

Symbolische Berechnung kürzester Wege

Daniel Sawitzki

Universität Dortmund, Informatik 2

DFG-Schwerpunkt 1126 – Jahres-Kolloquium 2004

23. Juli 2004

Übersicht

Einführung

Ein symbolischer APSP-Algorithmus (WG'04)
Breitenbeschränkte Funktionen

Symbolische SSSP-Algorithmen (WEA'04)

Zusammenfassung

Motivation

- ▶ Anwendungen erzeugen große Graphen (VLSI, Verkehr, WWW, ...)
 - ▶ \Rightarrow Konflikte mit interner Speichergröße
 - ▶ \Rightarrow Sogar effiziente Algos. nicht anwendbar
- ▶ Beobachtung: Die Graphen sind groß, aber auch **strukturiert**.
- ▶ Heuristischer Ansatz aus VLSI: **Symbolische Darstellung**
 - ▶ Vermeide explizite Aufzählung von Knoten/Kanten.
 - ▶ Betrachte Graph G als **charakteristische** boolesche Funktion C .
 - ▶ Stelle C durch (hoffentlich) kompakte Datenstruktur dar.
 - ▶ Löse Probleme in G durch **wenige** und **effiziente** Operationen auf C .

Motivation

- ▶ Anwendungen erzeugen große Graphen (VLSI, Verkehr, WWW, ...)
 - ▶ \Rightarrow Konflikte mit interner Speichergröße
 - ▶ \Rightarrow Sogar effiziente Algos. nicht anwendbar
- ▶ Beobachtung: Die Graphen sind groß, aber auch **strukturiert**.
- ▶ Heuristischer Ansatz aus VLSI: **Symbolische Darstellung**
 - ▶ Vermeide explizite Aufzählung von Knoten/Kanten.
 - ▶ Betrachte Graph G als **charakteristische** boolesche Funktion C .
 - ▶ Stelle C durch (hoffentlich) kompakte Datenstruktur dar.
 - ▶ Löse Probleme in G durch **wenige** und **effiziente** Operationen auf C .

Motivation

- ▶ Anwendungen erzeugen große Graphen (VLSI, Verkehr, WWW, ...)
 - ▶ \Rightarrow Konflikte mit interner Speichergröße
 - ▶ \Rightarrow Sogar effiziente Algos. nicht anwendbar
- ▶ Beobachtung: Die Graphen sind groß, aber auch **strukturiert**.
- ▶ Heuristischer Ansatz aus VLSI: **Symbolische Darstellung**
 - ▶ Vermeide explizite Aufzählung von Knoten/Kanten.
 - ▶ Betrachte Graph G als **charakteristische** boolesche Funktion C .
 - ▶ Stelle C durch (hoffentlich) kompakte Datenstruktur dar.
 - ▶ Löse Probleme in G durch **wenige** und **effiziente** Operationen auf C .

Motivation

- ▶ Anwendungen erzeugen große Graphen (VLSI, Verkehr, WWW, ...)
 - ▶ \Rightarrow Konflikte mit interner Speichergröße
 - ▶ \Rightarrow Sogar effiziente Algos. nicht anwendbar
- ▶ Beobachtung: Die Graphen sind groß, aber auch **strukturiert**.
- ▶ Heuristischer Ansatz aus VLSI: **Symbolische Darstellung**
 - ▶ Vermeide explizite Aufzählung von Knoten/Kanten.
 - ▶ Betrachte Graph G als **charakteristische** boolesche Funktion C .
 - ▶ Stelle C durch (hoffentlich) kompakte Datenstruktur dar.
 - ▶ Löse Probleme in G durch **wenige** und **effiziente** Operationen auf C .

Motivation

- ▶ Anwendungen erzeugen große Graphen (VLSI, Verkehr, WWW, ...)
 - ▶ \Rightarrow Konflikte mit interner Speichergröße
 - ▶ \Rightarrow Sogar effiziente Algos. nicht anwendbar
- ▶ Beobachtung: Die Graphen sind groß, aber auch **strukturiert**.
- ▶ Heuristischer Ansatz aus VLSI: **Symbolische Darstellung**
 - ▶ Vermeide explizite Aufzählung von Knoten/Kanten.
 - ▶ Betrachte Graph G als **charakteristische** boolesche Funktion C .
 - ▶ Stelle C durch (hoffentlich) kompakte Datenstruktur dar.
 - ▶ Löse Probleme in G durch **wenige** und **effiziente** Operationen auf C .

Motivation

- ▶ Anwendungen erzeugen große Graphen (VLSI, Verkehr, WWW, ...)
 - ▶ \Rightarrow Konflikte mit interner Speichergröße
 - ▶ \Rightarrow Sogar effiziente Algos. nicht anwendbar
- ▶ Beobachtung: Die Graphen sind groß, aber auch **strukturiert**.
- ▶ Heuristischer Ansatz aus VLSI: **Symbolische Darstellung**
 - ▶ Vermeide explizite Aufzählung von Knoten/Kanten.
 - ▶ Betrachte Graph G als **charakteristische** boolesche Funktion C .
 - ▶ Stelle C durch (hoffentlich) kompakte Datenstruktur dar.
 - ▶ Löse Probleme in G durch **wenige** und **effiziente** Operationen auf C .

Motivation

- ▶ Anwendungen erzeugen große Graphen (VLSI, Verkehr, WWW, ...)
 - ▶ \Rightarrow Konflikte mit interner Speichergröße
 - ▶ \Rightarrow Sogar effiziente Algos. nicht anwendbar
- ▶ Beobachtung: Die Graphen sind groß, aber auch **strukturiert**.
- ▶ Heuristischer Ansatz aus VLSI: **Symbolische Darstellung**
 - ▶ Vermeide explizite Aufzählung von Knoten/Kanten.
 - ▶ Betrachte Graph G als **charakteristische** boolesche Funktion C .
 - ▶ Stelle C durch (hoffentlich) kompakte Datenstruktur dar.
 - ▶ Löse Probleme in G durch **wenige** und **effiziente** Operationen auf C .

Motivation

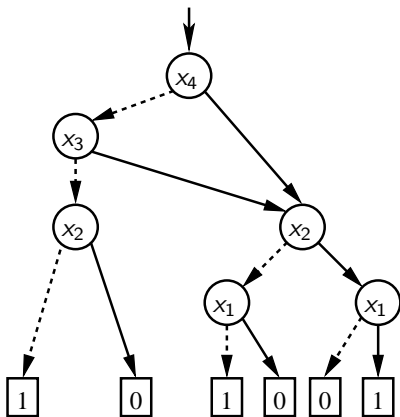
- ▶ Anwendungen erzeugen große Graphen (VLSI, Verkehr, WWW, ...)
 - ▶ \Rightarrow Konflikte mit interner Speichergröße
 - ▶ \Rightarrow Sogar effiziente Algos. nicht anwendbar
- ▶ Beobachtung: Die Graphen sind groß, aber auch **strukturiert**.
- ▶ Heuristischer Ansatz aus VLSI: **Symbolische Darstellung**
 - ▶ Vermeide explizite Aufzählung von Knoten/Kanten.
 - ▶ Betrachte Graph G als **charakteristische** boolesche Funktion C .
 - ▶ Stelle C durch (hoffentlich) kompakte Datenstruktur dar.
 - ▶ Löse Probleme in G durch **wenige** und **effiziente** Operationen auf C .

Motivation

- ▶ Anwendungen erzeugen große Graphen (VLSI, Verkehr, WWW, ...)
 - ▶ \Rightarrow Konflikte mit interner Speichergröße
 - ▶ \Rightarrow Sogar effiziente Algos. nicht anwendbar
- ▶ Beobachtung: Die Graphen sind groß, aber auch **strukturiert**.
- ▶ Heuristischer Ansatz aus VLSI: **Symbolische Darstellung**
 - ▶ Vermeide explizite Aufzählung von Knoten/Kanten.
 - ▶ Betrachte Graph G als **charakteristische** boolesche Funktion C .
 - ▶ Stelle C durch (hoffentlich) kompakte Datenstruktur dar.
 - ▶ Löse Probleme in G durch **wenige** und **effiziente** Operationen auf C .

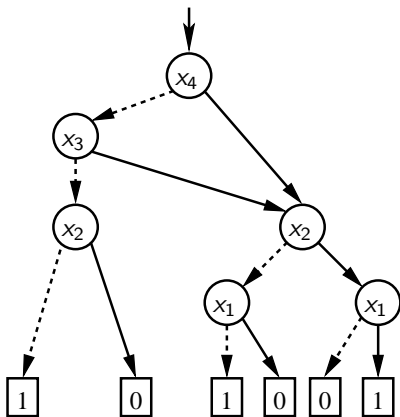
Ordered Binary Decision Diagrams (OBDDs)

- ▶ Datenstruktur für boolesche Funktionen $f: \{0, 1\}^n \rightarrow \{0, 1\}$ (Bryant, 1985)
- ▶ **Innere Knoten** tragen Variablen, 0-/1-Kanten.
- ▶ **Senken** entsprechen Wert $f(x_1, \dots, x_n)$.
- ▶ Lies x_1, \dots, x_n bez. $\pi \in \Sigma_n$.
- ▶ Worst-case Größe: $\mathcal{O}(2^n/n)$.
- ▶ $C(x, y) = 1 \Leftrightarrow (x, y) \in E$



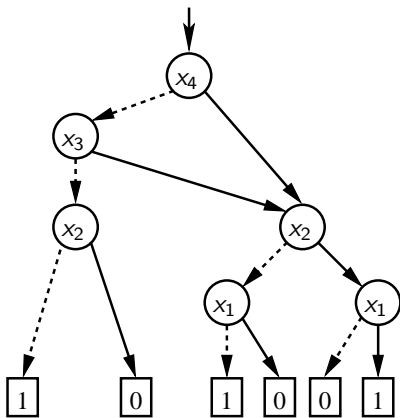
Ordered Binary Decision Diagrams (OBDDs)

- ▶ Datenstruktur für boolesche Funktionen $f: \{0, 1\}^n \rightarrow \{0, 1\}$ (Bryant, 1985)
- ▶ **Innere Knoten** tragen Variablen, 0-/1-Kanten.
- ▶ **Senken** entsprechen Wert $f(x_1, \dots, x_n)$.
- ▶ Lies x_1, \dots, x_n bez. $\pi \in \Sigma_n$.
- ▶ Worst-case Größe: $\mathcal{O}(2^n/n)$.
- ▶ $C(x, y) = 1 \Leftrightarrow (x, y) \in E$



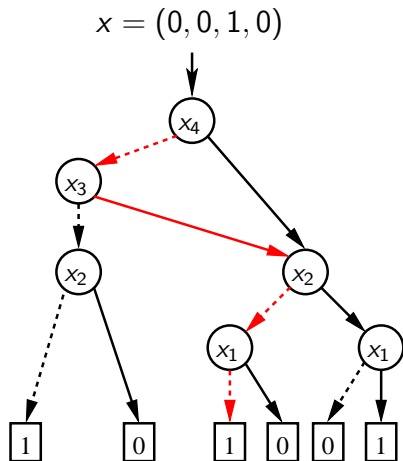
Ordered Binary Decision Diagrams (OBDDs)

- ▶ Datenstruktur für boolesche Funktionen $f: \{0, 1\}^n \rightarrow \{0, 1\}$ (Bryant, 1985)
- ▶ **Innere Knoten** tragen Variablen, 0-/1-Kanten.
- ▶ **Senken** entsprechen Wert $f(x_1, \dots, x_n)$.
- ▶ Lies x_1, \dots, x_n bez. $\pi \in \Sigma_n$.
- ▶ Worst-case Größe: $\mathcal{O}(2^n/n)$.
- ▶ $C(x, y) = 1 \Leftrightarrow (x, y) \in E$



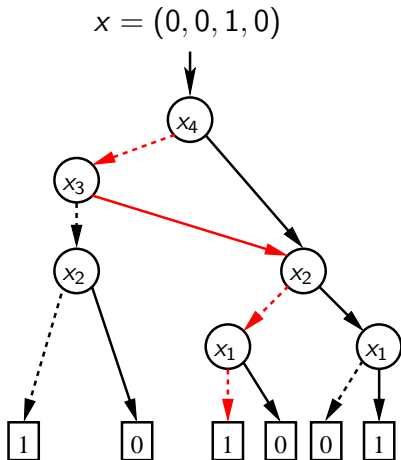
Ordered Binary Decision Diagrams (OBDDs)

- ▶ Datenstruktur für boolesche Funktionen $f: \{0, 1\}^n \rightarrow \{0, 1\}$ (Bryant, 1985)
- ▶ **Innere Knoten** tragen Variablen, 0-/1-Kanten.
- ▶ **Senken** entsprechen Wert $f(x_1, \dots, x_n)$.
- ▶ Lies x_1, \dots, x_n bez. $\pi \in \Sigma_n$.
- ▶ Worst-case Größe: $\mathcal{O}(2^n/n)$.
- ▶ $C(x, y) = 1 \Leftrightarrow (x, y) \in E$



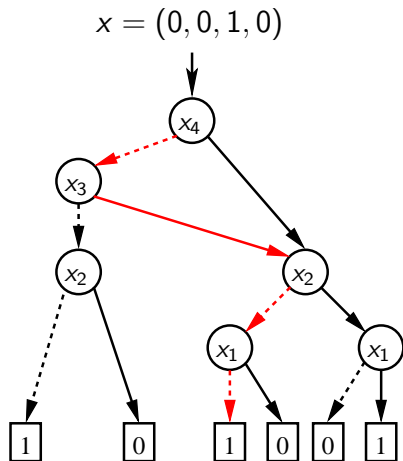
Ordered Binary Decision Diagrams (OBDDs)

- ▶ Datenstruktur für boolesche Funktionen $f: \{0, 1\}^n \rightarrow \{0, 1\}$ (Bryant, 1985)
- ▶ **Innere Knoten** tragen Variablen, 0-/1-Kanten.
- ▶ **Senken** entsprechen Wert $f(x_1, \dots, x_n)$.
- ▶ Lies x_1, \dots, x_n bez. $\pi \in \Sigma_n$.
- ▶ Worst-case Größe: $\mathcal{O}(2^n/n)$.
- ▶ $C(x, y) = 1 \Leftrightarrow (x, y) \in E$



Ordered Binary Decision Diagrams (OBDDs)

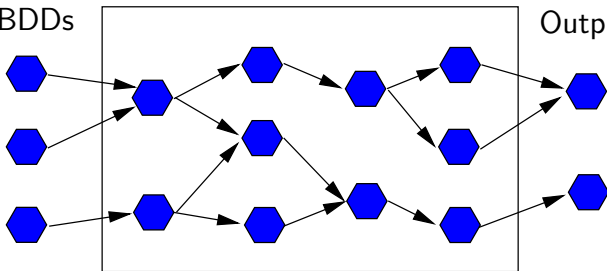
- ▶ Datenstruktur für boolesche Funktionen $f: \{0, 1\}^n \rightarrow \{0, 1\}$ (Bryant, 1985)
- ▶ **Innere Knoten** tragen Variablen, 0-/1-Kanten.
- ▶ **Senken** entsprechen Wert $f(x_1, \dots, x_n)$.
- ▶ Lies x_1, \dots, x_n bez. $\pi \in \Sigma_n$.
- ▶ Worst-case Größe: $\mathcal{O}(2^n/n)$.
- ▶ $C(x, y) = 1 \Leftrightarrow (x, y) \in E$



Symbolische Graphalgorithmen

- ▶ Löse Graphprobleme durch wenige effiziente OBDD-Operationen:
 - ▶ Synthese: \wedge , \vee , \oplus , $=$, \dots
 - ▶ Quantifizierung: $(\exists x) F(x, y)$, $(\forall x) F(x, y)$
 - ▶ Erfüllbarkeit, Äquivalenz, Var.-ersetzung
- ▶ Exp. blow-up durch $\mathcal{O}(\log |V|)$ Operationen möglich.

Input OBDDs

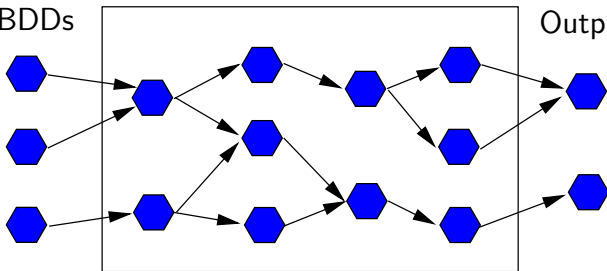


Output OBDDs

Symbolische Graphalgorithmen

- ▶ Löse Graphprobleme durch wenige effiziente OBDD-Operationen:
 - ▶ Synthese: \wedge , \vee , \oplus , $=$, ...
 - ▶ Quantifizierung: $(\exists x) F(x, y)$, $(\forall x) F(x, y)$
 - ▶ Erfüllbarkeit, Äquivalenz, Var.-ersetzung
- ▶ Exp. blow-up durch $\mathcal{O}(\log |V|)$ Operationen möglich.

Input OBDDs

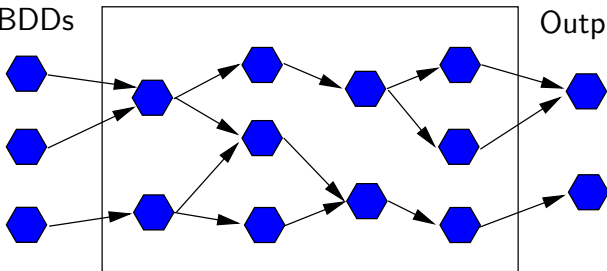


Output OBDDs

Symbolische Graphalgorithmen

- ▶ Löse Graphprobleme durch wenige effiziente OBDD-Operationen:
 - ▶ Synthese: \wedge , \vee , \oplus , $=$, \dots
 - ▶ Quantifizierung: $(\exists x) F(x, y)$, $(\forall x) F(x, y)$
 - ▶ Erfüllbarkeit, Äquivalenz, Var.-ersetzung
- ▶ Exp. blow-up durch $\mathcal{O}(\log |V|)$ Operationen möglich.

Input OBDDs

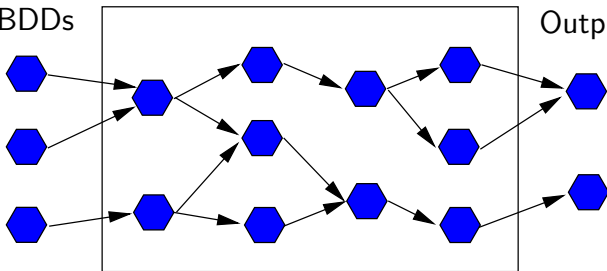


Output OBDDs

Symbolische Graphalgorithmen

- ▶ Löse Graphprobleme durch wenige effiziente OBDD-Operationen:
 - ▶ Synthese: \wedge , \vee , \oplus , $=$, \dots
 - ▶ Quantifizierung: $(\exists x) F(x, y)$, $(\forall x) F(x, y)$
 - ▶ Erfüllbarkeit, Äquivalenz, Var.-ersetzung
- ▶ Exp. blow-up durch $\mathcal{O}(\log |V|)$ Operationen möglich.

Input OBDDs

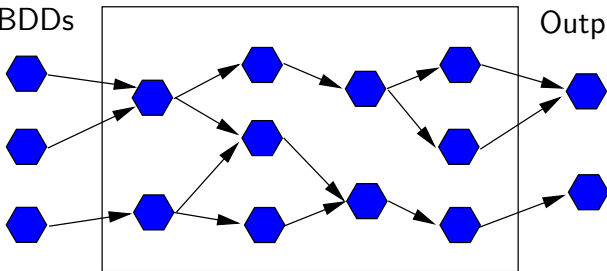


Output OBDDs

Symbolische Graphalgorithmen

- ▶ Löse Graphprobleme durch wenige effiziente OBDD-Operationen:
 - ▶ Synthese: \wedge , \vee , \oplus , $=$, ...
 - ▶ Quantifizierung: $(\exists x) F(x, y)$, $(\forall x) F(x, y)$
 - ▶ Erfüllbarkeit, Äquivalenz, Var.-ersetzung
- ▶ Exp. blow-up durch $\mathcal{O}(\log |V|)$ Operationen möglich.

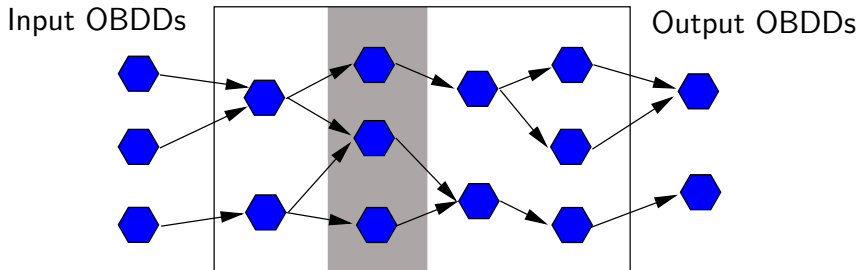
Input OBDDs



Output OBDDs

Symbolische Graphalgorithmen

- ▶ Löse Graphprobleme durch wenige effiziente OBDD-Operationen:
 - ▶ Synthese: \wedge , \vee , \oplus , $=$, \dots
 - ▶ Quantifizierung: $(\exists x) F(x, y)$, $(\forall x) F(x, y)$
 - ▶ Erfüllbarkeit, Äquivalenz, Var.-ersetzung
- ▶ Exp. blow-up durch $\mathcal{O}(\log |V|)$ Operationen möglich.



Bisherige Ergebnisse

- ▶ VLSI: Erreichbarkeit in Zustandsgraphen
- ▶ Flussmaximierung (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topologisches Sortieren (Woelfel, 2003)
- ▶ Starker Zusammenhang, Zweizusammenhang (Gentilini et al., 2003)
- ▶ Kürzeste Wege mit ADDs (Bahar et al., 1993)

Ziel der vorgestellten Arbeit:

Polylog. Laufzeit auf Graphen spezieller Struktur.

Bisherige Ergebnisse

- ▶ VLSI: Erreichbarkeit in Zustandsgraphen
- ▶ Flussmaximierung (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topologisches Sortieren (Woelfel, 2003)
- ▶ Starker Zusammenhang, Zweizusammenhang (Gentilini et al., 2003)
- ▶ Kürzeste Wege mit ADDs (Bahar et al., 1993)

Ziel der vorgestellten Arbeit:

Polylog. Laufzeit auf Graphen spezieller Struktur.

Bisherige Ergebnisse

- ▶ VLSI: Erreichbarkeit in Zustandsgraphen
- ▶ Flussmaximierung (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topologisches Sortieren (Woelfel, 2003)
- ▶ Starker Zusammenhang, Zweizusammenhang (Gentilini et al., 2003)
- ▶ Kürzeste Wege mit ADDs (Bahar et al., 1993)

Ziel der vorgestellten Arbeit:

Polylog. Laufzeit auf Graphen spezieller Struktur.

Bisherige Ergebnisse

- ▶ VLSI: Erreichbarkeit in Zustandsgraphen
- ▶ Flussmaximierung (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topologisches Sortieren (Woelfel, 2003)
- ▶ Starker Zusammenhang, Zweizusammenhang (Gentilini et al., 2003)
- ▶ Kürzeste Wege mit ADDs (Bahar et al., 1993)

Ziel der vorgestellten Arbeit:

Polylog. Laufzeit auf Graphen spezieller Struktur.

Bisherige Ergebnisse

- ▶ VLSI: Erreichbarkeit in Zustandsgraphen
- ▶ Flussmaximierung (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topologisches Sortieren (Woelfel, 2003)
- ▶ Starker Zusammenhang, Zweizusammenhang (Gentilini et al., 2003)
- ▶ Kürzeste Wege mit ADDs (Bahar et al., 1993)

Ziel der vorgestellten Arbeit:

Polylog. Laufzeit auf Graphen spezieller Struktur.

Bisherige Ergebnisse

- ▶ VLSI: Erreichbarkeit in Zustandsgraphen
- ▶ Flussmaximierung (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topologisches Sortieren (Woelfel, 2003)
- ▶ Starker Zusammenhang, Zweizusammenhang (Gentilini et al., 2003)
- ▶ Kürzeste Wege mit ADDs (Bahar et al., 1993)

Ziel der vorgestellten Arbeit:

Polylog. Laufzeit auf Graphen spezieller Struktur.

Übersicht

Einführung

Ein symbolischer APSP-Algorithmus (WG'04)
Breitenbeschränkte Funktionen

Symbolische SSSP-Algorithmen (WEA'04)

Zusammenfassung

Das symbolische APSP-Problem – Definition und Ansatz

Eingabe:

Gewichteter Graph $G = (V, E, \ell)$, $\ell: E \rightarrow \mathbb{N}_{>0}$

$$C(x, y, d) = 1 \Leftrightarrow [(x, y) \in E] \wedge [\ell(x, y) = d]$$

Ausgabe:

APSP-Längen $\Delta: V^2 \rightarrow \mathbb{N}_0$

$$S(x, y, d) = 1 \Leftrightarrow \Delta(x, y) = d$$

► Ansatz: Berechne iterativ S^{i+1} aus S^i mit

$$S^i(x, y, d) = 1 \Leftrightarrow [\Delta(x, y) = d] \wedge [d < 2^i].$$

► $S^1(x, y, d) := [(d = 1) \wedge C(x, y, d)] \vee [(d = 0) \wedge (x = y)]$

Das symbolische APSP-Problem – Definition und Ansatz

Eingabe:

Gewichteter Graph $G = (V, E, \ell)$, $\ell: E \rightarrow \mathbb{N}_{>0}$

$$C(x, y, d) = 1 \Leftrightarrow [(x, y) \in E] \wedge [\ell(x, y) = d]$$

Ausgabe:

APSP-Längen $\Delta: V^2 \rightarrow \mathbb{N}_0$

$$S(x, y, d) = 1 \Leftrightarrow \Delta(x, y) = d$$

► Ansatz: Berechne iterativ S^{i+1} aus S^i mit

$$S^i(x, y, d) = 1 \Leftrightarrow [\Delta(x, y) = d] \wedge [d < 2^i].$$

► $S^1(x, y, d) := [(d = 1) \wedge C(x, y, d)] \vee [(d = 0) \wedge (x = y)]$

Das symbolische APSP-Problem – Definition und Ansatz

Eingabe:

Gewichteter Graph $G = (V, E, \ell)$, $\ell: E \rightarrow \mathbb{N}_{>0}$

$$C(x, y, d) = 1 \Leftrightarrow [(x, y) \in E] \wedge [\ell(x, y) = d]$$

Ausgabe:

APSP-Längen $\Delta: V^2 \rightarrow \mathbb{N}_0$

$$S(x, y, d) = 1 \Leftrightarrow \Delta(x, y) = d$$

- ▶ Ansatz: Berechne iterativ S^{i+1} aus S^i mit

$$S^i(x, y, d) = 1 \Leftrightarrow [\Delta(x, y) = d] \wedge [d < 2^i].$$

- ▶ $S^1(x, y, d) := [(d = 1) \wedge C(x, y, d)] \vee [(d = 0) \wedge (x = y)]$

Das symbolische APSP-Problem – Definition und Ansatz

Eingabe:

Gewichteter Graph $G = (V, E, \ell)$, $\ell: E \rightarrow \mathbb{N}_{>0}$

$$C(x, y, d) = 1 \Leftrightarrow [(x, y) \in E] \wedge [\ell(x, y) = d]$$

Ausgabe:

APSP-Längen $\Delta: V^2 \rightarrow \mathbb{N}_0$

$$S(x, y, d) = 1 \Leftrightarrow \Delta(x, y) = d$$

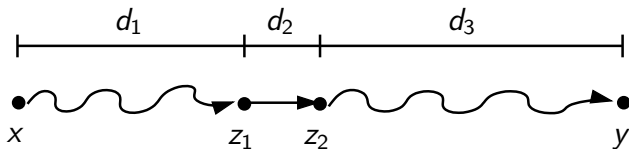
- ▶ Ansatz: Berechne iterativ S^{i+1} aus S^i mit

$$S^i(x, y, d) = 1 \Leftrightarrow [\Delta(x, y) = d] \wedge [d < 2^i].$$

- ▶ $S^1(x, y, d) := [(d = 1) \wedge C(x, y, d)] \vee [(d = 0) \wedge (x = y)]$

Berechnung von S^{i+1}

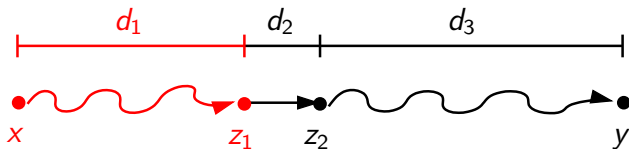
$$S^{i+1}(x, y, d) := (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S^i(x, z_1, d_1) \\ \wedge C(z_1, z_2, d_2) \\ \wedge S^i(z_2, y, d_3)] \\ \wedge (d < 2^{i+1}) [\dots]$$



- ▶ $\log(|V| \cdot \ell^{\max})$ Iterationen a $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ Operationen
- ▶ Insgesamt: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ Operationen

Berechnung von S^{i+1}

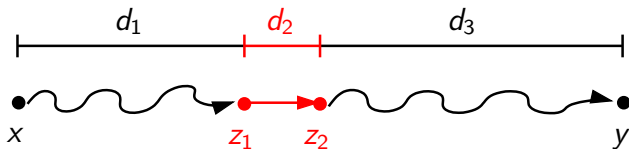
$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) [\dots]
 \end{aligned}$$



- $\log(|V| \cdot \ell^{\max})$ Iterationen à $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ Operationen
- Insgesamt: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ Operationen

Berechnung von S^{i+1}

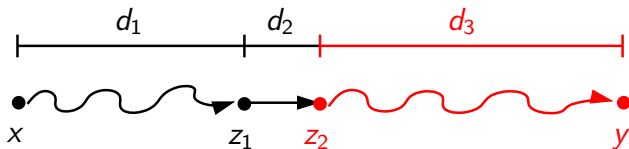
$$\begin{aligned} S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ & \wedge S^i(x, z_1, d_1) \\ & \wedge C(z_1, z_2, d_2) \\ & \wedge S^i(z_2, y, d_3)] \\ & \wedge (d < 2^{i+1}) [\dots] \end{aligned}$$



- ▶ $\log(|V| \cdot \ell^{\max})$ Iterationen a $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ Operationen
- ▶ Insgesamt: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ Operationen

Berechnung von S^{i+1}

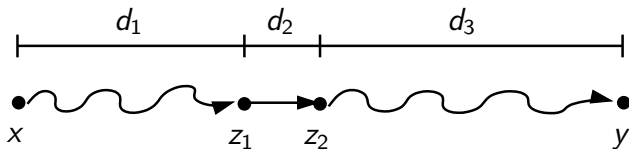
$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) [\dots]
 \end{aligned}$$



- $\log(|V| \cdot \ell^{\max})$ Iterationen a $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ Operationen
- Insgesamt: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ Operationen

Berechnung von S^{i+1}

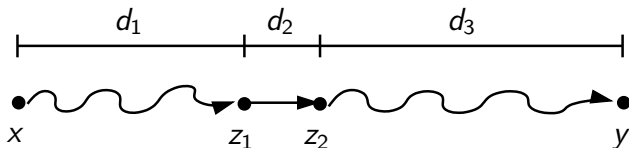
$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) [\dots]
 \end{aligned}$$



- $\log(|V| \cdot \ell^{\max})$ Iterationen à $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ Operationen
- Insgesamt: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ Operationen

Berechnung von S^{i+1}

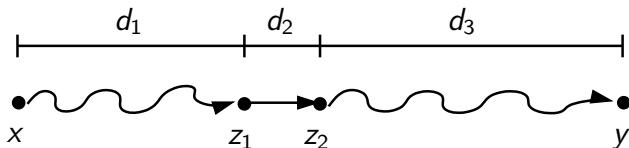
$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) [\dots]
 \end{aligned}$$



- $\log(|V| \cdot \ell^{\max})$ Iterationen a $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ Operationen
- Insgesamt: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ Operationen

Berechnung von S^{i+1}

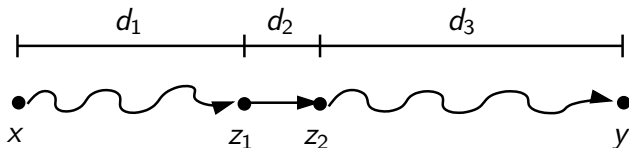
$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) [\dots]
 \end{aligned}$$



- ▶ $\log(|V| \cdot \ell^{\max})$ Iterationen a $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ Operationen
- ▶ Insgesamt: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ Operationen

Berechnung von S^{i+1}

$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) [\dots]
 \end{aligned}$$



- ▶ $\log(|V| \cdot \ell^{\max})$ Iterationen a $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ Operationen
- ▶ Insgesamt: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ Operationen

Übersicht

Einführung

Ein symbolischer APSP-Algorithmus (WG'04)
Breitenbeschränkte Funktionen

Symbolische SSSP-Algorithmen (WEA'04)

Zusammenfassung

Analyse symbolischer Algorithmen – Träume und Realität

Realität bis jetzt:

- ▶ Experimentelle Untersuchungen
- ▶ Zählen der OBDD-Operationen
- ▶ Analyse auf **sehr** speziellen Graphen

Träume:

- ▶ Polynomiell bez. OBDD-Größe
- ▶ Speziell strukturierte Eingabe und Ausgabe \Rightarrow Geringe Laufzeit

Ergebnis für den APSP-Algorithmus:

Eingabe/Ausgabe hat konstante Breite \Rightarrow Polylog. Zeit/Platz
(bez. $|V|$ und ℓ^{\max})

Analyse symbolischer Algorithmen – Träume und Realität

Realität bis jetzt:

- ▶ Experimentelle Untersuchungen
- ▶ Zählen der OBDD-Operationen
- ▶ Analyse auf **sehr** speziellen Graphen

Träume:

- ▶ Polynomiell bez. OBDD-Größe
- ▶ Speziell strukturierte Eingabe und Ausgabe \Rightarrow Geringe Laufzeit

Ergebnis für den APSP-Algorithmus:

Eingabe/Ausgabe hat konstante Breite \Rightarrow Polylog. Zeit/Platz
(bez. $|V|$ und ℓ^{\max})

Analyse symbolischer Algorithmen – Träume und Realität

Realität bis jetzt:

- ▶ Experimentelle Untersuchungen
- ▶ Zählen der OBDD-Operationen
- ▶ Analyse auf **sehr** speziellen Graphen

Träume:

- ▶ Polynomiell bez. OBDD-Größe
- ▶ Speziell strukturierte Eingabe und Ausgabe \Rightarrow Geringe Laufzeit

Ergebnis für den APSP-Algorithmus:

Eingabe/Ausgabe hat konstante Breite \Rightarrow Polylog. Zeit/Platz
(bez. $|V|$ und ℓ^{\max})

Analyse symbolischer Algorithmen – Träume und Realität

Realität bis jetzt:

- ▶ Experimentelle Untersuchungen
- ▶ Zählen der OBDD-Operationen
- ▶ Analyse auf **sehr** speziellen Graphen

Träume:

- ▶ Polynomiell bez. OBDD-Größe
- ▶ Speziell strukturierte Eingabe **und** Ausgabe \Rightarrow Geringe Laufzeit

Ergebnis für den APSP-Algorithmus:

Eingabe/Ausgabe hat konstante Breite \Rightarrow Polylog. Zeit/Platz
(bez. $|V|$ und ℓ^{\max})

Analyse symbolischer Algorithmen – Träume und Realität

Realität bis jetzt:

- ▶ Experimentelle Untersuchungen
- ▶ Zählen der OBDD-Operationen
- ▶ Analyse auf **sehr** speziellen Graphen

Träume:

- ▶ ~~Polynomiell bez. OBDD-Größe~~ \Leftarrow **PSPACE-vollständig**
- ▶ Speziell strukturierte Eingabe **und** Ausgabe \Rightarrow Geringe Laufzeit

Ergebnis für den APSP-Algorithmus:

Eingabe/Ausgabe hat konstante Breite \Rightarrow Polylog. Zeit/Platz
(bez. $|V|$ und ℓ^{\max})

Analyse symbolischer Algorithmen – Träume und Realität

Realität bis jetzt:

- ▶ Experimentelle Untersuchungen
- ▶ Zählen der OBDD-Operationen
- ▶ Analyse auf **sehr** speziellen Graphen

Träume:

- ▶ ~~Polynomiell bez. OBDD-Größe~~ \Leftarrow **PSPACE-vollständig**
- ▶ Speziell strukturierte Eingabe **und** Ausgabe \Rightarrow Geringe Laufzeit

Ergebnis für den APSP-Algorithmus:

Eingabe/Ausgabe hat konstante Breite \Rightarrow Polylog. Zeit/Platz
(bez. $|V|$ und ℓ^{\max})

Analyse symbolischer Algorithmen – Träume und Realität

Realität bis jetzt:

- ▶ Experimentelle Untersuchungen
- ▶ Zählen der OBDD-Operationen
- ▶ Analyse auf **sehr** speziellen Graphen

Träume:

- ▶ ~~Polynomiell bez. OBDD-Größe~~ \Leftarrow **PSPACE-vollständig**
- ▶ Speziell strukturierte Eingabe **und** Ausgabe \Rightarrow Geringe Laufzeit

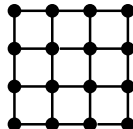
Ergebnis für den APSP-Algorithmus:

Eingabe/Ausgabe hat konstante Breite \Rightarrow Polylog. Zeit/Platz
(bez. $|V|$ und ℓ^{\max})

Breitenbeschränkte Funktionen

Definition.

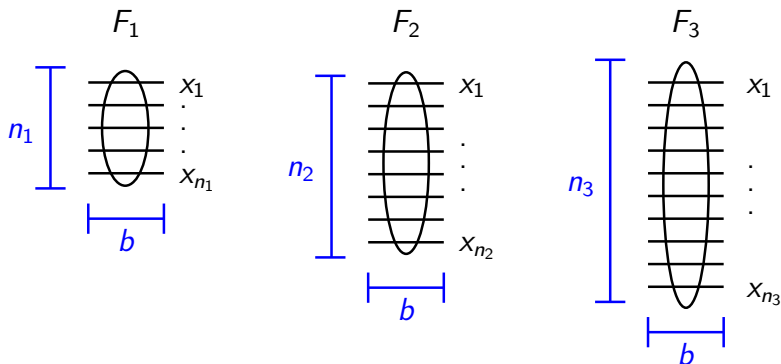
Eine Folge $f = (f_k)_k$ boolescher Fkt. heißt **breitenbeschränkt** wenn $\exists b$: Die **vollst.** OBDDs $F = (F_k)_k$ haben alle Breite $\leq b$.

 G_1  G_2  G_3 

Breitenbeschränkte Funktionen

Definition.

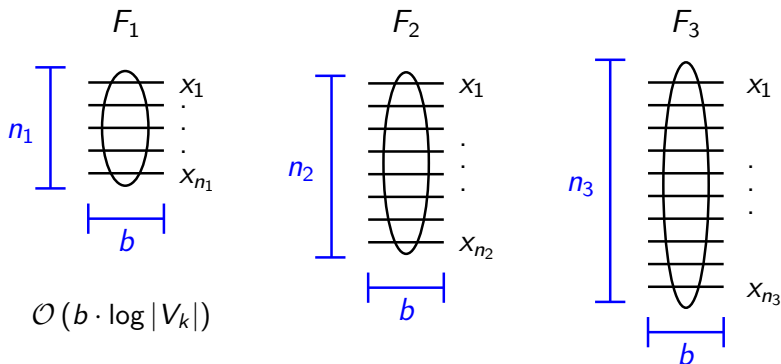
Eine Folge $f = (f_k)_k$ boolescher Fkt. heißt **breitenbeschränkt** wenn $\exists b$: Die **vollst.** OBDDs $F = (F_k)_k$ haben alle Breite $\leq b$.



Breitenbeschränkte Funktionen

Definition.

Eine Folge $f = (f_k)_k$ boolescher Fkt. heißt **breitenbeschränkt** wenn $\exists b$: Die **vollst.** OBDDs $F = (F_k)_k$ haben alle Breite $\leq b$.



Der APSP-Algo. auf breitenschränkten Funktionen

- ▶ Betrachte Graphsequenz $G = (G_k)_k$.
- ▶ $C = (C_k)_k$ und $S = (S_k)_k$ mit Breite b .

Satz.

Der symbolische APSP-Algorithmus berechnet S_k aus C_k in Zeit $\mathcal{O}(\log^3(|V_k| \cdot \ell_k^{\max}) \cdot \alpha(b))$.

Beweisskizze.

- ▶ Zeige Breite $\mathcal{O}(b)$ für $S_k^i(x, y, d) = S_k(x, y, d) \wedge (d < 2^i)$.
- ▶ Zeige Zeit $\mathcal{O}(\log(|V_k| \cdot \ell_k^{\max}) \cdot \alpha(b))$ für $S^i \rightarrow S^{i+1}$ -Schritt.

Der APSP-Algo. auf breitenschränkten Funktionen

- ▶ Betrachte Graphsequenz $G = (G_k)_k$.
- ▶ $C = (C_k)_k$ und $S = (S_k)_k$ mit Breite b .

Satz.

Der symbolische APSP-Algorithmus berechnet S_k aus C_k in Zeit $\mathcal{O}(\log^3(|V_k| \cdot \ell_k^{\max}) \cdot \alpha(b))$.

Beweisskizze.

- ▶ Zeige Breite $\mathcal{O}(b)$ für $S_k^i(x, y, d) = S_k(x, y, d) \wedge (d < 2^i)$.
- ▶ Zeige Zeit $\mathcal{O}(\log(|V_k| \cdot \ell_k^{\max}) \cdot \alpha(b))$ für $S^i \rightarrow S^{i+1}$ -Schritt.

Der APSP-Algo. auf breitenschränkten Funktionen

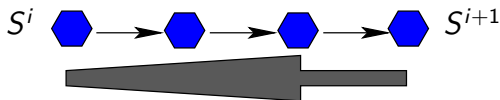
- ▶ Betrachte Graphsequenz $G = (G_k)_k$.
- ▶ $C = (C_k)_k$ und $S = (S_k)_k$ mit Breite b .

Satz.

Der symbolische APSP-Algorithmus berechnet S_k aus C_k in Zeit $\mathcal{O}(\log^3(|V_k| \cdot \ell_k^{\max}) \cdot \alpha(b))$.

Beweisskizze.

- ▶ Zeige Breite $\mathcal{O}(b)$ für $S_k^i(x, y, d) = S_k(x, y, d) \wedge (d < 2^i)$.
- ▶ Zeige Zeit $\mathcal{O}(\log(|V_k| \cdot \ell_k^{\max}) \cdot \alpha(b))$ für $S^i \rightarrow S^{i+1}$ -Schritt.



Der APSP-Algo. auf breitenschränkten Funktionen

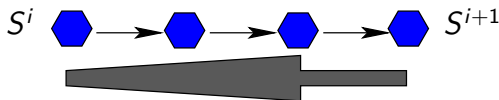
- ▶ Betrachte Graphsequenz $G = (G_k)_k$.
- ▶ $C = (C_k)_k$ und $S = (S_k)_k$ mit Breite b .

Satz.

Der symbolische APSP-Algorithmus berechnet S_k aus C_k in Zeit $\mathcal{O}(\log^3(|V_k| \cdot \ell_k^{\max}) \cdot \alpha(b))$.

Beweisskizze.

- ▶ Zeige Breite $\mathcal{O}(b)$ für $S_k^i(x, y, d) = S_k(x, y, d) \wedge (d < 2^i)$.
- ▶ Zeige Zeit $\mathcal{O}(\log(|V_k| \cdot \ell_k^{\max}) \cdot \alpha(b))$ für $S^i \rightarrow S^{i+1}$ -Schritt.



Graphen mit breitenbeschränkten Funktionen

Welche Graphen haben breitenbeschränkte Funktionen?

- ▶ Grundlegend: Paths, stars, fans, wheels, grids, cliques, bicliques, ...
- ▶ Spezielle Thresholdgraphen
- ▶ Multivariate threshold Funktionen:

$$f(x_1, \dots, x_r) = \left(\sum_{i=1}^r w_i \cdot x_i \geq T \right)$$

- ▶ Breitenbeschränkung ist abgeschlossen gegen ...
 - ▶ Vereinigungen: $V_3 := V_1 \cup V_2$.
 - ▶ Produkte: $V_3 := V_1 \times V_2$.

Graphen mit breitenbeschränkten Funktionen

Welche Graphen haben breitenbeschränkte Funktionen?

- ▶ Grundlegend: Paths, stars, fans, wheels, grids, cliques, bicliques, ...
- ▶ Spezielle Thresholdgraphen
- ▶ Multivariate threshold Funktionen:

$$f(x_1, \dots, x_r) = \left(\sum_{i=1}^r w_i \cdot x_i \geq T \right)$$

- ▶ Breitenbeschränkung ist abgeschlossen gegen ...
 - ▶ Vereinigungen: $V_3 := V_1 \cup V_2$.
 - ▶ Produkte: $V_3 := V_1 \times V_2$.

Graphen mit breitenbeschränkten Funktionen

Welche Graphen haben breitenbeschränkte Funktionen?

- ▶ Grundlegend: Paths, stars, fans, wheels, grids, cliques, bicliques, ...
- ▶ Spezielle Thresholdgraphen
- ▶ Multivariate threshold Funktionen:

$$f(x_1, \dots, x_r) = \left(\sum_{i=1}^r w_i \cdot x_i \geq T \right)$$

- ▶ Breitenbeschränkung ist abgeschlossen gegen ...
 - ▶ Vereinigungen: $V_3 := V_1 \cup V_2$.
 - ▶ Produkte: $V_3 := V_1 \times V_2$.

Graphen mit breitenbeschränkten Funktionen

Welche Graphen haben breitenbeschränkte Funktionen?

- ▶ Grundlegend: Paths, stars, fans, wheels, grids, cliques, bicliques, ...
- ▶ Spezielle Thresholdgraphen
- ▶ Multivariate threshold Funktionen:

$$f(x_1, \dots, x_r) = \left(\sum_{i=1}^r w_i \cdot x_i \geq T \right)$$

- ▶ Breitenbeschränkung ist abgeschlossen gegen ...
 - ▶ Vereinigungen: $V_3 := V_1 \cup V_2$.
 - ▶ Produkte: $V_3 := V_1 \times V_2$.

Graphen mit breitenbeschränkten Funktionen

Welche Graphen haben breitenbeschränkte Funktionen?

- ▶ Grundlegend: Paths, stars, fans, wheels, grids, cliques, bicliques, ...
- ▶ Spezielle Thresholdgraphen
- ▶ Multivariate threshold Funktionen:

$$f(x_1, \dots, x_r) = \left(\sum_{i=1}^r w_i \cdot x_i \geq T \right)$$

- ▶ Breitenbeschränkung ist abgeschlossen gegen ...
 - ▶ Vereinigungen: $V_3 := V_1 \cup V_2$.
 - ▶ Produkte: $V_3 := V_1 \times V_2$.

Graphen mit breitenbeschränkten Funktionen

Welche Graphen haben breitenbeschränkte Funktionen?

- ▶ Grundlegend: Paths, stars, fans, wheels, grids, cliques, bicliques, ...
- ▶ Spezielle Thresholdgraphen
- ▶ Multivariate threshold Funktionen:

$$f(x_1, \dots, x_r) = \left(\sum_{i=1}^r w_i \cdot x_i \geq T \right)$$

- ▶ Breitenbeschränkung ist abgeschlossen gegen ...
 - ▶ Vereinigungen: $V_3 := V_1 \cup V_2$.
 - ▶ Produkte: $V_3 := V_1 \times V_2$.

Ein alternativer Ansatz

- ▶ Ist die Einschränkung $\ell(e) > 0$ notwendig?
- ▶ Alternativ: $S^i(x, y, d) = 1 \Leftrightarrow x$ - y -Pfad mit $\leq 2^i$ Kanten.

$$S^{i+1}(x, y, d) := (\exists z, d_1, d_2) [(d_1 + d_2 = d) \\ \wedge S^i(x, z, d_1) \wedge S^i(z, y, d_2)]$$

- ▶ Gegenbeispiel $G^* = (G_k^*)_k$:
 - ▶ $C = (C_k)_k$ und $S = (S_k)_k$ sind breitenbeschränkt.
 - ▶ OBDDs $S^i = (S_k^i)_k$ sind nicht breitenbeschränkt.

Vermutung:

Beschränkte Breite **und** $\ell(e) > 0$ notwendig für polylog. Laufzeit.

Ein alternativer Ansatz

- ▶ Ist die Einschränkung $\ell(e) > 0$ notwendig?
- ▶ Alternativ: $S^i(x, y, d) = 1 \Leftrightarrow x$ - y -Pfad mit $\leq 2^i$ **Kanten**.

$$S^{i+1}(x, y, d) := (\exists z, d_1, d_2) [(d_1 + d_2 = d) \\ \wedge S^i(x, z, d_1) \wedge S^i(z, y, d_2)]$$

- ▶ Gegenbeispiel $G^* = (G_k^*)_k$:
 - ▶ $C = (C_k)_k$ und $S = (S_k)_k$ sind breitenbeschränkt.
 - ▶ OBDDs $S^i = (S_k^i)_k$ sind **nicht** breitenbeschränkt.

Vermutung:

Beschränkte Breite **und** $\ell(e) > 0$ notwendig für polylog. Laufzeit.

Ein alternativer Ansatz

- ▶ Ist die Einschränkung $\ell(e) > 0$ notwendig?
- ▶ Alternativ: $S^i(x, y, d) = 1 \Leftrightarrow x$ - y -Pfad mit $\leq 2^i$ **Kanten**.

$$S^{i+1}(x, y, d) := (\exists z, d_1, d_2) [(d_1 + d_2 = d) \\ \wedge S^i(x, z, d_1) \wedge S^i(z, y, d_2)]$$

- ▶ Gegenbeispiel $G^* = (G_k^*)_k$:
 - ▶ $C = (C_k)_k$ und $S = (S_k)_k$ sind breitenbeschränkt.
 - ▶ OBDDs $S^i = (S_k^i)_k$ sind **nicht** breitenbeschränkt.

Vermutung:

Beschränkte Breite **und** $\ell(e) > 0$ notwendig für polylog. Laufzeit.

Ein alternativer Ansatz

- ▶ Ist die Einschränkung $\ell(e) > 0$ notwendig?
- ▶ Alternativ: $S^i(x, y, d) = 1 \Leftrightarrow x$ - y -Pfad mit $\leq 2^i$ **Kanten**.

$$S^{i+1}(x, y, d) := (\exists z, d_1, d_2) [(d_1 + d_2 = d) \\ \wedge S^i(x, z, d_1) \wedge S^i(z, y, d_2)]$$

- ▶ Gegenbeispiel $G^* = (G_k^*)_k$:
 - ▶ $C = (C_k)_k$ und $S = (S_k)_k$ sind breitenbeschränkt.
 - ▶ OBDDs $S^i = (S_k^i)_k$ sind **nicht** breitenbeschränkt.

Vermutung:

Beschränkte Breite **und** $\ell(e) > 0$ notwendig für polylog. Laufzeit.

Ein alternativer Ansatz

- ▶ Ist die Einschränkung $\ell(e) > 0$ notwendig?
- ▶ Alternativ: $S^i(x, y, d) = 1 \Leftrightarrow x$ - y -Pfad mit $\leq 2^i$ **Kanten**.

$$S^{i+1}(x, y, d) := (\exists z, d_1, d_2) [(d_1 + d_2 = d) \\ \wedge S^i(x, z, d_1) \wedge S^i(z, y, d_2)]$$

- ▶ Gegenbeispiel $G^* = (G_k^*)_k$:
 - ▶ $C = (C_k)_k$ und $S = (S_k)_k$ sind breitenbeschränkt.
 - ▶ OBDDs $S^i = (S_k^i)_k$ sind **nicht** breitenbeschränkt.

Vermutung:

Beschränkte Breite **und** $\ell(e) > 0$ notwendig für polylog. Laufzeit.

Ein alternativer Ansatz

- ▶ Ist die Einschränkung $\ell(e) > 0$ notwendig?
- ▶ Alternativ: $S^i(x, y, d) = 1 \Leftrightarrow x$ - y -Pfad mit $\leq 2^i$ **Kanten**.

$$S^{i+1}(x, y, d) := (\exists z, d_1, d_2) [(d_1 + d_2 = d) \\ \wedge S^i(x, z, d_1) \wedge S^i(z, y, d_2)]$$

- ▶ Gegenbeispiel $G^* = (G_k^*)_k$:
 - ▶ $C = (C_k)_k$ und $S = (S_k)_k$ sind breitenbeschränkt.
 - ▶ OBDDs $S^i = (S_k^i)_k$ sind **nicht** breitenbeschränkt.

Vermutung:

Beschränkte Breite **und** $\ell(e) > 0$ notwendig für polylog. Laufzeit.

Übersicht

Einführung

Ein symbolischer APSP-Algorithmus (WG'04)
Breitenbeschränkte Funktionen

Symbolische SSSP-Algorithmen (WEA'04)

Zusammenfassung

Geänderte Zielsetzung

Bisher:

Polylog. auf speziellen Graphen – ineffizient in den meisten Fällen

Nun:

Auf den meisten Graphen effizient (linear bez. OBDD-Eingabe)

- ▶ Ansatz: Umsetzung populärer expliziter SSSP-Algos.
- ▶ Sequenzielles Vorgehen $\Rightarrow \Omega(|V|)$ OBDD-Operationen
- ▶ Average-case Analyse (momentan) nicht realistisch
- ▶ Ausweg: Experimentelle Analyse auf zufälligen Graphen

Geänderte Zielsetzung

Bisher:

Polylog. auf speziellen Graphen – ineffizient in den meisten Fällen

Nun:

Auf den meisten Graphen effizient (linear bez. OBDD-Eingabe)

- ▶ Ansatz: Umsetzung populärer expliziter SSSP-Algos.
 - ▶ Dijkstra, Bellman-Ford
- ▶ Sequenzielles Vorgehen $\Rightarrow \Omega(|V|)$ OBDD-Operationen
- ▶ Average-case Analyse (momentan) nicht realistisch
- ▶ Ausweg: Experimentelle Analyse auf zufälligen Graphen

Geänderte Zielsetzung

Bisher:

Polylog. auf speziellen Graphen – ineffizient in den meisten Fällen

Nun:

Auf den meisten Graphen effizient (linear bez. OBDD-Eingabe)

- ▶ Ansatz: Umsetzung populärer expliziter SSSP-Algos.
 - ▶ Dijkstra, Bellman-Ford
- ▶ Sequenzielles Vorgehen $\Rightarrow \Omega(|V|)$ OBDD-Operationen
- ▶ Average-case Analyse (momentan) nicht realistisch
- ▶ Ausweg: Experimentelle Analyse auf zufälligen Graphen

Geänderte Zielsetzung

Bisher:

Polylog. auf speziellen Graphen – ineffizient in den meisten Fällen

Nun:

Auf den meisten Graphen effizient (linear bez. OBDD-Eingabe)

- ▶ Ansatz: Umsetzung populärer expliziter SSSP-Algos.
 - ▶ Dijkstra, Bellman-Ford
- ▶ Sequenzielles Vorgehen $\Rightarrow \Omega(|V|)$ OBDD-Operationen
- ▶ Average-case Analyse (momentan) nicht realistisch
- ▶ Ausweg: Experimentelle Analyse auf zufälligen Graphen

Geänderte Zielsetzung

Bisher:

Polylog. auf speziellen Graphen – ineffizient in den meisten Fällen

Nun:

Auf den meisten Graphen effizient (linear bez. OBDD-Eingabe)

- ▶ Ansatz: Umsetzung populärer expliziter SSSP-Algos.
 - ▶ Dijkstra, Bellman-Ford
- ▶ Sequenzielles Vorgehen $\Rightarrow \Omega(|V|)$ OBDD-Operationen
- ▶ Average-case Analyse (momentan) nicht realistisch
- ▶ Ausweg: Experimentelle Analyse auf zufälligen Graphen

Geänderte Zielsetzung

Bisher:

Polylog. auf speziellen Graphen – ineffizient in den meisten Fällen

Nun:

Auf den meisten Graphen effizient (linear bez. OBDD-Eingabe)

- ▶ Ansatz: Umsetzung populärer expliziter SSSP-Algos.
 - ▶ Dijkstra, Bellman-Ford
- ▶ Sequenzielles Vorgehen $\Rightarrow \Omega(|V|)$ OBDD-Operationen
- ▶ Average-case Analyse (momentan) nicht realistisch
- ▶ Ausweg: Experimentelle Analyse auf zufälligen Graphen

Geänderte Zielsetzung

Bisher:

Polylog. auf speziellen Graphen – ineffizient in den meisten Fällen

Nun:

Auf den meisten Graphen effizient (linear bez. OBDD-Eingabe)

- ▶ Ansatz: Umsetzung populärer expliziter SSSP-Algos.
 - ▶ Dijkstra, Bellman-Ford
- ▶ Sequenzielles Vorgehen $\Rightarrow \Omega(|V|)$ OBDD-Operationen
- ▶ Average-case Analyse (momentan) nicht realistisch
- ▶ Ausweg: Experimentelle Analyse auf zufälligen Graphen

Experimentelle Ergebnisse auf Zufallsgraphen (1)

- ▶ Was erwarten wir von beiden Algorithmen?
- ▶ Dijkstra ...
 - ▶ ...arbeitet Knoten für Knoten \Rightarrow viele Operationen
 - ▶ ...strukturierte Mengen \Rightarrow effiziente Operationen
- ▶ Bellman-Ford ...
 - ▶ ...aktualisiert Kanten parallel \Rightarrow wenige Operationen
 - ▶ ...unstrukturierte Mengen $RELAX(x, y, d) \Rightarrow$ aufwendige Operationen
- ▶ Experimentelle Beobachtungen:
 - ▶ Dijkstra benötigt linearen Platz
 - ▶ Bellman-Ford benötigt lineare Zeit

Experimentelle Ergebnisse auf Zufallsgraphen (1)

- ▶ Was erwarten wir von beiden Algorithmen?
- ▶ Dijkstra ...
 - ▶ ...arbeitet Knoten für Knoten \Rightarrow viele Operationen
 - ▶ ...strukturierte Mengen \Rightarrow effiziente Operationen
- ▶ Bellman-Ford ...
 - ▶ ...aktualisiert Kanten parallel \Rightarrow wenige Operationen
 - ▶ ...unstrukturierte Mengen $RELAX(x, y, d) \Rightarrow$ aufwendige Operationen
- ▶ Experimentelle Beobachtungen:
 - ▶ Dijkstra benötigt linearen Platz
 - ▶ Bellman-Ford benötigt lineare Zeit

Experimentelle Ergebnisse auf Zufallsgraphen (1)

- ▶ Was erwarten wir von beiden Algorithmen?
- ▶ Dijkstra ...
 - ▶ ...arbeitet Knoten für Knoten \Rightarrow viele Operationen
 - ▶ ...strukturierte Mengen \Rightarrow effiziente Operationen
- ▶ Bellman-Ford ...
 - ▶ ...aktualisiert Kanten parallel \Rightarrow wenige Operationen
 - ▶ ...unstrukturierte Mengen $RELAX(x, y, d) \Rightarrow$ aufwendige Operationen
- ▶ Experimentelle Beobachtungen:
 - ▶ Dijkstra benötigt linearen Platz
 - ▶ Bellman-Ford benötigt lineare Zeit

Experimentelle Ergebnisse auf Zufallsgraphen (1)

- ▶ Was erwarten wir von beiden Algorithmen?
- ▶ Dijkstra ...
 - ▶ ...arbeitet Knoten für Knoten \Rightarrow viele Operationen
 - ▶ ...strukturierte Mengen \Rightarrow effiziente Operationen
- ▶ Bellman-Ford ...
 - ▶ ...aktualisiert Kanten parallel \Rightarrow wenige Operationen
 - ▶ ...unstrukturierte Mengen $RELAX(x, y, d) \Rightarrow$ aufwendige Operationen
- ▶ Experimentelle Beobachtungen:
 - ▶ Dijkstra benötigt linearen Platz
 - ▶ Bellman-Ford benötigt lineare Zeit

Experimentelle Ergebnisse auf Zufallsgraphen (1)

- ▶ Was erwarten wir von beiden Algorithmen?
- ▶ Dijkstra ...
 - ▶ ...arbeitet Knoten für Knoten \Rightarrow viele Operationen
 - ▶ ...strukturierte Mengen \Rightarrow effiziente Operationen
- ▶ Bellman-Ford ...
 - ▶ ...aktualisiert Kanten parallel \Rightarrow wenige Operationen
 - ▶ ...unstrukturierte Mengen $RELAX(x, y, d) \Rightarrow$ aufwendige Operationen
- ▶ Experimentelle Beobachtungen:
 - ▶ Dijkstra benötigt linearen Platz
 - ▶ Bellman-Ford benötigt lineare Zeit

Experimentelle Ergebnisse auf Zufallsgraphen (1)

- ▶ Was erwarten wir von beiden Algorithmen?
- ▶ Dijkstra ...
 - ▶ ...arbeitet Knoten für Knoten \Rightarrow viele Operationen
 - ▶ ...strukturierte Mengen \Rightarrow effiziente Operationen
- ▶ Bellman-Ford ...
 - ▶ ...aktualisiert Kanten parallel \Rightarrow wenige Operationen
 - ▶ ...unstrukturierte Mengen $RELAX(x, y, d) \Rightarrow$ aufwendige Operationen
- ▶ Experimentelle Beobachtungen:
 - ▶ Dijkstra benötigt linearen Platz
 - ▶ Bellman-Ford benötigt lineare Zeit

Experimentelle Ergebnisse auf Zufallsgraphen (1)

- ▶ Was erwarten wir von beiden Algorithmen?
- ▶ Dijkstra ...
 - ▶ ...arbeitet Knoten für Knoten \Rightarrow viele Operationen
 - ▶ ...strukturierte Mengen \Rightarrow effiziente Operationen
- ▶ Bellman-Ford ...
 - ▶ ...aktualisiert Kanten parallel \Rightarrow wenige Operationen
 - ▶ ...unstrukturierte Mengen $RELAX(x, y, d) \Rightarrow$ aufwendige Operationen
- ▶ Experimentelle Beobachtungen:
 - ▶ Dijkstra benötigt linearen Platz
 - ▶ Bellman-Ford benötigt lineare Zeit

Experimentelle Ergebnisse auf Zufallsgraphen (1)

- ▶ Was erwarten wir von beiden Algorithmen?
- ▶ Dijkstra ...
 - ▶ ...arbeitet Knoten für Knoten \Rightarrow viele Operationen
 - ▶ ...strukturierte Mengen \Rightarrow effiziente Operationen
- ▶ Bellman-Ford ...
 - ▶ ...aktualisiert Kanten parallel \Rightarrow wenige Operationen
 - ▶ ...unstrukturierte Mengen $RELAX(x, y, d) \Rightarrow$ aufwendige Operationen
- ▶ Experimentelle Beobachtungen:
 - ▶ Dijkstra benötigt linearen Platz
 - ▶ Bellman-Ford benötigt lineare Zeit

Experimentelle Ergebnisse auf Zufallsgraphen (1)

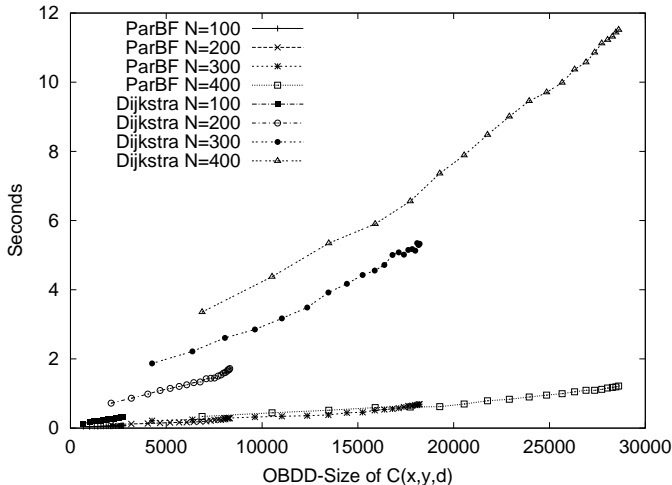
- ▶ Was erwarten wir von beiden Algorithmen?
- ▶ Dijkstra ...
 - ▶ ...arbeitet Knoten für Knoten \Rightarrow viele Operationen
 - ▶ ...strukturierte Mengen \Rightarrow effiziente Operationen
- ▶ Bellman-Ford ...
 - ▶ ...aktualisiert Kanten parallel \Rightarrow wenige Operationen
 - ▶ ...unstrukturierte Mengen $RELAX(x, y, d) \Rightarrow$ aufwendige Operationen
- ▶ Experimentelle Beobachtungen:
 - ▶ Dijkstra benötigt linearen Platz
 - ▶ Bellman-Ford benötigt lineare Zeit

Experimentelle Ergebnisse auf Zufallsgraphen (1)

- ▶ Was erwarten wir von beiden Algorithmen?
- ▶ Dijkstra ...
 - ▶ ...arbeitet Knoten für Knoten \Rightarrow viele Operationen
 - ▶ ...strukturierte Mengen \Rightarrow effiziente Operationen
- ▶ Bellman-Ford ...
 - ▶ ...aktualisiert Kanten parallel \Rightarrow wenige Operationen
 - ▶ ...unstrukturierte Mengen $RELAX(x, y, d) \Rightarrow$ aufwendige Operationen
- ▶ Experimentelle Beobachtungen:
 - ▶ Dijkstra benötigt linearen Platz
 - ▶ Bellman-Ford benötigt lineare Zeit

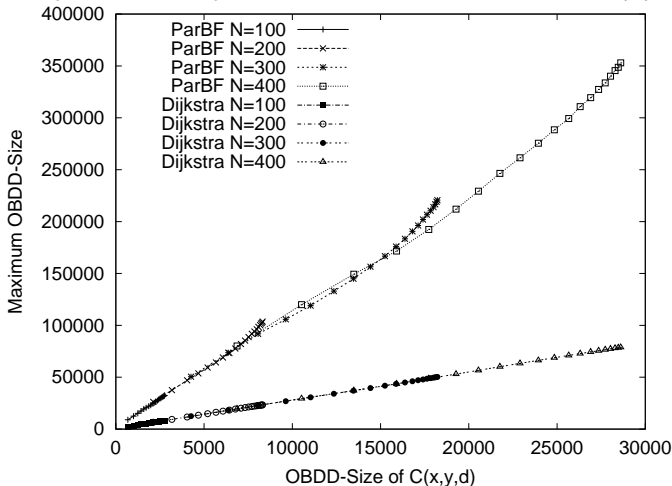
Experimentelle Ergebnisse auf Zufallsgraphen (2)

- ▶ $V \in \{100, \dots, 400\}$, Dichten $0.05, \dots, 1$, Gewicht $\ell(e) = 1$



Experimentelle Ergebnisse auf Zufallsgraphen (2)

- ▶ $V \in \{100, \dots, 400\}$, Dichten $0.05, \dots, 1$, Gewicht $\ell(e) = 1$



Zusammenfassung

- ▶ Symbolische Algorithmen arbeiten auf OBDDs von Graphen.
- ▶ Heuristiken für große, strukturierte Eingaben
- ▶ Der Symbolischer APSP-Algorithmus:
 - ▶ $O(\log^2(|V| \cdot \ell^{\max}))$ OBDD-Operationen
 - ▶ Beschränkte Breite \Rightarrow Zeit/Platz $O(\log^3(|V| \cdot \ell^{\max}))$
- ▶ Symbolische SSSP-Algorithmen:
 - ▶ Sind expliziten Algorithmen nachempfunden.
 - ▶ Sequenzielles Verhalten $\Rightarrow \Omega(|V|)$ Operationen
 - ▶ Lineares Verhalten auf OBDDs zufälliger Graphen

Zusammenfassung

- ▶ Symbolische Algorithmen arbeiten auf OBDDs von Graphen.
- ▶ Heuristiken für große, strukturierte Eingaben
- ▶ Der Symbolischer APSP-Algorithmus:
 - ▶ $O(\log^2(|V| \cdot \ell^{\max}))$ OBDD-Operationen
 - ▶ Beschränkte Breite \Rightarrow Zeit/Platz $O(\log^3(|V| \cdot \ell^{\max}))$
- ▶ Symbolische SSSP-Algorithmen:
 - ▶ Sind expliziten Algorithmen nachempfunden.
 - ▶ Sequenzielles Verhalten $\Rightarrow \Omega(|V|)$ Operationen
 - ▶ Lineares Verhalten auf OBDDs zufälliger Graphen

Zusammenfassung

- ▶ Symbolische Algorithmen arbeiten auf OBDDs von Graphen.
- ▶ Heuristiken für große, strukturierte Eingaben
- ▶ Der Symbolischer APSP-Algorithmus:
 - ▶ $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ OBDD-Operationen
 - ▶ Beschränkte Breite \Rightarrow Zeit/Platz $\mathcal{O}(\log^3(|V| \cdot \ell^{\max}))$
- ▶ Symbolische SSSP-Algorithmen:
 - ▶ Sind expliziten Algorithmen nachempfunden.
 - ▶ Sequenzielles Verhalten $\Rightarrow \Omega(|V|)$ Operationen
 - ▶ Lineares Verhalten auf OBDDs zufälliger Graphen

Zusammenfassung

- ▶ Symbolische Algorithmen arbeiten auf OBDDs von Graphen.
- ▶ Heuristiken für große, strukturierte Eingaben
- ▶ Der Symbolischer APSP-Algorithmus:
 - ▶ $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ OBDD-Operationen
 - ▶ Beschränkte Breite \Rightarrow Zeit/Platz $\mathcal{O}(\log^3(|V| \cdot \ell^{\max}))$
- ▶ Symbolische SSSP-Algorithmen:
 - ▶ Sind expliziten Algorithmen nachempfunden.
 - ▶ Sequenzielles Verhalten $\Rightarrow \Omega(|V|)$ Operationen
 - ▶ Lineares Verhalten auf OBDDs zufälliger Graphen

Zusammenfassung

- ▶ Symbolische Algorithmen arbeiten auf OBDDs von Graphen.
- ▶ Heuristiken für große, strukturierte Eingaben
- ▶ Der Symbolischer APSP-Algorithmus:
 - ▶ $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ OBDD-Operationen
 - ▶ Beschränkte Breite \Rightarrow Zeit/Platz $\mathcal{O}(\log^3(|V| \cdot \ell^{\max}))$
- ▶ Symbolische SSSP-Algorithmen:
 - ▶ Sind expliziten Algorithmen nachempfunden.
 - ▶ Sequenzielles Verhalten $\Rightarrow \Omega(|V|)$ Operationen
 - ▶ Lineares Verhalten auf OBDDs zufälliger Graphen

Zusammenfassung

- ▶ Symbolische Algorithmen arbeiten auf OBDDs von Graphen.
- ▶ Heuristiken für große, strukturierte Eingaben
- ▶ Der Symbolischer APSP-Algorithmus:
 - ▶ $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ OBDD-Operationen
 - ▶ Beschränkte Breite \Rightarrow Zeit/Platz $\mathcal{O}(\log^3(|V| \cdot \ell^{\max}))$
- ▶ Symbolische SSSP-Algorithmen:
 - ▶ Sind expliziten Algorithmen nachempfunden.
 - ▶ Sequenzielles Verhalten $\Rightarrow \Omega(|V|)$ Operationen
 - ▶ Lineares Verhalten auf OBDDs zufälliger Graphen

Zusammenfassung

- ▶ Symbolische Algorithmen arbeiten auf OBDDs von Graphen.
- ▶ Heuristiken für große, strukturierte Eingaben
- ▶ Der Symbolischer APSP-Algorithmus:
 - ▶ $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ OBDD-Operationen
 - ▶ Beschränkte Breite \Rightarrow Zeit/Platz $\mathcal{O}(\log^3(|V| \cdot \ell^{\max}))$
- ▶ Symbolische SSSP-Algorithmen:
 - ▶ Sind expliziten Algorithmen nachempfunden.
 - ▶ Sequenzielles Verhalten $\Rightarrow \Omega(|V|)$ Operationen
 - ▶ Lineares Verhalten auf OBDDs zufälliger Graphen

Zusammenfassung

- ▶ Symbolische Algorithmen arbeiten auf OBDDs von Graphen.
- ▶ Heuristiken für große, strukturierte Eingaben
- ▶ Der Symbolischer APSP-Algorithmus:
 - ▶ $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ OBDD-Operationen
 - ▶ Beschränkte Breite \Rightarrow Zeit/Platz $\mathcal{O}(\log^3(|V| \cdot \ell^{\max}))$
- ▶ Symbolische SSSP-Algorithmen:
 - ▶ Sind expliziten Algorithmen nachempfunden.
 - ▶ Sequenzielles Verhalten $\Rightarrow \Omega(|V|)$ Operationen
 - ▶ Lineares Verhalten auf OBDDs zufälliger Graphen

Zusammenfassung

- ▶ Symbolische Algorithmen arbeiten auf OBDDs von Graphen.
- ▶ Heuristiken für große, strukturierte Eingaben
- ▶ Der Symbolischer APSP-Algorithmus:
 - ▶ $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ OBDD-Operationen
 - ▶ Beschränkte Breite \Rightarrow Zeit/Platz $\mathcal{O}(\log^3(|V| \cdot \ell^{\max}))$
- ▶ Symbolische SSSP-Algorithmen:
 - ▶ Sind expliziten Algorithmen nachempfunden.
 - ▶ Sequenzielles Verhalten $\Rightarrow \Omega(|V|)$ Operationen
 - ▶ Lineares Verhalten auf OBDDs zufälliger Graphen

Danke fürs Zuhören!

