

A Symbolic Approach to the All-Pairs Shortest-Paths Problem

Daniel Sawitzki

Computer Science 2
University of Dortmund, Germany

Workshop on Graph-Theoretic Concepts in Computer Science
June 21–23, 2004

Overview

- ▶ Introduction to Symbolic Algorithms
- ▶ A Symbolic APSP-Algorithm
- ▶ Analysis on Bounded-Width Functions
- ▶ Concluding Remarks

Overview

- ▶ Introduction to Symbolic Algorithms
- ▶ A Symbolic APSP-Algorithm
- ▶ Analysis on Bounded-Width Functions
- ▶ Concluding Remarks

Overview

- ▶ Introduction to Symbolic Algorithms
- ▶ A Symbolic APSP-Algorithm
- ▶ Analysis on Bounded-Width Functions
- ▶ Concluding Remarks

Overview

- ▶ Introduction to Symbolic Algorithms
- ▶ A Symbolic APSP-Algorithm
- ▶ Analysis on Bounded-Width Functions
- ▶ Concluding Remarks

Motivation

- ▶ Today's applications produce large graphs (VLSI, traffic, WWW, ...).
 - ▶ \Rightarrow Conflicts with memory size
 - ▶ \Rightarrow Even efficient algorithms not applicable
- ▶ Observation: Applications produce large, but **structured** graphs.
- ▶ Heuristic approach from VLSI: **Symbolic representation**
 - ▶ Do not enumerate nodes and edges explicitly.
 - ▶ Consider graph G as a **characteristic** Boolean function C .
 - ▶ Represent C by a (hopefully) compact data structure.
 - ▶ Solve problems in G by **few** and **efficient** operations on C .

Motivation

- ▶ Today's applications produce large graphs (VLSI, traffic, WWW, ...).
 - ▶ \Rightarrow Conflicts with memory size
 - ▶ \Rightarrow Even efficient algorithms not applicable
- ▶ Observation: Applications produce large, but **structured** graphs.
- ▶ Heuristic approach from VLSI: **Symbolic representation**
 - ▶ Do not enumerate nodes and edges explicitly.
 - ▶ Consider graph G as a **characteristic** Boolean function C .
 - ▶ Represent C by a (hopefully) compact data structure.
 - ▶ Solve problems in G by **few** and **efficient** operations on C .

Motivation

- ▶ Today's applications produce large graphs (VLSI, traffic, WWW, ...).
 - ▶ \Rightarrow Conflicts with memory size
 - ▶ \Rightarrow Even efficient algorithms not applicable
- ▶ Observation: Applications produce large, but **structured** graphs.
- ▶ Heuristic approach from VLSI: **Symbolic representation**
 - ▶ Do not enumerate nodes and edges explicitly.
 - ▶ Consider graph G as a **characteristic** Boolean function C .
 - ▶ Represent C by a (hopefully) compact data structure.
 - ▶ Solve problems in G by **few** and **efficient** operations on C .

Motivation

- ▶ Today's applications produce large graphs (VLSI, traffic, WWW, ...).
 - ▶ \Rightarrow Conflicts with memory size
 - ▶ \Rightarrow Even efficient algorithms not applicable
- ▶ Observation: Applications produce large, but **structured** graphs.
- ▶ Heuristic approach from VLSI: **Symbolic representation**
 - ▶ Do not enumerate nodes and edges explicitly.
 - ▶ Consider graph G as a **characteristic** Boolean function C .
 - ▶ Represent C by a (hopefully) compact data structure.
 - ▶ Solve problems in G by **few and efficient** operations on C .

Motivation

- ▶ Today's applications produce large graphs (VLSI, traffic, WWW, ...).
 - ▶ \Rightarrow Conflicts with memory size
 - ▶ \Rightarrow Even efficient algorithms not applicable
- ▶ Observation: Applications produce large, but **structured** graphs.
- ▶ Heuristic approach from VLSI: **Symbolic representation**
 - ▶ Do not enumerate nodes and edges explicitly.
 - ▶ Consider graph G as a **characteristic** Boolean function C .
 - ▶ Represent C by a (hopefully) compact data structure.
 - ▶ Solve problems in G by **few** and **efficient** operations on C .

Motivation

- ▶ Today's applications produce large graphs (VLSI, traffic, WWW, ...).
 - ▶ \Rightarrow Conflicts with memory size
 - ▶ \Rightarrow Even efficient algorithms not applicable
- ▶ Observation: Applications produce large, but **structured** graphs.
- ▶ Heuristic approach from VLSI: **Symbolic representation**
 - ▶ Do not enumerate nodes and edges explicitly.
 - ▶ Consider graph G as a **characteristic** Boolean function C .
 - ▶ Represent C by a (hopefully) compact data structure.
 - ▶ Solve problems in G by **few** and **efficient** operations on C .

Motivation

- ▶ Today's applications produce large graphs (VLSI, traffic, WWW, ...).
 - ▶ \Rightarrow Conflicts with memory size
 - ▶ \Rightarrow Even efficient algorithms not applicable
- ▶ Observation: Applications produce large, but **structured** graphs.
- ▶ Heuristic approach from VLSI: **Symbolic representation**
 - ▶ Do not enumerate nodes and edges explicitly.
 - ▶ Consider graph G as a **characteristic** Boolean function C .
 - ▶ Represent C by a (hopefully) compact data structure.
 - ▶ Solve problems in G by **few** and **efficient** operations on C .

Motivation

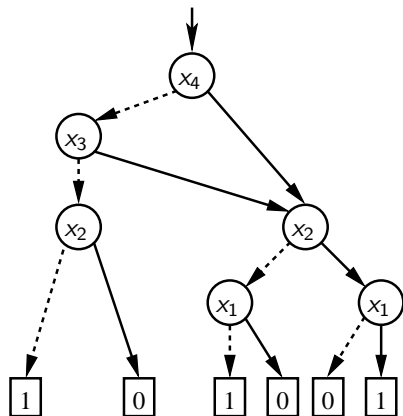
- ▶ Today's applications produce large graphs (VLSI, traffic, WWW, ...).
 - ▶ \Rightarrow Conflicts with memory size
 - ▶ \Rightarrow Even efficient algorithms not applicable
- ▶ Observation: Applications produce large, but **structured** graphs.
- ▶ Heuristic approach from VLSI: **Symbolic representation**
 - ▶ Do not enumerate nodes and edges explicitly.
 - ▶ Consider graph G as a **characteristic** Boolean function C .
 - ▶ Represent C by a (hopefully) compact data structure.
 - ▶ Solve problems in G by **few** and **efficient** operations on C .

Motivation

- ▶ Today's applications produce large graphs (VLSI, traffic, WWW, ...).
 - ▶ \Rightarrow Conflicts with memory size
 - ▶ \Rightarrow Even efficient algorithms not applicable
- ▶ Observation: Applications produce large, but **structured** graphs.
- ▶ Heuristic approach from VLSI: **Symbolic representation**
 - ▶ Do not enumerate nodes and edges explicitly.
 - ▶ Consider graph G as a **characteristic** Boolean function C .
 - ▶ Represent C by a (hopefully) compact data structure.
 - ▶ Solve problems in G by **few** and **efficient** operations on C .

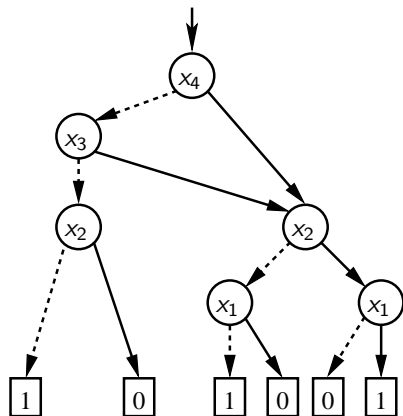
Ordered Binary Decision Diagrams (OBDDs)

- ▶ Data structure for Boolean functions
 $f: \{0, 1\}^n \rightarrow \{0, 1\}$
(Bryant, 1985)
- ▶ Inner nodes labeled with variables, left by 0-/1-edges.
- ▶ Sink nodes correspond to value $f(x_1, \dots, x_n)$.
- ▶ Variables x_1, \dots, x_n are read w. r. t. $\pi \in \Sigma_n$.



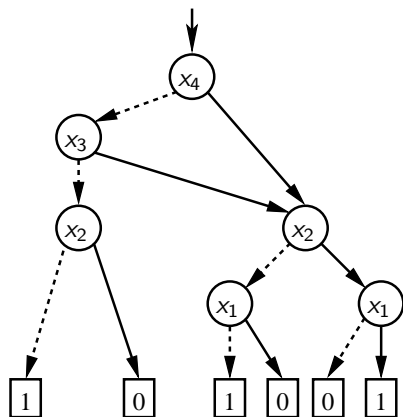
Ordered Binary Decision Diagrams (OBDDs)

- ▶ Data structure for Boolean functions
 $f: \{0, 1\}^n \rightarrow \{0, 1\}$
(Bryant, 1985)
- ▶ Inner nodes labeled with variables, left
by 0-/1-edges.
- ▶ Sink nodes correspond to value
 $f(x_1, \dots, x_n)$.
- ▶ Variables x_1, \dots, x_n are read w. r. t.
 $\pi \in \Sigma_n$.



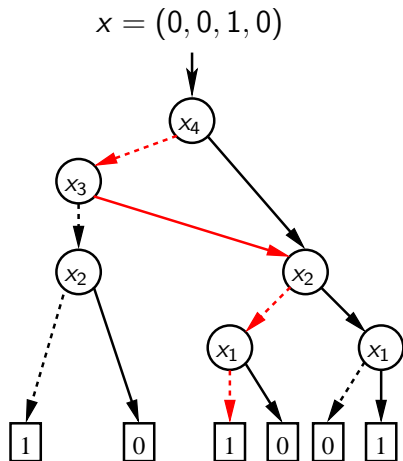
Ordered Binary Decision Diagrams (OBDDs)

- ▶ Data structure for Boolean functions
 $f: \{0, 1\}^n \rightarrow \{0, 1\}$
(Bryant, 1985)
- ▶ Inner nodes labeled with variables, left
by 0-/1-edges.
- ▶ Sink nodes correspond to value
 $f(x_1, \dots, x_n)$.
- ▶ Variables x_1, \dots, x_n are read w. r. t.
 $\pi \in \Sigma_n$.



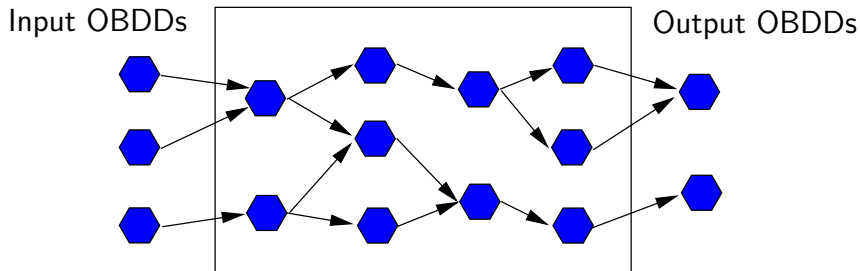
Ordered Binary Decision Diagrams (OBDDs)

- ▶ Data structure for Boolean functions
 $f: \{0, 1\}^n \rightarrow \{0, 1\}$
(Bryant, 1985)
- ▶ Inner nodes labeled with variables, left
by 0-/1-edges.
- ▶ Sink nodes correspond to value
 $f(x_1, \dots, x_n)$.
- ▶ Variables x_1, \dots, x_n are read w. r. t.
 $\pi \in \Sigma_n$.



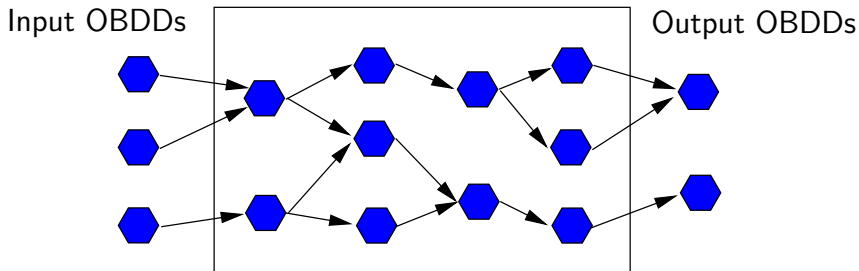
Symbolic Graph Algorithms

- ▶ Solve graph problems by functional operations:
 - ▶ Synthesis: \wedge , \vee , \oplus , $=$, ...
 - ▶ Quantification: $(\exists x) F(x, y)$, $(\forall x) F(x, y)$
 - ▶ Satisfiability, equivalence, variable replacement $F(x, 0)$
- ▶ Process large symbolic sets; avoid touching many nodes/edges.



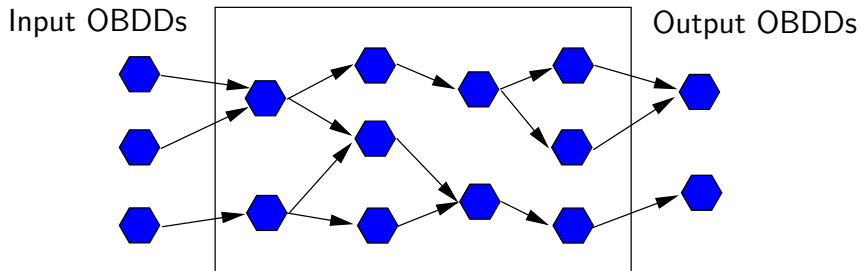
Symbolic Graph Algorithms

- ▶ Solve graph problems by functional operations:
 - ▶ Synthesis: \wedge , \vee , \oplus , $=$, \dots
 - ▶ Quantification: $(\exists x) F(x, y)$, $(\forall x) F(x, y)$
 - ▶ Satisfiability, equivalence, variable replacement $F(x, 0)$
- ▶ Process large symbolic sets; avoid touching many nodes/edges.



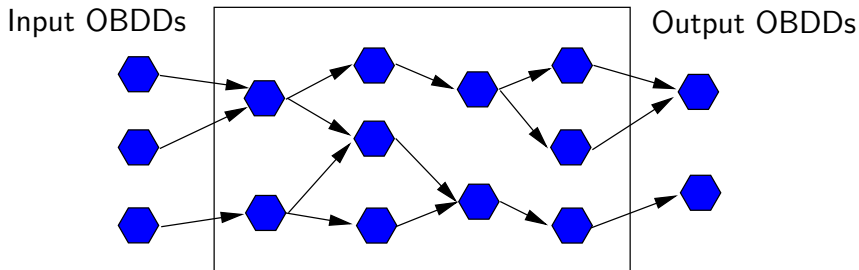
Symbolic Graph Algorithms

- ▶ Solve graph problems by functional operations:
 - ▶ Synthesis: \wedge , \vee , \oplus , $=$, \dots
 - ▶ Quantification: $(\exists x) F(x, y)$, $(\forall x) F(x, y)$
 - ▶ Satisfiability, equivalence, variable replacement $F(x, 0)$
- ▶ Process large symbolic sets; avoid touching many nodes/edges.



Symbolic Graph Algorithms

- ▶ Solve graph problems by functional operations:
 - ▶ Synthesis: \wedge , \vee , \oplus , $=$, \dots
 - ▶ Quantification: $(\exists x) F(x, y)$, $(\forall x) F(x, y)$
 - ▶ Satisfiability, equivalence, variable replacement $F(x, 0)$
- ▶ Process large symbolic sets; avoid touching many nodes/edges.



Previous Work

- ▶ VLSI: Reachability in state transition graphs
- ▶ Maximum flows (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting (Woelfel, 2003)
- ▶ Strongly connected components (Gentilini et al., 2003)
- ▶ Single-source shortest-paths (S., 2004)

This work's motivation:

Polylogarithmic runtime on graphs of special structure.

Previous Work

- ▶ VLSI: Reachability in state transition graphs
- ▶ Maximum flows (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting (Woelfel, 2003)
- ▶ Strongly connected components (Gentilini et al., 2003)
- ▶ Single-source shortest-paths (S., 2004)

This work's motivation:

Polylogarithmic runtime on graphs of special structure.

Previous Work

- ▶ VLSI: Reachability in state transition graphs
- ▶ Maximum flows (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting (Woelfel, 2003)
- ▶ Strongly connected components (Gentilini et al., 2003)
- ▶ Single-source shortest-paths (S., 2004)

This work's motivation:

Polylogarithmic runtime on graphs of special structure.

Previous Work

- ▶ VLSI: Reachability in state transition graphs
- ▶ Maximum flows (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting (Woelfel, 2003)
- ▶ Strongly connected components (Gentilini et al., 2003)
- ▶ Single-source shortest-paths (S., 2004)

This work's motivation:

Polylogarithmic runtime on graphs of special structure.

Previous Work

- ▶ VLSI: Reachability in state transition graphs
- ▶ Maximum flows (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting (Woelfel, 2003)
- ▶ Strongly connected components (Gentilini et al., 2003)
- ▶ Single-source shortest-paths (S., 2004)

This work's motivation:

Polylogarithmic runtime on graphs of special structure.

Previous Work

- ▶ VLSI: Reachability in state transition graphs
- ▶ Maximum flows (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting (Woelfel, 2003)
- ▶ Strongly connected components (Gentilini et al., 2003)
- ▶ Single-source shortest-paths (S., 2004)

This work's motivation:

Polylogarithmic runtime on graphs of special structure.

The Symbolic APSP-Problem—Definition and Approach

Input:

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

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

Output:

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

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

► Approach: Iteratively compute S^{i+1} from S^i with

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

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

The Symbolic APSP-Problem—Definition and Approach

Input:

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

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

Output:

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

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

► Approach: Iteratively compute S^{i+1} from S^i with

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

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

The Symbolic APSP-Problem—Definition and Approach

Input:

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

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

Output:

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

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

- ▶ Approach: Iteratively compute S^{i+1} from S^i with

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

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

The Symbolic APSP-Problem—Definition and Approach

Input:

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

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

Output:

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

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

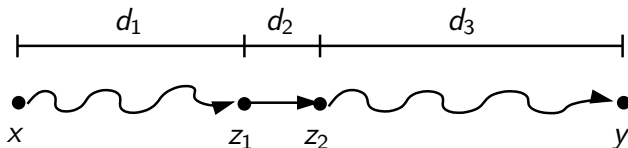
- ▶ Approach: Iteratively compute S^{i+1} from S^i with

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

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

Computing S^{i+1}

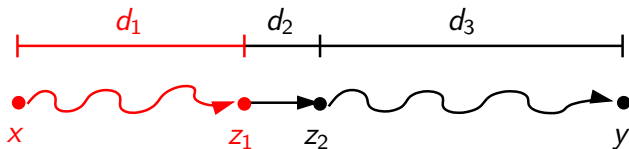
$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) [\dots]
 \end{aligned}$$



- $\log(|V| \cdot \ell^{\max})$ iterations with $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ ops. each
- Over-all: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ ops.

Computing S^{i+1}

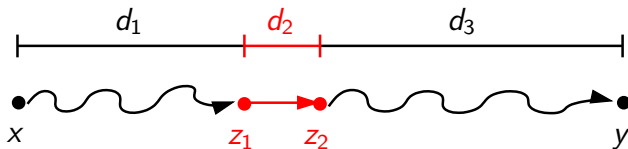
$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) [\dots]
 \end{aligned}$$



- $\log(|V| \cdot \ell^{\max})$ iterations with $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ ops. each
- Over-all: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ ops.

Computing S^{i+1}

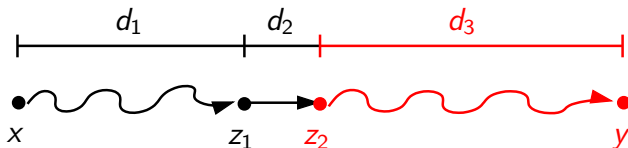
$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) [\dots]
 \end{aligned}$$



- $\log(|V| \cdot \ell^{\max})$ iterations with $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ ops. each
- Over-all: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ ops.

Computing S^{i+1}

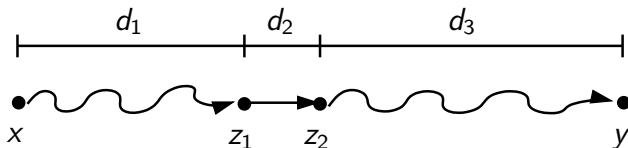
$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) [\dots]
 \end{aligned}$$



- $\log(|V| \cdot \ell^{\max})$ iterations with $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ ops. each
- Over-all: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ ops.

Computing S^{i+1}

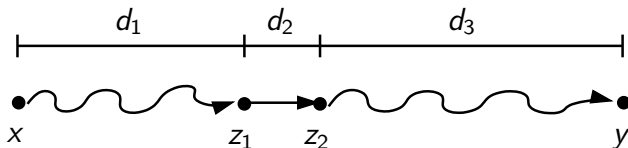
$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) \dots
 \end{aligned}$$



- $\log(|V| \cdot \ell^{\max})$ iterations with $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ ops. each
- Over-all: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ ops.

Computing S^{i+1}

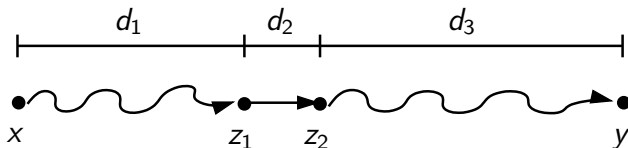
$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) [\dots]
 \end{aligned}$$



- $\log(|V| \cdot \ell^{\max})$ iterations with $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ ops. each
- Over-all: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ ops.

Computing S^{i+1}

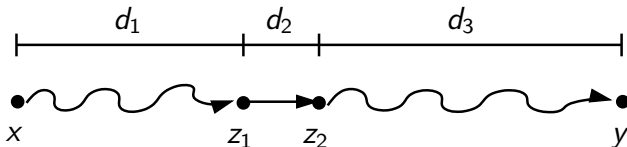
$$\begin{aligned}
 S^{i+1}(x, y, d) := & (\exists z_1, z_2, d_1, d_2, d_3) [(d_1 + d_2 + d_3 = d) \\
 & \wedge S^i(x, z_1, d_1) \\
 & \wedge C(z_1, z_2, d_2) \\
 & \wedge S^i(z_2, y, d_3)] \\
 & \wedge (d < 2^{i+1}) [\dots]
 \end{aligned}$$



- ▶ $\log(|V| \cdot \ell^{\max})$ iterations with $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ ops. each
- ▶ Over-all: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ ops.

Computing S^{i+1}

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



- ▶ $\log(|V| \cdot \ell^{\max})$ iterations with $\mathcal{O}(\log(|V| \cdot \ell^{\max}))$ ops. each
- ▶ Over-all: $\mathcal{O}(\log^2(|V| \cdot \ell^{\max}))$ ops.

Analysis of Symbolic Algorithms—Dreams and Reality

Reality up to now:

- ▶ Experimental evaluations
- ▶ Counting OBDD-operations
- ▶ Analysis on **very** special graphs

Dreams:

- ▶ Polynomial w. r. t. input OBDD-size
- ▶ Special input and output \Rightarrow low runtime

Result for APSP-algorithm:

Input/output has constant width \Rightarrow Polylogarithmic time/space
(w. r. t. $|V|$ and ℓ^{\max})

Analysis of Symbolic Algorithms—Dreams and Reality

Reality up to now:

- ▶ Experimental evaluations
- ▶ Counting OBDD-operations
- ▶ Analysis on **very** special graphs

Dreams:

- ▶ Polynomial w. r. t. input OBDD-size
- ▶ Special input **and** output \Rightarrow low runtime

Result for APSP-algorithm:

Input/output has constant width \Rightarrow Polylogarithmic time/space
(w. r. t. $|V|$ and ℓ^{\max})

Analysis of Symbolic Algorithms—Dreams and Reality

Reality up to now:

- ▶ Experimental evaluations
- ▶ Counting OBDD-operations
- ▶ Analysis on **very** special graphs

Dreams:

- ▶ Polynomial w. r. t. input OBDD-size
- ▶ Special input and output \Rightarrow low runtime

Result for APSP-algorithm:

Input/output has constant width \Rightarrow Polylogarithmic time/space
(w. r. t. $|V|$ and ℓ^{\max})

Analysis of Symbolic Algorithms—Dreams and Reality

Reality up to now:

- ▶ Experimental evaluations
- ▶ Counting OBDD-operations
- ▶ Analysis on **very** special graphs

Dreams:

- ▶ Polynomial w. r. t. input OBDD-size
- ▶ Special input **and** output \Rightarrow low runtime

Result for APSP-algorithm:

Input/output has constant width \Rightarrow Polylogarithmic time/space
(w. r. t. $|V|$ and ℓ^{\max})

Analysis of Symbolic Algorithms—Dreams and Reality

Reality up to now:

- ▶ Experimental evaluations
- ▶ Counting OBDD-operations
- ▶ Analysis on **very** special graphs

Dreams:

- ▶ ~~Polynomial w. r. t. input OBDD-size~~ \Leftarrow **PSPACE-complete**
- ▶ Special input **and** output \Rightarrow low runtime

Result for APSP-algorithm:

Input/output has constant width \Rightarrow Polylogarithmic time/space
(w. r. t. $|V|$ and ℓ^{\max})

Analysis of Symbolic Algorithms—Dreams and Reality

Reality up to now:

- ▶ Experimental evaluations
- ▶ Counting OBDD-operations
- ▶ Analysis on **very** special graphs

Dreams:

- ▶ ~~Polynomial w. r. t. input OBDD-size~~ \Leftarrow **PSPACE-complete**
- ▶ Special input **and** output \Rightarrow low runtime

Result for APSP-algorithm:

Input/output has constant width \Rightarrow Polylogarithmic time/space
(w. r. t. $|V|$ and ℓ^{\max})

Analysis of Symbolic Algorithms—Dreams and Reality

Reality up to now:

- ▶ Experimental evaluations
- ▶ Counting OBDD-operations
- ▶ Analysis on **very** special graphs

Dreams:

- ▶ ~~Polynomial w. r. t. input OBDD-size~~ \Leftarrow **PSPACE-complete**
- ▶ Special input **and** output \Rightarrow low runtime

Result for APSP-algorithm:

Input/output has constant width \Rightarrow Polylogarithmic time/space
(w. r. t. $|V|$ and ℓ^{\max})

Bounded-Width Functions

Definition.

Boolean functions $f = (f_n)_n$ have **bounded width** if $\exists b$: Complete OBDDs $F = (F_n)_n$ all have width b .

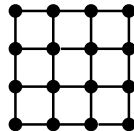
G_1



G_2



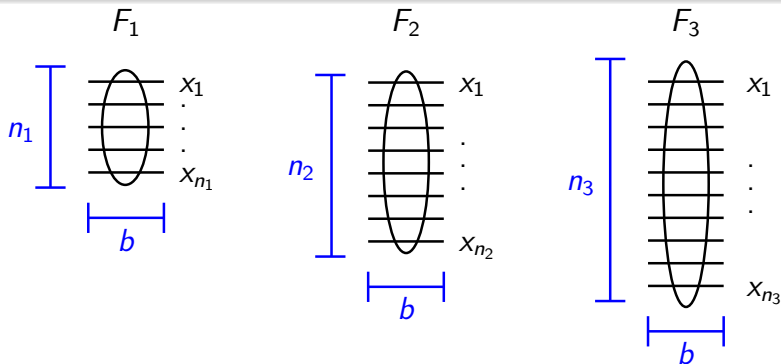
G_3



Bounded-Width Functions

Definition.

Boolean functions $f = (f_n)_n$ have **bounded width** if $\exists b$: Complete OBDDs $F = (F_n)_n$ all have width b .



The APSP-Algorithm on Bounded-Width Functions

- ▶ Consider graph sequence $G = (G_n)_n$.
- ▶ Let $C = (C_n)_n$ and $S = (S_n)_n$ be of bounded width b .

Theorem.

The symbolic APSP-algorithm computes S_n from C_n in time/space $\mathcal{O}(\log^3(|V_n| \cdot \ell_n^{\max}) \cdot \alpha(b))$.

Sketch of proof.

- ▶ Show width $\mathcal{O}(b)$ for $S_n^i(x, y, d) = S_n(x, y, d) \wedge (d < 2^i)$.
- ▶ Show time $\mathcal{O}(\log(|V_n| \cdot \ell_n^{\max}) \cdot \alpha(b))$ for $S^i \rightarrow S^{i+1}$ -step.

The APSP-Algorithm on Bounded-Width Functions

- ▶ Consider graph sequence $G = (G_n)_n$.
- ▶ Let $C = (C_n)_n$ and $S = (S_n)_n$ be of bounded width b .

Theorem.

The symbolic APSP-algorithm computes S_n from C_n in time/space $\mathcal{O}(\log^3(|V_n| \cdot \ell_n^{\max}) \cdot \alpha(b))$.

Sketch of proof.

- ▶ Show width $\mathcal{O}(b)$ for $S_n^i(x, y, d) = S_n(x, y, d) \wedge (d < 2^i)$.
- ▶ Show time $\mathcal{O}(\log(|V_n| \cdot \ell_n^{\max}) \cdot \alpha(b))$ for $S^i \rightarrow S^{i+1}$ -step.

The APSP-Algorithm on Bounded-Width Functions

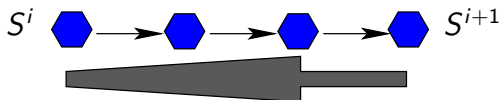
- ▶ Consider graph sequence $G = (G_n)_n$.
- ▶ Let $C = (C_n)_n$ and $S = (S_n)_n$ be of bounded width b .

Theorem.

The symbolic APSP-algorithm computes S_n from C_n in time/space $\mathcal{O}(\log^3(|V_n| \cdot \ell_n^{\max}) \cdot \alpha(b))$.

Sketch of proof.

- ▶ Show width $\mathcal{O}(b)$ for $S_n^i(x, y, d) = S_n(x, y, d) \wedge (d < 2^i)$.
- ▶ Show time $\mathcal{O}(\log(|V_n| \cdot \ell_n^{\max}) \cdot \alpha(b))$ for $S^i \rightarrow S^{i+1}$ -step.



The APSP-Algorithm on Bounded-Width Functions

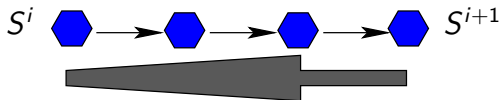
- ▶ Consider graph sequence $G = (G_n)_n$.
- ▶ Let $C = (C_n)_n$ and $S = (S_n)_n$ be of bounded width b .

Theorem.

The symbolic APSP-algorithm computes S_n from C_n in time/space $\mathcal{O}(\log^3(|V_n| \cdot \ell_n^{\max}) \cdot \alpha(b))$.

Sketch of proof.

- ▶ Show width $\mathcal{O}(b)$ for $S_n^i(x, y, d) = S_n(x, y, d) \wedge (d < 2^i)$.
- ▶ Show time $\mathcal{O}(\log(|V_n| \cdot \ell_n^{\max}) \cdot \alpha(b))$ for $S^i \rightarrow S^{i+1}$ -step.



Graphs with Characteristic Bounded-Width Functions

Which graphs have bounded-width functions?

- ▶ Basic graphs: Paths, stars, fans, wheels, grids, cliques, bicliques, ...
- ▶ Restricted threshold graphs
- ▶ Graphs of threshold functions:

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

- ▶ Bounded-width property is closed under ...
 - ▶ Join operations: $V_3 := V_1 \cup V_2$.
 - ▶ Product operations: $V_3 := V_1 \times V_2$.

Graphs with Characteristic Bounded-Width Functions

Which graphs have bounded-width functions?

- ▶ Basic graphs: Paths, stars, fans, wheels, grids, cliques, bicliques, ...
- ▶ Restricted threshold graphs
- ▶ Graphs of threshold functions:

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

- ▶ Bounded-width property is closed under ...
 - ▶ Join operations: $V_3 := V_1 \cup V_2$.
 - ▶ Product operations: $V_3 := V_1 \times V_2$.

Graphs with Characteristic Bounded-Width Functions

Which graphs have bounded-width functions?

- ▶ Basic graphs: Paths, stars, fans, wheels, grids, cliques, bicliques, ...
- ▶ Restricted threshold graphs
- ▶ Graphs of threshold functions:

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

- ▶ Bounded-width property is closed under ...
 - ▶ Join operations: $V_3 := V_1 \cup V_2$.
 - ▶ Product operations: $V_3 := V_1 \times V_2$.

Graphs with Characteristic Bounded-Width Functions

Which graphs have bounded-width functions?

- ▶ Basic graphs: Paths, stars, fans, wheels, grids, cliques, bicliques, ...
- ▶ Restricted threshold graphs
- ▶ Graphs of threshold functions:

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

- ▶ Bounded-width property is closed under ...
 - ▶ Join operations: $V_3 := V_1 \cup V_2$.
 - ▶ Product operations: $V_3 := V_1 \times V_2$.

Graphs with Characteristic Bounded-Width Functions

Which graphs have bounded-width functions?

- ▶ Basic graphs: Paths, stars, fans, wheels, grids, cliques, bicliques, ...
- ▶ Restricted threshold graphs
- ▶ Graphs of threshold functions:

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

- ▶ Bounded-width property is closed under ...
 - ▶ Join operations: $V_3 := V_1 \cup V_2$.
 - ▶ Product operations: $V_3 := V_1 \times V_2$.

Graphs with Characteristic Bounded-Width Functions

Which graphs have bounded-width functions?

- ▶ Basic graphs: Paths, stars, fans, wheels, grids, cliques, bicliques, ...
- ▶ Restricted threshold graphs
- ▶ Graphs of threshold functions:

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

- ▶ Bounded-width property is closed under ...
 - ▶ Join operations: $V_3 := V_1 \cup V_2$.
 - ▶ Product operations: $V_3 := V_1 \times V_2$.

An Alternative Approach

- ▶ Why to restrict to positive edge weights?
- ▶ Alternative: $S^i(x, y, d)$ has at most 2^i edges.

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

- ▶ Counterexample:
 - ▶ $C = (C_n)_n$ and $S = (S_n)_n$ have bounded width.
 - ▶ OBDDs $S^i = (S_n^i)_n$ have no constant width bound.
- ▶ Conjecture: $\ell(e) > 0$ is necessary for polylog. runtime.

An Alternative Approach

- ▶ Why to restrict to positive edge weights?
- ▶ Alternative: $S^i(x, y, d)$ has at most 2^i edges.

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

- ▶ Counterexample:
 - ▶ $C = (C_n)_n$ and $S = (S_n)_n$ have bounded width.
 - ▶ OBDDs $S^i = (S_n^i)_n$ have no constant width bound.
- ▶ Conjecture: $\ell(e) > 0$ is necessary for polylog. runtime.

An Alternative Approach

- ▶ Why to restrict to positive edge weights?
- ▶ Alternative: $S^i(x, y, d)$ has at most 2^i edges.

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

- ▶ Counterexample:
 - ▶ $C = (C_n)_n$ and $S = (S_n)_n$ have bounded width.
 - ▶ OBDDs $S^i = (S_n^i)_n$ have no constant width bound.
- ▶ Conjecture: $\ell(e) > 0$ is necessary for polylog. runtime.

An Alternative Approach

- ▶ Why to restrict to positive edge weights?
- ▶ Alternative: $S^i(x, y, d)$ has at most 2^i edges.

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

- ▶ Counterexample:
 - ▶ $C = (C_n)_n$ and $S = (S_n)_n$ have bounded width.
 - ▶ OBDDs $S^i = (S_n^i)_n$ have no constant width bound.
- ▶ Conjecture: $\ell(e) > 0$ is necessary for polylog. runtime.

An Alternative Approach

- ▶ Why to restrict to positive edge weights?
- ▶ Alternative: $S^i(x, y, d)$ has at most 2^i edges.

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

- ▶ Counterexample:
 - ▶ $C = (C_n)_n$ and $S = (S_n)_n$ have bounded width.
 - ▶ OBDDs $S^i = (S_n^i)_n$ have **no** constant width bound.
- ▶ Conjecture: $\ell(e) > 0$ is necessary for polylog. runtime.

An Alternative Approach

- ▶ Why to restrict to positive edge weights?
- ▶ Alternative: $S^i(x, y, d)$ has at most 2^i edges.

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

- ▶ Counterexample:
 - ▶ $C = (C_n)_n$ and $S = (S_n)_n$ have bounded width.
 - ▶ OBDDs $S^i = (S_n^i)_n$ have **no** constant width bound.
- ▶ Conjecture: $\ell(e) > 0$ is necessary for polylog. runtime.

Related Problems

Symbolic APSP-algorithm can be adapted to . . .

- ▶ all-pairs lightest shortest paths.
- ▶ all-pairs almost shortest paths.
- ▶ all-pairs small-stretch paths.
- ▶ dynamic edge weight decrease.

Related Problems

Symbolic APSP-algorithm can be adapted to . . .

- ▶ all-pairs lightest shortest paths.
- ▶ all-pairs almost shortest paths.
- ▶ all-pairs small-stretch paths.
- ▶ dynamic edge weight decrease.

Related Problems

Symbolic APSP-algorithm can be adapted to . . .

- ▶ all-pairs lightest shortest paths.
- ▶ all-pairs almost shortest paths.
- ▶ all-pairs small-stretch paths.
- ▶ dynamic edge weight decrease.

Related Problems

Symbolic APSP-algorithm can be adapted to . . .

- ▶ all-pairs lightest shortest paths.
- ▶ all-pairs almost shortest paths.
- ▶ all-pairs small-stretch paths.
- ▶ dynamic edge weight decrease.

Thank you for listening!

