

Symbolische Berechnung kürzester Wege

Daniel Sawitzki

Lehrstuhl 2 – Diplomanden-/Doktoranden-Seminar
13. Juli 2004

Übersicht

Einführung

Ein symbolischer APSP-Algorithmus

Analyse auf breitenbeschränkten Funktionen

Anmerkungen

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

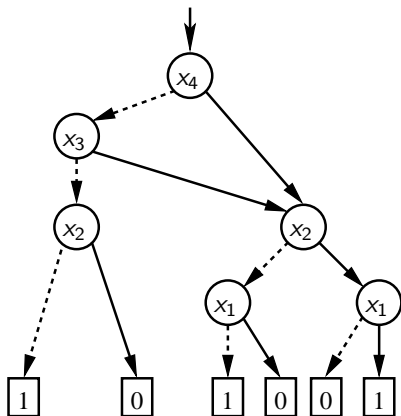
- ▶ Anwendungen erzeugen große Graphen (VLSI, Verkehr, WWW, ...)
 - ▶ \Rightarrow Konflikte mit interner Speichergröße
 - ▶ \Rightarrow Sogar effiziente Algos. nicht anwendbar
- ▶ Beobachtung: Die Graphen sind groß, aber auch **strukturiert**.
- ▶ Heuristischer Ansatz aus VLSI: **Symbolische Darstellung**
 - ▶ 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 .

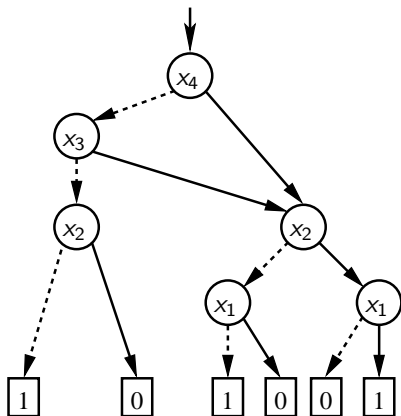
Ordered Binary Decision Diagrams (OBDDs)

- ▶ 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. $\pi \in \Sigma_n$.



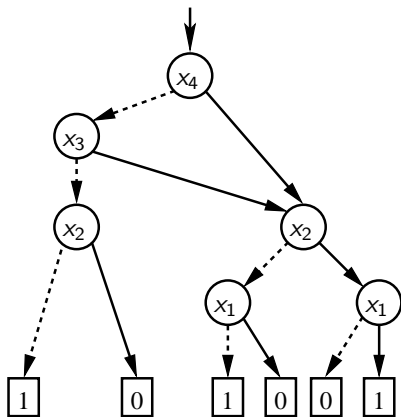
Ordered Binary Decision Diagrams (OBDDs)

- ▶ 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. $\pi \in \Sigma_n$.



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. $\pi \in \Sigma_n$.

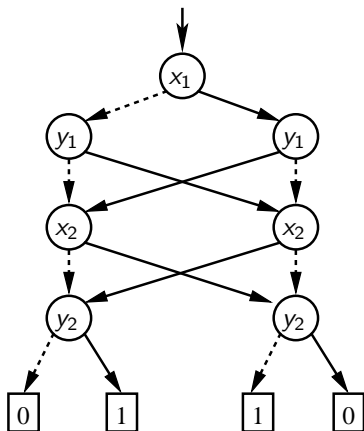
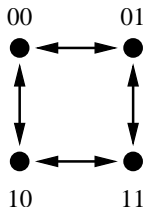


Beispielgraph mit OBDD

- Stelle $G = (V, E)$ durch
 $C: \{0, 1\}^{2n} \rightarrow \{0, 1\}$ dar mit

$$C(x, y) = 1 \Leftrightarrow (x, y) \in E.$$

- Beispiel:
 $C(x, y) := (x_1 = y_1) \oplus (x_2 = y_2)$

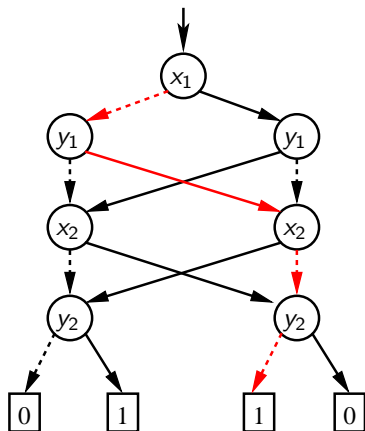
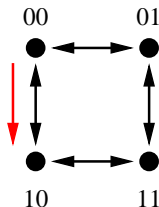


Beispielgraph mit OBDD

- Stelle $G = (V, E)$ durch
 $C: \{0, 1\}^{2n} \rightarrow \{0, 1\}$ dar mit

$$C(x, y) = 1 \Leftrightarrow (x, y) \in E.$$

- Beispiel:
 $C(x, y) := (x_1 = y_1) \oplus (x_2 = y_2)$



Eigenschaften symbolischer OBDD-Darstellungen

- ▶ $f: \{0, 1\}^n \rightarrow \{0, 1\}$ hat OBDD-Größe $(2 + o(1))2^n/n$.
- ▶ Kodiere Knoten $V = \{v_0, \dots, v_{|V|-1}\}$ durch $n := \lceil \log |V| \rceil$ Bits.
- ▶ $\Rightarrow C: \{0, 1\}^{2^n} \rightarrow \{0, 1\}$ hat OBDD-Größe $\mathcal{O}(|V|^2 / \log |V|)$.
- ▶ Ebenfalls klar: Obere Schranke von $\mathcal{O}(|E| \log |V|)$
- ▶ OBDD-Größe hängt von π und Knotennummerierung ab.
- ▶ Üblich: **Gemischte Var.-ordnung**

$$\pi = (x_{i_0}, y_{i_0}, \dots, x_{i_{n-1}}, y_{i_{n-1}})$$

Eigenschaften symbolischer OBDD-Darstellungen

- ▶ $f: \{0, 1\}^n \rightarrow \{0, 1\}$ hat OBDD-Größe $(2 + o(1))2^n/n$.
- ▶ Kodiere Knoten $V = \{v_0, \dots, v_{|V|-1}\}$ durch $n := \lceil \log |V| \rceil$ Bits.
- ▶ $\Rightarrow C: \{0, 1\}^{2^n} \rightarrow \{0, 1\}$ hat OBDD-Größe $\mathcal{O}(|V|^2 / \log |V|)$.
- ▶ Ebenfalls klar: Obere Schranke von $\mathcal{O}(|E| \log |V|)$
- ▶ OBDD-Größe hängt von π und Knotennummerierung ab.
- ▶ Üblich: **Gemischte Var.-ordnung**

$$\pi = (x_{i_0}, y_{i_0}, \dots, x_{i_{n-1}}, y_{i_{n-1}})$$

Eigenschaften symbolischer OBDD-Darstellungen

- ▶ $f: \{0, 1\}^n \rightarrow \{0, 1\}$ hat OBDD-Größe $(2 + o(1))2^n/n$.
- ▶ Kodiere Knoten $V = \{v_0, \dots, v_{|V|-1}\}$ durch $n := \lceil \log |V| \rceil$ Bits.
- ▶ $\Rightarrow C: \{0, 1\}^{2^n} \rightarrow \{0, 1\}$ hat OBDD-Größe $\mathcal{O}(|V|^2 / \log |V|)$.
- ▶ Ebenfalls klar: Obere Schranke von $\mathcal{O}(|E| \log |V|)$
- ▶ OBDD-Größe hängt von π und Knotennummerierung ab.
- ▶ Üblich: **Gemischte Var.-ordnung**

$$\pi = (x_{i_0}, y_{i_0}, \dots, x_{i_{n-1}}, y_{i_{n-1}})$$

Eigenschaften symbolischer OBDD-Darstellungen

- ▶ $f: \{0, 1\}^n \rightarrow \{0, 1\}$ hat OBDD-Größe $(2 + o(1)) 2^n/n$.
- ▶ Kodiere Knoten $V = \{v_0, \dots, v_{|V|-1}\}$ durch $n := \lceil \log |V| \rceil$ Bits.
- ▶ $\Rightarrow C: \{0, 1\}^{2^n} \rightarrow \{0, 1\}$ hat OBDD-Größe $\mathcal{O}(|V|^2 / \log |V|)$.
- ▶ Ebenfalls klar: Obere Schranke von $\mathcal{O}(|E| \log |V|)$
- ▶ OBDD-Größe hängt von π und Knotennummerierung ab.
- ▶ Üblich: **Gemischte Var.-ordnung**

$$\pi = (x_{i_0}, y_{i_0}, \dots, x_{i_{n-1}}, y_{i_{n-1}})$$

Eigenschaften symbolischer OBDD-Darstellungen

- ▶ $f: \{0, 1\}^n \rightarrow \{0, 1\}$ hat OBDD-Größe $(2 + o(1))2^n/n$.
- ▶ Kodiere Knoten $V = \{v_0, \dots, v_{|V|-1}\}$ durch $n := \lceil \log |V| \rceil$ Bits.
- ▶ $\Rightarrow C: \{0, 1\}^{2^n} \rightarrow \{0, 1\}$ hat OBDD-Größe $\mathcal{O}(|V|^2 / \log |V|)$.
- ▶ Ebenfalls klar: Obere Schranke von $\mathcal{O}(|E| \log |V|)$
- ▶ OBDD-Größe hängt von π und Knotennummerierung ab.
- ▶ Üblich: **Gemischte Var.-ordnung**

$$\pi = (x_{i_0}, y_{i_0}, \dots, x_{i_{n-1}}, y_{i_{n-1}})$$

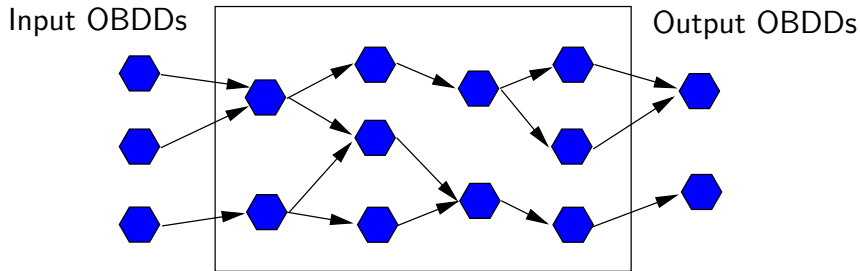
Eigenschaften symbolischer OBDD-Darstellungen

- ▶ $f: \{0, 1\}^n \rightarrow \{0, 1\}$ hat OBDD-Größe $(2 + o(1)) 2^n/n$.
- ▶ Kodiere Knoten $V = \{v_0, \dots, v_{|V|-1}\}$ durch $n := \lceil \log |V| \rceil$ Bits.
- ▶ $\Rightarrow C: \{0, 1\}^{2^n} \rightarrow \{0, 1\}$ hat OBDD-Größe $\mathcal{O}(|V|^2 / \log |V|)$.
- ▶ Ebenfalls klar: Obere Schranke von $\mathcal{O}(|E| \log |V|)$
- ▶ OBDD-Größe hängt von π und Knotennummerierung ab.
- ▶ Üblich: **Gemischte Var.-ordnung**

$$\pi = (x_{i_0}, y_{i_0}, \dots, x_{i_{n-1}}, y_{i_{n-1}})$$

Symbolische Graphalgorithmen

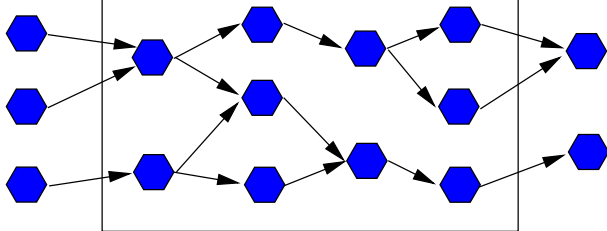
- ▶ Löse Graphprobleme durch effiziente OBDD-Operationen.
- ▶ Verarbeitete viele Knoten/Kanten durch wenige Operationen.
- ▶ Vermeide sequenzielle Behandlung von Knoten/Kanten.
- ▶ Exp. blow-up durch $\mathcal{O}(\log |V|)$ Operationen möglich.



Symbolische Graphalgorithmen

- ▶ Löse Graphprobleme durch effiziente OBDD-Operationen.
- ▶ Verarbeitete viele Knoten/Kanten durch wenige Operationen.
- ▶ Vermeide sequenzielle Behandlung von Knoten/Kanten.
- ▶ Exp. blow-up durch $\mathcal{O}(\log |V|)$ Operationen möglich.

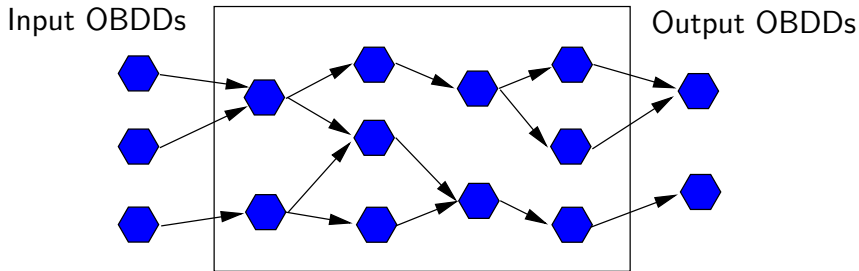
Input OBDDs



Output OBDDs

Symbolische Graphalgorithmen

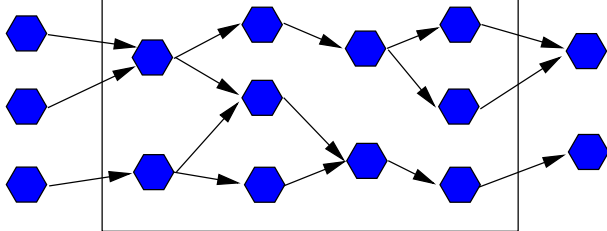
- ▶ Löse Graphprobleme durch effiziente OBDD-Operationen.
- ▶ Verarbeitete viele Knoten/Kanten durch wenige Operationen.
- ▶ Vermeide sequenzielle Behandlung von Knoten/Kanten.
- ▶ Exp. blow-up durch $\mathcal{O}(\log |V|)$ Operationen möglich.



Symbolische Graphalgorithmen

- ▶ Löse Graphprobleme durch effiziente OBDD-Operationen.
- ▶ Verarbeitete viele Knoten/Kanten durch wenige Operationen.
- ▶ Vermeide sequenzielle Behandlung von Knoten/Kanten.
- ▶ Exp. blow-up durch $\mathcal{O}(\log |V|)$ Operationen möglich.

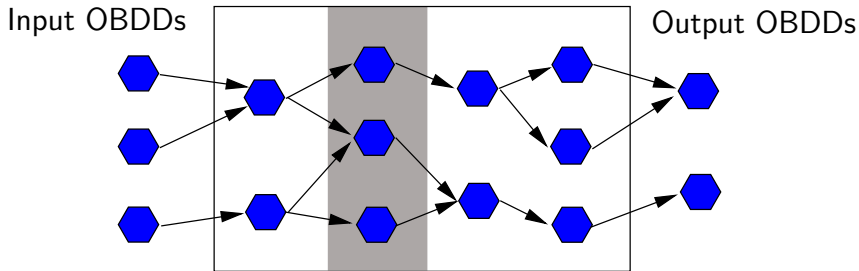
Input OBDDs



Output OBDDs

Symbolische Graphalgorithmen

- ▶ Löse Graphprobleme durch effiziente OBDD-Operationen.
- ▶ Verarbeitete viele Knoten/Kanten durch wenige Operationen.
- ▶ Vermeide sequenzielle Behandlung von Knoten/Kanten.
- ▶ Exp. blow-up durch $\mathcal{O}(\log |V|)$ Operationen möglich.



Effiziente OBDD-Operationen

- ▶ **Synthese:** $F_1(x) \otimes F_2(x)$, $\otimes \in \{\wedge, \vee, \oplus, =, \dots\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| \cdot |F_2|)$
- ▶ **Quantifizierung:** $(\exists x_i) F(x)$, $(\forall x_i) F(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F|^2)$
- ▶ **Äquivalenztest:** $F_1(x) \equiv F_2(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| + |F_2|)$
- ▶ **Var.-ersetzung:** $F(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_{n-1})$, $a \in \{0, 1\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F|)$
- ▶ **Erfüllbarkeit:** $F(x) \neq 0$
 - ▶ Zeit/Platz $\mathcal{O}(1)$
- ▶ **Argumenttausch:** $F_2(x, y) := F_1(y, x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1|)$

Effiziente OBDD-Operationen

- ▶ **Synthese:** $F_1(x) \otimes F_2(x)$, $\otimes \in \{\wedge, \vee, \oplus, =, \dots\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| \cdot |F_2|)$
- ▶ **Quantifizierung:** $(\exists x_j) F(x)$, $(\forall x_j) F(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F|^2)$
- ▶ **Äquivalenztest:** $F_1(x) \equiv F_2(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| + |F_2|)$
- ▶ **Var.-ersetzung:** $F(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_{n-1})$, $a \in \{0, 1\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F|)$
- ▶ **Erfüllbarkeit:** $F(x) \neq 0$
 - ▶ Zeit/Platz $\mathcal{O}(1)$
- ▶ **Argumenttausch:** $F_2(x, y) := F_1(y, x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1|)$

Effiziente OBDD-Operationen

- ▶ **Synthese:** $F_1(x) \otimes F_2(x)$, $\otimes \in \{\wedge, \vee, \oplus, =, \dots\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| \cdot |F_2|)$
- ▶ **Quantifizierung:** $(\exists x_i) F(x)$, $(\forall x_i) F(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F|^2)$
- ▶ **Äquivalenztest:** $F_1(x) \equiv F_2(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| + |F_2|)$
- ▶ **Var.-ersetzung:** $F(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_{n-1})$, $a \in \{0, 1\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F|)$
- ▶ **Erfüllbarkeit:** $F(x) \neq 0$
 - ▶ Zeit/Platz $\mathcal{O}(1)$
- ▶ **Argumenttausch:** $F_2(x, y) := F_1(y, x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1|)$

Effiziente OBDD-Operationen

- ▶ **Synthese:** $F_1(x) \otimes F_2(x)$, $\otimes \in \{\wedge, \vee, \oplus, =, \dots\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| \cdot |F_2|)$
- ▶ **Quantifizierung:** $(\exists x_i) F(x)$, $(\forall x_i) F(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F|^2)$
- ▶ **Äquivalenztest:** $F_1(x) \equiv F_2(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| + |F_2|)$
- ▶ **Var.-ersetzung:** $F(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_{n-1})$, $a \in \{0, 1\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F|)$
- ▶ **Erfüllbarkeit:** $F(x) \neq 0$
 - ▶ Zeit/Platz $\mathcal{O}(1)$
- ▶ **Argumenttausch:** $F_2(x, y) := F_1(y, x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1|)$

Effiziente OBDD-Operationen

- ▶ **Synthese:** $F_1(x) \otimes F_2(x)$, $\otimes \in \{\wedge, \vee, \oplus, =, \dots\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| \cdot |F_2|)$
- ▶ **Quantifizierung:** $(\exists x_i) F(x)$, $(\forall x_i) F(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F|^2)$
- ▶ **Äquivalenztest:** $F_1(x) \equiv F_2(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| + |F_2|)$
- ▶ **Var.-ersetzung:** $F(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_{n-1})$, $a \in \{0, 1\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F|)$
- ▶ **Erfüllbarkeit:** $F(x) \neq 0$
 - ▶ Zeit/Platz $\mathcal{O}(1)$
- ▶ **Argumenttausch:** $F_2(x, y) := F_1(y, x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1|)$

Effiziente OBDD-Operationen

- ▶ **Synthese:** $F_1(x) \otimes F_2(x)$, $\otimes \in \{\wedge, \vee, \oplus, =, \dots\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| \cdot |F_2|)$
- ▶ **Quantifizierung:** $(\exists x_i) F(x)$, $(\forall x_i) F(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F|^2)$
- ▶ **Äquivalenztest:** $F_1(x) \equiv F_2(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| + |F_2|)$
- ▶ **Var.-ersetzung:** $F(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_{n-1})$, $a \in \{0, 1\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F|)$
- ▶ **Erfüllbarkeit:** $F(x) \neq 0$
 - ▶ Zeit/Platz $\mathcal{O}(1)$
- ▶ **Argumenttausch:** $F_2(x, y) := F_1(y, x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1|)$

Effiziente OBDD-Operationen

- ▶ **Synthese:** $F_1(x) \otimes F_2(x)$, $\otimes \in \{\wedge, \vee, \oplus, =, \dots\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| \cdot |F_2|)$
- ▶ **Quantifizierung:** $(\exists x_i) F(x)$, $(\forall x_i) F(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F|^2)$
- ▶ **Äquivalenztest:** $F_1(x) \equiv F_2(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| + |F_2|)$
- ▶ **Var.-ersetzung:** $F(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_{n-1})$, $a \in \{0, 1\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F|)$
- ▶ **Erfüllbarkeit:** $F(x) \neq 0$
 - ▶ Zeit/Platz $\mathcal{O}(1)$
- ▶ **Argumenttausch:** $F_2(x, y) := F_1(y, x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1|)$

Effiziente OBDD-Operationen

- ▶ **Synthese:** $F_1(x) \otimes F_2(x)$, $\otimes \in \{\wedge, \vee, \oplus, =, \dots\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| \cdot |F_2|)$
- ▶ **Quantifizierung:** $(\exists x_i) F(x)$, $(\forall x_i) F(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F|^2)$
- ▶ **Äquivalenztest:** $F_1(x) \equiv F_2(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| + |F_2|)$
- ▶ **Var.-ersetzung:** $F(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_{n-1})$, $a \in \{0, 1\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F|)$
- ▶ **Erfüllbarkeit:** $F(x) \neq 0$
 - ▶ Zeit/Platz $\mathcal{O}(1)$
- ▶ **Argumenttausch:** $F_2(x, y) := F_1(y, x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1|)$

Effiziente OBDD-Operationen

- ▶ **Synthese:** $F_1(x) \otimes F_2(x)$, $\otimes \in \{\wedge, \vee, \oplus, =, \dots\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| \cdot |F_2|)$
- ▶ **Quantifizierung:** $(\exists x_i) F(x)$, $(\forall x_i) F(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F|^2)$
- ▶ **Äquivalenztest:** $F_1(x) \equiv F_2(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| + |F_2|)$
- ▶ **Var.-ersetzung:** $F(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_{n-1})$, $a \in \{0, 1\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F|)$
- ▶ **Erfüllbarkeit:** $F(x) \neq 0$
 - ▶ Zeit/Platz $\mathcal{O}(1)$
- ▶ **Argumenttausch:** $F_2(x, y) := F_1(y, x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1|)$

Effiziente OBDD-Operationen

- ▶ **Synthese:** $F_1(x) \otimes F_2(x)$, $\otimes \in \{\wedge, \vee, \oplus, =, \dots\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| \cdot |F_2|)$
- ▶ **Quantifizierung:** $(\exists x_i) F(x)$, $(\forall x_i) F(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F|^2)$
- ▶ **Äquivalenztest:** $F_1(x) \equiv F_2(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| + |F_2|)$
- ▶ **Var.-ersetzung:** $F(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_{n-1})$, $a \in \{0, 1\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F|)$
- ▶ **Erfüllbarkeit:** $F(x) \neq 0$
 - ▶ Zeit/Platz $\mathcal{O}(1)$
- ▶ **Argumenttausch:** $F_2(x, y) := F_1(y, x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1|)$

Effiziente OBDD-Operationen

- ▶ **Synthese:** $F_1(x) \otimes F_2(x)$, $\otimes \in \{\wedge, \vee, \oplus, =, \dots\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| \cdot |F_2|)$
- ▶ **Quantifizierung:** $(\exists x_i) F(x)$, $(\forall x_i) F(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F|^2)$
- ▶ **Äquivalenztest:** $F_1(x) \equiv F_2(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| + |F_2|)$
- ▶ **Var.-ersetzung:** $F(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_{n-1})$, $a \in \{0, 1\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F|)$
- ▶ **Erfüllbarkeit:** $F(x) \neq 0$
 - ▶ Zeit/Platz $\mathcal{O}(1)$
- ▶ **Argumenttausch:** $F_2(x, y) := F_1(y, x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1|)$

Effiziente OBDD-Operationen

- ▶ **Synthese:** $F_1(x) \otimes F_2(x)$, $\otimes \in \{\wedge, \vee, \oplus, =, \dots\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| \cdot |F_2|)$
- ▶ **Quantifizierung:** $(\exists x_i) F(x)$, $(\forall x_i) F(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F|^2)$
- ▶ **Äquivalenztest:** $F_1(x) \equiv F_2(x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1| + |F_2|)$
- ▶ **Var.-ersetzung:** $F(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots, x_{n-1})$, $a \in \{0, 1\}$
 - ▶ Zeit/Platz $\mathcal{O}(|F|)$
- ▶ **Erfüllbarkeit:** $F(x) \neq 0$
 - ▶ Zeit/Platz $\mathcal{O}(1)$
- ▶ **Argumenttausch:** $F_2(x, y) := F_1(y, x)$
 - ▶ Zeit/Platz $\mathcal{O}(|F_1|)$

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 (Bahar et al., 1993), (S., 2004)

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 (Bahar et al., 1993), (S., 2004)

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 (Bahar et al., 1993), (S., 2004)

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 (Bahar et al., 1993), (S., 2004)

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 (Bahar et al., 1993), (S., 2004)

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 (Bahar et al., 1993), (S., 2004)

Ziel der vorgestellten Arbeit:

Polylog. Laufzeit auf Graphen spezieller Struktur.

Übersicht

Einführung

Ein symbolischer APSP-Algorithmus

Analyse auf breitenbeschränkten Funktionen

Anmerkungen

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

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

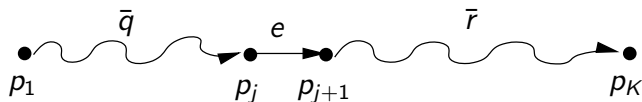
- ▶ $S^1(x, y, d) := [(d = 1) \wedge C(x, y, d)] \vee [(d = 0) \wedge (x = y)]$

Pfadzerlegung

Wegen $\ell(e) > 0$:

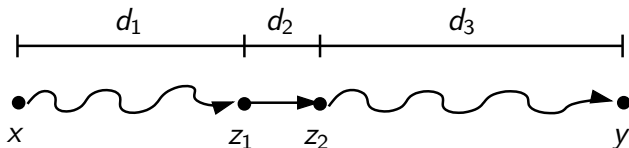
Lemma.

Jeder Pfad $\bar{p} = (p_1, \dots, p_K)$ mit $|\bar{p}| < 2^{i+1}$ enthält eine Kante $e := (p_j, p_{j+1})$, so dass $\bar{q} := (p_1, \dots, p_j)$ und $\bar{r} := (p_{j+1}, \dots, p_K)$ kürzer sind als 2^i .



Berechnung von S^{i+1}

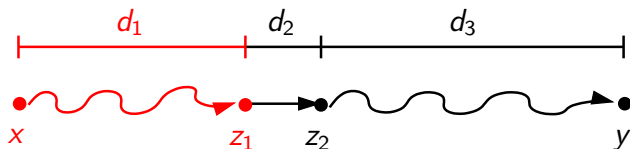
$$S^{i+1}(x, y, d) := (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S^i(x, z_1, d_1) \\ \wedge C(z_1, z_2, d_2) \\ \wedge S^i(z_2, y, d_3)] \\ \wedge (d < 2^{i+1}) [\dots]$$



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

Berechnung von S^{i+1}

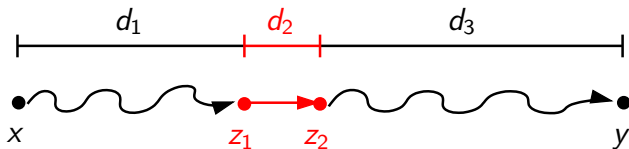
$$S^{i+1}(x, y, d) := (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S^i(x, z_1, d_1) \\ \wedge C(z_1, z_2, d_2) \\ \wedge S^i(z_2, y, d_3)] \\ \wedge (d < 2^{i+1}) [\dots]$$



- ▶ $\log(|V| \cdot \ell^{\max})$ 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}

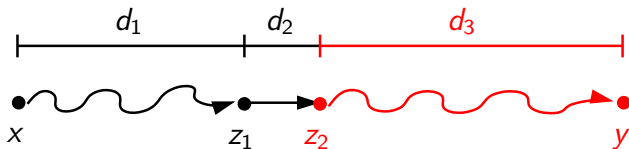
$$\begin{aligned} S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ & \wedge S^i(x, z_1, d_1) \\ & \wedge C(z_1, z_2, d_2) \\ & \wedge S^i(z_2, y, d_3)] \\ & \wedge (d < 2^{i+1}) [\dots] \end{aligned}$$



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

Berechnung von S^{i+1}

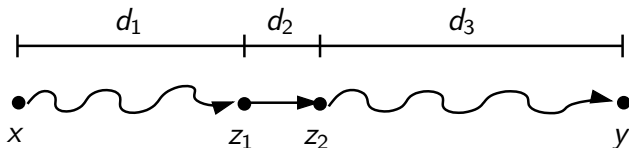
$$S^{i+1}(x, y, d) := (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S^i(x, z_1, d_1) \\ \wedge C(z_1, z_2, d_2) \\ \wedge S^i(z_2, y, d_3)] \\ \wedge (d < 2^{i+1}) [\dots]$$



- ▶ $\log(|V| \cdot \ell^{\max})$ 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}

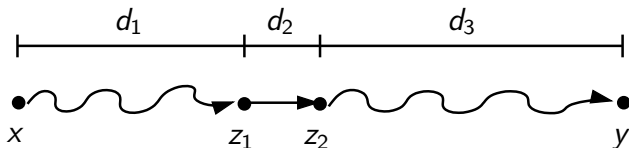
$$S^{i+1}(x, y, d) := (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S^i(x, z_1, d_1) \\ \wedge C(z_1, z_2, d_2) \\ \wedge S^i(z_2, y, d_3)] \\ \wedge (d < 2^{i+1}) [\dots]$$



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

Berechnung von S^{i+1}

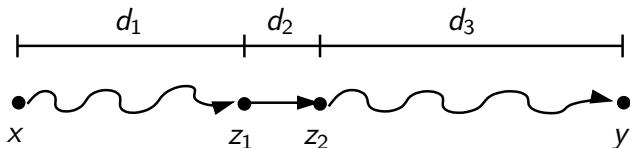
$$S^{i+1}(x, y, d) := (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S^i(x, z_1, d_1) \\ \wedge C(z_1, z_2, d_2) \\ \wedge S^i(z_2, y, d_3)] \\ \wedge (d < 2^{i+1}) [\dots]$$



- $\log(|V| \cdot \ell^{\max})$ 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}

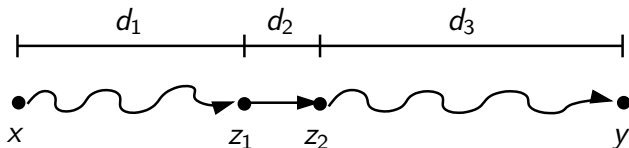
$$S^{i+1}(x, y, d) := (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S^i(x, z_1, d_1) \\ \wedge C(z_1, z_2, d_2) \\ \wedge S^i(z_2, y, d_3)] \\ \wedge (d < 2^{i+1}) [\dots]$$



- ▶ $\log(|V| \cdot \ell^{\max})$ 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}

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



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

Berechnung der Pfadkanten

Wir haben ...

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

Wir wollen ...

$$P(w, x, y, z) = 1 :\Leftrightarrow \text{Kante } (w, x) \text{ liegt auf kürzesten } y\text{-}z\text{-Pfad.}$$

$$P(w, x, y, z) := (\exists d, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S(y, w, d_1) \wedge C(w, x, d_2) \wedge S(x, z, d_3) \\ \wedge S(y, z, d)]$$

► Berechnung konkreter Pfade \bar{p} in Zeit $O(|P| \cdot |\bar{p}| \cdot \log |V|)$

Berechnung der Pfadkanten

Wir haben ...

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

Wir wollen ...

$$P(w, x, y, z) = 1 :\Leftrightarrow \text{Kante } (w, x) \text{ liegt auf kürzesten } y\text{-}z\text{-Pfad.}$$

$$P(w, x, y, z) := (\exists d, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S(y, w, d_1) \wedge C(w, x, d_2) \wedge S(x, z, d_3) \\ \wedge S(y, z, d)]$$

► Berechnung konkreter Pfade \bar{p} in Zeit $O(|P| \cdot |\bar{p}| \cdot \log |V|)$

Berechnung der Pfadkanten

Wir haben ...

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

Wir wollen ...

$P(w, x, y, z) = 1 :\Leftrightarrow$ Kante (w, x) liegt auf kürzesten y - z -Pfad.

$$P(w, x, y, z) := (\exists d, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S(y, w, d_1) \wedge C(w, x, d_2) \wedge S(x, z, d_3) \\ \wedge S(y, z, d)]$$

► Berechnung konkreter Pfade \bar{p} in Zeit $\mathcal{O}(|P| \cdot |\bar{p}| \cdot \log |V|)$

Berechnung der Pfadkanten

Wir haben ...

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

Wir wollen ...

$P(w, x, y, z) = 1 :\Leftrightarrow$ Kante (w, x) liegt auf kürzesten y - z -Pfad.

$$P(w, x, y, z) := (\exists d, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S(y, w, d_1) \wedge C(w, x, d_2) \wedge S(x, z, d_3) \\ \wedge S(y, z, d)]$$

► Berechnung konkreter Pfade \bar{p} in Zeit $\mathcal{O}(|P| \cdot |\bar{p}| \cdot \log |V|)$

Berechnung der Pfadkanten

Wir haben ...

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

Wir wollen ...

$P(w, x, y, z) = 1 :\Leftrightarrow$ Kante (w, x) liegt auf kürzesten y - z -Pfad.

$$P(w, x, y, z) := (\exists d, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S(y, w, d_1) \wedge C(w, x, d_2) \wedge S(x, z, d_3) \\ \wedge S(y, z, d)]$$

► Berechnung konkreter Pfade \bar{p} in Zeit $\mathcal{O}(|P| \cdot |\bar{p}| \cdot \log |V|)$

Berechnung der Pfadkanten

Wir haben ...

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

Wir wollen ...

$P(w, x, y, z) = 1 :\Leftrightarrow$ Kante (w, x) liegt auf kürzesten y - z -Pfad.

$$P(w, x, y, z) := (\exists d, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S(y, w, d_1) \wedge C(w, x, d_2) \wedge S(x, z, d_3) \\ \wedge S(y, z, d)]$$

► Berechnung konkreter Pfade \bar{p} in Zeit $\mathcal{O}(|P| \cdot |\bar{p}| \cdot \log |V|)$

Berechnung der Pfadkanten

Wir haben ...

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

Wir wollen ...

$P(w, x, y, z) = 1 :\Leftrightarrow$ Kante (w, x) liegt auf kürzesten y - z -Pfad.

$$P(w, x, y, z) := (\exists d, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S(y, w, d_1) \wedge C(w, x, d_2) \wedge S(x, z, d_3) \\ \wedge S(y, z, d)]$$

► Berechnung konkreter Pfade \bar{p} in Zeit $\mathcal{O}(|P| \cdot |\bar{p}| \cdot \log |V|)$

Berechnung der Pfadkanten

Wir haben ...

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

Wir wollen ...

$P(w, x, y, z) = 1 :\Leftrightarrow$ Kante (w, x) liegt auf kürzesten y - z -Pfad.

$$P(w, x, y, z) := (\exists d, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S(y, w, d_1) \wedge C(w, x, d_2) \wedge S(x, z, d_3) \\ \wedge S(y, z, d)]$$

► Berechnung konkreter Pfade \bar{p} in Zeit $\mathcal{O}(|P| \cdot |\bar{p}| \cdot \log |V|)$

Berechnung der Pfadkanten

Wir haben ...

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

Wir wollen ...

$P(w, x, y, z) = 1 :\Leftrightarrow$ Kante (w, x) liegt auf kürzesten y - z -Pfad.

$$P(w, x, y, z) := (\exists d, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S(y, w, d_1) \wedge C(w, x, d_2) \wedge S(x, z, d_3) \\ \wedge S(y, z, d)]$$

► Berechnung konkreter Pfade \bar{p} in Zeit $\mathcal{O}(|P| \cdot |\bar{p}| \cdot \log |V|)$

Berechnung der Pfadkanten

Wir haben ...

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

Wir wollen ...

$P(w, x, y, z) = 1 :\Leftrightarrow$ Kante (w, x) liegt auf kürzesten y - z -Pfad.

$$P(w, x, y, z) := (\exists d, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\ \wedge S(y, w, d_1) \wedge C(w, x, d_2) \wedge S(x, z, d_3) \\ \wedge S(y, z, d)]$$

- Berechnung konkreter Pfade \bar{p} in Zeit $\mathcal{O}(|P| \cdot |\bar{p}| \cdot \log |V|)$

Übersicht

Einführung

Ein symbolischer APSP-Algorithmus

Analyse auf breitenbeschränkten Funktionen

Anmerkungen

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 $f = (f_n)_n$ boolescher Fkt. heißt **breitenbeschränkt** wenn $\exists b$: Die **vollst.** OBDDs $F = (F_n)_n$ haben alle Breite $\leq b$.

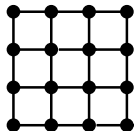
G_1



G_2



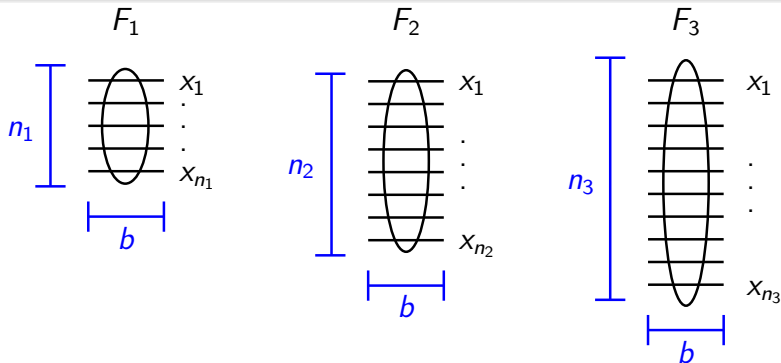
G_3



Breitenbeschränkte Funktionen

Definition.

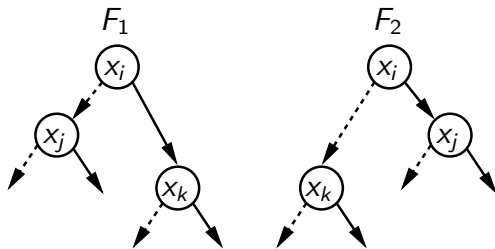
Eine Folge $f = (f_n)_n$ boolescher Fkt. heißt **breitenbeschränkt** wenn $\exists b$: Die **vollst.** OBDDs $F = (F_n)_n$ haben alle Breite $\leq b$.



Breitenbeschränkte Funktionen – Synthese

$$F_1 \otimes F_2 = \left[x_i \wedge [F_1(x_i = 1) \otimes F_2(x_i = 1)] \right] \\ \vee \left[\bar{x}_i \wedge [F_1(x_i = 0) \otimes F_2(x_i = 0)] \right]$$

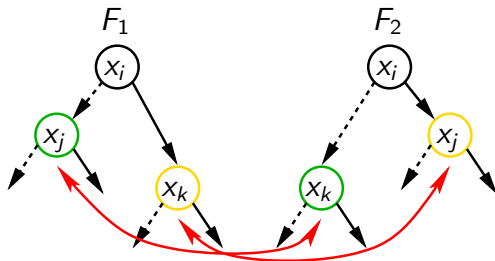
- ▶ Allgemein: $|F_1 \otimes F_2| = \Omega(|F_1| \cdot |F_2|)$
- ▶ Breitenbeschränkt: $|F_1|, |F_2| \leq n \cdot b, |F_1 \otimes F_2| = \mathcal{O}(n \cdot b^2)$.



Breitenbeschränkte Funktionen – Synthese

$$F_1 \otimes F_2 = \left[x_i \wedge [F_1(x_i = 1) \otimes F_2(x_i = 1)] \right] \\ \vee \left[\bar{x}_i \wedge [F_1(x_i = 0) \otimes F_2(x_i = 0)] \right]$$

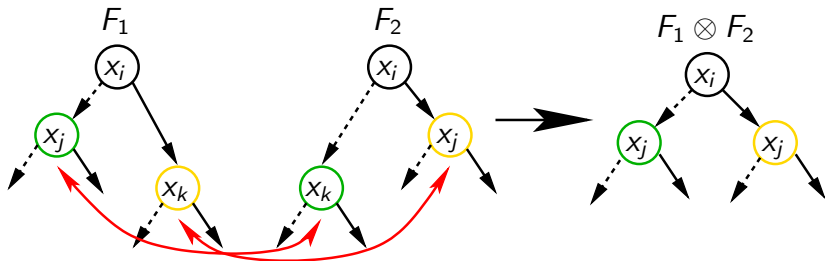
- ▶ Allgemein: $|F_1 \otimes F_2| = \Omega(|F_1| \cdot |F_2|)$
- ▶ Breitenbeschränkt: $|F_1|, |F_2| \leq n \cdot b, |F_1 \otimes F_2| = \mathcal{O}(n \cdot b^2)$.



Breitenbeschränkte Funktionen – Synthese

$$F_1 \otimes F_2 = \left[x_i \wedge [F_1(x_i = 1) \otimes F_2(x_i = 1)] \right] \\ \vee \left[\bar{x}_i \wedge [F_1(x_i = 0) \otimes F_2(x_i = 0)] \right]$$

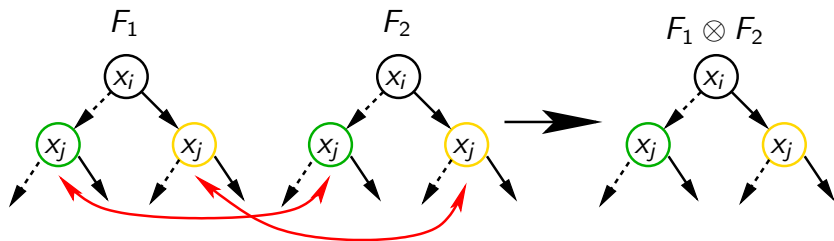
- ▶ Allgemein: $|F_1 \otimes F_2| = \Omega(|F_1| \cdot |F_2|)$
- ▶ Breitenbeschränkt: $|F_1|, |F_2| \leq n \cdot b, |F_1 \otimes F_2| = \mathcal{O}(n \cdot b^2)$.



Breitenbeschränkte Funktionen – Synthese

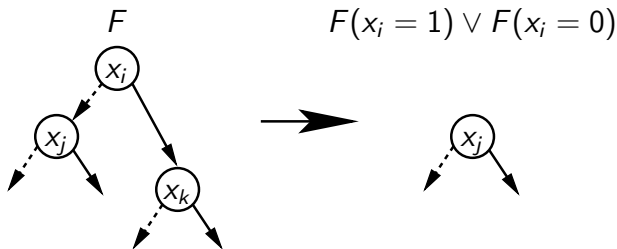
$$F_1 \otimes F_2 = \left[x_i \wedge [F_1(x_i = 1) \otimes F_2(x_i = 1)] \right] \\ \vee \left[\bar{x}_i \wedge [F_1(x_i = 0) \otimes F_2(x_i = 0)] \right]$$

- ▶ Allgemein: $|F_1 \otimes F_2| = \Omega(|F_1| \cdot |F_2|)$
- ▶ Breitenbeschränkt: $|F_1|, |F_2| \leq n \cdot b$, $|F_1 \otimes F_2| = \mathcal{O}(n \cdot b^2)$.



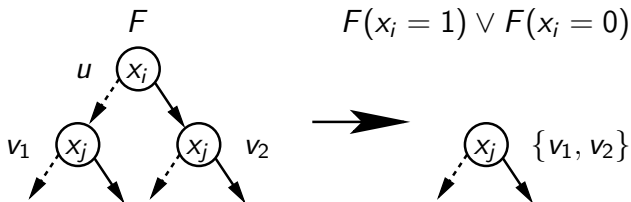
Breitenbeschränkte Funktionen – Quantifizierung

- ▶ Allgemeiner Fall: $|\langle \exists x_{i_1}, \dots, x_{i_r} \rangle F| = \Omega(|F|^{2^r})$
- ▶ Breitenbeschränkt: $|\langle \exists x_{i_1}, \dots, x_{i_r} \rangle F| = \mathcal{O}(n \cdot 2^b)$



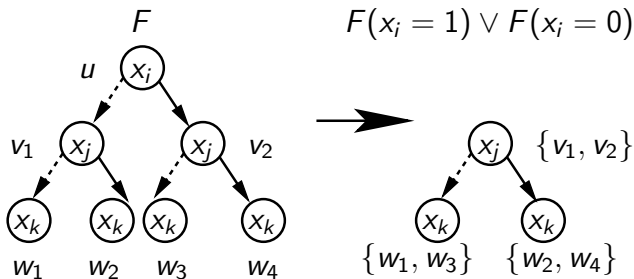
Breitenbeschränkte Funktionen – Quantifizierung

- ▶ Allgemeiner Fall: $|(\exists x_{i_1}, \dots, x_{i_r}) F| = \Omega(|F|^{2^r})$
- ▶ Breitenbeschränkt: $|(\exists x_{i_1}, \dots, x_{i_r}) F| = \mathcal{O}(n \cdot 2^b)$



Breitenbeschränkte Funktionen – Quantifizierung

- ▶ Allgemeiner Fall: $|\{\exists x_{i_1}, \dots, x_{i_r}\} F| = \Omega(|F|^{2^r})$
- ▶ Breitenbeschränkt: $|\{\exists x_{i_1}, \dots, x_{i_r}\} F| = \mathcal{O}(n \cdot 2^b)$



Der APSP-Algo. auf breitenbeschränkten Funktionen

- ▶ Betrachte Graphsequenz $G = (G_n)_n$.
- ▶ $C = (C_n)_n$ und $S = (S_n)_n$ mit Breite b .

Satz.

Der symbolische APSP-Algorithmus berechnet S_n aus C_n in Zeit/Platz $\mathcal{O}(\log^3(|V_n| \cdot \ell_n^{\max}) \cdot \alpha(b))$.

Beweisskizze.

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

Der APSP-Algo. auf breitenbeschränkten Funktionen

- ▶ Betrachte Graphsequenz $G = (G_n)_n$.
- ▶ $C = (C_n)_n$ und $S = (S_n)_n$ mit Breite b .

Satz.

Der symbolische APSP-Algorithmus berechnet S_n aus C_n in Zeit/Platz $\mathcal{O}(\log^3(|V_n| \cdot \ell_n^{\max}) \cdot \alpha(b))$.

Beweisskizze.

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

Der APSP-Algo. auf breitenbeschränkten Funktionen

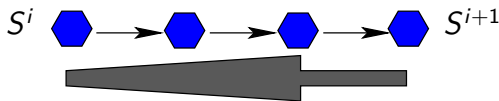
- ▶ Betrachte Graphsequenz $G = (G_n)_n$.
- ▶ $C = (C_n)_n$ und $S = (S_n)_n$ mit Breite b .

Satz.

Der symbolische APSP-Algorithmus berechnet S_n aus C_n in Zeit/Platz $\mathcal{O}(\log^3(|V_n| \cdot \ell_n^{\max}) \cdot \alpha(b))$.

Beweisskizze.

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



Der APSP-Algo. auf breitenbeschränkten Funktionen

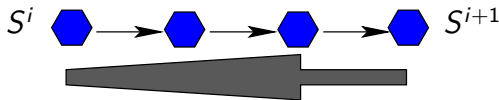
- ▶ Betrachte Graphsequenz $G = (G_n)_n$.
- ▶ $C = (C_n)_n$ und $S = (S_n)_n$ mit Breite b .

Satz.

Der symbolische APSP-Algorithmus berechnet S_n aus C_n in Zeit/Platz $\mathcal{O}(\log^3(|V_n| \cdot \ell_n^{\max}) \cdot \alpha(b))$.

Beweisskizze.

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



Analyse des $S^i \rightarrow S^{i+1}$ -Schritts

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

- ▶ $S^i(x, z_1, d_1)$ und $C(x, y, d)$ haben Breite $\mathcal{O}(b)$.
- ▶ $(d_1 + d_2 + d_3 = d)$ und $(d < 2^{i+1})$ haben Breite $\mathcal{O}(1)$.
- ▶ Breitenwachstum:
 - ▶ Synthese: Faktor b
 - ▶ Quantifizierung: Exponenzierung
- ▶ Insgesamt: Breite $\alpha(b) = 2^{2^{\mathcal{O}(b^3)}}$
- ▶ Durch Umformulierung: Breite $\alpha(b) = 2^{\mathcal{O}(b^3)}$

Analyse des $S^i \rightarrow S^{i+1}$ -Schritts

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

- ▶ $S^i(x, z_1, d_1)$ und $C(x, y, d)$ haben Breite $\mathcal{O}(b)$.
- ▶ $(d_1 + d_2 + d_3 = d)$ und $(d < 2^{i+1})$ haben Breite $\mathcal{O}(1)$.
- ▶ Breitenwachstum:
 - ▶ Synthese: Faktor b
 - ▶ Quantifizierung: Exponenzierung
- ▶ Insgesamt: Breite $\alpha(b) = 2^{2^{\mathcal{O}(b^3)}}$
- ▶ Durch Umformulierung: Breite $\alpha(b) = 2^{\mathcal{O}(b^3)}$

Analyse des $S^i \rightarrow S^{i+1}$ -Schritts

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

- ▶ $S^i(x, z_1, d_1)$ und $C(x, y, d)$ haben Breite $\mathcal{O}(b)$.
- ▶ $(d_1 + d_2 + d_3 = d)$ und $(d < 2^{i+1})$ haben Breite $\mathcal{O}(1)$.
- ▶ Breitenwachstum:
 - ▶ Synthese: Faktor b
 - ▶ Quantifizierung: Exponenzierung
- ▶ Insgesamt: Breite $\alpha(b) = 2^{2^{\mathcal{O}(b^3)}}$
- ▶ Durch Umformulierung: Breite $\alpha(b) = 2^{\mathcal{O}(b^3)}$

Analyse des $S^i \rightarrow S^{i+1}$ -Schritts

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

- ▶ $S^i(x, z_1, d_1)$ und $C(x, y, d)$ haben Breite $\mathcal{O}(b)$.
- ▶ $(d_1 + d_2 + d_3 = d)$ und $(d < 2^{i+1})$ haben Breite $\mathcal{O}(1)$.
- ▶ Breitenwachstum:
 - ▶ Synthese: Faktor b
 - ▶ Quantifizierung: Exponenzierung
- ▶ Insgesamt: Breite $\alpha(b) = 2^{2^{\mathcal{O}(b^3)}}$
- ▶ Durch Umformulierung: Breite $\alpha(b) = 2^{\mathcal{O}(b^3)}$

Analyse des $S^i \rightarrow S^{i+1}$ -Schritts

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

- ▶ $S^i(x, z_1, d_1)$ und $C(x, y, d)$ haben Breite $\mathcal{O}(b)$.
- ▶ $(d_1 + d_2 + d_3 = d)$ und $(d < 2^{i+1})$ haben Breite $\mathcal{O}(1)$.
- ▶ Breitenwachstum:
 - ▶ Synthese: Faktor b
 - ▶ Quantifizierung: Exponenzierung
- ▶ Insgesamt: Breite $\alpha(b) = 2^{2^{\mathcal{O}(b^3)}}$
- ▶ Durch Umformulierung: Breite $\alpha(b) = 2^{\mathcal{O}(b^3)}$

Analyse des $S^i \rightarrow S^{i+1}$ -Schritts

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

- ▶ $S^i(x, z_1, d_1)$ und $C(x, y, d)$ haben Breite $\mathcal{O}(b)$.
- ▶ $(d_1 + d_2 + d_3 = d)$ und $(d < 2^{i+1})$ haben Breite $\mathcal{O}(1)$.
- ▶ Breitenwachstum:
 - ▶ Synthese: Faktor b
 - ▶ Quantifizierung: Exponenzierung
- ▶ Insgesamt: Breite $\alpha(b) = 2^{2^{\mathcal{O}(b^3)}}$
- ▶ Durch Umformulierung: Breite $\alpha(b) = 2^{\mathcal{O}(b^3)}$

Analyse des $S^i \rightarrow S^{i+1}$ -Schritts

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

- ▶ $S^i(x, z_1, d_1)$ und $C(x, y, d)$ haben Breite $\mathcal{O}(b)$.
- ▶ $(d_1 + d_2 + d_3 = d)$ und $(d < 2^{i+1})$ haben Breite $\mathcal{O}(1)$.
- ▶ Breitenwachstum:
 - ▶ Synthese: Faktor b
 - ▶ Quantifizierung: Exponenzierung
- ▶ Insgesamt: Breite $\alpha(b) = 2^{2^{\mathcal{O}(b^3)}}$
- ▶ Durch Umformulierung: Breite $\alpha(b) = 2^{\mathcal{O}(b^3)}$

Graphen mit breitenbeschränkten Funktionen

Welche Graphen haben breitenbeschränkte Funktionen?

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

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

- ▶ 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:

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

- ▶ 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:

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

- ▶ 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:

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

- ▶ 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:

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

- ▶ 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:

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

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

Produktkomposition von Graphen (1)

Definition.

G_3 ist das **Produkt** von G_1 und G_2 wenn $V_3 = V_1 \times V_2$,

$$E_3 = \left\{ ((t, u), (v, w)) \mid ((t, v) \in E_1 \wedge (u = w)) \right. \\ \left. \vee ((t = v) \wedge (u, w) \in E_2) \right\},$$

und $l_3((t, u), (v, w)) = l_1(t, v)$ für $u = w$ bzw.
 $l_3((t, u), (v, w)) = l_2(u, w)$ für $t = v$.

Produktkomposition von Graphen (2)

Symbolische Formulierung:

- ▶ Eingabegraph $C_3(x_1, x_2, y_1, y_2, d)$:

$$C_3(x_1, x_2, y_1, y_2, d) = [C_1(x_1, y_1, d) \wedge (x_2 = y_2)] \\ \vee [(x_1 = y_1) \wedge C_2(x_2, y_2, d)] ,$$

- ▶ Distanzfunktion $S_3(x_1, x_2, y_1, y_2, d)$:

$$S_3(x_1, x_2, y_1, y_2, d) = (\exists d_1, d_2) [(d_1 + d_2 = d) \\ \wedge S_1(x_1, y_1, d_1) \wedge S_2(x_2, y_2, d_2)] .$$

- ▶ Abschluss gegen \times folgt aus Abschluss gegen \otimes und \exists .

Produktkomposition von Graphen (2)

Symbolische Formulierung:

- ▶ Eingabegraph $C_3(x_1, x_2, y_1, y_2, d)$:

$$C_3(x_1, x_2, y_1, y_2, d) = [C_1(x_1, y_1, d) \wedge (x_2 = y_2)] \\ \vee [(x_1 = y_1) \wedge C_2(x_2, y_2, d)] ,$$

- ▶ Distanzfunktion $S_3(x_1, x_2, y_1, y_2, d)$:

$$S_3(x_1, x_2, y_1, y_2, d) = (\exists d_1, d_2) [(d_1 + d_2 = d) \\ \wedge S_1(x_1, y_1, d_1) \wedge S_2(x_2, y_2, d_2)] .$$

- ▶ Abschluss gegen \times folgt aus Abschluss gegen \otimes und \exists .

Produktkomposition von Graphen (2)

Symbolische Formulierung:

- ▶ Eingabegraph $C_3(x_1, x_2, y_1, y_2, d)$:

$$C_3(x_1, x_2, y_1, y_2, d) = [C_1(x_1, y_1, d) \wedge (x_2 = y_2)] \\ \vee [(x_1 = y_1) \wedge C_2(x_2, y_2, d)] ,$$

- ▶ Distanzfunktion $S_3(x_1, x_2, y_1, y_2, d)$:

$$S_3(x_1, x_2, y_1, y_2, d) = (\exists d_1, d_2) [(d_1 + d_2 = d) \\ \wedge S_1(x_1, y_1, d_1) \wedge S_2(x_2, y_2, d_2)] .$$

- ▶ Abschluss gegen \times folgt aus Abschluss gegen \otimes und \exists .

Übersicht

Einführung

Ein symbolischer APSP-Algorithmus

Analyse auf breitenbeschränkten Funktionen

Anmerkungen

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 $G^* = (G_n^*)_n$:
 - ▶ $C = (C_n)_n$ und $S = (S_n)_n$ sind breitenbeschränkt.
 - ▶ OBDDs $S^i = (S_n^i)_n$ sind nicht breitenbeschränkt.
- ▶ Vermutung: $\ell(e) > 0$ ist 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 $G^* = (G_n^*)_n$:
 - ▶ $C = (C_n)_n$ und $S = (S_n)_n$ sind breitenbeschränkt.
 - ▶ OBDDs $S^i = (S_n^i)_n$ sind nicht breitenbeschränkt.
- ▶ Vermutung: $\ell(e) > 0$ ist 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 $G^* = (G_n^*)_n$:
 - ▶ $C = (C_n)_n$ und $S = (S_n)_n$ sind breitenbeschränkt.
 - ▶ OBDDs $S^i = (S_n^i)_n$ sind **nicht** breitenbeschränkt.
- ▶ Vermutung: $\ell(e) > 0$ ist 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 $G^* = (G_n^*)_n$:
 - ▶ $C = (C_n)_n$ und $S = (S_n)_n$ sind breitenbeschränkt.
 - ▶ OBDDs $S^i = (S_n^i)_n$ sind nicht breitenbeschränkt.
- ▶ Vermutung: $\ell(e) > 0$ ist 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 $G^* = (G_n^*)_n$:
 - ▶ $C = (C_n)_n$ und $S = (S_n)_n$ sind breitenbeschränkt.
 - ▶ OBDDs $S^i = (S_n^i)_n$ sind **nicht** breitenbeschränkt.
- ▶ Vermutung: $\ell(e) > 0$ ist notwendig für polylog. Laufzeit.

Ein alternativer Ansatz

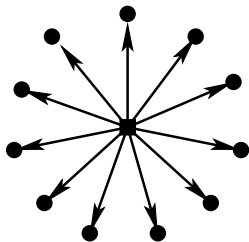
- ▶ Ist die Einschränkung $\ell(e) > 0$ notwendig?
- ▶ Alternativ: $S^i(x, y, d) = 1 \Leftrightarrow x$ - y -Pfad mit $\leq 2^i$ Kanten.

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

- ▶ Gegenbeispiel $G^* = (G_n^*)_n$:
 - ▶ $C = (C_n)_n$ und $S = (S_n)_n$ sind breitenbeschränkt.
 - ▶ OBDDs $S^i = (S_n^i)_n$ sind **nicht** breitenbeschränkt.
- ▶ Vermutung: $\ell(e) > 0$ ist notwendig für polylog. Laufzeit.

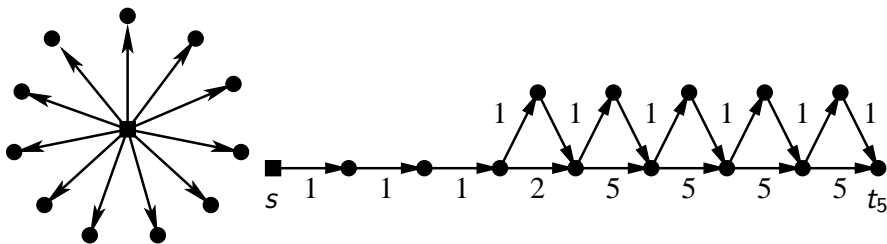
Konstruktion von G_n^*

- ▶ Stern aus s - t_i -Pfad: Länge i^2 , 2^n Kanten.
- ▶ Kürzester s - t_i -Weg kürzt ab: Länge $2^n + i$, $> 2^n$ Kanten.
- ▶ $\Rightarrow S_n^n$ darf keine Abkürzungen benutzen.



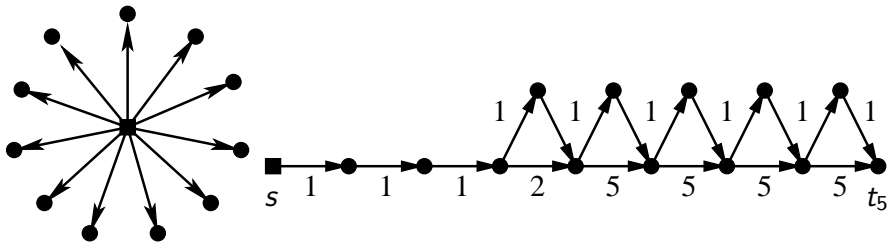
Konstruktion von G_n^*

- ▶ Stern aus $s-t_i$ -Pfad: Länge i^2 , 2^n Kanten.
- ▶ Kürzester $s-t_i$ -Weg kürzt ab: Länge $2^n + i$, $> 2^n$ Kanten.
- ▶ $\Rightarrow S_n^n$ darf keine Abkürzungen benutzen.



Konstruktion von G_n^*

- ▶ Stern aus $s-t_i$ -Pfad: Länge i^2 , 2^n Kanten.
- ▶ Kürzester $s-t_i$ -Weg kürzt ab: Länge $2^n + i$, $> 2^n$ Kanten.
- ▶ $\Rightarrow S_n^n$ darf keine Abkürzungen benutzen.

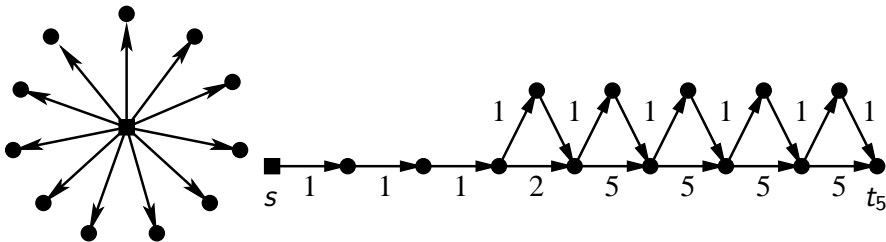


Konstruktion von G_n^*

- ▶ Stern aus $s-t_i$ -Pfad: Länge i^2 , 2^n Kanten.
- ▶ Kürzester $s-t_i$ -Weg kürzt ab: Länge $2^n + i$, $> 2^n$ Kanten.
- ▶ $\Rightarrow S_n^n$ darf keine Abkürzungen benutzen.

Idee.

S_n^n ist breitenbeschränkt $\Rightarrow SQU_{i,n}$ ist breitenbeschränkt. ⚡



Verwandte Probleme

Der vorgestellte Algorithmus kann angepasst werden an ...

- ▶ all-pairs lightest shortest paths.
- ▶ all-pairs almost shortest paths.
- ▶ all-pairs small-stretch paths.
- ▶ dynamische Gewichtsverringierung.

$$F(x, y, d) := (\exists x_1, x_2, d_1, d_2, d_3) \\ [(d_1 + d_2 + d_3 = d) \\ \wedge S(x, x_1, d_1) \wedge C'(x_1, x_2, d_2) \wedge S(x_2, y, d_3)]$$

Verwandte Probleme

Der vorgestellte Algorithmus kann angepasst werden an ...

- ▶ all-pairs lightest shortest paths.
- ▶ all-pairs almost shortest paths.
- ▶ all-pairs small-stretch paths.
- ▶ dynamische Gewichtsverringernug.

$$F(x, y, d) := (\exists x_1, x_2, d_1, d_2, d_3) \\ [(d_1 + d_2 + d_3 = d) \\ \wedge S(x, x_1, d_1) \wedge C'(x_1, x_2, d_2) \wedge S(x_2, y, d_3)]$$

Verwandte Probleme

Der vorgestellte Algorithmus kann angepasst werden an ...

- ▶ all-pairs lightest shortest paths.
- ▶ all-pairs almost shortest paths.
- ▶ all-pairs small-stretch paths.
- ▶ dynamische Gewichtsverringierung.

$$F(x, y, d) := (\exists x_1, x_2, d_1, d_2, d_3) \\ [(d_1 + d_2 + d_3 = d) \\ \wedge S(x, x_1, d_1) \wedge C'(x_1, x_2, d_2) \wedge S(x_2, y, d_3)]$$

Verwandte Probleme

Der vorgestellte Algorithmus kann angepasst werden an ...

- ▶ all-pairs lightest shortest paths.
- ▶ all-pairs almost shortest paths.
- ▶ all-pairs small-stretch paths.
- ▶ dynamische Gewichtsverringernug.

$$F(x, y, d) := (\exists x_1, x_2, d_1, d_2, d_3) \\ [(d_1 + d_2 + d_3 = d) \\ \wedge S(x, x_1, d_1) \wedge C'(x_1, x_2, d_2) \wedge S(x_2, y, d_3)]$$

Verwandte Probleme

Der vorgestellte Algorithmus kann angepasst werden an ...

- ▶ all-pairs lightest shortest paths.
- ▶ all-pairs almost shortest paths.
- ▶ all-pairs small-stretch paths.
- ▶ dynamische Gewichtsverringierung.

$$F(x, y, d) := (\exists x_1, x_2, d_1, d_2, d_3) \\ [(d_1 + d_2 + d_3 = d) \\ \wedge S(x, x_1, d_1) \wedge C'(x_1, x_2, d_2) \wedge S(x_2, y, d_3)]$$

Danke fürs Zuhören!

