Efficient Problem Solving on Tree Decompositions using Binary Decision Diagrams

Günther Charwat   Stefan Woltran

Database and Artificial Intelligence Group
Institute of Information Systems
TU Wien

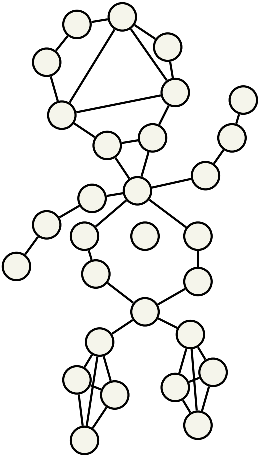
LPNMR'15 - 30 September 2015

Motivation

Solve your favorite intractable (graph) problem...

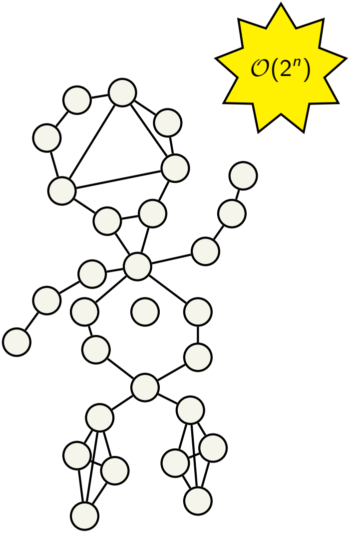
Motivation

Solve your favorite intractable (graph) problem...



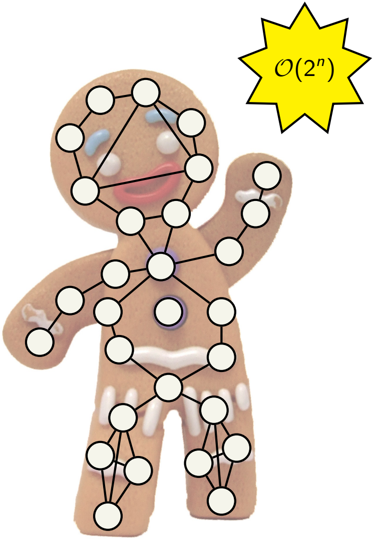
Motivation

Solve your favorite intractable (graph) problem...



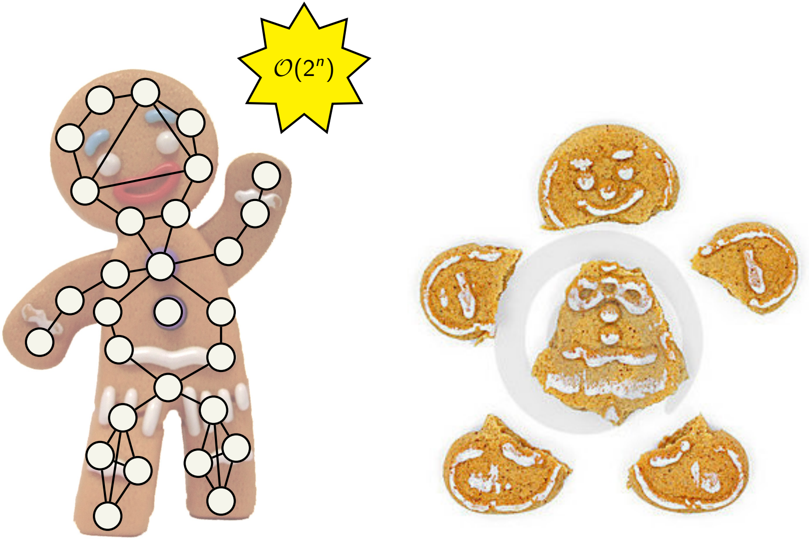
Motivation

Solve your favorite intractable (graph) problem...



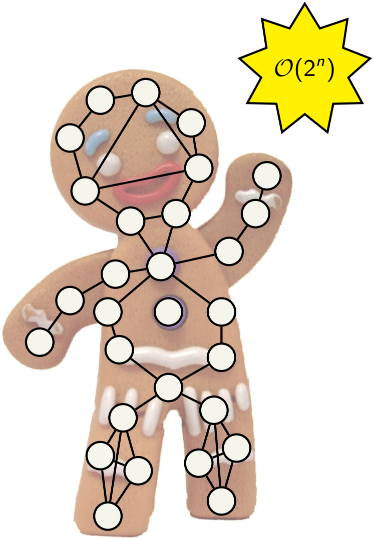
Motivation

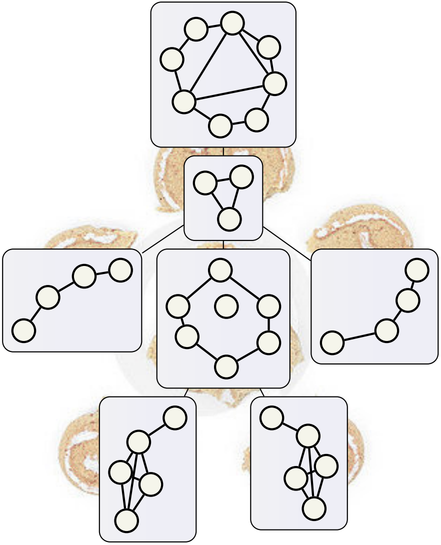
Solve your favorite intractable (graph) problem...



Motivation

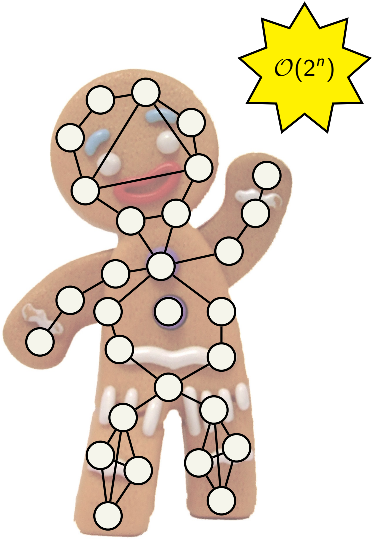
Solve your favorite intractable (graph) problem...

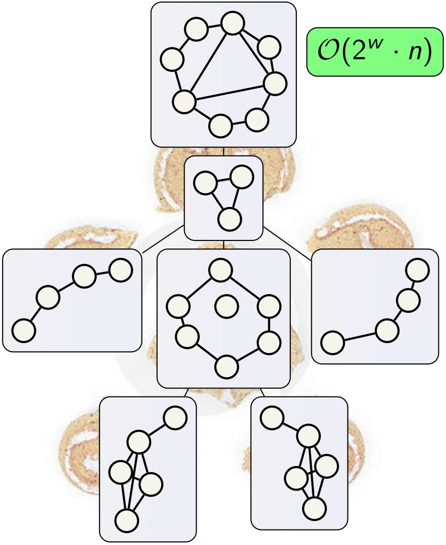




Motivation

Solve your favorite intractable (graph) problem...





Motivation

Dynamic programming on tree decompositions in a nutshell

Basic idea

- For hard problems exploit structural properties of instance
- Confine complexity to a parameter
- Many problems are fixed-parameter tractable (fpt) w.r.t. tree-width w, i.e. solvable in time

$$f(w) \cdot n^{\mathcal{O}(1)}$$

Motivation

Dynamic programming on tree decompositions in a nutshell

Basic idea

- For hard problems exploit structural properties of instance
- Confine complexity to a parameter
- Many problems are fixed-parameter tractable (fpt) w.r.t. tree-width w, i.e. solvable in time

$$f(w) \cdot n^{\mathcal{O}(1)}$$

General Approach

1. Decompose instance
2. Solve partial problems
3. Get result at final node

Motivation

Practical realization

- Intermediate results stored in *tables*
- Computation via *manipulation of rows*

Problem: Large memory footprint

Motivation

Practical realization

- Intermediate results stored in *tables*
- Computation via *manipulation of rows*

Problem: Large memory footprint

Our paradigm

- *Native* support for efficient storage
- *Logic-based* algorithm specifications
- Algorithms define how *sets* of partial solutions are computed

We use **Binary Decision Diagrams** as data structure

Background

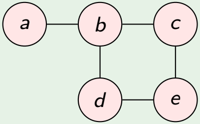
Tree decompositions

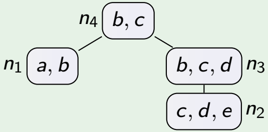
Definition

A *tree decomposition* is a tree obtained from an arbitrary graph s.t.

1. Each vertex must occur in some *bag*.
2. For each edge, there is a bag containing both endpoints.
3. If vertex v appears in bags of nodes n_0 and n_1, then v is also in the bag of each node on the path between n_0 and n_1.

Example





Background

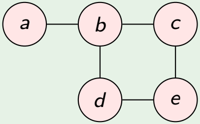
Tree decompositions

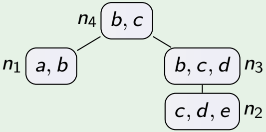
Definition

A *tree decomposition* is a tree obtained from an arbitrary graph s.t.

1. Each vertex must occur in some *bag*.
2. For each edge, there is a bag containing both endpoints.
3. If vertex v appears in bags of nodes n_0 and n_1, then v is also in the bag of each node on the path between n_0 and n_1.

Example





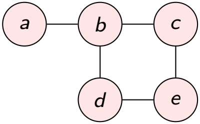
- *Width*: Size of largest bag minus 1
- *Tree-width*: Minimum width over all possible tree decompositions

Background

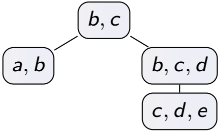
Tree decompositions

Definition

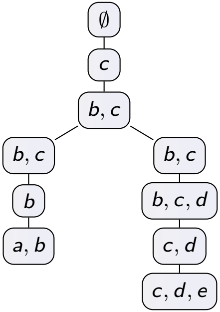
Each node in a *normalized tree decomposition* is of one of the following types: leaf, introduction, removal, or join node.



Input instance



Tree decomposition



Normalized tree decomposition

Background

Binary Decision Diagrams (BDDs)

- Data structure for storing Boolean functions
- Representation as rooted DAG
- Reduced Ordered BDDs [Bryant, 1986] particularly space efficient

Background

Binary Decision Diagrams (BDDs)

- Data structure for storing Boolean functions
- Representation as rooted DAG
- Reduced Ordered BDDs [Bryant, 1986] particularly space efficient

Example (BDD and ROBDD)

Let formula $\phi = (a \land b \land c) \lor (a \land \neg b \land c) \lor (\neg a \land b \land c)$.

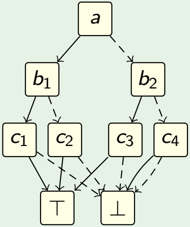
Background

Binary Decision Diagrams (BDDs)

- Data structure for storing Boolean functions
- Representation as rooted DAG
- Reduced Ordered BDDs [Bryant, 1986] particularly space efficient

Example (BDD and ROBDD)

Let formula $\phi = (a \wedge b \wedge c) \vee (a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c)$.



OBBD of ϕ.

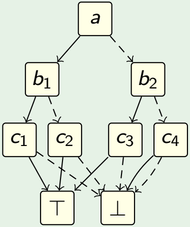
Background

Binary Decision Diagrams (BDDs)

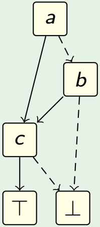
- Data structure for storing Boolean functions
- Representation as rooted DAG
- Reduced Ordered BDDs [Bryant, 1986] particularly space efficient

Example (BDD and ROBDD)

Let formula $\phi = (a \wedge b \wedge c) \vee (a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c)$.



OBBD of ϕ.



ROBBD of ϕ.

Background

Binary Decision Diagrams (BDDs)

Advantages of BDDs

- Well-studied concept (applied to model checking, planning, software verification, . . .)
- Efficient implementations available
- Memory-efficient storage handled directly by data structure
- Logic-based algorithm specification

Background

Binary Decision Diagrams (BDDs)

BDDs support

- Standard logical operators ($\wedge$, $\vee$, $\neg$, $\leftrightarrow$, $\ldots$)
- *Existential quantification* ($\exists V \mathcal{B}$)
- *Restriction* and *renaming* ($\mathcal{B}[v/\cdot]$ where $\cdot \in \{\top, \bot, v'\}$)

Background

Binary Decision Diagrams (BDDs)

BDDs support

- Standard logical operators ($\wedge$, $\vee$, $\neg$, $\leftrightarrow$, ...)
- *Existential quantification* ($\exists V \mathcal{B}$)
- *Restriction* and *renaming* ($\mathcal{B}[v/\cdot]$ where $\cdot \in \{\top, \bot, v'\}$)

Size of ROBDDs

- Bounded by $\mathcal{O}(2^{|V_{\mathcal{B}}|})$
- Heavily depends on variable ordering
- Finding optimal ordering is NP-complete [Bollig and Wegener, 1996]
- But there are good heuristics (e.g., [Rudell, 1993])

In practice often only polynomially large! [Friedman and Supowit, 1987]

Dynamic Programming using BDDs

Concept comparison

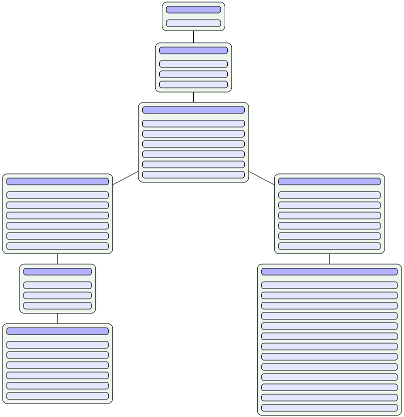


Table-based Dynamic Programming

Dynamic Programming using BDDs

Concept comparison

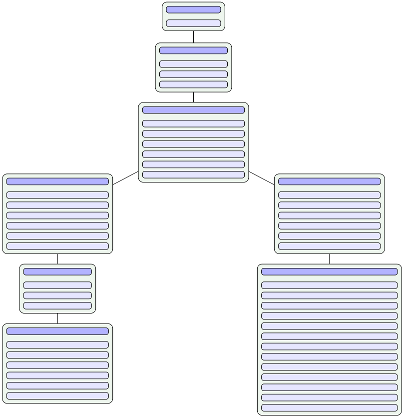
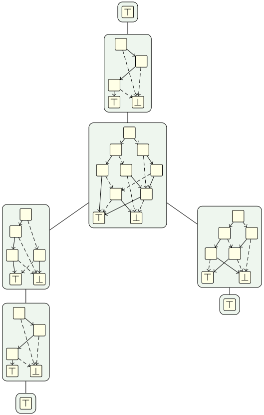


Table-based Dynamic Programming



BDD-based Dynamic Programming

Dynamic Programming using BDDs

Approach

Preparation

- Specify problem-dependent BDD manipulation operations $\mathcal{B}^*$
- Distinguish between node types, here: $* \in \{l, i, r, j\}$ (leaf, introduction, removal, join)

Dynamic Programming using BDDs

Approach

Preparation

- Specify problem-dependent BDD manipulation operations $\mathcal{B}^*$
- Distinguish between node types, here: $* \in \{l, i, r, j\}$ (leaf, introduction, removal, join)

Solve problem

1. Decompose instance to obtain tree decomposition $\mathcal{T}$
2. Traverse $\mathcal{T}$ in post-order and for each node n in $\mathcal{T}$, compute $\mathcal{B}^*_n$ based on node type $*$
3. In root node r of $\mathcal{T}$, either $\mathcal{B}_r = \top$ or $\mathcal{B}_r = \bot$ holds

3-Colorability

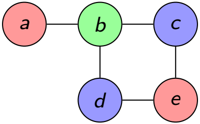
Problem

Given a graph $G = (V, E)$, is G 3-colorable, i.e.:

- each vertex gets assigned exactly one color, and
- neighboring vertices have different colors?

Variables

Color assignment: c_x for all $c \in C = \{r, g, b\}, x \in V$



3-Colorability

$$\mathcal{B}_n^l = \bigwedge_{c \in C} \bigwedge_{\{x,y\} \in E_n} \neg(c_x \wedge c_y) \wedge \bigwedge_{x \in X_n} (r_x \vee g_x \vee b_x) \wedge$$

$$\bigwedge_{x \in X_n} \Big(\neg(r_x \wedge g_x) \wedge \neg(r_x \wedge b_x) \wedge \neg(g_x \wedge b_x) \Big)$$

3-Colorability

$$\mathcal{B}^l_n = \bigwedge_{c \in C} \bigwedge_{\{x,y\} \in E_n} \neg(c_x \wedge c_y) \wedge \bigwedge_{x \in X_n} (r_x \vee g_x \vee b_x) \wedge$$
$$\bigwedge_{x \in X_n} \Big(\neg(r_x \wedge g_x) \wedge \neg(r_x \wedge b_x) \wedge \neg(g_x \wedge b_x) \Big)$$

$$\mathcal{B}^i_n = \mathcal{B}_{n'} \wedge \bigwedge_{c \in C} \bigwedge_{\{x,u\} \in E_n} \neg(c_x \wedge c_u) \wedge (r_u \vee g_u \vee b_u) \wedge$$
$$\neg(r_u \wedge g_u) \wedge \neg(r_u \wedge b_u) \wedge \neg(g_u \wedge b_u)$$

(Here, u is the introduced vertex.)

3-Colorability

$$\mathcal{B}_n^l = \bigwedge_{c \in C} \bigwedge_{\{x,y\} \in E_n} \neg(c_x \wedge c_y) \wedge \bigwedge_{x \in X_n} (r_x \vee g_x \vee b_x) \wedge$$

$$\bigwedge_{x \in X_n} \Big(\neg(r_x \wedge g_x) \wedge \neg(r_x \wedge b_x) \wedge \neg(g_x \wedge b_x) \Big)$$

$$\mathcal{B}_n^i = \mathcal{B}_{n'} \wedge \bigwedge_{c \in C} \bigwedge_{\{x,u\} \in E_n} \neg(c_x \wedge c_u) \wedge (r_u \vee g_u \vee b_u) \wedge$$

$$\neg(r_u \wedge g_u) \wedge \neg(r_u \wedge b_u) \wedge \neg(g_u \wedge b_u)$$

$$\mathcal{B}_n^r = \exists r_u g_u b_u [\mathcal{B}_{n'}]$$

(Here, u is the removed vertex.)

3-Colorability

$$\mathcal{B}_n^l = \bigwedge_{c \in C} \bigwedge_{\{x,y\} \in E_n} \neg(c_x \wedge c_y) \wedge \bigwedge_{x \in X_n} (r_x \vee g_x \vee b_x) \wedge$$
$$\bigwedge_{x \in X_n} \Big(\neg(r_x \wedge g_x) \wedge \neg(r_x \wedge b_x) \wedge \neg(g_x \wedge b_x) \Big)$$

$$\mathcal{B}_n^i = \mathcal{B}_{n'} \wedge \bigwedge_{c \in C} \bigwedge_{\{x,u\} \in E_n} \neg(c_x \wedge c_u) \wedge (r_u \vee g_u \vee b_u) \wedge$$
$$\neg(r_u \wedge g_u) \wedge \neg(r_u \wedge b_u) \wedge \neg(g_u \wedge b_u)$$

$$\mathcal{B}_n^r = \exists r_u g_u b_u [\mathcal{B}_{n'}]$$

$$\mathcal{B}_n^j = \mathcal{B}_{n'} \wedge \mathcal{B}_{n''}$$

Dynamic Programming using BDDs

Algorithm design choices

Early Decision Method (EDM)

- Information is incorporated in introduction nodes
- Comparable to “classical” table-based implementations
- `Unsatisfiable` instances: Conflicts are detected earlier

Dynamic Programming using BDDs

Algorithm design choices

Early Decision Method (EDM)

- Information is incorporated in introduction nodes
- Comparable to "classical" table-based implementations
- `Unsatisfiable` instances: Conflicts are detected earlier

Late Decision Method (LDM)

- BDD manipulation is delayed until removal of vertices
- Typically yields smaller BDDs and less computational effort
- Particularly useful for "complicated" algorithms
- Usually more concise algorithm specification

3-Colorability

Late Decision Method

$$\mathcal{B}^{l}_{n} = \top$$

3-Colorability

Late Decision Method

$\mathcal{B}_n^l = \top \qquad \mathcal{B}_n^i = \mathcal{B}_{n'}$

3-Colorability

Late Decision Method

$$\mathcal{B}_n^l = \top \qquad \mathcal{B}_n^i = \mathcal{B}_{n'} \qquad \mathcal{B}_n^j = \mathcal{B}_{n'} \land \mathcal{B}_{n''}$$

3-Colorability

Late Decision Method

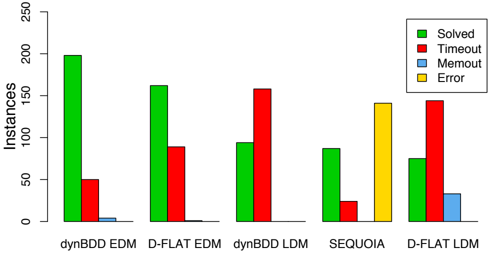
$$\mathcal{B}^l_n = \top \qquad \mathcal{B}^i_n = \mathcal{B}_{n'} \qquad \mathcal{B}^j_n = \mathcal{B}_{n'} \wedge \mathcal{B}_{n''}$$

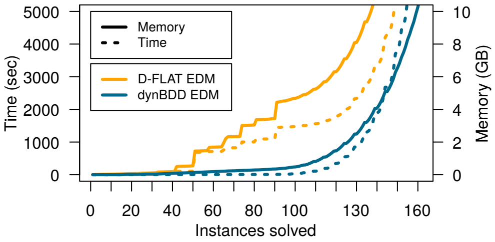
$$\begin{aligned}\mathcal{B}^r_n = &\Big(\mathcal{B}_{n'}[r_u/\top, g_u/\bot, b_u/\bot] \wedge \bigwedge_{\{x,u\}\in E_{n'}} \neg r_x\Big) \vee \\ &\Big(\mathcal{B}_{n'}[r_u/\bot, g_u/\top, b_u/\bot] \wedge \bigwedge_{\{x,u\}\in E_{n'}} \neg g_x\Big) \vee \\ &\Big(\mathcal{B}_{n'}[r_u/\bot, g_u/\bot, b_u/\top] \wedge \bigwedge_{\{x,u\}\in E_{n'}} \neg b_x\Big)\end{aligned}$$

(Here, u is the removed vertex.)

3-COLORABILITY

Preliminary experimental results





Related work

Practical realizations for DP on TDs

- Some problem-specific implementations (e.g. graph optimization, argumentation, . . .)
- SEQUOIA (2011): Takes MSO formula and does DP internally
- D-FLAT (2012): Specify algorithm for particular problem in ASP

Related work

Practical realizations for DP on TDs

- Some problem-specific implementations (e.g. graph optimization, argumentation, . . .)
- SEQUOIA (2011): Takes MSO formula and does DP internally
- D-FLAT (2012): Specify algorithm for particular problem in ASP

Further related approaches

- Branch and Bound on TDs [Allouche et al., 2015]
- Trees-of-BDDs [Fargier and Marquis, 2009]
- Optimization with decision diagrams [Bergman et al., 2015]

Conclusion

Current results

- Feasible for problems that are fpt w.r.t. tree-width w
  - Size of BDDs bounded by $\mathcal{O}(2^{w \cdot c})$
- So far, NP-complete problems were considered:
  - 3-Colorability: only variables with *fixed* truth value
  - Dominating Set variant: variables with *changing* truth value
  - Hamiltonian Cycle: handle *connectedness* in DP algorithm
- Development and study of design patterns EDM and LDM

Conclusion

Current results

- Feasible for problems that are fpt w.r.t. tree-width w
  - Size of BDDs bounded by $\mathcal{O}(2^{w \cdot c})$
- So far, NP-complete problems were considered:
  - 3-COLORABILITY: only variables with *fixed* truth value
  - DOMINATING SET variant: variables with *changing* truth value
  - HAMILTONIAN CYCLE: handle *connectedness* in DP algorithm
- Development and study of design patterns EDM and LDM

Future work

- Consider problems harder than NP (via *sets of BDDs*)
- Optimization problems (use alternatives to BDDs)
- Support for high-level algorithm specification
- Visualization and debugging support for algorithm development

http://dbai.tuwien.ac.at/proj/decodyn/dynbdd

http://dbai.tuwien.ac.at/proj/decodyn/dynbdd

References I

Allouche, D., De Givry, S., Katsirelos, G., Schiex, T., and Zytnicki, M. (2015).
Anytime hybrid best-first search with tree decomposition for weighted CSP.
In *Principles and Practice of Constraint Programming*, pages 12–29. Springer.

Bergman, D., Cire, A. A., van Hoeve, W.-J., and Hooker, J. (2015).
Discrete optimization with decision diagrams.
To appear in INFORMS Journal on Computing.

Bollig, B. and Wegener, I. (1996).
Improving the variable ordering of OBDDs is NP-complete.
IEEE Trans. Comp., 45(9):993–1002.

References II

Bryant, R. E. (1986).
Graph-based algorithms for boolean function manipulation.
IEEE Transactions on Computers, 100(8):677–691.

Fargier, H. and Marquis, P. (2009).
Knowledge compilation properties of Trees-of-BDDs, revisited.
In *Proc. IJCAI*, pages 772–777.

Friedman, S. J. and Supowit, K. J. (1987).
Finding the optimal variable ordering for binary decision diagrams.
In *Proc. IEEE Design Automation Conference*, pages 348–356. ACM.

Rudell, R. (1993).
Dynamic variable ordering for ordered binary decision diagrams.
In *Proc. ICCAD*, pages 42–47. IEEE CSP.