Knowledge Compilation und #SAT

Narek Bojikian

Humboldt University of Berlin

08.01.2019

Definitions

- The SAT Problem (**SAT**).

SAT

– Given a Boolean formula φ of n variables.

? Find an assignment that satisfies φ.

Definitions

- The SAT Problem (**SAT**).
- Counting SAT Problem (**#SAT**).

#SAT

– Given a Boolean formula φ of n variables.

? How many assignments in $2^{\mathrm{Var}(\varphi)}$ satisfy φ?

Definitions

- The SAT Problem (**SAT**).
- Counting SAT Problem (**#SAT**).

Notation

Let $\mathrm{SAT}(\chi) \subseteq 2^{\mathrm{VAR}(\chi)}$ be the set of all satisfying assignments of χ

$$\mathrm{SAT}(\chi) = \{\rho : \mathrm{VAR}(\chi) \to \{0,1\} : \rho(\chi) = 1\}.$$

Definitions

- The SAT Problem (**SAT**).
- Counting SAT Problem (**#SAT**).

Notation

Let $\mathrm{SAT}(\chi) \subseteq 2^{\mathrm{VAR}(\chi)}$ be the set of all satisfying assignments of χ

$$\mathrm{SAT}(\chi) = \{\rho : \mathrm{VAR}(\chi) \to \{0,1\} : \rho(\chi) = 1\}.$$

SAT: Is $\mathrm{SAT}(\varphi) = \emptyset$. #SAT: Find $|\mathrm{SAT}(\varphi)|$.

Definitions

- The SAT Problem (**SAT**).
- Counting SAT Problem (**#SAT**).

Example

$$\varphi = X_1 \wedge (X_2 \vee \neg X_3)$$

Clearly, $\#\mathrm{SAT}(\varphi) = 3$.

Definitions

- The SAT Problem (**SAT**).
- Counting SAT Problem (**#SAT**).
- Negation Normal Form (**NNF**).

Negation Normal Form

A Boolean formula φ is in NNF form, if it contains only disjunctions and conjunctions over a set of positive and negative literals.
Example. $\varphi = X_1 \vee \neg X_2$.

Definitions

- The SAT Problem (**SAT**).
- Counting SAT Problem (**#SAT**).
- Negation Normal Form (**NNF**).
- Conjunctive Normal Form (**CNF**).

Conjunctive Normal Form

A Boolean formula φ is in CNF, if it is a conjunction of one or more clauses, where each clauses is a disjunction of one or more literals. Note that each CNF formula is an NNF formula as well.

Definitions

- The SAT Problem (**SAT**).
- Counting SAT Problem (**#SAT**).
- Negation Normal Form (**NNF**).
- Conjunctive Normal Form (**CNF**).
- Decomposable Negation Normal Form (**DNNF**).

Decomposable Negation Normal Form

A Boolean formula φ is in DNNF, if it is in NNF and for each conjunction subformula *phi'* $:= \psi_1 \wedge \psi_2$ we have $\mathrm{VAR}(\psi_1) \cap \mathrm{VAR}(\psi_2) = \emptyset$.

- Satisfying each subformula is independent.

Definitions

- The SAT Problem (**SAT**).
- Counting SAT Problem (**#SAT**).
- Negation Normal Form (**NNF**).
- Conjunctive Normal Form (**CNF**).
- Decomposable Negation Normal Form (**DNNF**).
- deterministic Decomposable Negation Normal Form (**d-DNNF**).

deterministic Decomposable Negation Normal Form

A Boolean formula φ is in d-DNNF, if it is in DNNF and for each disjunction subformula $\varphi' = \psi_1 \vee \psi_2$ we have $\mathrm{SAT}(\psi_1) \cap \mathrm{SAT}(\psi_2) = \emptyset$.

Definitions

- The SAT Problem (**SAT**).
- Counting SAT Problem (**#SAT**).
- Negation Normal Form (**NNF**).
- Conjunctive Normal Form (**CNF**).
- Decomposable Negation Normal Form (**DNNF**).
- deterministic Decomposable Negation Normal Form (**d-DNNF**).
- decision Decomposable Negation Normal Form (**dec-DNNF**).

decision Decomposable Negation Normal Form

A Boolean formula φ is in dec-DNNF, if it is in DNNF and each disjunction subformula φ' is of the form $\varphi' = (X \wedge \psi_1) \vee (\neg X \wedge \psi_2)$ for some variable $X \in \mathrm{VAR}(\varphi)$.

Definitions

- The SAT Problem (**SAT**).
- Counting SAT Problem (**#SAT**).
- Negation Normal Form (**NNF**).
- Conjunctive Normal Form (**CNF**).
- Decomposable Negation Normal Form (**DNNF**).
- deterministic Decomposable Negation Normal Form (**d-DNNF**).
- decision Decomposable Negation Normal Form (**dec-DNNF**).

decision Decomposable Negation Normal Form

A Boolean formula φ is in dec-DNNF, if it is in DNNF and each disjunction subformula φ' is of the form $\varphi' = (X \wedge \psi_1) \vee (\neg X \wedge \psi_2)$ for some variable $X \in \mathrm{VAR}(\varphi)$.

Note. Each dec-DNNF is a d-DNNF.

Examples

[On the blackboard..]

Goals for today

Build a dec-DNNF of polynomial size for *β-acyclic* formulas.

Show a subclass of β-acyclic formulas, where each *structured* d-DNNF has exponential size.

Assignments

- Given a CNF Formula φ, an **assignment** for C is a function $\tau : \mathrm{VAR}(C) \to \{0, 1\}$.

Assignments

- Given a CNF Formula φ, an **assignment** for C is a function $\tau : \mathrm{VAR}(C) \to \{0, 1\}$.
- For $V' \subseteq \mathrm{VAR}(C)$, we define the **partial assignment** $\tau_{|V'} : V' \to \{0, 1\}$ as τ restricted to the variables in V'.

Assignments

- Given a CNF Formula φ, an **assignment** for C is a function $\tau : \mathrm{VAR}(C) \to \{0, 1\}$.
- For $V' \subseteq \mathrm{VAR}(C)$, we define the **partial assignment** $\tau_{|V'} : V' \to \{0, 1\}$ as τ restricted to the variables in V'.
- A partial assignment $\tau_{|V'}$ satisfies a CNF-formula φ ($\tau_{|V'} \models \varphi$), if for each clause $C \in \varphi$ there is a variable $v \in \mathrm{VAR}(C) \cap V'$ such that $\tau_{|V'}(v) = 1$ if and only if v appears in C as a positive literal.

Assignments

- Given a CNF Formula φ, an **assignment** for C is a function $\tau : \mathrm{VAR}(C) \to \{0,1\}$.
- For $V' \subseteq \mathrm{VAR}(C)$, we define the **partial assignment** $\tau_{|V'} : V' \to \{0,1\}$ as τ restricted to the variables in V'.
- A partial assignment $\tau_{|V'}$ satisfies a CNF-formula φ ($\tau_{|V'} \models \varphi$), if for each clause $C \in \varphi$ there is a variable $v \in \mathrm{VAR}(C) \cap V'$ such that $\tau_{|V'}(v) = 1$ if and only if v appears in C as a positive literal.

Example

$$\varphi := (v_1 \vee \neg v_2 \vee v_3) \wedge (v_1 \vee v_2) \wedge (\neg v_2 \vee \neg v_3)$$

For $V' = \{v_1, v_2\}, \tau_{|V'}(v_1) = 1, \tau_{|V'}(v_2) = 0$, the partial assignment $\tau_{|V'}$ satisfies φ.

Hypergraphs

- Hypergraph $\mathcal{H}$.
  - A set of vertices $V(\mathcal{H})$.
  - Edges $E(\mathcal{H})$, defined as subsets over $V(\mathcal{H})$.

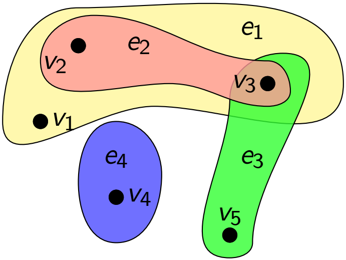


Figure: https://tex.stackexchange.com/a/1195/163902

Hypergraphs

- Hypergraph $\mathcal{H}$.
  - A set of vertices $V(\mathcal{H})$.
  - Edges $E(\mathcal{H})$, defined as subsets over $V(\mathcal{H})$.
- A walk is sequence $(e_1, x_1, \ldots, x_n, e_{n+1})$, $e_i \in \mathcal{H}, x_i \in V(\mathcal{H})$ and $x_i \in e_i \cap e_{i-1}$ for all $i \in [n]$.

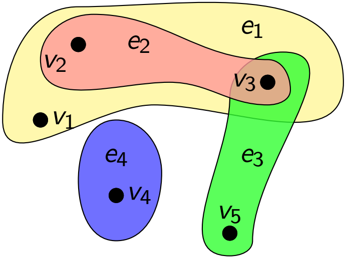


Figure:
https://tex.stackexchange.com/a/1195/163902

Hypergraphs

- Hypergraph $\mathcal{H}$.
  - A set of vertices $V(\mathcal{H})$.
  - Edges $E(\mathcal{H})$, defined as subsets over $V(\mathcal{H})$.
- A walk is sequence $(e_1, x_1, \ldots, x_n, e_{n+1})$, $e_i \in \mathcal{H}, x_i \in V(\mathcal{H})$ and $x_i \in e_i \cap e_{i-1}$ for all $i \in [n]$.
- A **path** is a walk that never goes twice through the same vertex nor the same edge.

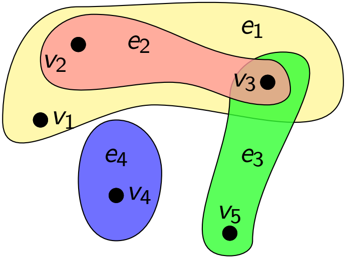


Figure:
https://tex.stackexchange.com/a/1195/163902

Hypergraphs

- Hypergraph $\mathcal{H}$.
  - A set of vertices $V(\mathcal{H})$.
  - Edges $E(\mathcal{H})$, defined as subsets over $V(\mathcal{H})$.
- A walk is sequence $(e_1, x_1, \ldots, x_n, e_{n+1})$, $e_i \in \mathcal{H}, x_i \in V(\mathcal{H})$ and $x_i \in e_i \cap e_{i-1}$ for all $i \in [n]$.
- A **path** is a walk that never goes twice through the same vertex nor the same edge.
- Different ways to translate

  acyclicity to hypergraphs.

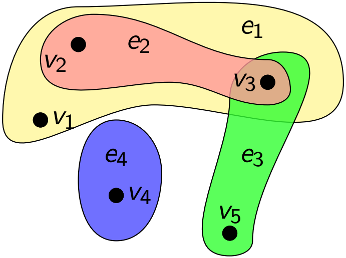
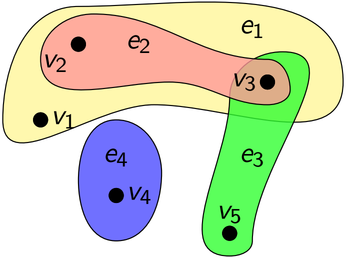


Figure:
https://tex.stackexchange.com/a/1195/163902

β-acyclic graphs

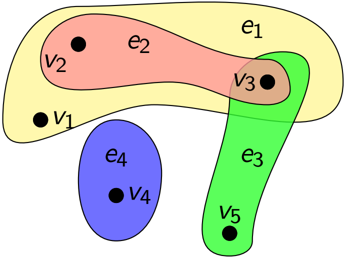
- Let $\rho := v_1, \dots v_n$ be an enumeration of the vertices.



[1] $e_{|\geq i} := e \cap \{v_i, \dots v_n\}$.

β-acyclic graphs

- Let $\rho := v_1, \dots v_n$ be an enumeration of the vertices.
- ρ is a β-elimination, if for all[1]

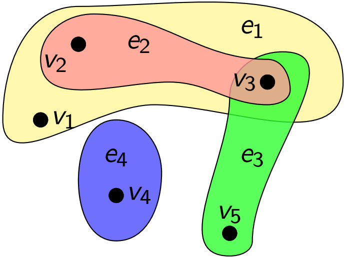


[1] $e_{|\geq i} := e \cap \{v_i, \dots v_n\}$.

β-acyclic graphs

- Let $\rho := v_1, \dots v_n$ be an enumeration of the vertices.
- ρ is a β-elimination, if for all[1]

$$e_1, e_2 \in E(\mathcal{H}) \text{ and } v_i \in e_1 \cap e_2,$$
$$e_{1|\geq i} \subseteq e_2 \text{ or } e_{2|\geq i} \subseteq e_1.$$



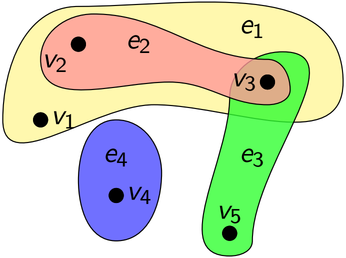
[1] $e_{|\geq i} := e \cap \{v_i, \dots v_n\}$.

β-acyclic graphs

- Let $\rho := v_1, \dots v_n$ be an enumeration of the vertices.
- ρ is a β-elimination, if for all[1]

$$e_1, e_2 \in E(\mathcal{H}) \text{ and } v_i \in e_1 \cap e_2,$$
$$e_{1|\geq i} \subseteq e_2 \text{ or } e_{2|\geq i} \subseteq e_1.$$

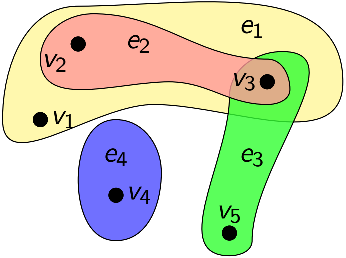
- A hypergraph is β-acyclic, if it admits a β-elimination.



[1] $e_{|\geq i} := e \cap \{v_i, \dots v_n\}$.

β-acyclic graphs

- Let $\rho := v_1, \dots v_n$ be an enumeration of the vertices.
- ρ is a β-elimination, if for all[1]
  $$e_1, e_2 \in E(\mathcal{H}) \text{ and } v_i \in e_1 \cap e_2,$$
  $$e_{1|\geq i} \subseteq e_2 \text{ or } e_{2|\geq i} \subseteq e_1.$$
- A hypergraph is β-acyclic, if it admits a β-elimination.
- We define $V_{\leq v_i} := \{v_j ; j \leq i\}$.

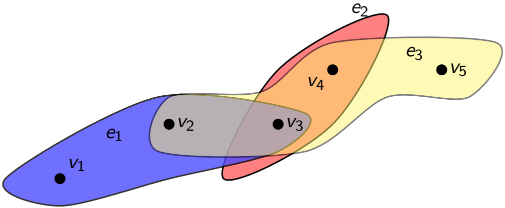


[1] $e_{|\geq i} := e \cap \{v_i, \dots v_n\}$.

Polynomial upper-bound on the practical method

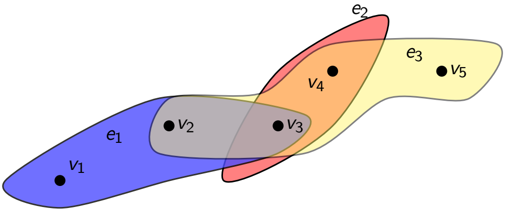
Lemmas on β-acyclic graphs

- Let $\mathcal{H}$ be a β-acyclic graph and $v_1, \ldots v_n$ a β-elimination.



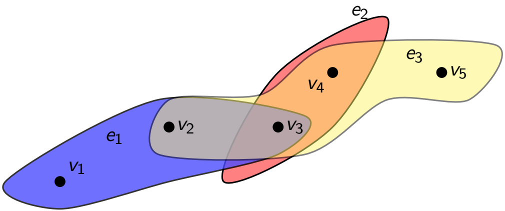
Lemmas on β-acyclic graphs

- Let $\mathcal{H}$ be a β-acyclic graph and $v_1, \dots v_n$ a β-elimination.
- For two edge $e, f \in \mathcal{H}$, $e < f$, if and only if $\max\{e \Delta f\} \in f$



Lemmas on β-acyclic graphs

- Let $\mathcal{H}$ be a β-acyclic graph and $v_1, \dots v_n$ a β-elimination.
- For two edge $e, f \in \mathcal{H}$, $e < f$, if and only if $\max\{e \Delta f\} \in f$
- $\mathcal{H}_e^x$ denotes the subgraph of $\mathcal{H}$, that contain the edges f, such that there is a walk from f to e that goes only through edges smaller than e and vertices smaller than (or equal to) x.



Lemmas on β-acyclic graphs

- Let $\mathcal{H}$ be a β-acyclic graph and $v_1, \dots v_n$ a β-elimination.
- For two edge $e, f \in \mathcal{H}$, $e < f$, if and only if $\max\{e \Delta f\} \in f$
- $\mathcal{H}_e^x$ denotes the subgraph of $\mathcal{H}$, that contain the edges f, such that there is a walk from f to e that goes only through edges smaller than e and vertices smaller than (or equal to) x.

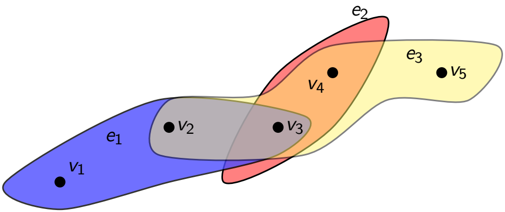


Figure: Note that $e_2 \notin \mathcal{H}_{e_3}^{v_2}$ meanwhile $e_2 \in \mathcal{H}_{e_3}^{v_3}$

Lemmas on β-acyclic graphs

Lemma (lemma 2)

For $x, y \in V(\mathcal{H}), x \leq y$ and for $e, f \in \mathcal{H}, e \leq f$,

$$\text{if } V(\mathcal{H}_e^x) \cap V(\mathcal{H}_f^y) \cap V_{\leq x} \neq \emptyset, \text{ then } \mathcal{H}_e^x \subseteq \mathcal{H}_f^y.$$

In particular, for all $y \in V(\mathcal{H})$,

$$\text{if } e \in \mathcal{H}_f^y, \text{ then } \mathcal{H}_e^y \subseteq \mathcal{H}_f^y$$

Lemmas on β-acyclic graphs

Lemma (lemma 2)

For $x, y \in V(\mathcal{H}), x \leq y$ and for $e, f \in \mathcal{H}, e \leq f$,

$$\text{if } V(\mathcal{H}_e^x) \cap V(\mathcal{H}_f^y) \cap V_{\leq x} \neq \emptyset, \text{ then } \mathcal{H}_e^x \subseteq \mathcal{H}_f^y.$$

In particular, for all $y \in V(\mathcal{H})$,

$$\text{if } e \in \mathcal{H}_f^y, \text{ then } \mathcal{H}_e^y \subseteq \mathcal{H}_f^y$$

Proof sketch. For $g \in \mathcal{H}_e^x$, there is a path from g to e using edges smaller than e and vertices smaller than x.
There is also a path from e to f. Concatenate both paths to get a path from g to f.

Lemmas on β-acyclic graphs

Lemma (lemma 4)

For $e, f \in \mathcal{H}, e \leq f$, If there exists a vertex $x \in V(\mathcal{H})$, such that $x \in e \cap f$, then $e \cap V_{\geq x} \subseteq f$.

Lemmas on β-acyclic graphs

Lemma (lemma 4)

For $e, f \in \mathcal{H}, e \leq f$, If there exists a vertex $x \in V(\mathcal{H})$, such that $x \in e \cap f$, then $e \cap V_{\geq x} \subseteq f$.

Proof sketch. If $y \in e \setminus f$ such that $y > x$, then $\mathcal{H}$ is not β-acyclic.

Lemmas on β-acyclic graphs

A path $(e_1, x_1, \dots e_{n+1})$ is called decreasing, if $e_i > e_{i+1}$ and $x_i > x_{i+1}$ for all i.

Lemma (lemma 5)

For $x \in V(\mathcal{H}), e \in \mathcal{H}$ and $f \in \mathcal{H}_e^x$, there exists a decreasing path from e to f going through vertices smaller than x.

Lemmas on β-acyclic graphs

A path $(e_1, x_1, \dots e_{n+1})$ is called decreasing, if $e_i > e_{i+1}$ and $x_i > x_{i+1}$ for all i.

Lemma (lemma 5)

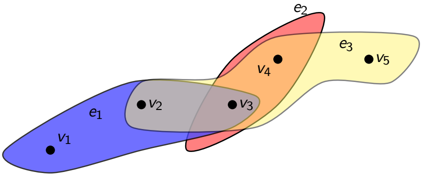
For $x \in V(\mathcal{H}), e \in \mathcal{H}$ and $f \in \mathcal{H}_e^x$, there exists a decreasing path from e to f going through vertices smaller than x.

Proof sketch. Any shortest path from e to f is decreasing. A path exists by definition.

Lemmas on β-acyclic graphs

Theorem (theorem 3)

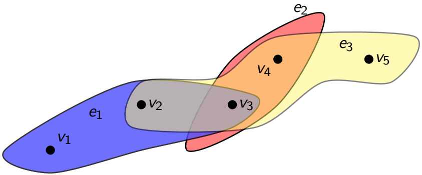
For every $x \in V(\mathcal{H})$ and $e \in \mathcal{H}$, $V(\mathcal{H}_e^x) \cap V_{\geq x} \subseteq e$



Lemmas on β-acyclic graphs

Theorem (theorem 3)

For every $x \in V(\mathcal{H})$ and $e \in \mathcal{H}$, $V(\mathcal{H}_e^x) \cap V_{\geq x} \subseteq e$

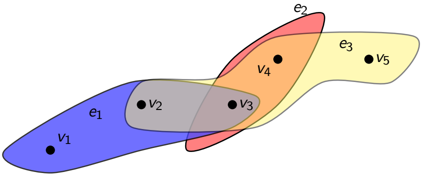


Proof sketch. Prove that all edges of a decreasing path are subsets of the first edge by induction over the length of the path.

Lemmas on β-acyclic graphs

Theorem (theorem 3)

For every $x \in V(\mathcal{H})$ and $e \in \mathcal{H}$, $V(\mathcal{H}_e^x) \cap V_{\geq x} \subseteq e$



Proof sketch. Prove that all edges of a decreasing path are subsets of the first edge by induction over the length of the path.
Intuitively, this allows us to use dynamic programming, since all variables in $\mathcal{H}_e^x$ not contained in e are smaller than x.

Solving #SAT in β-acyclic graphs

- The hypergraph of a CNF-formula:

Solving #SAT in β-acyclic graphs

- The hypergraph of a CNF-formula:

  Variables of each clause correspond to an edge.

Solving #SAT in β-acyclic graphs

- The hypergraph of a CNF-formula:

  Variables of each clause correspond to an edge.

  Two clauses might correspond to the same edge.

Solving #SAT in β-acyclic graphs

- The hypergraph of a CNF-formula:

  Variables of each clause correspond to an edge.

  Two clauses might correspond to the same edge.
- A β-acyclic CNF-formula F is given.

Solving #SAT in β-acyclic graphs

- The hypergraph of a CNF-formula:

  Variables of each clause correspond to an edge.

  Two clauses might correspond to the same edge.
- A β-acyclic CNF-formula F is given.
- Let $\mathcal{H}$ be the hypergraph of F.

Solving #SAT in β-acyclic graphs

- The hypergraph of a CNF-formula:

  Variables of each clause correspond to an edge.

  Two clauses might correspond to the same edge.
- A β-acyclic CNF-formula F is given.
- Let $\mathcal{H}$ be the hypergraph of F.
- Let $v_1, \ldots v_n$ be an elimination order in $\mathcal{H}$.

Solving #SAT in β-acyclic graphs

- The hypergraph of a CNF-formula:

  Variables of each clause correspond to an edge.

  Two clauses might correspond to the same edge.
- A β-acyclic CNF-formula F is given.
- Let $\mathcal{H}$ be the hypergraph of F.
- Let $v_1, \dots v_n$ be an elimination order in $\mathcal{H}$.
- Let F_e^x be the subformula of F that corresponds to $\mathcal{H}_e^x$,

  i.e. $C \in F_e^x$, if $\mathrm{VAR}(C) \in \mathcal{H}_e^x$.

Solving #SAT in β-acyclic graphs

- The hypergraph of a CNF-formula:

  Variables of each clause correspond to an edge.

  Two clauses might correspond to the same edge.
- A β-acyclic CNF-formula F is given.
- Let $\mathcal{H}$ be the hypergraph of F.
- Let $v_1, \ldots v_n$ be an elimination order in $\mathcal{H}$.
- Let F_e^x be the subformula of F that corresponds to $\mathcal{H}_e^x$,

  i.e. $C \in F_e^x$, if $\mathrm{VAR}(C) \in \mathcal{H}_e^x$.
- For a clause C, the partial assignment τ_C is defined over $\mathrm{VAR}(C)$ as the only assignment that does not satisfy C,

Solving #SAT in β-acyclic graphs

- The hypergraph of a CNF-formula:

  Variables of each clause correspond to an edge.

  Two clauses might correspond to the same edge.
- A β-acyclic CNF-formula F is given.
- Let $\mathcal{H}$ be the hypergraph of F.
- Let $v_1, \ldots v_n$ be an elimination order in $\mathcal{H}$.
- Let F_e^x be the subformula of F that corresponds to $\mathcal{H}_e^x$,

  i.e. $C \in F_e^x$, if $\mathrm{VAR}(C) \in \mathcal{H}_e^x$.
- For a clause C, the partial assignment τ_C is defined over $\mathrm{VAR}(C)$ as the only assignment that does not satisfy C,

  i.e. for $x \in C$, $\tau_C(x) = 1$ if and only if x appears as a negative literal in C.

Solving #SAT in β-acyclic graphs

- The hypergraph of a CNF-formula:

  Variables of each clause correspond to an edge.

  Two clauses might correspond to the same edge.
- A β-acyclic CNF-formula F is given.
- Let $\mathcal{H}$ be the hypergraph of F.
- Let $v_1, \dots v_n$ be an elimination order in $\mathcal{H}$.
- Let F_e^x be the subformula of F that corresponds to $\mathcal{H}_e^x$,

  i.e. $C \in F_e^x$, if $\mathrm{VAR}(C) \in \mathcal{H}_e^x$.
- For a clause C, the partial assignment τ_C is defined over $\mathrm{VAR}(C)$ as the only assignment that does not satisfy C,

  i.e. for $x \in C$, $\tau_C(x) = 1$ if and only if x appears as a negative literal in C.
- We define $\tau_C^x := \tau_{C|\geq x}$,

Solving #SAT in β-acyclic graphs

- The hypergraph of a CNF-formula:

  Variables of each clause correspond to an edge.

  Two clauses might correspond to the same edge.
- A β-acyclic CNF-formula F is given.
- Let $\mathcal{H}$ be the hypergraph of F.
- Let $v_1, \ldots v_n$ be an elimination order in $\mathcal{H}$.
- Let F_e^x be the subformula of F that corresponds to $\mathcal{H}_e^x$,

  i.e. $C \in F_e^x$, if $\mathrm{VAR}(C) \in \mathcal{H}_e^x$.
- For a clause C, the partial assignment τ_C is defined over $\mathrm{VAR}(C)$ as the only assignment that does not satisfy C,

  i.e. for $x \in C$, $\tau_C(x) = 1$ if and only if x appears as a negative literal in C.
- We define $\tau_C^x := \tau_{C|\geq x}$,

  $F[\tau_C^x]$ results from F by removing all variables greater than x from each clause.

Solving #SAT in β-acyclic graphs

Lemma (lemma 6)

Let $x \neq x_1 \in \mathrm{VAR}(F)$ and let y be the predecessor of x for $<$. Let $e \in \mathcal{H}$ and $\tau : (e \cap V_{\geq x}) \to \{0,1\}$. Then either $F_e^x[\tau] \equiv 1$ or there exists $U \subseteq \mathcal{H}_e^x$ such that

$$F_e^x[\tau] \equiv \bigwedge_{g \in U} F_g^y[\tau_{C_g}^y],$$

where C_g is some clause in F_e^x such that $\mathrm{VAR}(C_g) = g$.
Moreover, all and-gates are decomposable and U can be computed in polynomial time.

$$F_e^x[\tau] \equiv \bigwedge_{g \in U} F_g^y[\tau_{C_g}^y],$$

Proof sketch.

- Let A be the set edges g, where $\tau \not\models C_g$ for some *corresponding* C_g.

Solving #SAT in β-acyclic graphs

$$F_e^x[\tau] \equiv \bigwedge_{g \in U} F_g^y[\tau_{C_g}^y],$$

Proof sketch.

- Let A be the set edges g, where $\tau \not\models C_g$ for some *corresponding* C_g.
- For each clause C such that $\mathrm{VAR}(C) \notin A$ we have $\tau \models C$.

Solving #SAT in β-acyclic graphs

$$F_e^x[\tau] \equiv \bigwedge_{g \in U} F_g^y[\tau_{C_g}^y],$$

Proof sketch.

- Let A be the set edges g, where $\tau \not\models C_g$ for some *corresponding* C_g.
- For each clause C such that $\mathrm{VAR}(C) \notin A$ we have $\tau \models C$.
- Choose U as the set of "maximal" edges $g \in A$,

Solving #SAT in β-acyclic graphs

$$F_e^x[\tau] \equiv \bigwedge_{g \in U} F_g^y[\tau_{C_g}^y],$$

Proof sketch.

- Let A be the set edges g, where $\tau \not\models C_g$ for some *corresponding* C_g.
- For each clause C such that $\mathrm{VAR}(C) \notin A$ we have $\tau \models C$.
- Choose U as the set of "maximal" edges $g \in A$,
- For each $f \in A$, there is $g \in U$ such that $f \in \mathcal{H}_g^y$.

Solving #SAT in β-acyclic graphs

$$F_e^x[\tau] \equiv \bigwedge_{g \in U} F_g^y[\tau_{C_g}^y],$$

Proof sketch.

- Let A be the set edges g, where $\tau \not\models C_g$ for some *corresponding* C_g.
- For each clause C such that $\mathrm{VAR}(C) \notin A$ we have $\tau \models C$.
- Choose U as the set of "maximal" edges $g \in A$,
- For each $f \in A$, there is $g \in U$ such that $f \in \mathcal{H}_g^y$.

  i.e. $g \not\subseteq \mathcal{H}_f^y$ for all $f \in A$, $f \neq g$.

$$F_e^x[\tau] \equiv \bigwedge_{g \in U} F_g^y[\tau_{C_g}^y],$$

Proof sketch.

- Let A be the set edges g, where $\tau \not\models C_g$ for some *corresponding* C_g.
- For each clause C such that $\mathrm{VAR}(C) \notin A$ we have $\tau \models C$.
- Choose U as the set of "maximal" edges $g \in A$,
- For each $f \in A$, there is $g \in U$ such that $f \in \mathcal{H}_g^y$.

  i.e. $g \not\subseteq \mathcal{H}_f^y$ for all $f \in A$, $f \neq g$.
- U can be computed in polynomial time.

Solving #SAT in β-acyclic graphs

$$F_e^x[\tau] \equiv \bigwedge_{g \in U} F_g^y[\tau_{C_g}^y],$$

Proof sketch.

- Let A be the set edges g, where $\tau \not\models C_g$ for some *corresponding* C_g.
- For each clause C such that $\mathrm{VAR}(C) \notin A$ we have $\tau \models C$.
- Choose U as the set of "maximal" edges $g \in A$,
- For each $f \in A$, there is $g \in U$ such that $f \in \mathcal{H}_g^y$.

  i.e. $g \not\subseteq \mathcal{H}_f^y$ for all $f \in A$, $f \neq g$.
- U can be computed in polynomial time.
- The edges in U are pairwise disjoint. Hence, the and-gate is decomposable.

Solving #SAT in β-acyclic graphs

Corollary (corollary 7)

Let $x \neq x_1 \in \mathrm{VAR}(F)$ and let y be the predecessor of x for $<$. For every $C \in \mathcal{H}$, there exist $U_0, U_1 \subseteq \mathcal{H}^x_{\mathrm{VAR}(C)}$ such that

$$F^x_{\mathrm{VAR}(C)}[\tau^x_C] \equiv (x \wedge \bigwedge_{g \in U_1} F^y_g[\tau^y_{C_g}]) \vee (\neg x \wedge \bigwedge_{g \in U_2} F^y_g[\tau^y_{C_g}]).$$

Moreover, all conjunctions are decomposable and U_0, U_1 can be computed in polynomial time.

Corollary (corollary 7)

Let $x \neq x_1 \in \mathrm{VAR}(F)$ and let y be the predecessor of x for $<$. For every $C \in \mathcal{H}$, there exist $U_0, U_1 \subseteq \mathcal{H}^x_{\mathrm{VAR}(C)}$ such that

$$F^x_{\mathrm{VAR}(C)}[\tau^x_C] \equiv (x \wedge \bigwedge_{g \in U_1} F^y_g[\tau^y_{C_g}]) \vee (\neg x \wedge \bigwedge_{g \in U_2} F^y_g[\tau^y_{C_g}]).$$

Moreover, all conjunctions are decomposable and U_0, U_1 can be computed in polynomial time.

Proof sketch. Let $\tau_1 := \tau^x_C \cup \{x \mapsto 1\}$ and $\tau_0 := \tau^x_C \cup \{x \mapsto 0\}$.

$$F^x_{\mathrm{VAR}(C)}[\tau^x_C] = (x \wedge F^x_{\mathrm{VAR}(C)}[\tau_1]) \vee (\neg x \wedge F^x_{\mathrm{VAR}(C)}[\tau_0])$$

Apply lemma 6 on each of the terms.

Solving #SAT in β-acyclic graphs

Theorem (theorem 8)

Let F be a β-acyclic CNF-formula. One can construct in polynomial time in $\mathrm{size}(F)$ a dec-DNNF D of size $O((size(F))$ and fanin at most $|\mathcal{H}|$ computing F.

Solving #SAT in β-acyclic graphs

Theorem (theorem 8)

Let F be a β-acyclic CNF-formula. One can construct in polynomial time in $\mathrm{size}(F)$ a dec-DNNF D of size $O((size(F))$ and fanin at most $|\mathcal{H}|$ computing F.

Proof sketch. Let D_i be a dec-DNNF of fanin $|\mathcal{H}|$ at most such that for each $e \in \mathcal{H}$, $C \in F$ such that $\mathrm{VAR}(C) = e$ and $j \leq i$, there exists a gate in D_i computing $F_e^{x_j}[\tau_C^{x_j}]$.

Solving #SAT in β-acyclic graphs

Theorem (theorem 8)

Let F be a β-acyclic CNF-formula. One can construct in polynomial time in $\mathrm{size}(F)$ a dec-DNNF D of size $O((size(F))$ and fanin at most $|\mathcal{H}|$ computing F.

Proof sketch. Let D_i be a dec-DNNF of fanin $|\mathcal{H}|$ at most such that for each $e \in \mathcal{H}, C \in F$ such that $\mathrm{VAR}(C) = e$ and $j \leq i$, there exists a gate in D_i computing $F_e^{x_j}[\tau_C^{x_j}]$.

- Construct D_i inductively over i.

Theorem (theorem 8)

Let F be a β-acyclic CNF-formula. One can construct in polynomial time in $\mathrm{size}(F)$ a dec-DNNF D of size $O((size(F))$ and fanin at most $|\mathcal{H}|$ computing F.

Proof sketch. Let D_i be a dec-DNNF of fanin $|\mathcal{H}|$ at most such that for each $e \in \mathcal{H}$, $C \in F$ such that $\mathrm{VAR}(C) = e$ and $j \leq i$, there exists a gate in D_i computing $F_e^{x_j}[\tau_C^{x_j}]$.

- Construct D_i inductively over i.
- For an edge e, distinguish the cases whether $v_i \in e$ or not.

Solving #SAT in β-acyclic graphs

Theorem (theorem 8)

Let F be a β-acyclic CNF-formula. One can construct in polynomial time in $\mathrm{size}(F)$ a dec-DNNF D of size $O((size(F))$ and fanin at most $|\mathcal{H}|$ computing F.

Proof sketch. Let D_i be a dec-DNNF of fanin $|\mathcal{H}|$ at most such that for each $e \in \mathcal{H}, C \in F$ such that $\mathrm{VAR}(C) = e$ and $j \leq i$, there exists a gate in D_i computing $F_e^{x_j}[\tau_c^{x_j}]$.

- Construct D_i inductively over i.
- For an edge e, distinguish the cases whether $v_i \in e$ or not.
- For $i = 1$,

Solving #SAT in β-acyclic graphs

Theorem (theorem 8)

Let F be a β-acyclic CNF-formula. One can construct in polynomial time in $\mathrm{size}(F)$ a dec-DNNF D of size $O((size(F))$ and fanin at most $|\mathcal{H}|$ computing F.

Proof sketch. Let D_i be a dec-DNNF of fanin $|\mathcal{H}|$ at most such that for each $e \in \mathcal{H}$, $C \in F$ such that $\mathrm{VAR}(C) = e$ and $j \leq i$, there exists a gate in D_i computing $F_e^{x_j}[\tau_c^{x_j}]$.

- Construct D_i inductively over i.
- For an edge e, distinguish the cases whether $v_i \in e$ or not.
- For $i = 1$,

$$v_1 \notin e\text{: } F_e^{v_1}[\tau_C] = 0.$$

Solving #SAT in β-acyclic graphs

Theorem (theorem 8)

Let F be a β-acyclic CNF-formula. One can construct in polynomial time in $\mathrm{size}(F)$ a dec-DNNF D of size $O((size(F))$ and fanin at most $|\mathcal{H}|$ computing F.

Proof sketch. Let D_i be a dec-DNNF of fanin $|\mathcal{H}|$ at most such that for each $e \in \mathcal{H}, C \in F$ such that $\mathrm{VAR}(C) = e$ and $j \leq i$, there exists a gate in D_i computing $F_e^{x_j}[\tau_c^{x_j}]$.

- Construct D_i inductively over i.
- For an edge e, distinguish the cases whether $v_i \in e$ or not.
- For $i = 1$,

$$v_1 \notin e\text{: } F_e^{v_1}[\tau_C] = 0.$$

$$v_1 \in e\text{: } F_e^{v_1} \in \{1, v_1, \neg v_1\}.$$

Proof sketch of theorem 8

- If $v_{i+1} \notin e$, then $F_e^{x_{i+1}} = F_e^{x_i}$.

Proof sketch of theorem 8

- If $v_{i+1} \notin e$, then $F_e^{x_{i+1}} = F_e^{x_i}$.
- If $v_{i+1} \in e$, then by corollary 7 we can compute $F_e^{x_{i+1}}$ by adding a decision gate and (two) fanin at most $\mathcal{H}$ decomposable and-gates.

Proof sketch of theorem 8

- If $v_{i+1} \notin e$, then $F_e^{x_{i+1}} = F_e^{x_i}$.
- If $v_{i+1} \in e$, then by corollary 7 we can compute $F_e^{x_{i+1}}$ by adding a decision gate and (two) fanin at most $\mathcal{H}$ decomposable and-gates.

For each other terms from corollary 7 there is already a gate in D_i that computes this term by induction hypothesis.

Proof sketch of theorem 8

- If $v_{i+1} \notin e$, then $F_e^{x_{i+1}} = F_e^{x_i}$.
- If $v_{i+1} \in e$, then by corollary 7 we can compute $F_e^{x_{i+1}}$ by adding a decision gate and (two) fanin at most $\mathcal{H}$ decomposable and-gates.

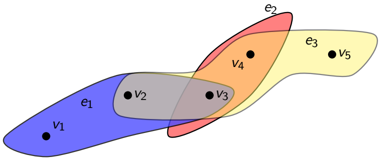
  For each other terms from corollary 7 there is already a gate in D_i that computes this term by induction hypothesis.
- For $e = \max \mathcal{H}$ and a clause C such that $\mathrm{VAR}(C) = e$, we have $\mathcal{H}_e^{v_n} = \mathcal{H}$ and $\tau_{\mathrm{VAR}(C)}^{x_n} = \emptyset$, hence there is a gate in D_n computing $F_e^{x_n}[\tau_C] = F$.

Proof sketch of theorem 8

- If $v_{i+1} \notin e$, then $F_e^{x_{i+1}} = F_e^{x_i}$.
- If $v_{i+1} \in e$, then by corollary 7 we can compute $F_e^{x_{i+1}}$ by adding a decision gate and (two) fanin at most $\mathcal{H}$ decomposable and-gates.

  For each other terms from corollary 7 there is already a gate in D_i that computes this term by induction hypothesis.
- For $e = \max \mathcal{H}$ and a clause C such that $\mathrm{VAR}(C) = e$, we have $\mathcal{H}_e^{v_n} = \mathcal{H}$ and $\tau_{\mathrm{VAR}(C)}^{x_n} = \emptyset$, hence there is a gate in D_n computing $F_e^{x_n}[\tau_C] = F$.
- We add at most 7 gates per edges per vertex.

Example



$$F = \{\{\overline{v_1}, v_2, v_3\}, \{\overline{v_3}, v_4\}, \{v_2, v_3, \overline{v_4}, \overline{v_5}\}\}$$

The rest on the blackboard..

concluding the practical method

- Exhaustive DPLL is a very-well used in practice method.

concluding the practical method

- Exhaustive DPLL is a very-well used in practice method.
  - Try to write F as a decomposable conjunction.

concluding the practical method

- Exhaustive DPLL is a very-well used in practice method.
  - Try to write F as a decomposable conjunction.
    Solve independently on each and multiply the results.

concluding the practical method

- Exhaustive DPLL is a very-well used in practice method.
  - Try to write F as a decomposable conjunction.
    Solve independently on each and multiply the results.
  - Choose a variable x.

concluding the practical method

- Exhaustive DPLL is a very-well used in practice method.
  - Try to write F as a decomposable conjunction.
    Solve independently on each and multiply the results.
  - Choose a variable x.
    Compute $\#F[x \mapsto 1] + \#F[x \mapsto 0]$.

concluding the practical method

- Exhaustive DPLL is a very-well used in practice method.
  - Try to write F as a decomposable conjunction.
    Solve independently on each and multiply the results.
  - Choose a variable x.
    Compute $\#F[x \mapsto 1] + \#F[x \mapsto 0]$.
- The method makes use of cashing (choose what values to keep).

concluding the practical method

- Exhaustive DPLL is a very-well used in practice method.
  - Try to write F as a decomposable conjunction.
    Solve independently on each and multiply the results.
  - Choose a variable x.
    Compute $\#F[x \mapsto 1] + \#F[x \mapsto 0]$.
- The method makes use of cashing (choose what values to keep).
- Tries to find a good candidate for x.

concluding the practical method

- Exhaustive DPLL is a very-well used in practice method.
  - Try to write F as a decomposable conjunction.
    Solve independently on each and multiply the results.
  - Choose a variable x.
    Compute $\#F[x \mapsto 1] + \#F[x \mapsto 0]$.
- The method makes use of cashing (choose what values to keep).
- Tries to find a good candidate for x.
- The previous dynamic programming is implicitly a run of DPLL.

concluding the practical method

- Exhaustive DPLL is a very-well used in practice method.
  - Try to write F as a decomposable conjunction.
    Solve independently on each and multiply the results.
  - Choose a variable x.
    Compute $\#F[x \mapsto 1] + \#F[x \mapsto 0]$.
- The method makes use of cashing (choose what values to keep).
- Tries to find a good candidate for x.
- The previous dynamic programming is implicitly a run of DPLL.
- The variables are chosen in e reversed β-elimination ordering.

concluding the practical method

- Exhaustive DPLL is a very-well used in practice method.
  - Try to write F as a decomposable conjunction.
    Solve independently on each and multiply the results.
  - Choose a variable x.
    Compute $\#F[x \mapsto 1] + \#F[x \mapsto 0]$.
- The method makes use of cashing (choose what values to keep).
- Tries to find a good candidate for x.
- The previous dynamic programming is implicitly a run of DPLL.
- The variables are chosen in e reversed β-elimination ordering.

conclusion

Exhaustive DPLL can yield efficient algorithms "theoretically", if we can find a good order to choose the variable (such an ordering must be computable in polynomial time) and a good method of cashing.

Lower-bound on the theoretical method

Branch decomposition and MIM-width

- A **branch decomposition** T of a graph $G = (V, E)$ is a binary rooted tree T, whose leaves are in one-to-one correspondence with V.

Branch decomposition and MIM-width

- A **branch decomposition** T of a graph $G = (V, E)$ is a binary rooted tree T, whose leaves are in one-to-one correspondence with V.
- The maximal-induced-matching width (MIM-width) of a vertex t of T is the size of a largest induced matching M of $G[V \setminus V_t, V_t]$.

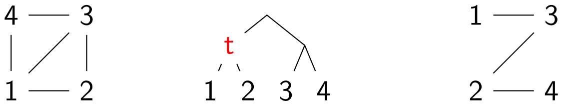
Branch decomposition and MIM-width

- A **branch decomposition** T of a graph $G = (V, E)$ is a binary rooted tree T, whose leaves are in one-to-one correspondence with V.
- The maximal-induced-matching width (MIM-width) of a vertex t of T is the size of a largest induced matching M of $G[V \setminus V_t, V_t]$.
- $mimw(T) = \max\{\mathrm{mimw}(t) : t \in V(T)\}$.

Branch decomposition and MIM-width

- A **branch decomposition** T of a graph $G = (V, E)$ is a binary rooted tree T, whose leaves are in one-to-one correspondence with V.
- The maximal-induced-matching width (MIM-width) of a vertex t of T is the size of a largest induced matching M of $G[V \setminus V_t, V_t]$.
- $mimw(T) = \max\{\mathrm{mimw}(t) : t \in V(T)\}$.

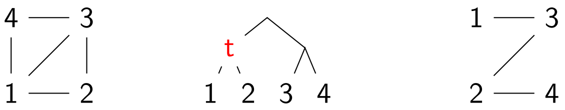
Example.



Branch decomposition and MIM-width

- A **branch decomposition** T of a graph $G = (V, E)$ is a binary rooted tree T, whose leaves are in one-to-one correspondence with V.
- The maximal-induced-matching width (MIM-width) of a vertex t of T is the size of a largest induced matching M of $G[V \setminus V_t, V_t]$.
- $mimw(T) = \max\{\mathrm{mimw}(t) : t \in V(T)\}$.

Example.



MIM-width$(t) = 2$.

Structuredness of a formula

- Let φ be a DNNF formula and let $V := \mathrm{VAR}(\varphi)$.

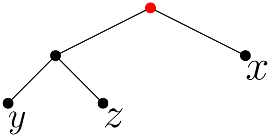
Structuredness of a formula

- Let φ be a DNNF formula and let $V := \mathrm{VAR}(\varphi)$.
- A **vTree** T is a binary tree where the leaves of the tree has a one-to-one correspondence to the variables of φ.

Structuredness of a formula

- Let φ be a DNNF formula and let $V := \mathrm{VAR}(\varphi)$.
- A **vTree** T is a binary tree where the leaves of the tree has a one-to-one correspondence to the variables of φ.
- The formula φ respects T if and only if for each subformula of φ of the form $\varphi' := \psi_1 \wedge \psi_2$, there is a vertex $v \in V(T)$ with two children v_1, v_2, where $\mathrm{VAR}(\psi_1) \subseteq V(T_{v_1})$ and $\mathrm{VAR}(\psi_2) \subseteq V(T_{v_2})$, where T_v is the subtree of T rooted at v. We say φ' respects v in this case.

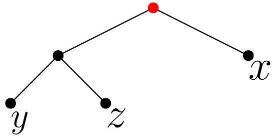
$(x \wedge (y \vee z)) \vee (z {\color{red}\wedge} \neg x)$



Structuredness of a formula

- Let φ be a DNNF formula and let $V := \mathrm{VAR}(\varphi)$.
- A **vTree** T is a binary tree where the leaves of the tree has a one-to-one correspondence to the variables of φ.
- The formula φ respects T if and only if for each subformula of φ of the form $\varphi' := \psi_1 \wedge \psi_2$, there is a vertex $v \in V(T)$ with two children v_1, v_2, where $\mathrm{VAR}(\psi_1) \subseteq V(T_{v_1})$ and $\mathrm{VAR}(\psi_2) \subseteq V(T_{v_2})$, where T_v is the subtree of T rooted at v. We say φ' respects v in this case.
- A formula φ is structured, if there is a *v*tree T over the vertices of φ, such that φ respects T.

$(x \wedge (y \vee z)) \vee (z \wedge \neg x)$



Incidence graphs and structure of formulas

- The **incidence graph** of $\mathcal{H}$ is a

  bipartite graph $(V(\mathcal{H}) \cup E(\mathcal{H}), E)$

  ,where $\{v, e\} \in E$ iff $v \in e$.

Incidence graphs and structure of formulas

- The **incidence graph** of $\mathcal{H}$ is a
  bipartite graph $(V(\mathcal{H}) \cup E(\mathcal{H}), E)$
  ,where $\{v, e\} \in E$ iff $v \in e$.

- The incidence graph of a CNF-Formula is
  the incidence graph of its hyper graph.

Incidence graphs and structure of formulas

- The **incidence graph** of $\mathcal{H}$ is a
  bipartite graph $(V(\mathcal{H}) \cup E(\mathcal{H}), E)$
  ,where $\{v, e\} \in E$ iff $v \in e$.

- The incidence graph of a CNF-Formula is
  the incidence graph of its hyper graph.

- The MIM-width of a CNF-formula is the MIM-width of its incidence graph.

Results on the structured d-DNNF

Theorem (theorem 9)

There exists an infinite family $\mathcal{F}$ of β-acyclic CNF-formulas such that for every $F \in \mathcal{F}$ having n variables, there is no structured DNNF of size less than $2^{\Omega(\sqrt{n})}$ computing F.

[1] Understanding Model Counting for beta-acyclic CNF-formulas, Brault-Baron et al., 2015.

Results on the structured d-DNNF

Theorem (theorem 9)

There exists an infinite family $\mathcal{F}$ of β-acyclic CNF-formulas such that for every $F \in \mathcal{F}$ having n variables, there is no structured DNNF of size less than $2^{\Omega(\sqrt{n})}$ computing F.

Theorem (theorem 1)[1]

There exists an infinite family of β-acyclic hypergraphs of incidence MIM-width $\Omega(n)$ where n is the number of vertices of the hypergraph.

[1] Understanding Model Counting for beta-acyclic CNF-formulas, Brault-Baron et al., 2015.

Results on the structured d-DNNF

- Let r be a boolean function over X and let (Y, Z) be a partition of X. We call r a (Y, Z)-rectangle if and only if for every $\tau, \tau' \in \{0,1\}^X$ such that $\tau \models r$ and $\tau' \models r$, we have $\tau|Y \cup \tau'|Z) \models r$.
- A (Y, Z)-rectangle cover of a boolean function f is a set $R = \{r_1, \ldots, r_q\}$ of (Y, Z)-rectangles such that $\mathrm{sat}(f) = \bigcup_{i=1}^{q} \mathrm{sat}(r_i)$.

[2] Knowledge Compilation Meets Communication Complexity, Bova et al., 2016.

[3] A Lower Bound on the Size of Decomposable Negation Normal Form, Pipatsrisawat and Darwiche, 2010.

Results on the structured d-DNNF

- Let r be a boolean function over X and let (Y, Z) be a partition of X. We call r a (Y, Z)-rectangle if and only if for every $\tau, \tau' \in \{0,1\}^X$ such that $\tau \models r$ and $\tau' \models r$, we have $\tau|Y \cup \tau'|Z) \models r$.
- A (Y, Z)-rectangle cover of a boolean function f is a set $R = \{r_1, \ldots, r_q\}$ of (Y, Z)-rectangles such that $\mathrm{sat}(f) = \bigcup_{i=1}^{q} \mathrm{sat}(r_i)$.

Theorem (theorem 11)[2,3]

Let D be a DNNF on variables X respecting the vtree T. For every vertex t of T, there exists a $(X_t, X \setminus X_t)$-rectangle cover of D of size at most $|D|$, where $X_t = \mathrm{VAR}(T_t)$.

[2] Knowledge Compilation Meets Communication Complexity, Bova et al., 2016.

[3] A Lower Bound on the Size of Decomposable Negation Normal Form, Pipatsrisawat and Darwiche, 2010.

Results on the structured d-DNNF

Let F be a CNF-formula. Let $\hat{F} := \{K \cup \{c_K\} | K \in F\}$ where we add a fresh variable to each clause.

Theorem (theorem 12)

Let F be a monotone formula of incidence MIM-width k. Any structured DNNF computing $\hat{F}$ is of size at least $2^{k/2}$.

Results on the structured d-DNNF

Let F be a CNF-formula. Let $\hat{F} := \{K \cup \{c_K\} | K \in F\}$ where we add a fresh variable to each clause.

Theorem (theorem 12)

Let F be a monotone formula of incidence MIM-width k. Any structured DNNF computing $\hat{F}$ is of size at least $2^{k/2}$.

Lemma (lemma 13)

Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be two disjoint sets of k variables. The number of (X, Y)-rectangles needed to cover the CNF-formula $F = \bigwedge_{i=1}^{k}(x_i \vee y_i)$ is at least 2^k.

Results on the structured d-DNNF

Let F be a CNF-formula. Let $\hat{F} := \{K \cup \{c_K\} | K \in F\}$ where we add a fresh variable to each clause.

Theorem (theorem 12)

Let F be a monotone formula of incidence MIM-width k. Any structured DNNF computing $\hat{F}$ is of size at least $2^{k/2}$.

Lemma (lemma 13)

Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be two disjoint sets of k variables. The number of (X, Y)-rectangles needed to cover the CNF-formula $F = \bigwedge_{i=1}^{k}(x_i \vee y_i)$ is at least 2^k.

Proof sketch (lemma 12). Find an assignment τ of $\hat{F}$ such that

$$\hat{F}[\tau] \equiv \bigwedge_{e \in N} (x_e \vee c_e).$$

Conclusion

Takeaway

- Building a structured d-DNNF is not always the best choice we have.

Takeaway

- Building a structured d-DNNF is not always the best choice we have.
- If the structure implies a good elimination ordering, exhaustive DPLL might be a better shot.