The Complexity of Problems on Implicitly Represented Inputs

Daniel Sawitzki

University of Dortmund, Computer Science 2

SOFSEM 2006, Merin

Contents

Implicit Data Representation

Definition

An implicit data representation avoids explicit enumeration of single elements.

- Focus: Representation by Boolean functions.
- Consider string $I \in \{0,1\}^n$ of length $n = 2^m$.
- Define the characteristic function $\chi_I : \{0,1\}^m \to \{0,1\}$ of I by

$$\chi_I(x) = I_{|x|}$$

for $x \in \{0,1\}^m$.

Implicit Data Representation

Definition

An implicit data representation avoids explicit enumeration of single elements.

- Focus: Representation by Boolean functions.
- Consider string $I \in \{0,1\}^n$ of length $n = 2^m$.
- Define the characteristic function $\chi_I : \{0,1\}^m \to \{0,1\}$ of I by

$$\chi_I(x) = I_{|x|}$$

for $x \in \{0,1\}^m$.

Implicit Data Representation

Definition

An implicit data representation avoids explicit enumeration of single elements.

- Focus: Representation by Boolean functions.
- Consider string $I \in \{0,1\}^n$ of length $n = 2^m$.
- Define the characteristic function $\chi_I : \{0,1\}^m \to \{0,1\}$ of I by

$$\chi_I(x) = I_{|x|}$$

for $x \in \{0,1\}^m$.

Implicit Algorithms and OBDDs

- Popular data structure for Boolean functions:
  Ordered Binary Decision Diagrams (OBDDs)
- OBDDs of structured functions are known to be very succinct.
- OBDDs offer efficient algorithms for functional operations.
- $\Rightarrow$ Implicit/symbolic algorithms:
  - Represent input $I \in \{0,1\}^n$ implicitly as OBDD of χ_I.
  - Compute OBDD of χ_O for output $O \in \{0,1\}^*$.
  - Process implicit data via functional OBDD operations.
- Hope: Efficient (sublinear) heuristic on large but structured problem instances

Implicit Algorithms and OBDDs

- Popular data structure for Boolean functions:
  Ordered Binary Decision Diagrams (OBDDs)
- OBDDs of structured functions are known to be very succinct.
- OBDDs offer efficient algorithms for functional operations.
- $\Rightarrow$ Implicit/symbolic algorithms:
  - Represent input $I \in \{0,1\}^n$ implicitly as OBDD of χ_I.
  - Compute OBDD of χ_O for output $O \in \{0,1\}^*$.
  - Process implicit data via functional OBDD operations.
- Hope: Efficient (sublinear) heuristic on large but structured problem instances

Implicit Algorithms and OBDDs

- Popular data structure for Boolean functions:
  Ordered Binary Decision Diagrams (OBDDs)
- OBDDs of structured functions are known to be very succinct.
- OBDDs offer efficient algorithms for functional operations.
- $\Rightarrow$ Implicit/symbolic algorithms:
  - Represent input $I \in \{0,1\}^n$ implicitly as OBDD of χ_I.
  - Compute OBDD of χ_O for output $O \in \{0,1\}^*$.
  - Process implicit data via functional OBDD operations.
- Hope: Efficient (sublinear) heuristic on large but structured problem instances

Implicit Algorithms and OBDDs

- Popular data structure for Boolean functions:
  Ordered Binary Decision Diagrams (OBDDs)
- OBDDs of structured functions are known to be very succinct.
- OBDDs offer efficient algorithms for functional operations.
- $\Rightarrow$ Implicit/symbolic algorithms:
  - Represent input $I \in \{0,1\}^n$ implicitly as OBDD of χ_I.
  - Compute OBDD of χ_O for output $O \in \{0,1\}^*$.
  - Process implicit data via functional OBDD operations.
- Hope: Efficient (sublinear) heuristic on large but structured problem instances

Implicit Algorithms and OBDDs

- Popular data structure for Boolean functions:
  Ordered Binary Decision Diagrams (OBDDs)
- OBDDs of structured functions are known to be very succinct.
- OBDDs offer efficient algorithms for functional operations.
- $\Rightarrow$ Implicit/symbolic algorithms:
  - Represent input $I \in \{0,1\}^n$ implicitly as OBDD of χ_I.
  - Compute OBDD of χ_O for output $O \in \{0,1\}^*$.
  - Process implicit data via functional OBDD operations.
- Hope: Efficient (sublinear) heuristic on large but structured problem instances

Implicit Algorithms and OBDDs

- Popular data structure for Boolean functions:
  Ordered Binary Decision Diagrams (OBDDs)
- OBDDs of structured functions are known to be very succinct.
- OBDDs offer efficient algorithms for functional operations.
- $\Rightarrow$ Implicit/symbolic algorithms:
  - Represent input $I \in \{0,1\}^n$ implicitly as OBDD of χ_I.
  - Compute OBDD of χ_O for output $O \in \{0,1\}^*$.
  - Process implicit data via functional OBDD operations.
- Hope: Efficient (sublinear) heuristic on large but structured problem instances

Implicit Algorithms and OBDDs

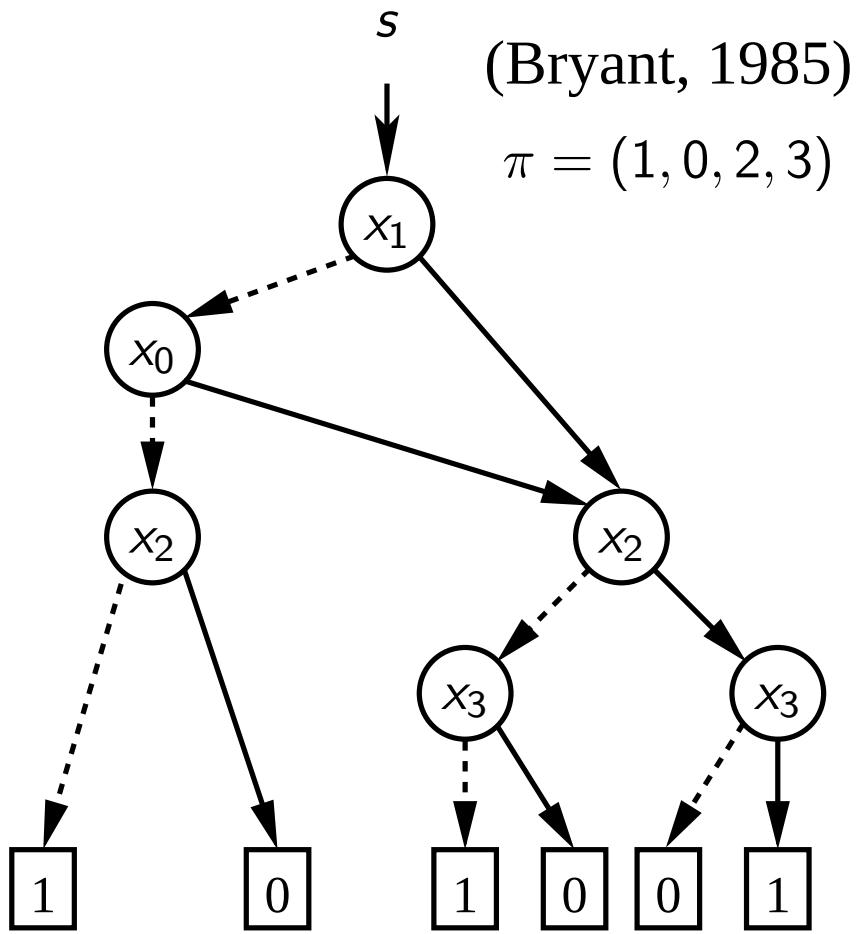
- Popular data structure for Boolean functions:
  Ordered Binary Decision Diagrams (OBDDs)
- OBDDs of structured functions are known to be very succinct.
- OBDDs offer efficient algorithms for functional operations.
- $\Rightarrow$ Implicit/symbolic algorithms:
  - Represent input $I \in \{0,1\}^n$ implicitly as OBDD of χ_I.
  - Compute OBDD of χ_O for output $O \in \{0,1\}^*$.
  - Process implicit data via functional OBDD operations.
- Hope: Efficient (sublinear) heuristic on large but structured problem instances

Implicit Algorithms and OBDDs

- Popular data structure for Boolean functions:
  Ordered Binary Decision Diagrams (OBDDs)
- OBDDs of structured functions are known to be very succinct.
- OBDDs offer efficient algorithms for functional operations.
- $\Rightarrow$ Implicit/symbolic algorithms:
  - Represent input $I \in \{0,1\}^n$ implicitly as OBDD of χ_I.
  - Compute OBDD of χ_O for output $O \in \{0,1\}^*$.
  - Process implicit data via functional OBDD operations.
- Hope: Efficient (sublinear) heuristic on large but structured problem instances

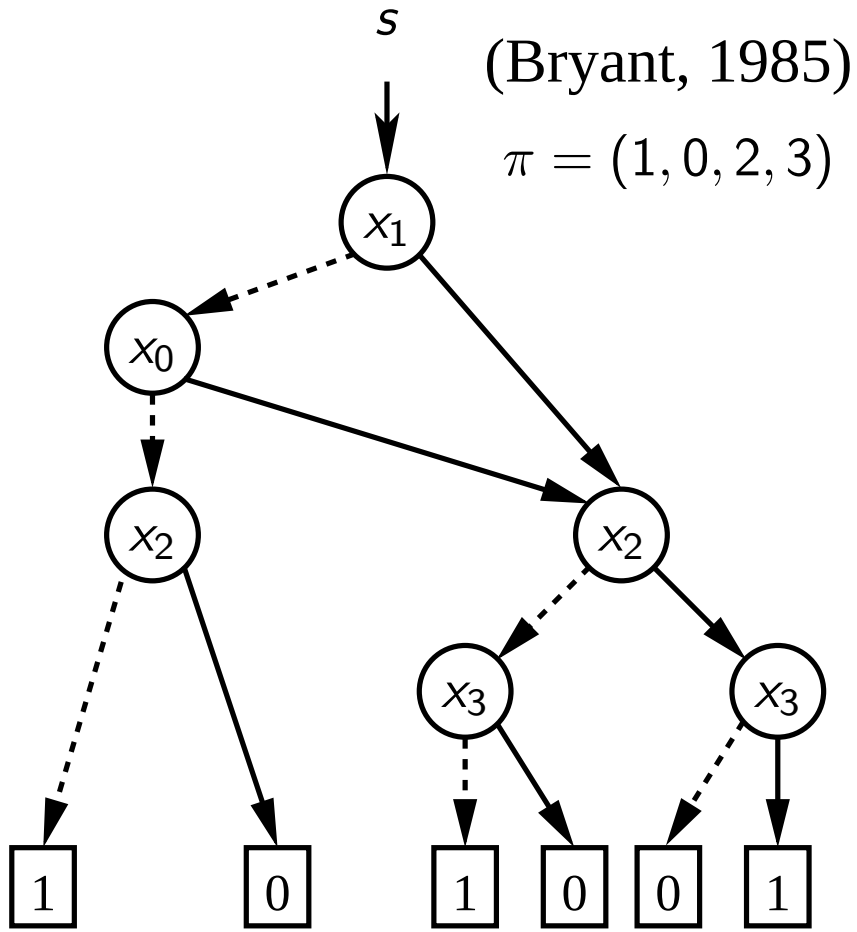
Ordered Binary Decision Diagrams (OBDDs)

- Data structure for $f : \{0,1\}^m \to \{0,1\}$ with Vars. $x_0, \ldots, x_{m-1} \in \{0,1\}$
- OBDD $\mathcal{G}_f$ is acyclic digraph having inner nodes and sinks.
- Inner nodes: Variable label, 0- and 1-edge
- Sink represents value $f(x_0, \ldots, x_{m-1})$.
- Source pointer s
- Reads vars. w. r. t. $\pi \in \Sigma_m$.



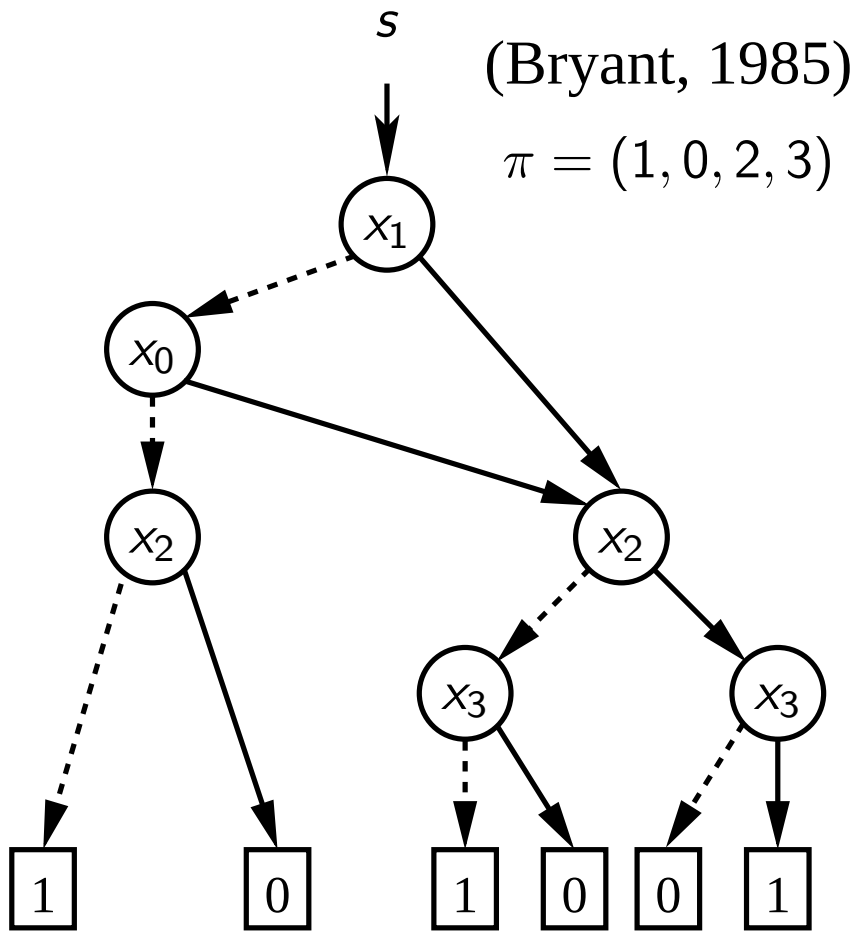
Ordered Binary Decision Diagrams (OBDDs)

- Data structure for $f : \{0,1\}^m \to \{0,1\}$ with Vars. $x_0, \ldots, x_{m-1} \in \{0,1\}$
- OBDD $\mathcal{G}_f$ is acyclic digraph having inner nodes and sinks.
- Inner nodes: Variable label, 0- and 1-edge
- Sink represents value $f(x_0, \ldots, x_{m-1})$.
- Source pointer s
- Reads vars. w. r. t. $\pi \in \Sigma_m$.



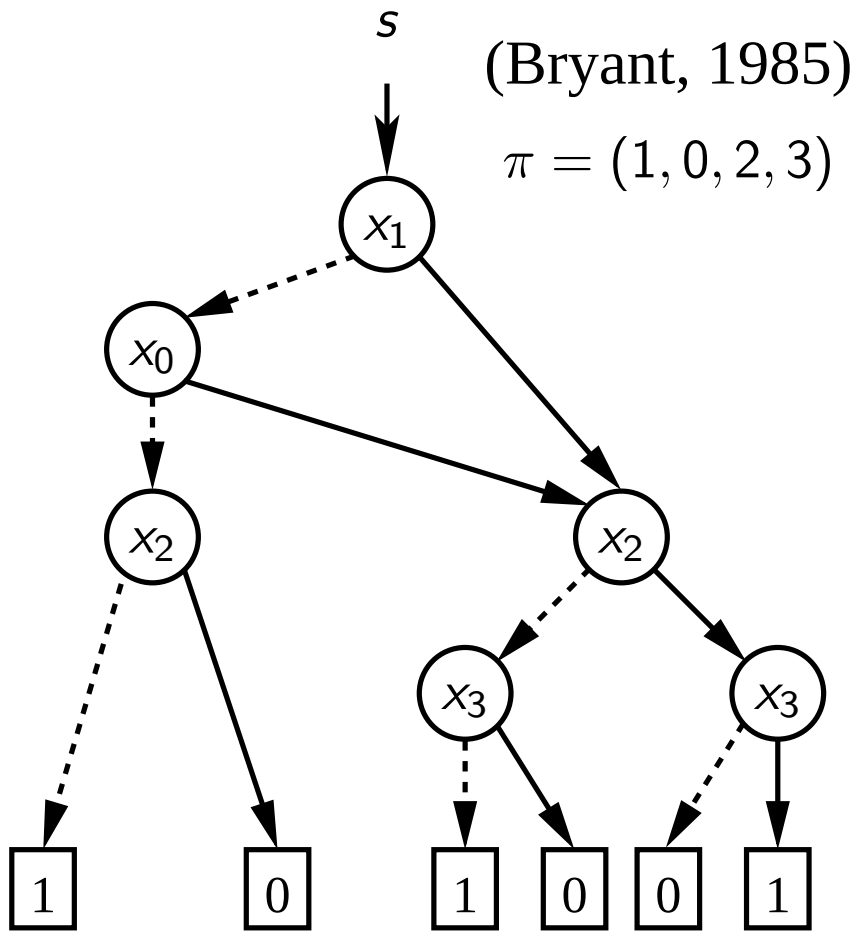
Ordered Binary Decision Diagrams (OBDDs)

- Data structure for $f : \{0,1\}^m \to \{0,1\}$ with Vars. $x_0, \ldots, x_{m-1} \in \{0,1\}$
- OBDD $\mathcal{G}_f$ is acyclic digraph having inner nodes and sinks.
- Inner nodes: Variable label, 0- and 1-edge
- Sink represents value $f(x_0, \ldots, x_{m-1})$.
- Source pointer s
- Reads vars. w. r. t. $\pi \in \Sigma_m$.



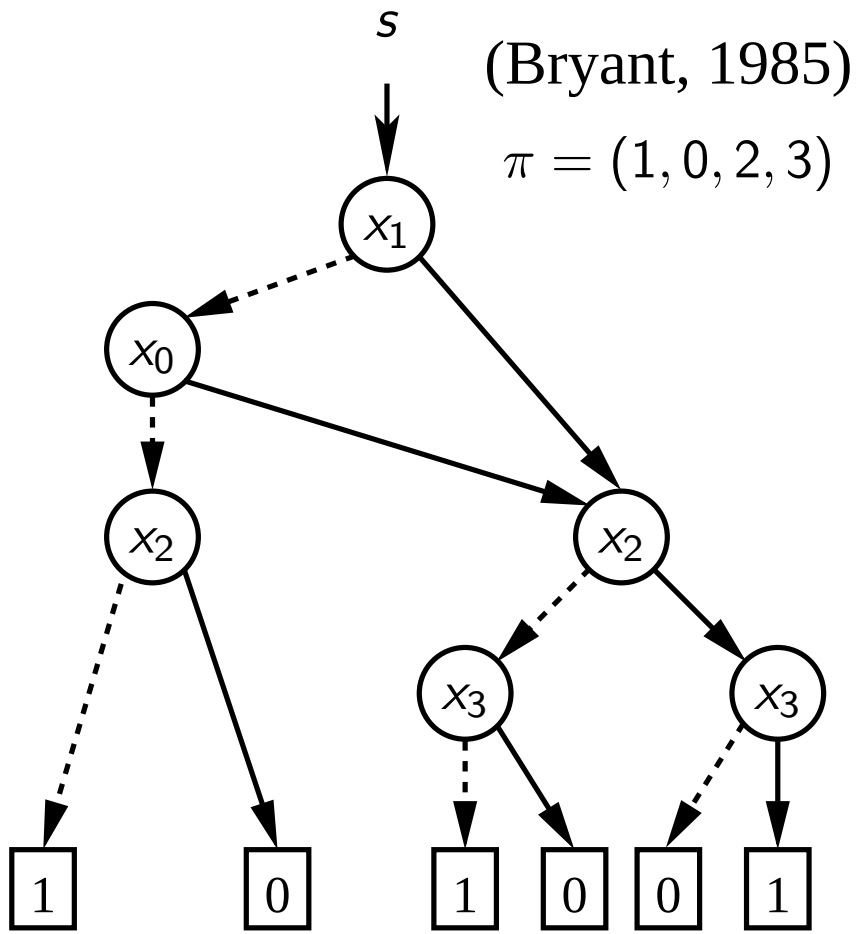
Ordered Binary Decision Diagrams (OBDDs)

- Data structure for $f : \{0,1\}^m \to \{0,1\}$ with Vars. $x_0, \ldots, x_{m-1} \in \{0,1\}$
- OBDD $\mathcal{G}_f$ is acyclic digraph having inner nodes and sinks.
- Inner nodes: Variable label, 0- and 1-edge
- Sink represents value $f(x_0, \ldots, x_{m-1})$.
- Source pointer s
- Reads vars. w. r. t. $\pi \in \Sigma_m$.



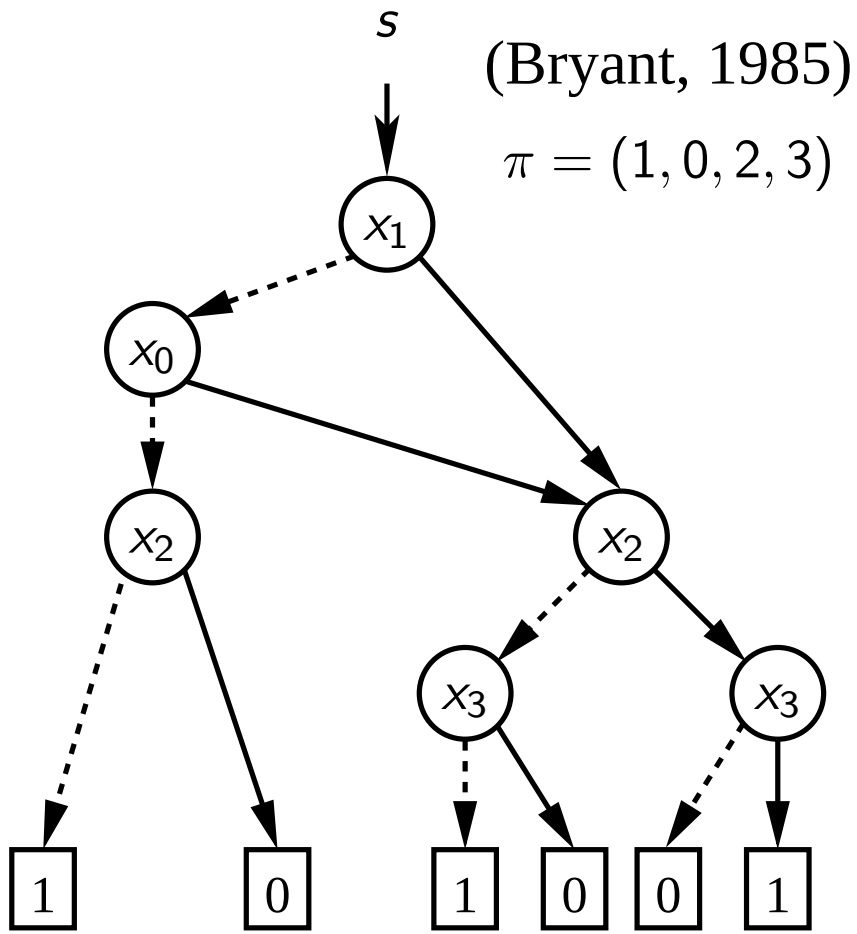
Ordered Binary Decision Diagrams (OBDDs)

- Data structure for $f : \{0,1\}^m \to \{0,1\}$ with Vars. $x_0, \ldots, x_{m-1} \in \{0,1\}$
- OBDD $\mathcal{G}_f$ is acyclic digraph having inner nodes and sinks.
- Inner nodes: Variable label, 0- and 1-edge
- Sink represents value $f(x_0, \ldots, x_{m-1})$.
- Source pointer s
- Reads vars. w. r. t. $\pi \in \Sigma_m$.



Ordered Binary Decision Diagrams (OBDDs)

- Data structure for $f : \{0,1\}^m \to \{0,1\}$ with Vars. $x_0, \ldots, x_{m-1} \in \{0,1\}$
- OBDD $\mathcal{G}_f$ is acyclic digraph having inner nodes and sinks.
- Inner nodes: Variable label, 0- and 1-edge
- Sink represents value $f(x_0, \ldots, x_{m-1})$.
- Source pointer s
- Reads vars. w. r. t. $\pi \in \Sigma_m$.



Algorithmic Properties of OBDDs

- Every function f on m vars. has at most OBDD size $\mathcal{O}(2^m/m)$.
- Hope for structured functions: OBDD size poly(m)
- Efficient operations for OBDDs $\mathcal{G}_f$ and $\mathcal{G}_h$:
  - Satisfiability: $f \not\equiv 0$
  - Equivalence: $f = h$
  - Variable replacement: $f_{|x_i=0/1}$
  - Binary synthesis: $f \otimes h$ for $\otimes = \vee, \wedge, \oplus, \ldots$
  - Quantification: $(\exists/\forall x_i)f$

Algorithmic Properties of OBDDs

- Every function f on m vars. has at most OBDD size $\mathcal{O}(2^m/m)$.
- Hope for structured functions: OBDD size poly(m)
- Efficient operations for OBDDs $\mathcal{G}_f$ and $\mathcal{G}_h$:
  - Satisfiability: $f \not\equiv 0$
  - Equivalence: $f = h$
  - Variable replacement: $f_{|x_i=0/1}$
  - Binary synthesis: $f \otimes h$ for $\otimes = \vee, \wedge, \oplus, \ldots$
  - Quantification: $(\exists/\forall x_i) f$

Algorithmic Properties of OBDDs

- Every function f on m vars. has at most OBDD size $\mathcal{O}(2^m/m)$.
- Hope for structured functions: OBDD size poly(m)
- Efficient operations for OBDDs $\mathcal{G}_f$ and $\mathcal{G}_h$:
  - Satisfiability: $f \not\equiv 0$
  - Equivalence: $f = h$
  - Variable replacement: $f_{|x_i=0/1}$
  - Binary synthesis: $f \otimes h$ for $\otimes = \vee, \wedge, \oplus, \ldots$
  - Quantification: $(\exists/\forall x_i)f$

Algorithmic Properties of OBDDs

- Every function f on m vars. has at most OBDD size $\mathcal{O}(2^m/m)$.
- Hope for structured functions: OBDD size poly(m)
- Efficient operations for OBDDs $\mathcal{G}_f$ and $\mathcal{G}_h$:
  - Satisfiability: $f \not\equiv 0$
  - Equivalence: $f = h$
  - Variable replacement: $f_{|x_i=0/1}$
  - Binary synthesis: $f \otimes h$ for $\otimes = \vee, \wedge, \oplus, \ldots$
  - Quantification: $(\exists/\forall x_i)f$

Algorithmic Properties of OBDDs

- Every function f on m vars. has at most OBDD size $\mathcal{O}(2^m/m)$.
- Hope for structured functions: OBDD size poly(m)
- Efficient operations for OBDDs $\mathcal{G}_f$ and $\mathcal{G}_h$:
  - Satisfiability: $f \not\equiv 0$
  - Equivalence: $f = h$
  - Variable replacement: $f_{|x_i=0/1}$
  - Binary synthesis: $f \otimes h$ for $\otimes = \vee, \wedge, \oplus, \ldots$
  - Quantification: $(\exists/\forall x_i)f$

Algorithmic Properties of OBDDs

- Every function f on m vars. has at most OBDD size $\mathcal{O}(2^m/m)$.
- Hope for structured functions: OBDD size poly(m)
- Efficient operations for OBDDs $\mathcal{G}_f$ and $\mathcal{G}_h$:
  - Satisfiability: $f \not\equiv 0$
  - Equivalence: $f = h$
  - Variable replacement: $f_{|x_i=0/1}$
  - Binary synthesis: $f \otimes h$ for $\otimes = \vee, \wedge, \oplus, \ldots$
  - Quantification: $(\exists/\forall x_i) f$

Algorithmic Properties of OBDDs

- Every function f on m vars. has at most OBDD size $\mathcal{O}(2^m/m)$.
- Hope for structured functions: OBDD size poly(m)
- Efficient operations for OBDDs $\mathcal{G}_f$ and $\mathcal{G}_h$:
  - Satisfiability: $f \not\equiv 0$
  - Equivalence: $f = h$
  - Variable replacement: $f_{|x_i=0/1}$
  - Binary synthesis: $f \otimes h$ for $\otimes = \vee, \wedge, \oplus, \ldots$
  - Quantification: $(\exists/\forall x_i)f$

Algorithmic Properties of OBDDs

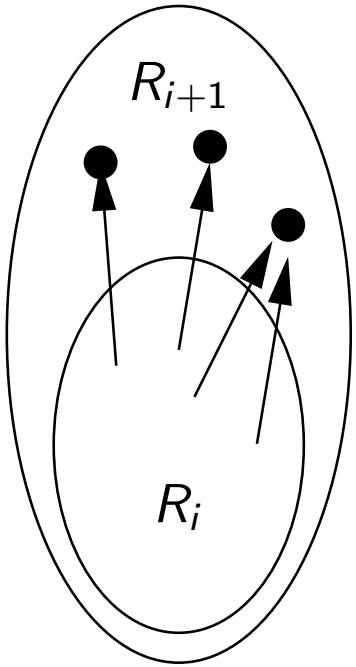
- Every function f on m vars. has at most OBDD size $\mathcal{O}(2^m/m)$.
- Hope for structured functions: OBDD size poly(m)
- Efficient operations for OBDDs $\mathcal{G}_f$ and $\mathcal{G}_h$:
  - Satisfiability: $f \not\equiv 0$
  - Equivalence: $f = h$
  - Variable replacement: $f_{|x_i=0/1}$
  - Binary synthesis: $f \otimes h$ for $\otimes = \vee, \wedge, \oplus, \ldots$
  - Quantification: $(\exists/\forall x_i)f$

Implicit Graph Algorithms: An Example

Example: An implicit BFS algorithm on χ_G for

$$\chi_G(x, y) = 1 \Leftrightarrow (v_{|x|}, v_{|y|}) \in E$$

$i := 0$; $R_0(x) := (|x| = s)$
repeat
  $N(x) := (\exists y)[\chi_G(y, x) \wedge R_i(y) \wedge \overline{R_i(x)}]$
  $R_{i+1}(x) := R_i(x) \vee N(x)$
  $i := i + 1$
until $R_i = R_{i-1}$



State of Affairs

- Situation until 2002:
  - OBDDs well established in CAD, Model Checking, ...
  - Pure heuristics for mostly application-specific problems
  - No theoretical analyses of time/space
- Recent contributions:
  - Implicit algorithms for many graph-theoretic problems (Flows, Shortest Paths, Topological Sorting, ...)
  - Polylogarithmic upper bounds for structured instances
  - Polynomial lower bounds for certain structured instances
- This talk:
  - A lower bound for P-complete problems
  - Fixed-parameter intractability of basic graph problems

State of Affairs

- Situation until 2002:
  - OBDDs well established in CAD, Model Checking, ...
  - Pure heuristics for mostly application-specific problems
  - No theoretical analyses of time/space
- Recent contributions:
  - Implicit algorithms for many graph-theoretic problems (Flows, Shortest Paths, Topological Sorting, ...)
  - Polylogarithmic upper bounds for structured instances
  - Polynomial lower bounds for certain structured instances
- This talk:
  - A lower bound for P-complete problems
  - Fixed-parameter intractability of basic graph problems

State of Affairs

- Situation until 2002:
  - OBDDs well established in CAD, Model Checking, ...
  - Pure heuristics for mostly application-specific problems
  - No theoretical analyses of time/space
- Recent contributions:
  - Implicit algorithms for many graph-theoretic problems (Flows, Shortest Paths, Topological Sorting, ...)
  - Polylogarithmic upper bounds for structured instances
  - Polynomial lower bounds for certain structured instances
- This talk:
  - A lower bound for P-complete problems
  - Fixed-parameter intractability of basic graph problems

State of Affairs

- Situation until 2002:
  - OBDDs well established in CAD, Model Checking, ...
  - Pure heuristics for mostly application-specific problems
  - No theoretical analyses of time/space
- Recent contributions:
  - Implicit algorithms for many graph-theoretic problems (Flows, Shortest Paths, Topological Sorting, ...)
  - Polylogarithmic upper bounds for structured instances
  - Polynomial lower bounds for certain structured instances
- This talk:
  - A lower bound for P-complete problems
  - Fixed-parameter intractability of basic graph problems

State of Affairs

- Situation until 2002:
  - OBDDs well established in CAD, Model Checking, ...
  - Pure heuristics for mostly application-specific problems
  - No theoretical analyses of time/space
- Recent contributions:
  - Implicit algorithms for many graph-theoretic problems (Flows, Shortest Paths, Topological Sorting, ...)
  - Polylogarithmic upper bounds for structured instances
  - Polynomial lower bounds for certain structured instances
- This talk:
  - A lower bound for P-complete problems
  - Fixed-parameter intractability of basic graph problems

State of Affairs

- Situation until 2002:
  - OBDDs well established in CAD, Model Checking, ...
  - Pure heuristics for mostly application-specific problems
  - No theoretical analyses of time/space
- Recent contributions:
  - Implicit algorithms for many graph-theoretic problems (Flows, Shortest Paths, Topological Sorting, ...)
  - Polylogarithmic upper bounds for structured instances
  - Polynomial lower bounds for certain structured instances
- This talk:
  - A lower bound for P-complete problems
  - Fixed-parameter intractability of basic graph problems

State of Affairs

- Situation until 2002:
  - OBDDs well established in CAD, Model Checking, ...
  - Pure heuristics for mostly application-specific problems
  - No theoretical analyses of time/space
- Recent contributions:
  - Implicit algorithms for many graph-theoretic problems (Flows, Shortest Paths, Topological Sorting, ...)
  - Polylogarithmic upper bounds for structured instances
  - Polynomial lower bounds for certain structured instances
- This talk:
  - A lower bound for P-complete problems
  - Fixed-parameter intractability of basic graph problems

State of Affairs

- Situation until 2002:
  - OBDDs well established in CAD, Model Checking, ...
  - Pure heuristics for mostly application-specific problems
  - No theoretical analyses of time/space
- Recent contributions:
  - Implicit algorithms for many graph-theoretic problems (Flows, Shortest Paths, Topological Sorting, ...)
  - Polylogarithmic upper bounds for structured instances
  - Polynomial lower bounds for certain structured instances
- This talk:
  - A lower bound for P-complete problems
  - Fixed-parameter intractability of basic graph problems

State of Affairs

- Situation until 2002:
  - OBDDs well established in CAD, Model Checking, ...
  - Pure heuristics for mostly application-specific problems
  - No theoretical analyses of time/space
- Recent contributions:
  - Implicit algorithms for many graph-theoretic problems (Flows, Shortest Paths, Topological Sorting, ...)
  - Polylogarithmic upper bounds for structured instances
  - Polynomial lower bounds for certain structured instances
- This talk:
  - A lower bound for P-complete problems
  - Fixed-parameter intractability of basic graph problems

State of Affairs

- Situation until 2002:
  - OBDDs well established in CAD, Model Checking, ...
  - Pure heuristics for mostly application-specific problems
  - No theoretical analyses of time/space
- Recent contributions:
  - Implicit algorithms for many graph-theoretic problems (Flows, Shortest Paths, Topological Sorting, ...)
  - Polylogarithmic upper bounds for structured instances
  - Polynomial lower bounds for certain structured instances
- This talk:
  - A lower bound for P-complete problems
  - Fixed-parameter intractability of basic graph problems

State of Affairs

- Situation until 2002:
  - OBDDs well established in CAD, Model Checking, ...
  - Pure heuristics for mostly application-specific problems
  - No theoretical analyses of time/space
- Recent contributions:
  - Implicit algorithms for many graph-theoretic problems (Flows, Shortest Paths, Topological Sorting, ...)
  - Polylogarithmic upper bounds for structured instances
  - Polynomial lower bounds for certain structured instances
- This talk:
  - A lower bound for P-complete problems
  - Fixed-parameter intractability of basic graph problems

Contents

The Number of Functional Operations

- Efficient implicit algos. execute few operations on small data structures.
- Many works just consider the number of operations (SCCs, Gentilini et al., SODA'03).
- General goal: Design algorithms with $\mathcal{O}(\log^k n)$ operations.
- New result: Impossible for P-complete problem (unless P=NC)!

The Number of Functional Operations

- Efficient implicit algos. execute few operations on small data structures.
- Many works just consider the number of operations (SCCs, Gentilini et al., SODA'03).
- General goal: Design algorithms with $\mathcal{O}(\log^k n)$ operations.
- New result: Impossible for P-complete problem (unless P=NC)!

The Number of Functional Operations

- Efficient implicit algos. execute few operations on small data structures.
- Many works just consider the number of operations (SCCs, Gentilini et al., SODA'03).
- General goal: Design algorithms with $\mathcal{O}(\log^k n)$ operations.
- New result: Impossible for P-complete problem (unless P=NC)!

The Number of Functional Operations

- Efficient implicit algos. execute few operations on small data structures.
- Many works just consider the number of operations (SCCs, Gentilini et al., SODA'03).
- General goal: Design algorithms with $\mathcal{O}(\log^k n)$ operations.
- New result: Impossible for P-complete problem (unless P=NC)!

A Framework for Implicit Algorithms

Definition

A symbolic register access machine (SRAM) is a RAM with additional symbolic regs. $S_0, S_1, \ldots$ each holding a Boolean function $f : \{0,1\}^m \to \{0,1\}$. It offers ops. to

- get/increase m,
- evaluate S_i due to $a \in \{0,1\}^m$,
- read f from standard registers into S_i.
- copy/negate symbolic registers,
- compute $S_i \otimes S_j$,
- $[\ldots]$

each of cost 1. Input χ_I and output χ_O are located in S_0.

A Framework for Implicit Algorithms

Definition

A symbolic register access machine (SRAM) is a RAM with additional symbolic regs. $S_0, S_1, \ldots$ each holding a Boolean function $f : \{0,1\}^m \to \{0,1\}$. It offers ops. to

- get/increase m,
- evaluate S_i due to $a \in \{0,1\}^m$,
- read f from standard registers into S_i.
- copy/negate symbolic registers,
- compute $S_i \otimes S_j$,
- $[\ldots]$

each of cost 1. Input χ_I and output χ_O are located in S_0.

A Framework for Implicit Algorithms

Definition

A symbolic register access machine (SRAM) is a RAM with additional symbolic regs. $S_0, S_1, \ldots$ each holding a Boolean function $f : \{0,1\}^m \to \{0,1\}$. It offers ops. to

- get/increase m,
- evaluate S_i due to $a \in \{0,1\}^m$,
- read f from standard registers into S_i.
- copy/negate symbolic registers,
- compute $S_i \otimes S_j$,
- $[\ldots]$

each of cost 1. Input χ_I and output χ_O are located in S_0.

A Framework for Implicit Algorithms

Definition

A symbolic register access machine (SRAM) is a RAM with additional symbolic regs. $S_0, S_1, \ldots$ each holding a Boolean function $f : \{0,1\}^m \to \{0,1\}$. It offers ops. to

- get/increase m,
- evaluate S_i due to $a \in \{0,1\}^m$,
- read f from standard registers into S_i.
- copy/negate symbolic registers,
- compute $S_i \otimes S_j$,
- $[\ldots]$

each of cost 1. Input χ_I and output χ_O are located in S_0.

A Framework for Implicit Algorithms

Definition

A symbolic register access machine (SRAM) is a RAM with additional symbolic regs. $S_0, S_1, \ldots$ each holding a Boolean function $f : \{0,1\}^m \to \{0,1\}$. It offers ops. to

- get/increase m,
- evaluate S_i due to $a \in \{0,1\}^m$,
- read f from standard registers into S_i.
- copy/negate symbolic registers,
- compute $S_i \otimes S_j$,
- $[\ldots]$

each of cost 1. Input χ_I and output χ_O are located in S_0.

A Framework for Implicit Algorithms

Definition

A symbolic register access machine (SRAM) is a RAM with additional symbolic regs. $S_0, S_1, \ldots$ each holding a Boolean function $f : \{0,1\}^m \to \{0,1\}$. It offers ops. to

- get/increase m,
- evaluate S_i due to $a \in \{0,1\}^m$,
- read f from standard registers into S_i.
- copy/negate symbolic registers,
- compute $S_i \otimes S_j$,
- $[\ldots]$

each of cost 1. Input χ_I and output χ_O are located in S_0.

A Framework for Implicit Algorithms

Definition

A symbolic register access machine (SRAM) is a RAM with additional symbolic regs. $S_0, S_1, \ldots$ each holding a Boolean function $f : \{0,1\}^m \to \{0,1\}$. It offers ops. to

- get/increase m,
- evaluate S_i due to $a \in \{0,1\}^m$,
- read f from standard registers into S_i.
- copy/negate symbolic registers,
- compute $S_i \otimes S_j$,
- $[\ldots]$

each of cost 1. Input χ_I and output χ_O are located in S_0.

A Framework for Implicit Algorithms

- SRAM model captures capabilties of "all" implicit (OBDD-based) algorithms.
- Implicit algorithm with $t(n)$ operations $\Rightarrow$ SRAM with time $\mathcal{O}(t(n))$.

Theorem

SRAM on input χ_I with time $t(n)$ and $m \leq k \log n$ variables can be simulated by PRAM in parallel time $\mathcal{O}((t(n))^2 \cdot \log^2 n)$ with $\mathcal{O}(n^k)$ processors on $I \in \{0,1\}^n$.

A Framework for Implicit Algorithms

- SRAM model captures capabilties of "all" implicit (OBDD-based) algorithms.
- Implicit algorithm with $t(n)$ operations $\Rightarrow$ SRAM with time $\mathcal{O}(t(n))$.

Theorem

SRAM on input χ_I with time $t(n)$ and $m \leq k \log n$ variables can be simulated by PRAM in parallel time $\mathcal{O}((t(n))^2 \cdot \log^2 n)$ with $\mathcal{O}(n^k)$ processors on $I \in \{0,1\}^n$.

A Framework for Implicit Algorithms

- SRAM model captures capabilties of "all" implicit (OBDD-based) algorithms.
- Implicit algorithm with $t(n)$ operations $\Rightarrow$ SRAM with time $\mathcal{O}(t(n))$.

Theorem

SRAM on input χ_I with time $t(n)$ and $m \leq k \log n$ variables can be simulated by PRAM in parallel time $\mathcal{O}((t(n))^2 \cdot \log^2 n)$ with $\mathcal{O}(n^k)$ processors on $I \in \{0,1\}^n$.

Simulating SRAMs by PRAMs

Theorem

SRAM on input χ_I with time $t(n)$ and $m \leq k \log n$ variables can be simulated by PRAM in parallel time $\mathcal{O}((t(n))^2 \cdot \log^2 n)$ with $\mathcal{O}(n^k)$ processors on $I \in \{0,1\}^n$.

Sketch of proof:

- Handle values $S_0(a), \ldots, S_r(a)$ by processor P_a for $a \in \{0,1\}^m$ and $r \leq t(n)$.
- $\Rightarrow \mathcal{O}(2^m) = \mathcal{O}(n^k)$ processors.
- Simulate each symbolic op. in parallel time $\mathcal{O}(t(n) \cdot \log^2 n)$.
- Example $\wedge$: Each P_a computes $S_i(a) \wedge S_j(a)$.

Simulating SRAMs by PRAMs

Theorem

SRAM on input χ_I with time $t(n)$ and $m \leq k \log n$ variables can be simulated by PRAM in parallel time $\mathcal{O}((t(n))^2 \cdot \log^2 n)$ with $\mathcal{O}(n^k)$ processors on $I \in \{0,1\}^n$.

Sketch of proof:

- Handle values $S_0(a), \ldots, S_r(a)$ by processor P_a for $a \in \{0,1\}^m$ and $r \leq t(n)$.
- $\Rightarrow \mathcal{O}(2^m) = \mathcal{O}(n^k)$ processors.
- Simulate each symbolic op. in parallel time $\mathcal{O}(t(n) \cdot \log^2 n)$.
- Example $\wedge$: Each P_a computes $S_i(a) \wedge S_j(a)$.

Simulating SRAMs by PRAMs

Theorem

SRAM on input χ_I with time $t(n)$ and $m \leq k \log n$ variables can be simulated by PRAM in parallel time $\mathcal{O}((t(n))^2 \cdot \log^2 n)$ with $\mathcal{O}(n^k)$ processors on $I \in \{0,1\}^n$.

Sketch of proof:

- Handle values $S_0(a), \ldots, S_r(a)$ by processor P_a for $a \in \{0,1\}^m$ and $r \leq t(n)$.
- $\Rightarrow \mathcal{O}(2^m) = \mathcal{O}(n^k)$ processors.
- Simulate each symbolic op. in parallel time $\mathcal{O}(t(n) \cdot \log^2 n)$.
- Example $\wedge$: Each P_a computes $S_i(a) \wedge S_j(a)$.

Simulating SRAMs by PRAMs

Theorem

SRAM on input χ_I with time $t(n)$ and $m \leq k \log n$ variables can be simulated by PRAM in parallel time $\mathcal{O}((t(n))^2 \cdot \log^2 n)$ with $\mathcal{O}(n^k)$ processors on $I \in \{0,1\}^n$.

Sketch of proof:

- Handle values $S_0(a), \ldots, S_r(a)$ by processor P_a for $a \in \{0,1\}^m$ and $r \leq t(n)$.
- $\Rightarrow \mathcal{O}(2^m) = \mathcal{O}(n^k)$ processors.
- Simulate each symbolic op. in parallel time $\mathcal{O}(t(n) \cdot \log^2 n)$.
- Example $\wedge$: Each P_a computes $S_i(a) \wedge S_j(a)$.

Result for P-complete Problems

Theorem

P-complete problems have no PRAMs with time $\mathcal{O}(\log^k n)$ on $\mathcal{O}(n^k)$ processors unless $P = NC$.

Corollary

P-complete problems have no implicit algorithms with $\mathcal{O}(\log^k n)$ functional operations on $\leq k \log n$ variables unless $P = NC$.

- Example: Flow maximization is P-complete.
- Open: Is 0-1 flow maximization P-complete?
- $\Rightarrow$ No polylog. implicit algo. yet. (S., SOFSEM'04)

Result for P-complete Problems

Theorem

P-complete problems have no PRAMs with time $\mathcal{O}(\log^k n)$ on $\mathcal{O}(n^k)$ processors unless $P = NC$.

Corollary

P-complete problems have no implicit algorithms with $\mathcal{O}(\log^k n)$ functional operations on $\leq k \log n$ variables unless $P = NC$.

- Example: Flow maximization is P-complete.
- Open: Is 0-1 flow maximization P-complete?
- $\Rightarrow$ No polylog. implicit algo. yet. (S., SOFSEM'04)

Result for P-complete Problems

Theorem

P-complete problems have no PRAMs with time $\mathcal{O}(\log^k n)$ on $\mathcal{O}(n^k)$ processors unless $P = NC$.

Corollary

P-complete problems have no implicit algorithms with $\mathcal{O}(\log^k n)$ functional operations on $\leq k \log n$ variables unless $P = NC$.

- Example: Flow maximization is P-complete.
- Open: Is 0-1 flow maximization P-complete?
- $\Rightarrow$ No polylog. implicit algo. yet. (S., SOFSEM'04)

Result for P-complete Problems

Theorem

P-complete problems have no PRAMs with time $\mathcal{O}(\log^k n)$ on $\mathcal{O}(n^k)$ processors unless $P = NC$.

Corollary

P-complete problems have no implicit algorithms with $\mathcal{O}(\log^k n)$ functional operations on $\leq k \log n$ variables unless $P = NC$.

- Example: Flow maximization is P-complete.
- Open: Is 0-1 flow maximization P-complete?
- $\Rightarrow$ No polylog. implicit algo. yet. (S., SOFSEM'04)

Result for P-complete Problems

Theorem

P-complete problems have no PRAMs with time $\mathcal{O}(\log^k n)$ on $\mathcal{O}(n^k)$ processors unless $P = NC$.

Corollary

P-complete problems have no implicit algorithms with $\mathcal{O}(\log^k n)$ functional operations on $\leq k \log n$ variables unless $P = NC$.

- Example: Flow maximization is P-complete.
- Open: Is 0-1 flow maximization P-complete?
- $\Rightarrow$ No polylog. implicit algo. yet. (S., SOFSEM'04)

Contents

s-t-Connectivity in OBDD-represented Graphs

- Input: $\chi_G(x, y) = 1 \Leftrightarrow (v_{|x|}, v_{|y|}) \in E$, $s, t \in V$
- Feigenbaum et al. (STACS'98): PSPACE-hard!
  - Technique: Construct small OBDD for configuration transition relation of pol. space bounded TM.
  - For $\Pi \in PSPACE$, TM M_Π and input $I \in \{0,1\}^m$: Construct OBDD $\chi_{\Pi,I}$ of size $\mathcal{O}(\text{poly}(m))$.
  - Ask if start config. is connected to accepting config.
- W. r. t. graph size: No $\mathcal{O}(\log^k |V|)$-algorithm.
- Question: Which input OBDD properties might enable polynomial complexity?

s-t-Connectivity in OBDD-represented Graphs

- Input: $\chi_G(x, y) = 1 \Leftrightarrow (v_{|x|}, v_{|y|}) \in E$, $s, t \in V$
- Feigenbaum et al. (STACS'98): PSPACE-hard!
  - Technique: Construct small OBDD for configuration transition relation of pol. space bounded TM.
  - For $\Pi \in \mathit{PSPACE}$, TM M_Π and input $I \in \{0,1\}^m$: Construct OBDD $\chi_{\Pi,I}$ of size $\mathcal{O}(\mathrm{poly}(m))$.
  - Ask if start config. is connected to accepting config.
- W. r. t. graph size: No $\mathcal{O}(\log^k |V|)$-algorithm.
- Question: Which input OBDD properties might enable polynomial complexity?

s-t-Connectivity in OBDD-represented Graphs

- Input: $\chi_G(x, y) = 1 \Leftrightarrow (v_{|x|}, v_{|y|}) \in E$, $s, t \in V$
- Feigenbaum et al. (STACS'98): PSPACE-hard!
  - Technique: Construct small OBDD for configuration transition relation of pol. space bounded TM.
    - For $\Pi \in \mathit{PSPACE}$, TM M_Π and input $I \in \{0,1\}^m$: Construct OBDD $\chi_{\Pi,I}$ of size $\mathcal{O}(\mathrm{poly}(m))$.
    - Ask if start config. is connected to accepting config.
- W. r. t. graph size: No $\mathcal{O}(\log^k |V|)$-algorithm.
- Question: Which input OBDD properties might enable polynomial complexity?

s-t-Connectivity in OBDD-represented Graphs

- Input: $\chi_G(x, y) = 1 \Leftrightarrow (v_{|x|}, v_{|y|}) \in E$, $s, t \in V$
- Feigenbaum et al. (STACS'98): PSPACE-hard!
  - Technique: Construct small OBDD for configuration transition relation of pol. space bounded TM.
  - For $\Pi \in \mathit{PSPACE}$, TM M_Π and input $I \in \{0,1\}^m$: Construct OBDD $\chi_{\Pi,I}$ of size $\mathcal{O}(\text{poly}(m))$.
  - Ask if start config. is connected to accepting config.
- W. r. t. graph size: No $\mathcal{O}(\log^k |V|)$-algorithm.
- Question: Which input OBDD properties might enable polynomial complexity?

s-t-Connectivity in OBDD-represented Graphs

- Input: $\chi_G(x, y) = 1 \Leftrightarrow (v_{|x|}, v_{|y|}) \in E$, $s, t \in V$
- Feigenbaum et al. (STACS'98): PSPACE-hard!
  - Technique: Construct small OBDD for configuration transition relation of pol. space bounded TM.
  - For $\Pi \in \mathit{PSPACE}$, TM M_Π and input $I \in \{0,1\}^m$: Construct OBDD $\chi_{\Pi,I}$ of size $\mathcal{O}(\text{poly}(m))$.
  - Ask if start config. is connected to accepting config.
- W. r. t. graph size: No $\mathcal{O}(\log^k |V|)$-algorithm.
- Question: Which input OBDD properties might enable polynomial complexity?

s-t-Connectivity in OBDD-represented Graphs

- Input: $\chi_G(x, y) = 1 \Leftrightarrow (v_{|x|}, v_{|y|}) \in E$, $s, t \in V$
- Feigenbaum et al. (STACS'98): PSPACE-hard!
  - Technique: Construct small OBDD for configuration transition relation of pol. space bounded TM.
  - For $\Pi \in \mathit{PSPACE}$, TM M_Π and input $I \in \{0,1\}^m$: Construct OBDD $\chi_{\Pi,I}$ of size $\mathcal{O}(\text{poly}(m))$.
  - Ask if start config. is connected to accepting config.
- W. r. t. graph size: No $\mathcal{O}(\log^k |V|)$-algorithm.
- Question: Which input OBDD properties might enable polynomial complexity?

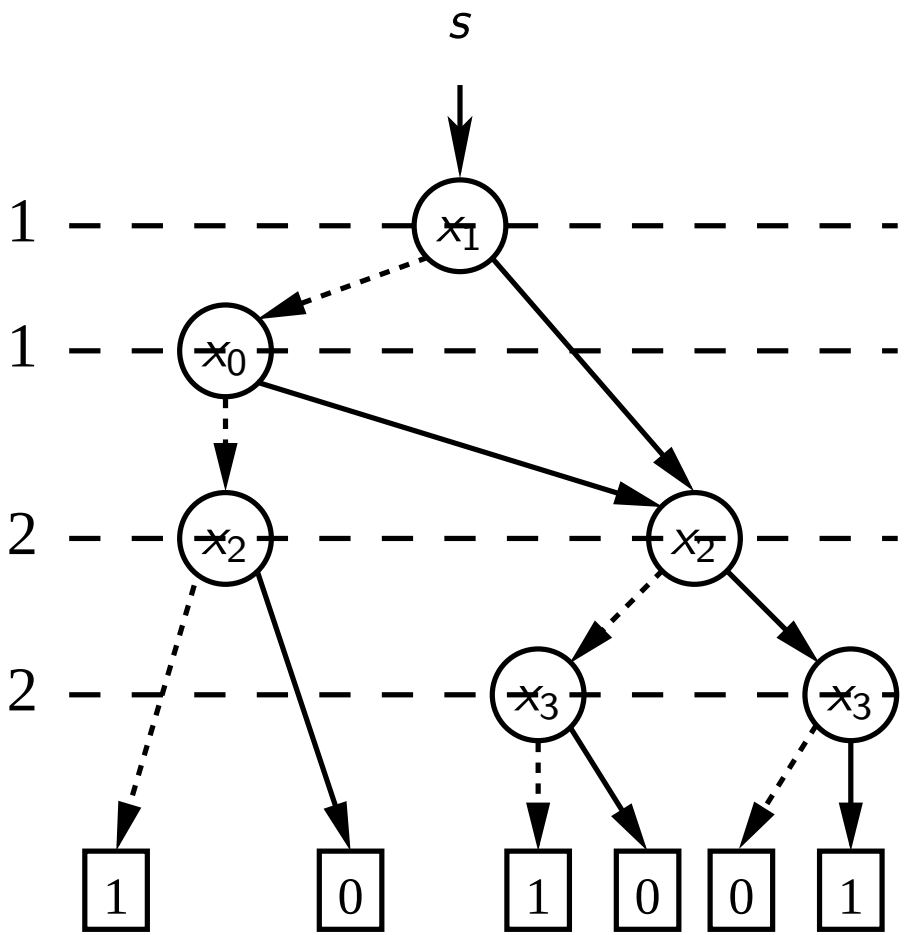
s-t-Connectivity in OBDD-represented Graphs

- Input: $\chi_G(x, y) = 1 \Leftrightarrow (v_{|x|}, v_{|y|}) \in E$, $s, t \in V$
- Feigenbaum et al. (STACS'98): PSPACE-hard!
  - Technique: Construct small OBDD for configuration transition relation of pol. space bounded TM.
  - For $\Pi \in \mathit{PSPACE}$, TM M_Π and input $I \in \{0,1\}^m$: Construct OBDD $\chi_{\Pi,I}$ of size $\mathcal{O}(\text{poly}(m))$.
  - Ask if start config. is connected to accepting config.
- W. r. t. graph size: No $\mathcal{O}(\log^k |V|)$-algorithm.
- Question: Which input OBDD properties might enable polynomial complexity?

Definition of OBDD Width

Definition

The OBDD width is the maximum number of nodes labeled the same variable.



OBDD Width as Fixed Parameter

- Are there efficient algorithms for inputs with small OBDD width W?
- For width W of χ_G and some function α:
- Parameterized complexity $\mathcal{O}(\log^k |V| \cdot \alpha(W))$ possible?
- Feigenbaum proof: $W = \mathcal{O}(1) \Rightarrow$ No FPT-algo. for s-t-conn.
- New contribution: Fixed-parameter intractability for further problems on OBDD-represented graphs.

Theorem

None of the problems s-t-conn., APSP, MaxFlow, Acyclicity, Connectivity, Bipartiteness, Eulerian-Cycle, and Planarity on OBDD-represented graphs has an FPT-algo. unless P=PSPACE.

OBDD Width as Fixed Parameter

- Are there efficient algorithms for inputs with small OBDD width W?
- For width W of χ_G and some function α:
  - Parameterized complexity $\mathcal{O}(\log^k |V| \cdot \alpha(W))$ possible?
  - Feigenbaum proof: $W = \mathcal{O}(1) \Rightarrow$ No FPT-algo. for s-t-conn.
  - New contribution: Fixed-parameter intractability for further problems on OBDD-represented graphs.

Theorem

None of the problems s-t-conn., APSP, MaxFlow, Acyclicity, Connectivity, Bipartiteness, Eulerian-Cycle, and Planarity on OBDD-represented graphs has an FPT-algo. unless P=PSPACE.

OBDD Width as Fixed Parameter

- Are there efficient algorithms for inputs with small OBDD width W?
- For width W of χ_G and some function α:
- Parameterized complexity $\mathcal{O}(\log^k |V| \cdot \alpha(W))$ possible?
- Feigenbaum proof: $W = \mathcal{O}(1) \Rightarrow$ No FPT-algo. for s-t-conn.
- New contribution: Fixed-parameter intractability for further problems on OBDD-represented graphs.

Theorem

None of the problems s-t-conn., APSP, MaxFlow, Acyclicity, Connectivity, Bipartiteness, Eulerian-Cycle, and Planarity on OBDD-represented graphs has an FPT-algo. unless P=PSPACE.

OBDD Width as Fixed Parameter

- Are there efficient algorithms for inputs with small OBDD width W?
- For width W of χ_G and some function α:
- Parameterized complexity $\mathcal{O}(\log^k |V| \cdot \alpha(W))$ possible?
- Feigenbaum proof: $W = \mathcal{O}(1) \Rightarrow$ No FPT-algo. for s-t-conn.
- New contribution: Fixed-parameter intractability for further problems on OBDD-represented graphs.

Theorem

None of the problems s-t-conn., APSP, MaxFlow, Acyclicity, Connectivity, Bipartiteness, Eulerian-Cycle, and Planarity on OBDD-represented graphs has an FPT-algo. unless P=PSPACE.

OBDD Width as Fixed Parameter

- Are there efficient algorithms for inputs with small OBDD width W?
- For width W of χ_G and some function α:
- Parameterized complexity $\mathcal{O}(\log^k |V| \cdot \alpha(W))$ possible?
- Feigenbaum proof: $W = \mathcal{O}(1) \Rightarrow$ No FPT-algo. for s-t-conn.
- New contribution: Fixed-parameter intractability for further problems on OBDD-represented graphs.

Theorem

None of the problems s-t-conn., APSP, MaxFlow, Acyclicity, Connectivity, Bipartiteness, Eulerian-Cycle, and Planarity on OBDD-represented graphs has an FPT-algo. unless P=PSPACE.

OBDD Width as Fixed Parameter

- Are there efficient algorithms for inputs with small OBDD width W?
- For width W of χ_G and some function α:
- Parameterized complexity $\mathcal{O}(\log^k |V| \cdot \alpha(W))$ possible?
- Feigenbaum proof: $W = \mathcal{O}(1) \Rightarrow$ No FPT-algo. for s-t-conn.
- New contribution: Fixed-parameter intractability for further problems on OBDD-represented graphs.

Theorem

None of the problems s-t-conn., APSP, MaxFlow, Acyclicity, Connectivity, Bipartiteness, Eulerian-Cycle, and Planarity on OBDD-represented graphs has an FPT-algo. unless P=PSPACE.

Width-Preserving Reductions from Π to Π'

Map ϕ is width-preserving reduction from Π to Π' iff

- it maps OBDD χ_G to OBDD $\chi_{G'}$ with

$$G \in \Pi \Leftrightarrow G' \in \Pi',$$

- width W' of $\chi_{G'}$ depends only on W rather than on $|V|$.

Theorem

A sequence of $\mathcal{O}(1)$ arbitrary OBDD operations is width-preserving.

Width-Preserving Reductions from Π to Π'

Map ϕ is width-preserving reduction from Π to Π' iff

- it maps OBDD χ_G to OBDD $\chi_{G'}$ with

$$G \in \Pi \Leftrightarrow G' \in \Pi',$$

- width W' of $\chi_{G'}$ depends only on W rather than on $|V|$.

Theorem

A sequence of $\mathcal{O}(1)$ arbitrary OBDD operations is width-preserving.

Width-Preserving Reductions from Π to Π'

Map ϕ is width-preserving reduction from Π to Π' iff

- it maps OBDD χ_G to OBDD $\chi_{G'}$ with

$$G \in \Pi \Leftrightarrow G' \in \Pi',$$

- width W' of $\chi_{G'}$ depends only on W rather than on $|V|$.

Theorem

A sequence of $\mathcal{O}(1)$ arbitrary OBDD operations is width-preserving.

Width-Preserving Reductions from Π to Π'

Map ϕ is width-preserving reduction from Π to Π' iff

- it maps OBDD χ_G to OBDD $\chi_{G'}$ with

$$G \in \Pi \Leftrightarrow G' \in \Pi',$$

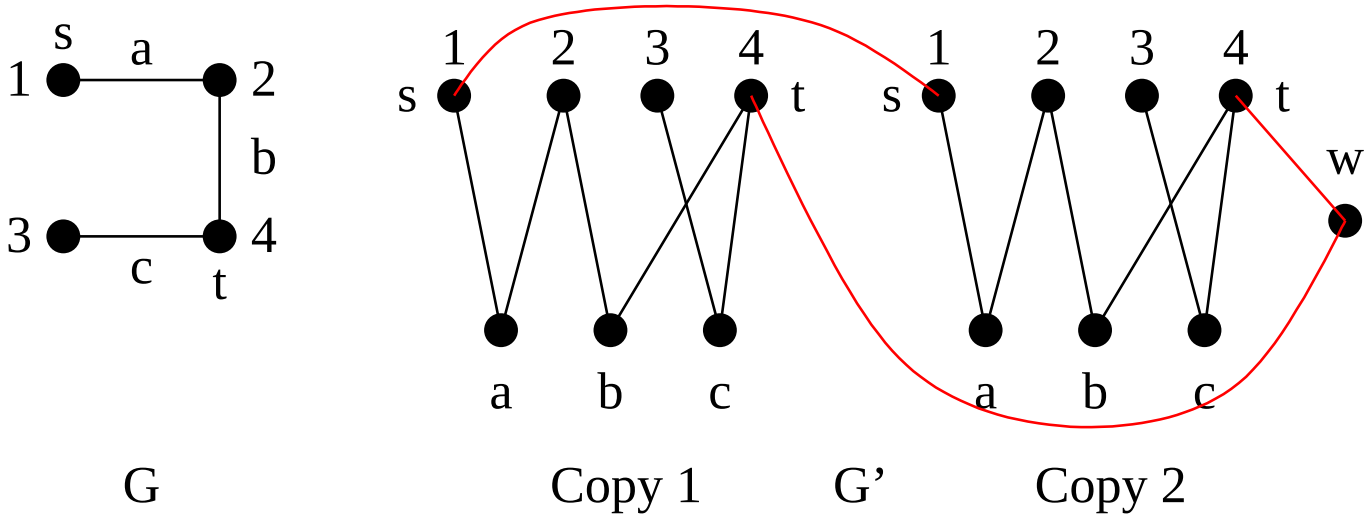
- width W' of $\chi_{G'}$ depends only on W rather than on $|V|$.

Theorem

A sequence of $\mathcal{O}(1)$ arbitrary OBDD operations is width-preserving.

Fixed-Parameter Intractability of Bipartiteness

Exemplarily reduction from undir. s-t-conn. to bipart.:



- Reduction has constant length expression $\Rightarrow$ width-preserving
- Constant width of χ_G implies constant width of $\chi_{G'}$.
- FPT algo. for bipart. would yield pol. algo. for s-t-conn.

Fixed-Parameter Intractability of Bipartiteness

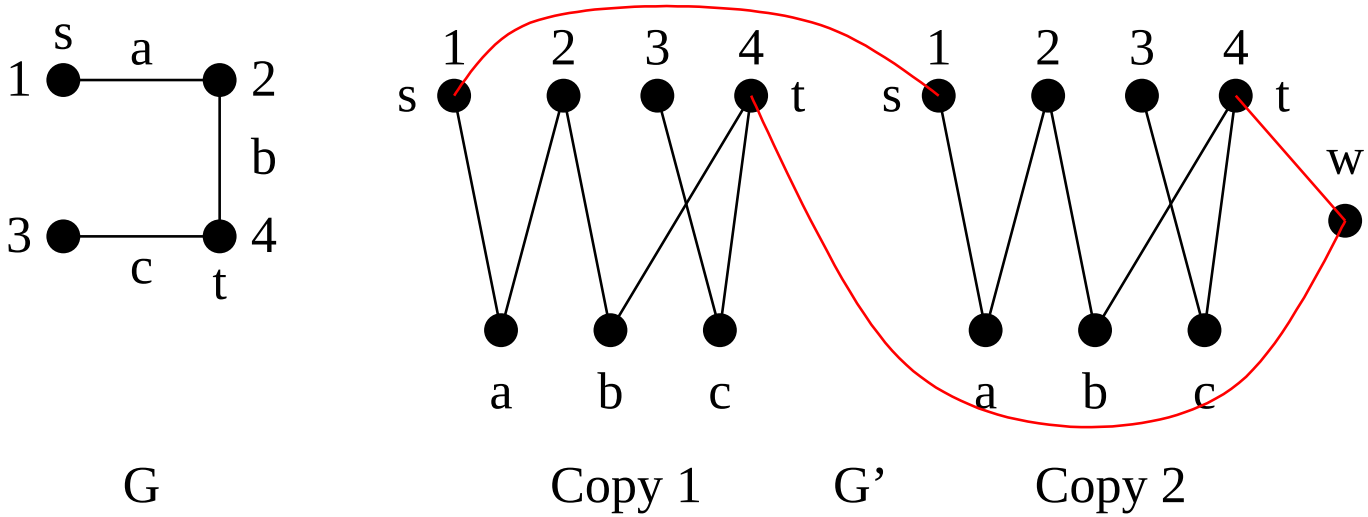
Exemplarily reduction from undir. s-t-conn. to bipart.:

$$\begin{aligned}\chi_{G'}(x,y) := &\big[(T(x)=v)\wedge(T(y)=e)\wedge(i(x)=i(y))\wedge(c(x)=c(y))\wedge\chi_G(i(y),j(y))\big]\\ \vee &\big[(T(x)=e)\wedge(T(y)=v)\wedge(j(x)=j(y))\wedge(c(x)=c(y))\wedge\chi_G(i(x),j(x))\big]\\ \vee &\big[(T(x)=T(y)=v)\wedge(v_{|i(x)|}=v_{|i(y)|}=s)\wedge(c(x)\neq c(y))\big]\\ \vee &\big[(T(x)=v)\wedge(T(y)=w)\wedge(v_{|i(x)|}=t)\big]\ ,\end{aligned}$$

- Reduction has constant length expression $\Rightarrow$ width-preserving
- Constant width of χ_G implies constant width of $\chi_{G'}$.
- FPT algo. for bipart. would yield pol. algo. for s-t-conn.

Fixed-Parameter Intractability of Bipartiteness

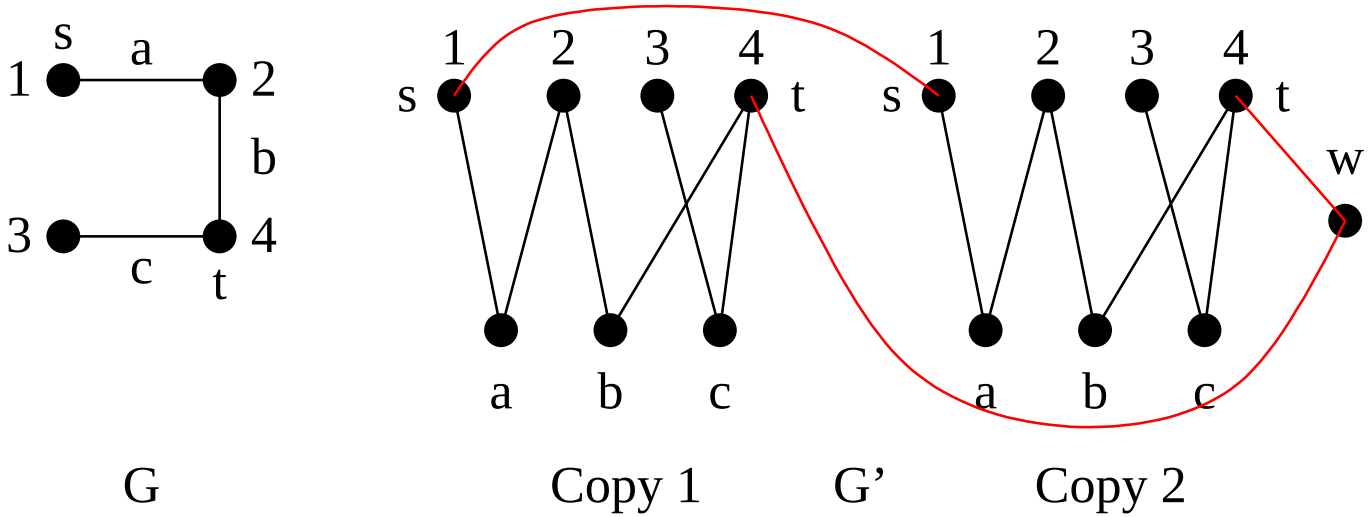
Exemplarily reduction from undir. s-t-conn. to bipart.:



- Reduction has constant length expression $\Rightarrow$ width-preserving
- Constant width of χ_G implies constant width of $\chi_{G'}$.
- FPT algo. for bipart. would yield pol. algo. for s-t-conn.

Fixed-Parameter Intractability of Bipartiteness

Exemplarily reduction from undir. s-t-conn. to bipart.:



- Reduction has constant length expression $\Rightarrow$ width-preserving
- Constant width of χ_G implies constant width of $\chi_{G'}$.
- FPT algo. for bipart. would yield pol. algo. for s-t-conn.

Contents

Summary

If $P \neq NC$ and $P \neq PSPACE$:

- P-complete problems cannot be solved by $\mathcal{O}(\log^k n)$ functional operations.
- Fundamental graph problems have no OBDD-based FPT algorithms w. r. t. fixed input OBDD width.
- Even constant input OBDD width does not suffice for polynomial time w. r. t. $m = \Theta(\log n)$.
- Technique works for many constant depth reductions and read-once projections.
- $\Rightarrow$ Practical success of OBDDs has to be explained by further instance properties.

Summary

If $P \neq NC$ and $P \neq PSPACE$:

- P-complete problems cannot be solved by $\mathcal{O}(\log^k n)$ functional operations.
- Fundamental graph problems have no OBDD-based FPT algorithms w. r. t. fixed input OBDD width.
- Even constant input OBDD width does not suffice for polynomial time w. r. t. $m = \Theta(\log n)$.
- Technique works for many constant depth reductions and read-once projections.
- $\Rightarrow$ Practical success of OBDDs has to be explained by further instance properties.

Summary

If $P \neq NC$ and $P \neq PSPACE$:

- P-complete problems cannot be solved by $\mathcal{O}(\log^k n)$ functional operations.
- Fundamental graph problems have no OBDD-based FPT algorithms w. r. t. fixed input OBDD width.
- Even constant input OBDD width does not suffice for polynomial time w. r. t. $m = \Theta(\log n)$.
- Technique works for many constant depth reductions and read-once projections.
- $\Rightarrow$ Practical success of OBDDs has to be explained by further instance properties.

Summary

If $P \neq NC$ and $P \neq PSPACE$:

- P-complete problems cannot be solved by $\mathcal{O}(\log^k n)$ functional operations.
- Fundamental graph problems have no OBDD-based FPT algorithms w. r. t. fixed input OBDD width.
- Even constant input OBDD width does not suffice for polynomial time w. r. t. $m = \Theta(\log n)$.
- Technique works for many constant depth reductions and read-once projections.
- $\Rightarrow$ Practical success of OBDDs has to be explained by further instance properties.

Summary

If $P \neq NC$ and $P \neq PSPACE$:

- P-complete problems cannot be solved by $\mathcal{O}(\log^k n)$ functional operations.
- Fundamental graph problems have no OBDD-based FPT algorithms w. r. t. fixed input OBDD width.
- Even constant input OBDD width does not suffice for polynomial time w. r. t. $m = \Theta(\log n)$.
- Technique works for many constant depth reductions and read-once projections.
- $\Rightarrow$ Practical success of OBDDs has to be explained by further instance properties.

“That’s all Folks!”