mit Profil-Abstand
- Konservierte Subgraphen haben ähnliche lokale Profile

6 Theoretische Implikationen

6.1 Determinismus versus Probabilismus

Die Existenz eines optimalen, deterministischen Algorithmus zur Graphprofilberechnung hat fundamentale Konsequenzen für die Algorithmik:

Satz 10 (Überlegenheit deterministischer Verfahren)**.** Sei $\mathcal{A}_{\text{det}}$ der in dieser Arbeit vorgestellte deterministische Algorithmus zur Profilberechnung und $\mathcal{A}_{\text{prob}}$ ein beliebiger probabilistischer Algorithmus. Dann gilt:

1. $\mathcal{A}_{\text{det}}$ liefert stets das exakte Ergebnis
2. $\mathcal{A}_{\text{prob}}$ liefert mit Wahrscheinlichkeit $p < 1$ ein korrektes Ergebnis
3. Für Anwendungen, die Korrektheit erfordern, ist $\mathcal{A}_{\text{det}}$ strikt zu bevorzugen

Konsequenz für die Praxis: In sicherheitskritischen Anwendungen (medizinische Diagnostik, Infrastruktur-Design, KI-Verifikation) sollten stochastische Methoden durch deterministische ersetzt werden, wo immer möglich.

6.2 Komplexitätstheoretische Einordnung

Die Profilberechnung ist ein Problem in **P** (polynomielle Zeit, deterministisch). Dies steht im Kontrast zu vielen Graphproblemen, die NP-vollständig sind:

- Hamiltonpfad: NP-vollständig
- Maximale Clique: NP-vollständig
- Graphfärbung: NP-vollständig
- **Graphprofil**: P (diese Arbeit, $O(n^3)$)

Korollar 3 (Praktikabilität)**.** Für Graphen mit $n \leq 10.000$ Knoten ist die Profilberechnung in Sekunden auf modernen Rechnern durchführbar. Für $n \leq 100.000$ in Minuten. Dies deckt die meisten praktischen Anwendungen ab.

6.3 Universalität der Methode

Die Signatur-Methode ist nicht auf die Profilberechnung beschränkt. Sie kann auf weitere Graphprobleme übertragen werden:

Satz 11 (Transitive Hülle)**.** Die transitive Hülle eines Graphen G (Erreichbarkeitsmatrix R) kann mit der Signatur-Methode in Zeit $O(n^3)$ berechnet werden.

Beweis. Die Erreichbarkeitsmatrix ist gegeben durch $R = A \vee A^2 \vee \cdots \vee A^{n-1}$. Berechne alle Potenzen A^k mittels Boolean Matrixmultiplikation (je $O(n^2)$), dann komponentenweise ODER-Verknüpfung in $O(n^2)$. Gesamt: $O(n) \cdot O(n^2) = O(n^3)$. □

Weitere Anwendungen:

- **Zykelerkennung**: Falls $A^k[i, i] = 1$ für ein $k \geq 2$, existiert ein Zyklus durch i
- **Starke Zusammenhangskomponenten**: Über Kombination von R und R^T (transponiert)
- **Kürzeste Pfade mit Gewichten**: Erweiterung auf gewichtete Graphen durch Anpassung der Signatur-Methode

7 Ausblick und offene Fragen

7.1 Parallelisierung

Die Signatur-Berechnung ist inhärent parallelisierbar:

- Zeilen- und Spalten-Signaturen können parallel berechnet werden
- Die Boolean Matrixmultiplikation lässt sich auf GPU-Architekturen abbilden
- Potenzielle Beschleunigung: Faktor p bei p Prozessoren

Offene Frage: Lässt sich durch geschickte Parallelisierung eine Laufzeit von $O(n^2)$ für die gesamte Profilberechnung erreichen?

7.2 Sparse Graphen

Viele reale Graphen sind *dünn besetzt* (sparse): $|E| = O(n)$ statt $O(n^2)$.

Frage 1. Kann die Signatur-Methode für sparse Graphen so angepasst werden, dass die Laufzeit von $O(n^3)$ auf $O(n^2)$ oder sogar $O(n \cdot |E|)$ reduziert wird?

Ansatz: Verwende komprimierte Darstellung der Adjazenzmatrix (z.B. Compressed Sparse Row Format) und passe Signatur-Berechnung entsprechend an.

7.3 Dynamische Graphen

In vielen Anwendungen ändern sich Graphen über Zeit (z.B. soziale Netzwerke, Verkehrsnetze).

Frage 2. Kann das Profil eines Graphen G' nach Hinzufügen/Entfernen einer Kante inkrementell aus dem Profil von G berechnet werden, ohne vollständige Neuberechnung?

Potenzielle Laufzeit inkrementeller Update: $O(n^2)$ statt $O(n^3)$.

7.4 Approximative Varianten

Für extrem große Graphen ($n > 10^6$) könnte eine approximative Profilberechnung sinnvoll sein:

Frage 3. Existiert ein Algorithmus, der in Zeit $O(n^2)$ ein Profil $(\tilde{D}, \tilde{L}, \tilde{\kappa})$ berechnet mit garantierter Güte $\|\tilde{D} - D\| < \epsilon$?

Dies würde einen Trade-off zwischen Laufzeit und Exaktheit ermöglichen.

7.5 Anwendung auf Quantengraphen

Mit dem Aufkommen von Quantencomputern stellt sich die Frage:

Frage 4. Lässt sich die Boolean Matrixmultiplikation auf Quantencomputern implementieren? Kann dadurch eine Laufzeit unterhalb von $O(n^2)$ erreicht werden?

Die Signatur-Methode basiert auf Bitoperationen, die sich potenziell auf Qubits übertragen lassen.

7.6 Universelle Graphdatenbank

Eine visionäre Anwendung wäre eine **universelle Datenbank aller bekannten Graphprofile**:

Beispiel 5 (Graph Profile Database)**.** Konstruktion einer Datenbank, die für Millionen bekannter Graphen (Protein-Netzwerke, soziale Netzwerke, Infrastrukturgraphen, etc.) die Profile (D, L, κ) speichert.

Funktionalität:

- Query: Finde alle Graphen mit $\kappa \in [1.0, 1.5]$ und Durchmesser < 10"
- Similarity Search: Finde die 10 ähnlichsten Graphen zu gegebenem Query-Graph"
- Pattern Discovery: Ëntdecke wiederkehrende Profil-Muster über Domänen hinweg"

Eine solche Datenbank wäre ein fundamentales Werkzeug für die Wissenschaft – vergleichbar mit Sequenzdatenbanken in der Bioinformatik.

8 Fazit

Die in dieser Arbeit entwickelte Methode zur optimalen Berechnung von Graphprofilen mittels der Signatur-Technik hat weitreichende theoretische und praktische Implikationen:

Theoretisch wird gezeigt, dass deterministische Algorithmen stochastischen Verfahren überlegen sein können, wenn sie in polynomieller Zeit exakte Ergebnisse liefern.

Praktisch eröffnen sich Anwendungen von der Neurowissenschaft über Künstliche Intelligenz bis hin zur Infrastruktur-Optimierung. Die Garantie optimaler Einordnung in die Graphprofilverteilung macht Aussagen über Graphstrukturen maximal informiert und nicht verbesserbar.

Die Kernbotschaft lautet: *Jeder Graph wird optimal charakterisiert. Darauf basierende Entscheidungen sind deterministisch und reproduzierbar.*

In einer Zeit zunehmender Komplexität und Vernetzung bietet diese Methode ein fundamentales Werkzeug zur Analyse und zum Verständnis strukturierter Systeme.