

Experimental Studies of Symbolic Shortest Path Algorithms

Daniel Sawitzki

Computer Science 2
University of Dortmund, Germany

Workshop on Efficient and Experimental Algorithms
May 25 to 28, 2004

- ▶ Introduction to Symbolic Methods
- ▶ Symbolic Shortest-Path Algorithms
- ▶ Experimental Analysis on Random Graphs
- ▶ Summary

- ▶ Introduction to Symbolic Methods
- ▶ Symbolic Shortest-Path Algorithms
- ▶ Experimental Analysis on Random Graphs
- ▶ Summary

- ▶ Introduction to Symbolic Methods
- ▶ Symbolic Shortest-Path Algorithms
- ▶ Experimental Analysis on Random Graphs
- ▶ Summary

- ▶ Introduction to Symbolic Methods
- ▶ Symbolic Shortest-Path Algorithms
- ▶ Experimental Analysis on Random Graphs
- ▶ Summary

Motivation

- ▶ Applications produce “large” graphs:
 - ▶ Logic design and verification (e. g. 10^{30} nodes)
 - ▶ Modeling traffic or the WWW
 - ▶ ...
- ▶ Possible conflicts with resource limitations:
 - ▶ Graphs do not fit into internal memory.
 - ▶ Even efficient algorithms get intractable.
- ▶ Observation: Many applications produce large, but structured graphs.
- ▶ Heuristic approach: **Symbolic algorithms**
 1. Consider graph G as a **characteristic** Boolean function C .
 2. Represent C by a compact data structure.
 3. Solve problem in G by few and **efficient** operations on C .

Motivation

- ▶ Applications produce “large” graphs:
 - ▶ Logic design and verification (e. g. 10^{30} nodes)
 - ▶ Modeling traffic or the WWW
 - ▶ ...
- ▶ Possible conflicts with resource limitations:
 - ▶ Graphs do not fit into internal memory.
 - ▶ Even efficient algorithms get intractable.
- ▶ Observation: Many applications produce large, but structured graphs.
- ▶ Heuristic approach: **Symbolic algorithms**
 1. Consider graph G as a **characteristic** Boolean function C .
 2. Represent C by a compact data structure.
 3. Solve problem in G by few and **efficient** operations on C .

Motivation

- ▶ Applications produce “large” graphs:
 - ▶ Logic design and verification (e. g. 10^{30} nodes)
 - ▶ Modeling traffic or the WWW
 - ▶ ...
- ▶ Possible conflicts with resource limitations:
 - ▶ Graphs do not fit into internal memory.
 - ▶ Even efficient algorithms get intractable.
- ▶ Observation: Many applications produce large, but structured graphs.
- ▶ Heuristic approach: **Symbolic algorithms**
 1. Consider graph G as a **characteristic** Boolean function C .
 2. Represent C by a compact data structure.
 3. Solve problem in G by few and **efficient** operations on C .

Motivation

- ▶ Applications produce “large” graphs:
 - ▶ Logic design and verification (e. g. 10^{30} nodes)
 - ▶ Modeling traffic or the WWW
 - ▶ ...
- ▶ Possible conflicts with resource limitations:
 - ▶ Graphs do not fit into internal memory.
 - ▶ Even efficient algorithms get intractable.
- ▶ Observation: Many applications produce large, but structured graphs.
- ▶ Heuristic approach: **Symbolic algorithms**
 1. Consider graph G as a **characteristic** Boolean function C .
 2. Represent C by a compact data structure.
 3. Solve problem in G by few and **efficient** operations on C .

Motivation

- ▶ Applications produce “large” graphs:
 - ▶ Logic design and verification (e. g. 10^{30} nodes)
 - ▶ Modeling traffic or the WWW
 - ▶ ...
- ▶ Possible conflicts with resource limitations:
 - ▶ Graphs do not fit into internal memory.
 - ▶ Even efficient algorithms get intractable.
- ▶ Observation: Many applications produce large, but structured graphs.
- ▶ Heuristic approach: **Symbolic algorithms**
 1. Consider graph G as a **characteristic** Boolean function C .
 2. Represent C by a compact data structure.
 3. Solve problem in G by few and **efficient** operations on C .

Motivation

- ▶ Applications produce “large” graphs:
 - ▶ Logic design and verification (e. g. 10^{30} nodes)
 - ▶ Modeling traffic or the WWW
 - ▶ ...
- ▶ Possible conflicts with resource limitations:
 - ▶ Graphs do not fit into internal memory.
 - ▶ Even efficient algorithms get intractable.
- ▶ Observation: Many applications produce large, but structured graphs.
- ▶ Heuristic approach: **Symbolic algorithms**
 1. Consider graph G as a **characteristic** Boolean function C .
 2. Represent C by a compact data structure.
 3. Solve problem in G by few and **efficient** operations on C .

Motivation

- ▶ Applications produce “large” graphs:
 - ▶ Logic design and verification (e. g. 10^{30} nodes)
 - ▶ Modeling traffic or the WWW
 - ▶ ...
- ▶ Possible conflicts with resource limitations:
 - ▶ Graphs do not fit into internal memory.
 - ▶ Even efficient algorithms get intractable.
- ▶ Observation: Many applications produce large, but structured graphs.
- ▶ Heuristic approach: **Symbolic algorithms**
 1. Consider graph G as a **characteristic** Boolean function C .
 2. Represent C by a compact data structure.
 3. Solve problem in G by few and **efficient** operations on C .

Motivation

- ▶ Applications produce “large” graphs:
 - ▶ Logic design and verification (e. g. 10^{30} nodes)
 - ▶ Modeling traffic or the WWW
 - ▶ ...
- ▶ Possible conflicts with resource limitations:
 - ▶ Graphs do not fit into internal memory.
 - ▶ Even efficient algorithms get intractable.
- ▶ Observation: Many applications produce large, but structured graphs.
- ▶ Heuristic approach: **Symbolic algorithms**
 1. Consider graph G as a characteristic Boolean function C .
 2. Represent C by a compact data structure.
 3. Solve problem in G by few and efficient operations on C .

Motivation

- ▶ Applications produce “large” graphs:
 - ▶ Logic design and verification (e. g. 10^{30} nodes)
 - ▶ Modeling traffic or the WWW
 - ▶ ...
- ▶ Possible conflicts with resource limitations:
 - ▶ Graphs do not fit into internal memory.
 - ▶ Even efficient algorithms get intractable.
- ▶ Observation: Many applications produce large, but structured graphs.
- ▶ Heuristic approach: **Symbolic algorithms**
 1. Consider graph G as a **characteristic** Boolean function C .
 2. Represent C by a compact data structure.
 3. Solve problem in G by **few** and **efficient** operations on C .

Motivation

- ▶ Applications produce “large” graphs:
 - ▶ Logic design and verification (e. g. 10^{30} nodes)
 - ▶ Modeling traffic or the WWW
 - ▶ ...
- ▶ Possible conflicts with resource limitations:
 - ▶ Graphs do not fit into internal memory.
 - ▶ Even efficient algorithms get intractable.
- ▶ Observation: Many applications produce large, but structured graphs.
- ▶ Heuristic approach: **Symbolic algorithms**
 1. Consider graph G as a **characteristic** Boolean function C .
 2. Represent C by a compact data structure.
 3. Solve problem in G by **few** and **efficient** operations on C .

Motivation

- ▶ Applications produce “large” graphs:
 - ▶ Logic design and verification (e. g. 10^{30} nodes)
 - ▶ Modeling traffic or the WWW
 - ▶ ...
- ▶ Possible conflicts with resource limitations:
 - ▶ Graphs do not fit into internal memory.
 - ▶ Even efficient algorithms get intractable.
- ▶ Observation: Many applications produce large, but structured graphs.
- ▶ Heuristic approach: **Symbolic algorithms**
 1. Consider graph G as a **characteristic** Boolean function C .
 2. Represent C by a compact data structure.
 3. Solve problem in G by **few** and **efficient** operations on C .

Motivation

- ▶ Applications produce “large” graphs:
 - ▶ Logic design and verification (e. g. 10^{30} nodes)
 - ▶ Modeling traffic or the WWW
 - ▶ ...
- ▶ Possible conflicts with resource limitations:
 - ▶ Graphs do not fit into internal memory.
 - ▶ Even efficient algorithms get intractable.
- ▶ Observation: Many applications produce large, but structured graphs.
- ▶ Heuristic approach: **Symbolic algorithms**
 1. Consider graph G as a **characteristic** Boolean function C .
 2. Represent C by a compact data structure.
 3. Solve problem in G by **few** and **efficient** operations on C .

Symbolic Graph Representation

- ▶ Consider directed weighted graph $G = (V, E, \ell)$ with

$$N := |V|, \quad M := |E|, \quad 0 \leq \ell(e) \leq T.$$

- ▶ Define $C: V \times V \times [T] \rightarrow \{0, 1\}$ by

$$C(u, v, d) = 1 \Leftrightarrow [(u, v) \in E] \wedge [\ell(u, v) = d].$$

- ▶ Graph-based data structure for Boolean functions:
Ordered Binary Decision Diagrams (OBDDs) (Bryant, 1985)
- ▶ OBDD C has worst-case size $\mathcal{O}((NT)^3 / \log(NT))$.
- ▶ Structured graphs hopefully have sublinear OBDD-size $o(NT)$.

Symbolic Graph Representation

- ▶ Consider directed weighted graph $G = (V, E, \ell)$ with

$$N := |V|, \quad M := |E|, \quad 0 \leq \ell(e) \leq T.$$

- ▶ Define $C: V \times V \times [T] \rightarrow \{0, 1\}$ by

$$C(u, v, d) = 1 \Leftrightarrow [(u, v) \in E] \wedge [\ell(u, v) = d].$$

- ▶ Graph-based data structure for Boolean functions:
Ordered Binary Decision Diagrams (OBDDs) (Bryant, 1985)
- ▶ OBDD C has worst-case size $\mathcal{O}((NT)^3 / \log(NT))$.
- ▶ Structured graphs hopefully have sublinear OBDD-size $o(NT)$.

Symbolic Graph Representation

- ▶ Consider directed weighted graph $G = (V, E, \ell)$ with

$$N := |V|, \quad M := |E|, \quad 0 \leq \ell(e) \leq T.$$

- ▶ Define $C: V \times V \times [T] \rightarrow \{0, 1\}$ by

$$C(u, v, d) = 1 \Leftrightarrow [(u, v) \in E] \wedge [\ell(u, v) = d].$$

- ▶ Graph-based data structure for Boolean functions:

Ordered Binary Decision Diagrams (OBDDs) (Bryant, 1985)

- ▶ OBDD C has worst-case size $\mathcal{O}((NT)^3 / \log(NT))$.
- ▶ Structured graphs hopefully have sublinear OBDD-size $o(NT)$.

Symbolic Graph Representation

- ▶ Consider directed weighted graph $G = (V, E, \ell)$ with

$$N := |V|, \quad M := |E|, \quad 0 \leq \ell(e) \leq T.$$

- ▶ Define $C: V \times V \times [T] \rightarrow \{0, 1\}$ by

$$C(u, v, d) = 1 \Leftrightarrow [(u, v) \in E] \wedge [\ell(u, v) = d].$$

- ▶ Graph-based data structure for Boolean functions:
Ordered Binary Decision Diagrams (OBDDs) (Bryant, 1985)
 - ▶ OBDD C has worst-case size $\mathcal{O}((NT)^3 / \log(NT))$.
 - ▶ Structured graphs hopefully have sublinear OBDD-size $o(NT)$.

Symbolic Graph Representation

- ▶ Consider directed weighted graph $G = (V, E, \ell)$ with

$$N := |V|, \quad M := |E|, \quad 0 \leq \ell(e) \leq T.$$

- ▶ Define $C: V \times V \times [T] \rightarrow \{0, 1\}$ by

$$C(u, v, d) = 1 \Leftrightarrow [(u, v) \in E] \wedge [\ell(u, v) = d].$$

- ▶ Graph-based data structure for Boolean functions:
Ordered Binary Decision Diagrams (OBDDs) (Bryant, 1985)
- ▶ OBDD C has worst-case size $\mathcal{O}((NT)^3 / \log(NT))$.
- ▶ Structured graphs hopefully have sublinear OBDD-size $o(NT)$.

Symbolic Graph Representation

- ▶ Consider directed weighted graph $G = (V, E, \ell)$ with

$$N := |V|, \quad M := |E|, \quad 0 \leq \ell(e) \leq T.$$

- ▶ Define $C: V \times V \times [T] \rightarrow \{0, 1\}$ by

$$C(u, v, d) = 1 \Leftrightarrow [(u, v) \in E] \wedge [\ell(u, v) = d].$$

- ▶ Graph-based data structure for Boolean functions:
Ordered Binary Decision Diagrams (OBDDs) (Bryant, 1985)
- ▶ OBDD C has worst-case size $\mathcal{O}((NT)^3 / \log(NT))$.
- ▶ Structured graphs hopefully have sublinear OBDD-size $o(NT)$.

Symbolic Graph Algorithms

- ▶ **Symbolic algorithms** work on the OBDDs of graphs.
- ▶ Are restricted to functional operations:
 - ▶ Satisfiability, equivalence, variable replacement
 - ▶ Synthesis $F \otimes G$, $\otimes \in \{\wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow\}$
 - ▶ Quantifications $(\exists x)F$, $(\forall x)F$
- ▶ Criteria for efficiency:
 - ▶ Runtime: Sum of sizes of generated OBDDs
 - ▶ Space: Maximum occurring OBDD-size
- ▶ Challenges in algorithm analysis:
 - ▶ $\log N$ operations may cause exponential blow-up.
 - ▶ OBDD-sizes are difficult to analyze.

Symbolic Graph Algorithms

- ▶ **Symbolic algorithms** work on the OBDDs of graphs.
- ▶ Are restricted to functional operations:
 - ▶ Satisfiability, equivalence, variable replacement
 - ▶ Synthesis $F \otimes G$, $\otimes \in \{\wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow\}$
 - ▶ Quantifications $(\exists x)F$, $(\forall x)F$
- ▶ Criteria for efficiency:
 - ▶ Runtime: Sum of sizes of generated OBDDs
 - ▶ Space: Maximum occurring OBDD-size
- ▶ Challenges in algorithm analysis:
 - ▶ $\log N$ operations may cause exponential blow-up.
 - ▶ OBDD-sizes are difficult to analyze.

Symbolic Graph Algorithms

- ▶ **Symbolic algorithms** work on the OBDDs of graphs.
- ▶ Are restricted to functional operations:
 - ▶ Satisfiability, equivalence, variable replacement
 - ▶ Synthesis $F \otimes G$, $\otimes \in \{\wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow\}$
 - ▶ Quantifications $(\exists x)F$, $(\forall x)F$
- ▶ Criteria for efficiency:
 - ▶ Runtime: Sum of sizes of generated OBDDs
 - ▶ Space: Maximum occurring OBDD-size
- ▶ Challenges in algorithm analysis:
 - ▶ $\log N$ operations may cause exponential blow-up.
 - ▶ OBDD-sizes are difficult to analyze.

Symbolic Graph Algorithms

- ▶ **Symbolic algorithms** work on the OBDDs of graphs.
- ▶ Are restricted to functional operations:
 - ▶ Satisfiability, equivalence, variable replacement
 - ▶ Synthesis $F \otimes G$, $\otimes \in \{\wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow\}$
 - ▶ Quantifications $(\exists x)F$, $(\forall x)F$
- ▶ Criteria for efficiency:
 - ▶ Runtime: Sum of sizes of generated OBDDs
 - ▶ Space: Maximum occurring OBDD-size
- ▶ Challenges in algorithm analysis:
 - ▶ $\log N$ operations may cause exponential blow-up.
 - ▶ OBDD-sizes are difficult to analyze.

Symbolic Graph Algorithms

- ▶ **Symbolic algorithms** work on the OBDDs of graphs.
- ▶ Are restricted to functional operations:
 - ▶ Satisfiability, equivalence, variable replacement
 - ▶ Synthesis $F \otimes G$, $\otimes \in \{\wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow\}$
 - ▶ Quantifications $(\exists x)F$, $(\forall x)F$
- ▶ Criteria for efficiency:
 - ▶ Runtime: Sum of sizes of generated OBDDs
 - ▶ Space: Maximum occurring OBDD-size
- ▶ Challenges in algorithm analysis:
 - ▶ $\log N$ operations may cause exponential blow-up.
 - ▶ OBDD-sizes are difficult to analyze.

Symbolic Graph Algorithms

- ▶ **Symbolic algorithms** work on the OBDDs of graphs.
- ▶ Are restricted to functional operations:
 - ▶ Satisfiability, equivalence, variable replacement
 - ▶ Synthesis $F \otimes G$, $\otimes \in \{\wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow\}$
 - ▶ Quantifications $(\exists x)F$, $(\forall x)F$
- ▶ **Criteria for efficiency:**
 - ▶ Runtime: Sum of sizes of generated OBDDs
 - ▶ Space: Maximum occurring OBDD-size
- ▶ **Challenges in algorithm analysis:**
 - ▶ $\log N$ operations may cause exponential blow-up.
 - ▶ OBDD-sizes are difficult to analyze.

Symbolic Graph Algorithms

- ▶ **Symbolic algorithms** work on the OBDDs of graphs.
- ▶ Are restricted to functional operations:
 - ▶ Satisfiability, equivalence, variable replacement
 - ▶ Synthesis $F \otimes G$, $\otimes \in \{\wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow\}$
 - ▶ Quantifications $(\exists x)F$, $(\forall x)F$
- ▶ Criteria for efficiency:
 - ▶ Runtime: Sum of sizes of generated OBDDs
 - ▶ Space: Maximum occurring OBDD-size
- ▶ Challenges in algorithm analysis:
 - ▶ $\log N$ operations may cause exponential blow-up.
 - ▶ OBDD-sizes are difficult to analyze.

Symbolic Graph Algorithms

- ▶ **Symbolic algorithms** work on the OBDDs of graphs.
- ▶ Are restricted to functional operations:
 - ▶ Satisfiability, equivalence, variable replacement
 - ▶ Synthesis $F \otimes G$, $\otimes \in \{\wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow\}$
 - ▶ Quantifications $(\exists x)F$, $(\forall x)F$
- ▶ Criteria for efficiency:
 - ▶ Runtime: Sum of sizes of generated OBDDs
 - ▶ Space: Maximum occurring OBDD-size
- ▶ Challenges in algorithm analysis:
 - ▶ $\log N$ operations may cause exponential blow-up.
 - ▶ OBDD-sizes are difficult to analyze.

Symbolic Graph Algorithms

- ▶ **Symbolic algorithms** work on the OBDDs of graphs.
- ▶ Are restricted to functional operations:
 - ▶ Satisfiability, equivalence, variable replacement
 - ▶ Synthesis $F \otimes G$, $\otimes \in \{\wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow\}$
 - ▶ Quantifications $(\exists x)F$, $(\forall x)F$
- ▶ Criteria for efficiency:
 - ▶ Runtime: Sum of sizes of generated OBDDs
 - ▶ Space: Maximum occurring OBDD-size
- ▶ Challenges in algorithm analysis:
 - ▶ $\log N$ operations may cause exponential blow-up.
 - ▶ OBDD-sizes are difficult to analyze.

Symbolic Graph Algorithms

- ▶ **Symbolic algorithms** work on the OBDDs of graphs.
- ▶ Are restricted to functional operations:
 - ▶ Satisfiability, equivalence, variable replacement
 - ▶ Synthesis $F \otimes G$, $\otimes \in \{\wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow\}$
 - ▶ Quantifications $(\exists x)F$, $(\forall x)F$
- ▶ Criteria for efficiency:
 - ▶ Runtime: Sum of sizes of generated OBDDs
 - ▶ Space: Maximum occurring OBDD-size
- ▶ Challenges in algorithm analysis:
 - ▶ $\log N$ operations may cause exponential blow-up.
 - ▶ OBDD-sizes are difficult to analyze.

Symbolic Graph Algorithms

- ▶ **Symbolic algorithms** work on the OBDDs of graphs.
- ▶ Are restricted to functional operations:
 - ▶ Satisfiability, equivalence, variable replacement
 - ▶ Synthesis $F \otimes G$, $\otimes \in \{\wedge, \vee, \oplus, \Rightarrow, \Leftrightarrow\}$
 - ▶ Quantifications $(\exists x)F$, $(\forall x)F$
- ▶ Criteria for efficiency:
 - ▶ Runtime: Sum of sizes of generated OBDDs
 - ▶ Space: Maximum occurring OBDD-size
- ▶ Challenges in algorithm analysis:
 - ▶ $\log N$ operations may cause exponential blow-up.
 - ▶ OBDD-sizes are difficult to analyze.

Previous Work

- ▶ Practical research in circuit verification:
 - ▶ Symbolic reachability analysis for state graphs
- ▶ Flow maximization: (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting: (Woelfel, 2003)
- ▶ Strongly connected components: (Gentilini et al., 2003)
- ▶ Shortest paths: (Bahar et al., 1993), (S., 2004)
- ▶ There are mainly two different design goals:
 1. Sublinear (polylogarithmic) runtime on special graphs
 2. Acceptable average-case behavior

Previous Work

- ▶ Practical research in circuit verification:
 - ▶ Symbolic reachability analysis for state graphs
- ▶ Flow maximization: (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting: (Woelfel, 2003)
- ▶ Strongly connected components: (Gentilini et al., 2003)
- ▶ Shortest paths: (Bahar et al., 1993), (S., 2004)
- ▶ There are mainly two different design goals:
 1. Sublinear (polylogarithmic) runtime on special graphs
 2. Acceptable average-case behavior

Previous Work

- ▶ Practical research in circuit verification:
 - ▶ Symbolic reachability analysis for state graphs
- ▶ Flow maximization: (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting: (Woelfel, 2003)
- ▶ Strongly connected components: (Gentilini et al., 2003)
- ▶ Shortest paths: (Bahar et al., 1993), (S., 2004)
- ▶ There are mainly two different design goals:
 1. Sublinear (polylogarithmic) runtime on special graphs
 2. Acceptable average-case behavior

Previous Work

- ▶ Practical research in circuit verification:
 - ▶ Symbolic reachability analysis for state graphs
- ▶ Flow maximization: (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting: (Woelfel, 2003)
- ▶ Strongly connected components: (Gentilini et al., 2003)
- ▶ Shortest paths: (Bahar et al., 1993), (S., 2004)
- ▶ There are mainly two different design goals:
 1. Sublinear (polylogarithmic) runtime on special graphs
 2. Acceptable average-case behavior

Previous Work

- ▶ Practical research in circuit verification:
 - ▶ Symbolic reachability analysis for state graphs
- ▶ Flow maximization: (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting: (Woelfel, 2003)
- ▶ Strongly connected components: (Gentilini et al., 2003)
- ▶ Shortest paths: (Bahar et al., 1993), (S., 2004)
- ▶ There are mainly two different design goals:
 1. Sublinear (polylogarithmic) runtime on special graphs
 2. Acceptable average-case behavior

Previous Work

- ▶ Practical research in circuit verification:
 - ▶ Symbolic reachability analysis for state graphs
- ▶ Flow maximization: (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting: (Woelfel, 2003)
- ▶ Strongly connected components: (Gentilini et al., 2003)
- ▶ Shortest paths: (Bahar et al., 1993), (S., 2004)
- ▶ There are mainly two different design goals:
 1. Sublinear (polylogarithmic) runtime on special graphs
 2. Acceptable average-case behavior

Previous Work

- ▶ Practical research in circuit verification:
 - ▶ Symbolic reachability analysis for state graphs
- ▶ Flow maximization: (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting: (Woelfel, 2003)
- ▶ Strongly connected components: (Gentilini et al., 2003)
- ▶ Shortest paths: (Bahar et al., 1993), (S., 2004)
- ▶ There are mainly two different design goals:
 1. Sublinear (polylogarithmic) runtime on special graphs
 2. Acceptable average-case behavior

Previous Work

- ▶ Practical research in circuit verification:
 - ▶ Symbolic reachability analysis for state graphs
- ▶ Flow maximization: (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting: (Woelfel, 2003)
- ▶ Strongly connected components: (Gentilini et al., 2003)
- ▶ Shortest paths: (Bahar et al., 1993), (S., 2004)
- ▶ There are mainly two different design goals:
 1. Sublinear (polylogarithmic) runtime on special graphs
 2. Acceptable average-case behavior

Previous Work

- ▶ Practical research in circuit verification:
 - ▶ Symbolic reachability analysis for state graphs
- ▶ Flow maximization: (Hachtel, Somenzi, 1997), (S., 2004)
- ▶ Topological sorting: (Woelfel, 2003)
- ▶ Strongly connected components: (Gentilini et al., 2003)
- ▶ Shortest paths: (Bahar et al., 1993), (S., 2004)
- ▶ There are mainly two different design goals:
 1. Sublinear (polylogarithmic) runtime on special graphs
 2. Acceptable average-case behavior

Symbolic Shortest-Path Algorithms

- ▶ Approach: Develop symbolic versions of Dijkstra and Bellman-Ford.
- ▶ **Input:** Graph $G = (V, E, \ell)$, source $s \in V$

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

- ▶ **Output:** Distance function $\text{dist}: V \rightarrow \mathbb{N} \cup \{\infty\}$ as OBDD

$$\text{DIST}(v, d) = 1 \Leftrightarrow \text{dist}(v) = d$$

Symbolic Shortest-Path Algorithms

- ▶ Approach: Develop symbolic versions of Dijkstra and Bellman-Ford.
- ▶ **Input:** Graph $G = (V, E, \ell)$, source $s \in V$

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

- ▶ **Output:** Distance function $\text{dist}: V \rightarrow \mathbb{N} \cup \{\infty\}$ as OBDD

$$\text{DIST}(v, d) = 1 \Leftrightarrow \text{dist}(v) = d$$

Symbolic Shortest-Path Algorithms

- ▶ Approach: Develop symbolic versions of Dijkstra and Bellman-Ford.
- ▶ **Input:** Graph $G = (V, E, \ell)$, source $s \in V$

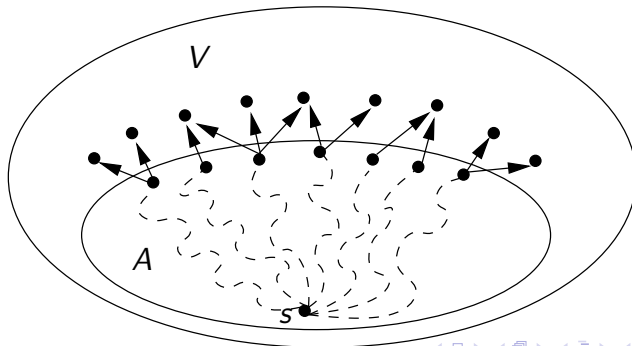
$$C(u, v, d) = 1 \Leftrightarrow [(u, v) \in E] \wedge [\ell(u, v) = d]$$

- ▶ **Output:** Distance function $\text{dist}: V \rightarrow \mathbb{N} \cup \{\infty\}$ as OBDD

$$\text{DIST}(v, d) = 1 \Leftrightarrow \text{dist}(v) = d$$

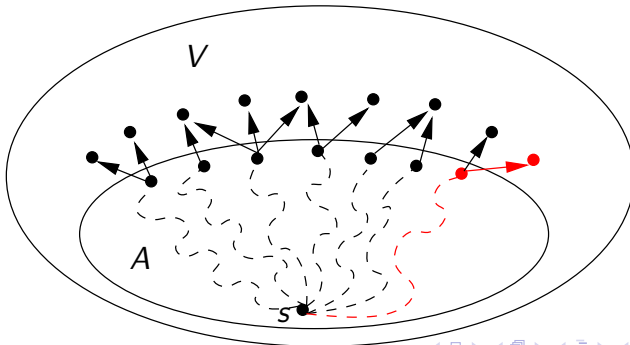
The Symbolic Dijkstra-Algorithm

- ▶ Nodes $v \in A \subset V$ already have shortest paths.
- ▶ Select $v^{\min} \in V \setminus A$ with minimal $\text{dist}(v^{\min})$.
- ▶ Add v^{\min} to A .



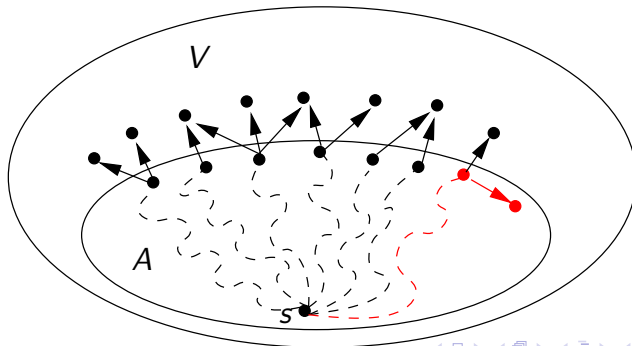
The Symbolic Dijkstra-Algorithm

- ▶ Nodes $v \in A \subset V$ already have shortest paths.
- ▶ Select $v^{\min} \in V \setminus A$ with minimal $\text{dist}(v^{\min})$.
- ▶ Add v^{\min} to A .



The Symbolic Dijkstra-Algorithm

- ▶ Nodes $v \in A \subset V$ already have shortest paths.
- ▶ Select $v^{\min} \in V \setminus A$ with minimal $\text{dist}(v^{\min})$.
- ▶ Add v^{\min} to A .

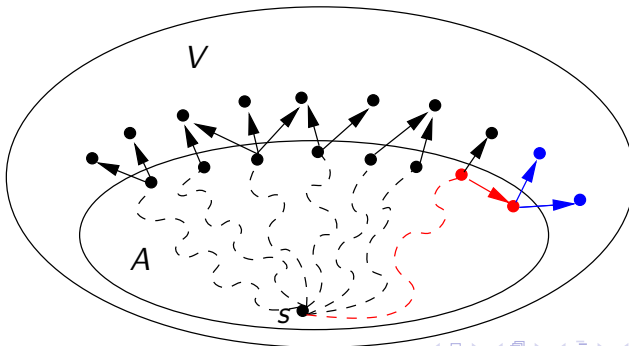


The Symbolic Dijkstra-Algorithm

- ▶ Update dist for neighbors w of v^{\min} :

$$\text{dist}(w) := \min\{\text{dist}(w), \text{dist}(v^{\min}) + \ell(v^{\min}, w)\}$$

- ▶ $\mathcal{O}(N)$ iterations and $\mathcal{O}(N \cdot \log(NT))$ OBDD-operations

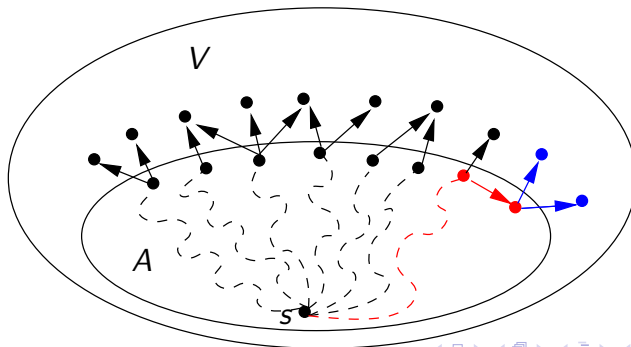


The Symbolic Dijkstra-Algorithm

- ▶ Update dist for neighbors w of v^{\min} :

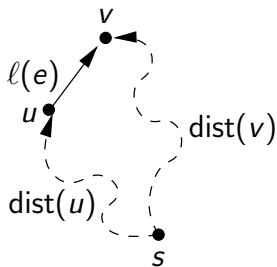
$$\text{dist}(w) := \min\{\text{dist}(w), \text{dist}(v^{\min}) + \ell(v^{\min}, w)\}$$

- ▶ $\mathcal{O}(N)$ iterations and $\mathcal{O}(N \cdot \log(NT))$ OBDD-operations



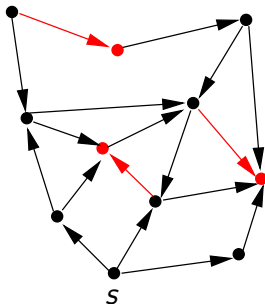
The Symbolic Bellman-Ford-Algorithm

- ▶ Relax edges $(u, v) \in E$ with $\text{dist}(u) + \ell(u, v) < \text{dist}(v)$.
- ▶ Update $\text{dist}(v) := d$.
- ▶ Compute OBDD
 $\text{RELAX}(u, v, d) = 1 \Leftrightarrow \text{dist}(u) + \ell(u, v) = d < \text{dist}(v)$.
- ▶ $\mathcal{O}(NM)$ iterations and $\mathcal{O}(NM \cdot \log(NT))$ OBDD-operations



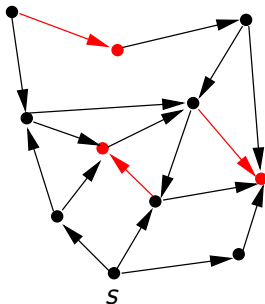
The Symbolic Bellman-Ford-Algorithm

- ▶ Relax edges $(u, v) \in E$ with $\text{dist}(u) + \ell(u, v) < \text{dist}(v)$.
- ▶ Update $\text{dist}(v) := d$.
- ▶ Compute OBDD
 $\text{RELAX}(u, v, d) = 1 \Leftrightarrow \text{dist}(u) + \ell(u, v) = d < \text{dist}(v)$.
- ▶ $\mathcal{O}(NM)$ iterations and $\mathcal{O}(NM \cdot \log(NT))$ OBDD-operations



The Symbolic Bellman-Ford-Algorithm

- ▶ Relax edges $(u, v) \in E$ with $\text{dist}(u) + \ell(u, v) < \text{dist}(v)$.
- ▶ Update $\text{dist}(v) := d$.
- ▶ Compute OBDD
 $RELAX(u, v, d) = 1 \Leftrightarrow \text{dist}(u) + \ell(u, v) = d < \text{dist}(v)$.
- ▶ $\mathcal{O}(NM)$ iterations and $\mathcal{O}(NM \cdot \log(NT))$ OBDD-operations



Computing $RELAX(u, v, d)$:

$$RELAX(u, v, d) := (\exists d^{(1)}, d^{(2)}) [DIST(u, d^{(1)}) \wedge C(u, v, d^{(2)}) \\ \wedge (d = d^{(1)} + d^{(2)})] \\ \wedge \overline{(\exists d^{(1)}) [DIST(v, d^{(1)}) \wedge (d^{(1)} \leq d)]}$$

Experimental Analysis on Random Graphs

- ▶ What do we expect from both algorithms?
- ▶ Dijkstra ...
 - ▶ ... works node by node \Rightarrow many operations.
 - ▶ ... handles structured symbolic sets \Rightarrow fast operations.
- ▶ Bellman-Ford ...
 - ▶ ... relaxes edges in parallel \Rightarrow less operations.
 - ▶ ... may handle large, unstructured sets $RELAX(x, y, d) \Rightarrow$ complex, slow operations.
- ▶ Expectations/hopes:
 - ▶ Dijkstra needs little space (at best linear in $size[C]$).
 - ▶ Bellman-Ford needs little runtime (at best linear in $size[C]$).

Experimental Analysis on Random Graphs

- ▶ What do we expect from both algorithms?
- ▶ Dijkstra ...
 - ▶ ... works node by node \Rightarrow many operations.
 - ▶ ... handles structured symbolic sets \Rightarrow fast operations.
- ▶ Bellman-Ford ...
 - ▶ ... relaxes edges in parallel \Rightarrow less operations.
 - ▶ ... may handle large, unstructured sets $RELAX(x, y, d) \Rightarrow$ complex, slow operations.
- ▶ Expectations/hopes:
 - ▶ Dijkstra needs little space (at best linear in $size[C]$).
 - ▶ Bellman-Ford needs little runtime (at best linear in $size[C]$).

Experimental Analysis on Random Graphs

- ▶ What do we expect from both algorithms?
- ▶ Dijkstra ...
 - ▶ ... works node by node \Rightarrow many operations.
 - ▶ ... handles structured symbolic sets \Rightarrow fast operations.
- ▶ Bellman-Ford ...
 - ▶ ... relaxes edges in parallel \Rightarrow less operations.
 - ▶ ... may handle large, unstructured sets $RELAX(x, y, d) \Rightarrow$ complex, slow operations.
- ▶ Expectations/hopes:
 - ▶ Dijkstra needs little space (at best linear in $size[C]$).
 - ▶ Bellman-Ford needs little runtime (at best linear in $size[C]$).

Experimental Analysis on Random Graphs

- ▶ What do we expect from both algorithms?
- ▶ Dijkstra ...
 - ▶ ...works node by node \Rightarrow many operations.
 - ▶ ...handles structured symbolic sets \Rightarrow fast operations.
- ▶ Bellman-Ford ...
 - ▶ ...relaxes edges in parallel \Rightarrow less operations.
 - ▶ ...may handle large, unstructured sets $RELAX(x, y, d) \Rightarrow$ complex, slow operations.
- ▶ Expectations/hopes:
 - ▶ Dijkstra needs little space (at best linear in $size[C]$).
 - ▶ Bellman-Ford needs little runtime (at best linear in $size[C]$).

Experimental Analysis on Random Graphs

- ▶ What do we expect from both algorithms?
- ▶ Dijkstra ...
 - ▶ ... works node by node \Rightarrow many operations.
 - ▶ ... handles structured symbolic sets \Rightarrow fast operations.
- ▶ Bellman-Ford ...
 - ▶ ... relaxes edges in parallel \Rightarrow less operations.
 - ▶ ... may handle large, unstructured sets $RELAX(x, y, d) \Rightarrow$ complex, slow operations.
- ▶ Expectations/hopes:
 - ▶ Dijkstra needs little space (at best linear in $size[C]$).
 - ▶ Bellman-Ford needs little runtime (at best linear in $size[C]$).

Experimental Analysis on Random Graphs

- ▶ What do we expect from both algorithms?
- ▶ Dijkstra ...
 - ▶ ... works node by node \Rightarrow many operations.
 - ▶ ... handles structured symbolic sets \Rightarrow fast operations.
- ▶ Bellman-Ford ...
 - ▶ ... relaxes edges in parallel \Rightarrow less operations.
 - ▶ ... may handle large, unstructured sets $RELAX(x, y, d) \Rightarrow$ complex, slow operations.
- ▶ Expectations/hopes:
 - ▶ Dijkstra needs little space (at best linear in $size[C]$).
 - ▶ Bellman-Ford needs little runtime (at best linear in $size[C]$).

Experimental Analysis on Random Graphs

- ▶ What do we expect from both algorithms?
- ▶ Dijkstra ...
 - ▶ ...works node by node \Rightarrow many operations.
 - ▶ ...handles structured symbolic sets \Rightarrow fast operations.
- ▶ Bellman-Ford ...
 - ▶ ...relaxes edges in parallel \Rightarrow less operations.
 - ▶ ...may handle large, unstructured sets $RELAX(x, y, d) \Rightarrow$ complex, slow operations.
- ▶ Expectations/hopes:
 - ▶ Dijkstra needs little space (at best linear in size $[C]$).
 - ▶ Bellman-Ford needs little runtime (at best linear in size $[C]$).

Experimental Analysis on Random Graphs

- ▶ What do we expect from both algorithms?
- ▶ Dijkstra ...
 - ▶ ...works node by node \Rightarrow many operations.
 - ▶ ...handles structured symbolic sets \Rightarrow fast operations.
- ▶ Bellman-Ford ...
 - ▶ ...relaxes edges in parallel \Rightarrow less operations.
 - ▶ ...may handle large, unstructured sets $RELAX(x, y, d) \Rightarrow$ complex, slow operations.
- ▶ Expectations/hopes:
 - ▶ Dijkstra needs little space (at best linear in size $[C]$).
 - ▶ Bellman-Ford needs little runtime (at best linear in size $[C]$).

Experimental Analysis on Random Graphs

- ▶ What do we expect from both algorithms?
- ▶ Dijkstra ...
 - ▶ ...works node by node \Rightarrow many operations.
 - ▶ ...handles structured symbolic sets \Rightarrow fast operations.
- ▶ Bellman-Ford ...
 - ▶ ...relaxes edges in parallel \Rightarrow less operations.
 - ▶ ...may handle large, unstructured sets $RELAX(x, y, d) \Rightarrow$ complex, slow operations.
- ▶ Expectations/hopes:
 - ▶ Dijkstra needs little space (at best linear in size $[C]$).
 - ▶ Bellman-Ford needs little runtime (at best linear in size $[C]$).

Experimental Analysis on Random Graphs

- ▶ What do we expect from both algorithms?
- ▶ Dijkstra ...
 - ▶ ...works node by node \Rightarrow many operations.
 - ▶ ...handles structured symbolic sets \Rightarrow fast operations.
- ▶ Bellman-Ford ...
 - ▶ ...relaxes edges in parallel \Rightarrow less operations.
 - ▶ ...may handle large, unstructured sets $RELAX(x, y, d) \Rightarrow$ complex, slow operations.
- ▶ Expectations/hopes:
 - ▶ Dijkstra needs little space (at best linear in size $[C]$).
 - ▶ Bellman-Ford needs little runtime (at best linear in size $[C]$).

- ▶ Random graph instances:
 - ▶ Directed, asymmetric, loopfree
 - ▶ $N \in \{100, 200, 300, 400\}$
 - ▶ Edge probability $p = q \cdot 0.05$, $q \in \{1, \dots, 20\}$.
 - ▶ 10 experiments per setting.

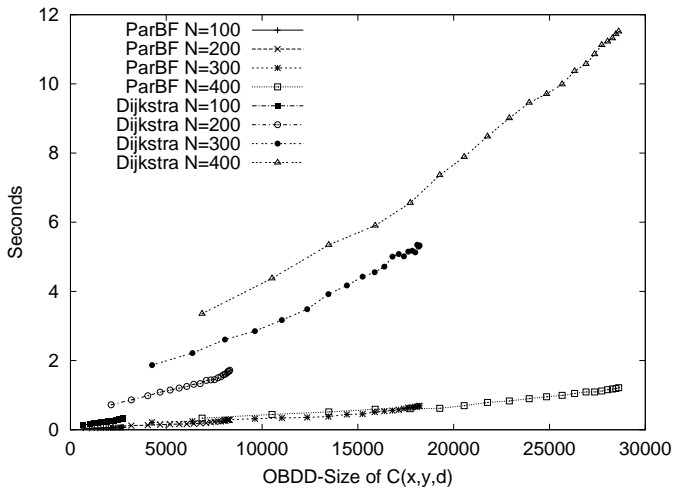
- ▶ Random graph instances:
 - ▶ Directed, asymmetric, loopfree
 - ▶ $N \in \{100, 200, 300, 400\}$
 - ▶ Edge probability $p = q \cdot 0.05$, $q \in \{1, \dots, 20\}$.
 - ▶ 10 experiments per setting.

- ▶ Random graph instances:
 - ▶ Directed, asymmetric, loopfree
 - ▶ $N \in \{100, 200, 300, 400\}$
 - ▶ Edge probability $p = q \cdot 0.05$, $q \in \{1, \dots, 20\}$.
 - ▶ 10 experiments per setting.

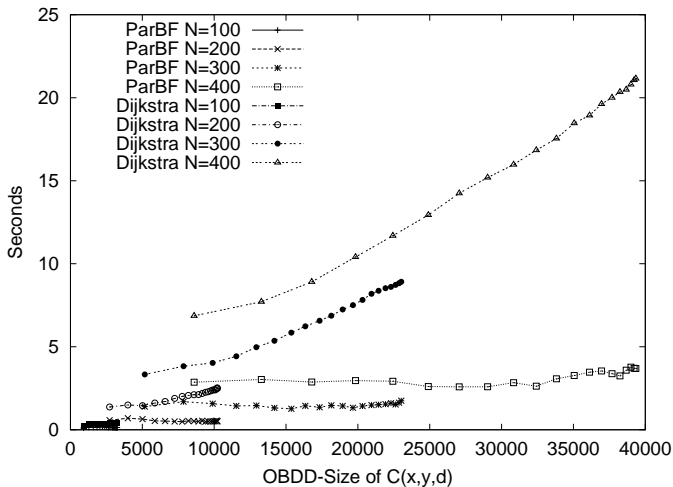
- ▶ Random graph instances:
 - ▶ Directed, asymmetric, loopfree
 - ▶ $N \in \{100, 200, 300, 400\}$
 - ▶ Edge probability $p = q \cdot 0.05$, $q \in \{1, \dots, 20\}$.
 - ▶ 10 experiments per setting.

- ▶ Random graph instances:
 - ▶ Directed, asymmetric, loopfree
 - ▶ $N \in \{100, 200, 300, 400\}$
 - ▶ Edge probability $p = q \cdot 0.05$, $q \in \{1, \dots, 20\}$.
 - ▶ 10 experiments per setting.

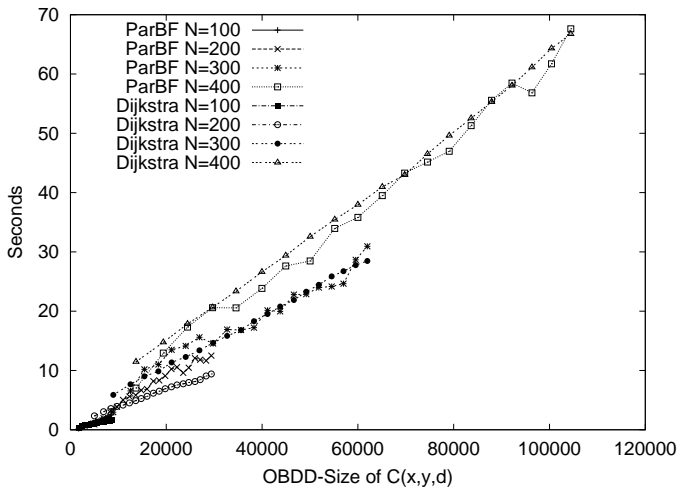
- ▶ Random edges, constant edge weights $\ell(e) = 1$.



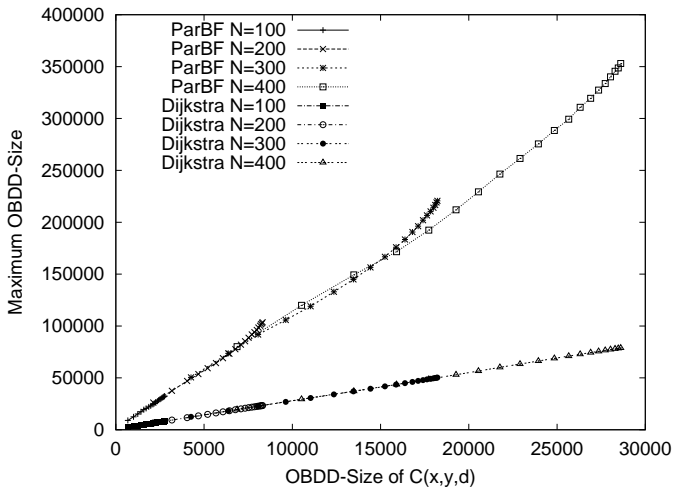
- ▶ Random edges, regular weights $\ell(v_a, v_b) = |a - b| \bmod 200$.



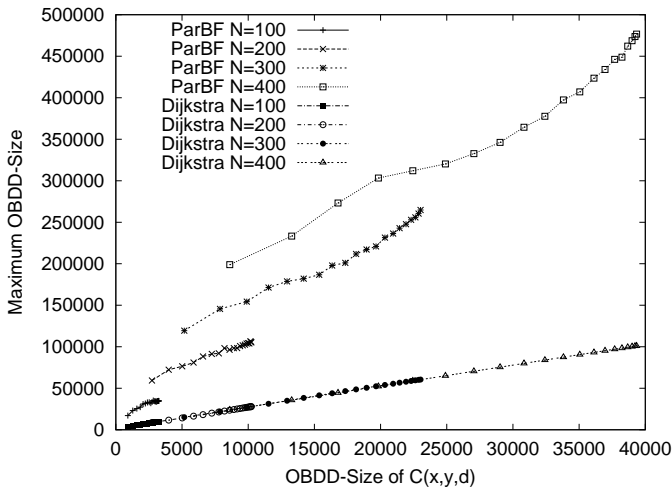
- ▶ Random edges, random edge weights $\ell(e) \in \{0, \dots, 200\}$.



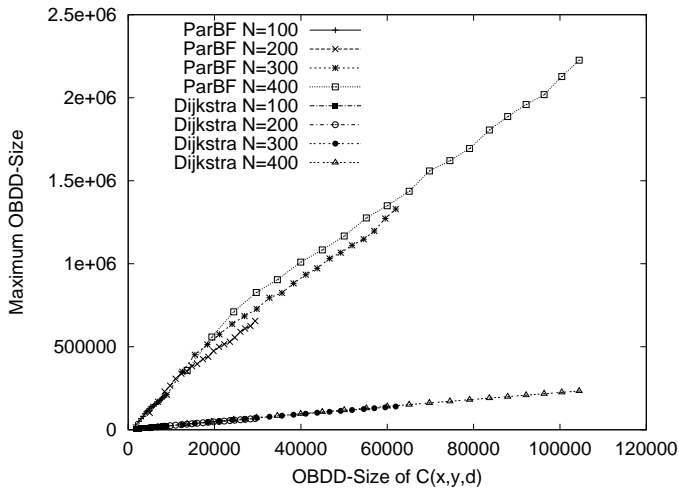
- ▶ Random edges, constant edge weights $\ell(e) = 1$.



- ▶ Random edges, regular weights $\ell(v_a, v_b) = |a - b| \bmod 200$.



- ▶ Random edges, random edge weights $\ell(e) \in \{0, \dots, 200\}$.



Summary

- ▶ Symbolic algorithms are heuristics to solve problems in large but structured data.
- ▶ Dijkstra and Bellman-Ford have been transformed into OBDD-algorithms.
- ▶ On random, grid, and threshold graphs ...
 - ▶ ... Dijkstra is dominant w. r. t. space.
 - ▶ ... Bellman-Ford is dominant w. r. t. runtime.
- ▶ Dijkstra even linear in input size $\text{size}[C]$.
- ▶ Future research:
 - ▶ Experiments on real-world graphs
 - ▶ Considering further SP-problems

Summary

- ▶ Symbolic algorithms are heuristics to solve problems in large but structured data.
- ▶ Dijkstra and Bellman-Ford have been transformed into OBDD-algorithms.
- ▶ On random, grid, and threshold graphs ...
 - ▶ ... Dijkstra is dominant w. r. t. space.
 - ▶ ... Bellman-Ford is dominant w. r. t. runtime.
- ▶ Dijkstra even linear in input size $\text{size}[C]$.
- ▶ Future research:
 - ▶ Experiments on real-world graphs
 - ▶ Considering further SP-problems

Summary

- ▶ Symbolic algorithms are heuristics to solve problems in large but structured data.
- ▶ Dijkstra and Bellman-Ford have been transformed into OBDD-algorithms.
- ▶ On random, grid, and threshold graphs ...
 - ▶ ... Dijkstra is dominant w. r. t. space.
 - ▶ ... Bellman-Ford is dominant w. r. t. runtime.
- ▶ Dijkstra even linear in input size $\text{size}[C]$.
- ▶ Future research:
 - ▶ Experiments on real-world graphs
 - ▶ Considering further SP-problems

Summary

- ▶ Symbolic algorithms are heuristics to solve problems in large but structured data.
- ▶ Dijkstra and Bellman-Ford have been transformed into OBDD-algorithms.
- ▶ On random, grid, and threshold graphs ...
 - ▶ ... Dijkstra is dominant w. r. t. space.
 - ▶ ... Bellman-Ford is dominant w. r. t. runtime.
- ▶ Dijkstra even linear in input size $\text{size}[C]$.
- ▶ Future research:
 - ▶ Experiments on real-world graphs
 - ▶ Considering further SP-problems

Summary

- ▶ Symbolic algorithms are heuristics to solve problems in large but structured data.
- ▶ Dijkstra and Bellman-Ford have been transformed into OBDD-algorithms.
- ▶ On random, grid, and threshold graphs ...
 - ▶ ... Dijkstra is dominant w. r. t. space.
 - ▶ ... Bellman-Ford is dominant w. r. t. runtime.
- ▶ Dijkstra even linear in input size $\text{size}[C]$.
- ▶ Future research:
 - ▶ Experiments on real-world graphs
 - ▶ Considering further SP-problems

Summary

- ▶ Symbolic algorithms are heuristics to solve problems in large but structured data.
- ▶ Dijkstra and Bellman-Ford have been transformed into OBDD-algorithms.
- ▶ On random, grid, and threshold graphs ...
 - ▶ ... Dijkstra is dominant w. r. t. space.
 - ▶ ... Bellman-Ford is dominant w. r. t. runtime.
- ▶ Dijkstra even linear in input size $\text{size}[C]$.
- ▶ Future research:
 - ▶ Experiments on real-world graphs
 - ▶ Considering further SP-problems

Summary

- ▶ Symbolic algorithms are heuristics to solve problems in large but structured data.
- ▶ Dijkstra and Bellman-Ford have been transformed into OBDD-algorithms.
- ▶ On random, grid, and threshold graphs ...
 - ▶ ... Dijkstra is dominant w. r. t. space.
 - ▶ ... Bellman-Ford is dominant w. r. t. runtime.
- ▶ Dijkstra even linear in input size $\text{size}[C]$.
- ▶ Future research:
 - ▶ Experiments on real-world graphs
 - ▶ Considering further SP-problems

Summary

- ▶ Symbolic algorithms are heuristics to solve problems in large but structured data.
- ▶ Dijkstra and Bellman-Ford have been transformed into OBDD-algorithms.
- ▶ On random, grid, and threshold graphs ...
 - ▶ ... Dijkstra is dominant w. r. t. space.
 - ▶ ... Bellman-Ford is dominant w. r. t. runtime.
- ▶ Dijkstra even linear in input size $\text{size}[C]$.
- ▶ Future research:
 - ▶ Experiments on real-world graphs
 - ▶ Considering further SP-problems

Summary

- ▶ Symbolic algorithms are heuristics to solve problems in large but structured data.
- ▶ Dijkstra and Bellman-Ford have been transformed into OBDD-algorithms.
- ▶ On random, grid, and threshold graphs ...
 - ▶ ... Dijkstra is dominant w. r. t. space.
 - ▶ ... Bellman-Ford is dominant w. r. t. runtime.
- ▶ Dijkstra even linear in input size $\text{size}[C]$.
- ▶ Future research:
 - ▶ Experiments on real-world graphs
 - ▶ Considering further SP-problems

Thank you for listening!