Bounded Treewidth in Knowledge Representation and Reasoning

Reinhard Pichler

Institut für Informationssysteme
Arbeitsbereich DBAI
Technische Universität Wien

Luminy, October 2010

1 Joint work with G. Gottlob, M. Jakl, S. Rümmele, F. Wei, and S. Woltran
Outline

1. Motivation

2. Treewidth of Finite Structures

3. Tractable Reasoning via Courcelle’s Theorem

4. FPT-Algorithms via Dynamic Programming

5. Conclusion
Outline

1. Motivation

2. Treewidth of Finite Structures

3. Tractable Reasoning via Courcelle’s Theorem

4. FPT-Algorithms via Dynamic Programming

5. Conclusion
Motivation

Complexity in non-monotonic reasoning

- Most forms of non-monotonic reasoning are intractable (even on the second or third level of the polynomial hierarchy)
- Examples: propositional abduction, closed world reasoning, answer set programming, belief revision
- treewidth as a key to fixed-parameter tractability: Courcelle’s Theorem and extensions
- theoretical tractability vs. efficient computation
- efficient algorithms via dynamic programming
2. Treewidth of Finite Structures
Treewidth of Finite Structures

Basic Notions and Results

- tree decomposition / treewidth of a graph
- finite structure over a signature $\tau$
- tree decomposition / treewidth of a finite structure
- computation of a tree decomposition of width $k$ is FPT w.r.t. $k$
- heuristic methods for efficient computation of a “good” tree decomposition exist
Treewidth of a Propositional Formula in CNF

Represent a CNF formula as a finite structure

Given a propositional formula $F$ in CNF, represent $F$ by a finite structure $A(F)$ over the signature $\tau$ with $\tau = \{cl, var, pos, neg\}$, where

- $cl(c)$, $var(x)$
  means that $c$ is a clause (resp. $x$ is a variable) in $F$.

- $pos(x, c)$, $neg(x, c)$
  means that $x$ occurs unnegated (resp. negated) in the clause $c$.

Treewidth of a CNF formula

We define $tw(F) := tw(A(F))$.

Remark. This definition corresponds to the incidence graph of $F$. 
# Treewidth of CNF

## Example

Given a propositional formula $F$ in CNF

$$(x_1 \lor \bar{x}_2 \lor x_3) \land (x_2 \lor \bar{x}_3 \lor x_4),$$

represent $F$ by a finite structure $A(F)$ over $\tau$: 

Treewidth of CNF

Example

Given a propositional formula \( F \) in CNF
\[
(x_1 \lor \bar{x}_2 \lor x_3) \land (x_2 \lor \bar{x}_3 \lor x_4),
\]
represent \( F \) by a finite structure \( A(F) \) over \( \tau \):

\( A(F) \) contains the following atoms:

- \( cl(c_1), \ cl(c_2), \)
- \( var(x_1), \ var(x_2), \ var(x_3), \ var(x_4), \)
- \( pos(x_1, c_1), \ pos(x_3, c_1), \ pos(x_2, c_2), \ pos(x_4, c_2), \)
- \( neg(x_2, c_1), \ neg(x_3, c_2). \)
Tree Decomposition

\[ A(F) \]

- \( cl(c_1), cl(c_2), \)
- \( var(x_1), var(x_2), \)
- \( var(x_3), var(x_4), \)
- \( pos(x_1, c_1), \)
- \( pos(x_3, c_1), \)
- \( pos(x_2, c_2), \)
- \( pos(x_4, c_2), \)
- \( neg(x_2, c_1), \)
- \( neg(x_3, c_2). \)
Tree Decomposition

$A(F)$

$cl(c_1), cl(c_2),\,$
$var(x_1), var(x_2),\,$
$var(x_3), var(x_4),\,$
$pos(x_1, c_1),\,$
$pos(x_3, c_1),\,$
$pos(x_2, c_2),\,$
$pos(x_4, c_2),\,$
$neg(x_2, c_1),\,$
$neg(x_3, c_2).\,$

Tree Decomposition of $A(F)$
Tree Decomposition

\[ A(F) \]

\[ cl(c_1), cl(c_2), \]
\[ var(x_1), var(x_2), \]
\[ var(x_3), var(x_4), \]
\[ pos(x_1, c_1), \]
\[ pos(x_3, c_1), \]
\[ pos(x_2, c_2), \]
\[ pos(x_4, c_2), \]
\[ neg(x_2, c_1), \]
\[ neg(x_3, c_2). \]

Remark. We have \( tw(F) := tw(A(F)) = 2. \)
Outline

1. Motivation

2. Treewidth of Finite Structures

3. Tractable Reasoning via Courcelle’s Theorem

4. FPT-Algorithms via Dynamic Programming

5. Conclusion
Monadic-Second Order Logic (MSO)

Expressive Power of MSO

- MSO extends First Order logic (FO) by the use of set variables (usually denoted by upper case letters), which range over sets of domain elements.
- Many interesting (intractable) properties of finite structures are expressible in MSO, e.g.: SAT, propositional abduction, closed world reasoning, belief revision, answer set programming, etc.
Monadic-Second Order Logic (MSO)

Expressive Power of MSO

- MSO extends First Order logic (FO) by the use of set variables (usually denoted by upper case letters), which range over sets of domain elements.
- Many interesting (intractable) properties of finite structures are expressible in MSO, e.g.: SAT, propositional abduction, closed world reasoning, belief revision, answer set programming, etc.

Theorem (Courcelle, 1990)

*Any property of finite structures, which is definable by an MSO sentence, can be decided in time \( O(f(k) \cdot n) \), where \( n \) is the size of the structure and \( f(k) \) depends only on the treewidth \( k \) of the structure, i.e., FPT w.r.t. the treewidth.*
MSO-Example 1: SAT-Problem

MSO-Encoding of the SAT-Problem (Courcelle et al. 2001)

**Idea.** Let $F$ be a propositional formula in CNF; an interpretation of $F$ can be represented as a set $X$ of variables (i.e., the variables which are true).
MSO-Example 1: SAT-Problem

**MSO-Encoding of the SAT-Problem (Courcelle et al. 2001)**

**Idea.** Let $F$ be a propositional formula in CNF; an interpretation of $F$ can be represented as a set $X$ of variables (i.e., the variables which are true).

MSO encoding that $X$ is a model of $F$:

$$
Mod(X, F) := (\forall c)cl(c) \rightarrow (\exists z)[(pos(z, c) \land z \in X) \lor (neg(z, c) \land z \notin X)]
$$
MSO-Example 1: SAT-Problem

MSO-Encoding of the SAT-Problem (Courcelle et al. 2001)

Idea. Let $F$ be a propositional formula in CNF; an interpretation of $F$ can be represented as a set $X$ of variables (i.e., the variables which are true).

MSO encoding that $X$ is a model of $F$:

$Mod(X, F) := (\forall c) cl(c) \rightarrow (\exists z)[(pos(z, c) \wedge z \in X) \vee (neg(z, c) \wedge z \notin X)]$

MSO encoding of the SAT-Problem:

$(\exists X) Mod(X, F)$
MSO-Example 2: Propositional Abduction

Definition
A propositional abduction problem (PAP) is a quadruple \( \mathcal{P} = \langle V, H, M, \mathcal{C} \rangle \) consisting of

- a finite set of propositional variables \( V \),
- a theory \( \mathcal{C} \), which is a consistent set of clauses over the variables \( V \),
- a set of hypotheses \( H \subseteq V \),
- and a set of manifestations \( M \subseteq V \).
**MSO-Example 2: Propositional Abduction**

**Definition**

A propositional abduction problem (PAP) is a quadruple \( P = \langle V, H, M, C \rangle \) consisting of:

- a finite set of propositional variables \( V \),
- a theory \( C \), which is a consistent set of clauses over the variables \( V \),
- a set of hypotheses \( H \subseteq V \),
- and a set of manifestations \( M \subseteq V \).

A set \( S \) satisfying \( S \subseteq H \) is a solution iff:

- \( C \cup S \) is consistent (i.e., satisfiable)
- and \( C \cup S \models M \) holds.
MSO-Example 2: Propositional Abduction

Abduction in System Diagnosis

A diagnosis problem can be represented by a propositional abduction problem \( P = \langle V, H, M, C \rangle \) as follows:

- The clausal theory \( C \) is the system description.
- The hypotheses \( H \subseteq V \) describe the possibly faulty system components.
- The manifestations \( M \subseteq V \) are the observed symptoms (describing some malfunction of the system).
- The solutions \( S \in Sol(P) \) are the possible explanations of the malfunction.
MSO-Example 2: Propositional Abduction

The main decision problems

Given a PAP $\mathcal{P}$ we ask:

- **Solvability.** Does there exist at least one solution for $\mathcal{P}$?
MSO-Example 2: Propositional Abduction

The main decision problems

Given a PAP $\mathcal{P}$ we ask:

- **Solvability.** Does there exist at least one solution for $\mathcal{P}$?

Given a PAP $\mathcal{P}$ and a hypothesis $h \in H$, we ask:

- **Relevance:** Is $h$ contained in at least one solution of $\mathcal{P}$?
- **Necessity:** Is $h$ contained in all solutions of $\mathcal{P}$?
MSO-Example 2: Propositional Abduction

The main decision problems

Given a PAP $\mathcal{P}$ we ask:

- **Solvability.** Does there exist at least one solution for $\mathcal{P}$?

Given a PAP $\mathcal{P}$ and a hypothesis $h \in H$, we ask:

- **Relevance:** Is $h$ contained in at least one solution of $\mathcal{P}$?
- **Necessity:** Is $h$ contained in all solutions of $\mathcal{P}$?

**Remark.** These problems are $\Sigma_2P$-complete (Solvability and Relevance) resp. $\Pi_2P$-complete (Necessity).
MSO-Example 2: Propositional Abduction

MSO-Encoding of the main Abduction problems

Let a PAP $\mathcal{P} = \langle V, H, M, C \rangle$ be represented as a $\tau$-structure with $\tau = \{cl, var, pos, neg, H, M\}$, where the predicates $cl, var, pos, neg$ represent the clause set $C$ and the unary predicates $H$ and $M$ identify the hypotheses and manifestations.
MSO-Example 2: Propositional Abduction

**MSO-Encoding of the main Abduction problems**

Let a PAP $\mathcal{P} = \langle V, H, M, C \rangle$ be represented as a $\tau$-structure with $\tau = \{ cl, \text{var}, \text{pos}, \text{neg}, H, M \}$, where the predicates $cl, \text{var}, \text{pos}, \text{neg}$ represent the clause set $C$ and the unary predicates $H$ and $M$ identify the hypotheses and manifestations.

MSO encoding of $Sol(S)$, i.e., “$S$ is a solution of $\mathcal{P}$”:

$S \subseteq H \land (\exists X)[\text{Mod}(X, C) \land S \subseteq X] \land (\forall Y)[(\text{Mod}(Y, C) \land S \subseteq Y) \rightarrow M \subseteq Y]$
MSO-Example 2: Propositional Abduction

MSO-Encoding of the main Abduction problems

Let a PAP \( \mathcal{P} = \langle V, H, M, C \rangle \) be represented as a \( \tau \)-structure with \( \tau = \{ cl, var, pos, neg, H, M \} \), where the predicates \( cl, var, pos, neg \) represent the clause set \( C \) and the unary predicates \( H \) and \( M \) identify the hypotheses and manifestations.

MSO encoding of \( \text{Sol}(S) \), i.e., “\( S \) is a solution of \( \mathcal{P} \)”:
\[
S \subseteq H \land (\exists X)[\text{Mod}(X, C) \land S \subseteq X] \land \\
(\forall Y)[(\text{Mod}(Y, C) \land S \subseteq Y) \rightarrow M \subseteq Y]
\]

MSO encoding of the main Abduction problems:

**Solvability:** \((\exists S)\text{Sol}(S)\)

**Relevance:** \((\exists S)[\text{Sol}(S) \land h \in S]\)

**Necessity:** \((\forall S)[\text{Sol}(S) \rightarrow h \in S]\)
### Further Results

#### Closed World Reasoning

- **Closed world assumption**: assume negative information if the positive information is not explicitly given by a theory $T$.
- Several forms of closed world assumption: CWA (Closed World Assumption), GCWA (Generalized CWA), EGCWA (Extended GCWA), CCWA (Careful CWA), and ECWA (Extended CWA).
- Crucial for MSO-encoding: define **minimal models**
Further Results

### Closed World Reasoning

- **Closed world assumption**: assume negative information if the positive information is not explicitly given by a theory $T$.
- Several forms of closed world assumption: CWA (Closed World Assumption), GCWA (Generalized CWA), EGCWA (Extended GCWA), CCWA (Careful CWA), and ECWA (Extended CWA).
- Crucial for MSO-encoding: define minimal models

### Other Forms of Reasoning

- Answer Set Programming
- Belief Revision
Extended MSO

Extension of Courcelle’s Theorem

- Extended MSO: weight functions, cardinality, sum, min, max, etc.
- (Arnborg/Lagergren/Seese, 1991) Linear extended MSO extremum problems over finite structures are FPT w.r.t. the treewidth.

Extended MSO-Encodings in Propositional Abduction

- search for small explanations, express probabilities or cost of repair
- restriction to solutions of minimal cardinality or minimal weight
Theoretical Tractability vs. Efficient Computation

Algorithms via Courcelle’s Theorem

In principle, Courcelle’s theorem can be used to effectively generate a concrete algorithm from an MSO description, see e.g. (Arnborg/Lagergren/Seese, 1991), (Flum/Frick/Grohe, 2002).

1. Translate the MSO evaluation problem over finite structures into an equivalent MSO evaluation problem over colored binary trees.

2. Solve this problem via the correspondence between MSO over trees and finite tree automata (FTA), see (Thatcher/Wright, 1968), (Doner, 1970).
Theoretical Tractability vs. Efficient Computation

Problem with this approach

- “State explosion” of the FTA even for relatively simple MSO formulae on trees.
- MSO formula over colored binary trees is significantly more complex than the original formula for the structure with bounded treewidth.
Theoretical Tractability vs. Efficient Computation

Problem with this approach

- “State explosion” of the FTA even for relatively simple MSO formulae on trees.
- MSO formula over colored binary trees is significantly more complex than the original formula for the structure with bounded treewidth.

Conclusion: (Grohe 1999), similarly (Niedermeier, 2006)

- Main benefit of Courcelle’s Theorem: “a simple way to recognize a property as being linear time computable”.
- Algorithms are “useless for practical applications”.
Outline

1. Motivation

2. Treewidth of Finite Structures

3. Tractable Reasoning via Courcelle’s Theorem

4. FPT-Algorithms via Dynamic Programming

5. Conclusion
Nice Tree Decompositions

A tree decomposition \( T \) is called nice if

- \( T \) is a rooted binary tree,
- nodes with 2 children have the same bags as their children,
- the bags of nodes with one child differ in exactly 1 element from the bag of the child.
Nice Tree Decompositions

A tree decomposition $\mathcal{T}$ is called nice if

- $\mathcal{T}$ is a rooted binary tree,
- nodes with 2 children have the same bags as their children,
- the bags of nodes with one child differ in exactly 1 element from the bag of the child.

![Tree Diagram]

$x_2, x_3, c_1$

$x_1, x_2, c_1$

$x_2, x_3, c_2$

$x_3, x_4, c_2$
Nice Tree Decompositions

A tree decomposition $\mathcal{T}$ is called nice if

- $\mathcal{T}$ is a rooted binary tree,
- nodes with 2 children have the same bags as their children,
- the bags of nodes with one child differ in exactly 1 element from the bag of the child.
Nice Tree Decompositions

We distinguish:

- Leaf nodes.
- Forget nodes (variable or clause).
- Introduce nodes (variable or clause).
- Branch nodes.
Nice Tree Decompositions

We distinguish:

- Leaf nodes.
- **Forget nodes** (variable or clause).
- Introduce nodes (variable or clause).
- Branch nodes.
Nice Tree Decompositions

We distinguish:

- Leaf nodes.
- Forget nodes (variable or clause).
- Introduce nodes (variable or clause).
- Branch nodes.
Nice Tree Decompositions

We distinguish:

- Leaf nodes.
- Forget nodes (variable or clause).
- Introduce nodes (variable or clause).
- Branch nodes.
### #SAT-Algorithm (Samer/Szeider, 2010)

#### Notation

- $F$ ... CNF formula.
- $(T, \chi, r)$ ... Nice tree decomposition of $F$.
- $\chi_V, \chi_C$ ... $\chi$ restricted to variables/clauses.
- $T_t$ ... Subtree of $T$ rooted at node $t$.
- $V_t, C_t$ ... Variables/Clauses in the bags of $T_t$.

#### Data Structure

For each truth assignment $\alpha : \chi_V(t) \rightarrow \{0, 1\}$ and subset $A \subseteq \chi_C(t)$ let $N(t, \alpha, A)$ be the set of truth assignments $\tau : V_t \rightarrow \{0, 1\}$, s.t.

- $\tau(x) = \alpha(x)$ for all $x \in \chi_V(t)$.
- $A$ is exactly the set of clauses in $C_t$ that are not satisfied by $\tau$.  

Reinhard Pichler
Luminy, October 2010
#SAT-Algorithm (Samer/Szeider, 2010)

**Notation**

- \( F \ldots \) CNF formula.
- \((T, \chi, r) \ldots \) Nice tree decomposition of \( F \).
- \( \chi_V, \chi_C \ldots \chi \) restricted to variables/clauses.
- \( T_t \ldots \) Subtree of \( T \) rooted at node \( t \).
- \( V_t, C_t \ldots \) Variables/Clauses in the bags of \( T_t \).

**Data Structure**

For each truth assignment \( \alpha : \chi_V(t) \rightarrow \{0, 1\} \) and subset \( A \subseteq \chi_C(t) \) let \( N(t, \alpha, A) \) be the set of truth assignments \( \tau : V_t \rightarrow \{0, 1\} \), s.t.

- \( \tau(x) = \alpha(x) \) for all \( x \in \chi_V(t) \).
- \( A \) is exactly the set of clauses in \( C_t \) that are not satisfied by \( \tau \).

Let \( n(t, \alpha, A) = |N(t, \alpha, A)| \).
Example

\[
(x_1 \lor \overline{x}_2 \lor x_3) \land (x_2 \lor \overline{x}_3 \lor x_4)
\]

\[
\begin{array}{ccc}
\alpha & A & n \\
- & - & 1 \\
x_1 & - & 1 \\
x_2 & c_1 & 1 \\
x_1, x_2 & - & 1 \\
\end{array}
\]
Example

\[ (x_1 \lor \overline{x}_2 \lor x_3) \land (x_2 \lor \overline{x}_3 \lor x_4) \]

\[ c_1 \land c_2 \]

\[
\begin{array}{ccc}
\alpha & A & n \\
- & - & 1 \\
x_1 & - & 1 \\
x_2 & c_1 & 1 \\
x_1, x_2 & - & 1 \\
- & - & 2 \\
x_2 & c_1 & 1 \\
x_2 & - & 1 \\
\end{array}
\]
4. FPT-Algorithms via Dynamic Programming

Example

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4)\]

\[
\begin{array}{c|c|c}
\alpha & A & n \\
\hline
- & - & 2 \\
x_2 & c_1 & 1 \\
x_2 & - & 1 \\
\hline
- & - & 2 \\
x_2 & c_1 & 1 \\
x_2 & - & 1 \\
x_3 & - & 2 \\
x_2, x_3 & - & 2 \\
\end{array}
\]
Example

\[(x_1 \lor \overline{x}_2 \lor x_3) \land (x_2 \lor \overline{x}_3 \lor x_4)\]

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$A$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$c_2$</td>
<td>1</td>
</tr>
<tr>
<td>$x_4$</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>$x_3, x_4$</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>
Example

\[ (x_1 \lor \overline{x_2} \lor x_3) \land (x_2 \lor \overline{x_3} \lor x_4) \]

\[ c_1 \land c_2 \]

\[
\begin{array}{ccc}
\alpha & A & n \\
- & - & 1 \\
x_3 & c_2 & 1 \\
x_4 & - & 1 \\
x_3, x_4 & - & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
\alpha & A & n \\
- & - & 2 \\
x_3 & c_2 & 1 \\
x_3 & - & 1 \\
\end{array}
\]
Example

\[
(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4)
\]

\[
\begin{array}{l|l|l}
\alpha & A & n \\
\hline
- & - & 2 \\
x_3 & c_2 & 1 \\
x_3 & - & 1 \\
- & - & 2 \\
x_2 & - & 2 \\
x_3 & c_2 & 1 \\
x_3 & - & 1 \\
x_2, x_3 & - & 2 \\
\end{array}
\]
Example

\begin{itemize}
  \item \( x_2, x_3, c_1 \)
  \item \( x_2, c_1 \)
  \item \( x_1, x_2, c_1 \)
  \item \( x_3, c_2 \)
  \item \( x_3, x_4, c_2 \)
  \item \((x_1 \lor \overline{x_2} \lor x_3) \land (x_2 \lor \overline{x_3} \lor x_4)\)
\end{itemize}

\begin{tabular}{c|c|c}
\( \alpha \) & A & n \\
\hline
- & - & 2 \\
x_2 & - & 2 \\
x_3 & c_2 & 1 \\
x_3 & - & 1 \\
x_2, x_3 & - & 2 \\
\end{tabular}

\begin{tabular}{c|c|c}
\( \alpha \) & A & n \\
\hline
- & - & 2 \\
x_2 & - & 2 \\
x_3 & - & 1 \\
x_2, x_3 & - & 2 \\
\end{tabular}
Example

\[
\alpha \quad A \quad n
\]

\[
\begin{array}{ccc}
- & - & 2 \\
\bar{x}_2 & - & 2 \\
\bar{x}_3 & - & 1 \\
x_2, x_3 & - & 2 \\
\end{array}
\]

\[
\alpha \quad A \quad n
\]

\[
\begin{array}{ccc}
- & - & 2 \\
x_2 & c_1 & 2 \\
x_3 & - & 1 \\
x_2, x_3 & - & 2 \\
\end{array}
\]
Example

\[
\begin{align*}
\alpha & \quad A & \quad n \\
- & \quad - & \quad 2 \\
\begin{array}{c} x_2 \\ c_1 \end{array} & \quad - & \quad 1 \\
\begin{array}{c} x_2 \\ - \end{array} & \quad - & \quad 1 \\
\begin{array}{c} x_3 \\ - \end{array} & \quad - & \quad 2 \\
\begin{array}{c} x_2, x_3 \\ - \end{array} & \quad - & \quad 2 \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\alpha & \quad A & \quad n \\
- & \quad - & \quad 4 \\
\begin{array}{c} x_2 \\ c_1 \end{array} & \quad - & \quad 2 \\
\begin{array}{c} x_2 \\ - \end{array} & \quad - & \quad 2 \\
\begin{array}{c} x_3 \\ - \end{array} & \quad - & \quad 2 \\
\begin{array}{c} x_2, x_3 \\ - \end{array} & \quad - & \quad 4 \\
\end{align*}
\]
Leaf and Join Nodes

**Lemma**

Let $t$ be a leaf node. Then, for each truth assignment $\alpha : \chi_V(t) \rightarrow \{0, 1\}$ and set $A \subseteq \chi_C(t)$, we have

$$n(t, \alpha, A) = \begin{cases} 1 & \text{if } A = \{ c \in \chi_C(t) : \alpha \text{ does not satisfy } c \}; \\ 0 & \text{otherwise}. \end{cases}$$

**Lemma**

Let $t$ be a join node of $T$ with children $t_1, t_2$. Then, for each truth assignment $\alpha : \chi_V(t) \rightarrow \{0, 1\}$ and set $A \subseteq \chi_C(t)$, we have

$$n(t, \alpha, A) = \sum_{A_1, A_2 \subseteq \chi_C(t), A_1 \cap A_2 = A} n(t_1, \alpha, A_1) \cdot n(t_2, \alpha, A_2).$$
Variable Introduce Nodes

Lemma

Let \( t \) be a variable introduce node with child \( t' \) and \( \chi(t) = \chi(t') \cup \{x\} \) for a variable \( x \). Then, for each truth assignment \( \alpha : \chi_V(t) \rightarrow \{0,1\} \) and set \( A \subseteq \chi_C(t) \), we have

\[
\begin{align*}
    n(t, \alpha_{x=0}, A) &= \begin{cases} 
    0 & \text{if } \neg x \in c \text{ for some } c \in A; \\
    \sum_{B' \subseteq \bar{B}} n(t', \alpha, A \cup B') & \text{otherwise};
    \end{cases} \\
    \text{where } \bar{B} &= \{ c \in \chi_C(t) : \neg x \in c \};
\end{align*}
\]

\[
\begin{align*}
    n(t, \alpha_{x=1}, A) &= \begin{cases} 
    0 & \text{if } x \in c \text{ for some } c \in A; \\
    \sum_{B' \subseteq B} n(t', \alpha, A \cup B') & \text{otherwise};
    \end{cases} \\
    \text{where } B &= \{ c \in \chi_C(t) : x \in c \}.
\end{align*}
\]
Clause Introduce Nodes

Lemma

Let \( t \) be a clause introduce node with child \( t' \) and \( \chi(t) = \chi(t') \cup \{c\} \) for a clause \( c \). Then, for each truth assignment \( \alpha : \chi_V(t) \to \{0, 1\} \) and set \( A \subseteq \chi_C(t) \), we have

\[
n(t, \alpha, A) = \begin{cases} 
n(t', \alpha, A) & \text{if } c \notin A \text{ and } \alpha \text{ satisfies } c; \\
n(t', \alpha, A \setminus \{c\}) & \text{if } c \in A \text{ and } \alpha \text{ does not satisfy } c; \\
0 & \text{otherwise.}
\end{cases}
\]
Forget Nodes

**Lemma**

Let \( t \) be a variable forget node with child \( t' \) and \( \chi(t) = \chi(t') \setminus \{x\} \). Then, for each truth assignment \( \alpha : \chi_V(t) \rightarrow \{0, 1\} \) and set \( A \subseteq \chi_C(t) \), we have

\[
n(t, \alpha, A) = n(t', \alpha_{x=0}, A) + n(t', \alpha_{x=1}, A).
\]

**Lemma**

Let \( t \) be a clause forget node with child \( t' \) and \( \chi(t) = \chi(t') \setminus \{c\} \). Then, for each truth assignment \( \alpha : \chi_V(t) \rightarrow \{0, 1\} \) and set \( A \subseteq \chi_C(t) \), we have

\[
n(t, \alpha, A) = n(t', \alpha, A).
\]
Summary

Data Structure

In this dynamic programming algorithm, we have to maintain the following data structure at each node $t$ in the tree decomposition $T$: $(\alpha, A, n)$, where $\alpha : \chi_V(t) \rightarrow \{0, 1\}$, $A \subseteq \chi_C(t)$, and $n$ is an integer.
Summary

Data Structure

In this dynamic programming algorithm, we have to maintain the following data structure at each node $t$ in the tree decomposition $T$: $(\alpha, A, n)$, where $\alpha : \chi_V(t) \to \{0, 1\}$, $A \subseteq \chi_C(t)$, and $n$ is an integer.

Alternative view: represent $(\alpha, A)$ as a subset of $\chi(t)$. Hence, if $T$ has width $\leq w - 1$, we have to store a table of size $O(w^2 w)$ at each node.
Summary

Data Structure

In this dynamic programming algorithm, we have to maintain the following data structure at each node $t$ in the tree decomposition $T$: $(\alpha, A, n)$, where $\alpha : \chi_V(t) \rightarrow \{0, 1\}$, $A \subseteq \chi_C(t)$, and $n$ is an integer.

Alternative view: represent ($\alpha, A$) as a subset of $\chi(t)$. Hence, if $T$ has width $\leq w - 1$, we have to store a table of size $O(w^2)$ at each node.

Theorem (Samer/Szeider, 2010)

The \#SAT-problem can be solved in time $O(w^4 N)$, where $N$ denotes the number of nodes in a tree decomposition of width $w - 1$ for an input CNF formula $F$ (assuming unit cost for arithmetic operations).
Summary

Lessons learned

- The models of a CNF-formula $F$ can be computed by means of a single bottom-up traversal of a tree decomposition $T$ of $F$:
  - assign truth value to the variables in the bags at leaf nodes and
  - extend the truth assignments at the variable introduce nodes.
Summary

Lessons learned

- The models of a CNF-formula $F$ can be computed by means of a single bottom-up traversal of a tree decomposition $T$ of $F$:
  - assign truth value to the variables in the bags at leaf nodes and
  - extend the truth assignments at the variable introduce nodes.

- We cannot afford to store all truth assignments on $V_t$.

- It suffices to store the “projection” of each truth assignment on $V_t$ to the variables and clauses in $\chi(t)$.

- The resulting data structure is single-exponential in the treewidth.

- Each “projection” may represent many truth assignments on $V_t$: these are indistinguishable for the further bottom-up traversal.
Dynamic Programming in Propositional Abduction

Definition

A propositional abduction problem (PAP) is a quadruple $P = \langle V, H, M, C \rangle$ consisting of

- a finite set of propositional variables $V$,
- a theory $C$, which is a consistent set of clauses over the variables $V$,
- a set of hypotheses $H \subseteq V$,
- and a set of manifestations $M \subseteq V$. 
Dynamic Programming in Propositional Abduction

Definition

A propositional abduction problem (PAP) is a quadruple $\mathcal{P} = \langle V, H, M, \mathcal{C} \rangle$ consisting of

- a finite set of propositional variables $V$,
- a theory $\mathcal{C}$, which is a consistent set of clauses over the variables $V$,
- a set of hypotheses $H \subseteq V$,
- and a set of manifestations $M \subseteq V$.

A set $S$ satisfying $S \subseteq H$ is a solution iff

- $\mathcal{C} \cup S$ is consistent (i.e., satisfiable)
- and $\mathcal{C} \cup S \models M$ holds.
Dynamic Programming in Propositional Abduction

Intuition of Computing a Solution

- The clause set $\mathcal{C}$ has, in general, many models.
- There may exist models of $\mathcal{C}$ where a manifestation $m \in M$ is false.
- Idea of constructing a solution $S$: adding a variable $h \in H$ to $S$ means that we discard all models of $\mathcal{C}$ where $h$ is false.
Dynamic Programming in Propositional Abduction

Intuition of Computing a Solution

- The clause set $C$ has, in general, many models.
- There may exist models of $C$ where a manifestation $m \in M$ is false.
- **Idea of constructing a solution $S$:** adding a variable $h \in H$ to $S$ means that we discard all models of $C$ where $h$ is false.
- **Difficulty** (and reason for $\Sigma_2$-completeness):
  - $S$ must be “big enough” so that every manifestation $m \in M$ is true in the remaining models of $C$.
  - $S$ must be “small enough” so that at least one model of $C$ is left.
Dynamic Programming in Propositional Abduction

Dynamic Programming Algorithm

- In a bottom-up traversal of the tree decomposition $T$, at each node $t \in T$, consider all subsets $S \subseteq H_t$.
- For every such $S$ do:
  - Consider all possible truth assignments $I$ for $C \cup S$.
  - Keep track if some manifestation $m \in M$ is set to false in $I$.
- Read off the solutions at the root node:
  - $C \cup S$ has at least one model.
  - In all models of $C \cup S$, no $m \in M$ is set to false.
Data Structure and Complexity

- Recall from the \#SAT-algorithm the idea of projecting truth assignments on $V_t$ onto the bag $\chi(t)$. 
FPT-Algorithm for Propositional Abduction

Data Structure and Complexity

- Recall from the \#SAT-algorithm the idea of projecting truth assignments on $V_t$ onto the bag $\chi(t)$.
- Data structure required for Abduction at each node $t \in T$:
  - Restriction of each set $S \subseteq H_t$ to $S \cap \chi(t)$.
  - Set of projections of all possible truth assignments for $C \cup S$.
  - One bit for each such projection indicating if some $m \in M_t$ is false in the corresponding truth assignments.
- Formal definition of the data structure: $2^{\chi(t)} \times 2^{2^{\chi(t)}} \times \{0, 1\}$.
- Complexity of the algorithm: double-exponential in the treewidth.
Extensions

Other types of problems

- **Optimization Problems**: during the bottom-up traversal, keep track of the intermediate value of the target function.

- **Counting Problems**: compute the number of solutions in abduction, the number of minimal models, the number of answer sets, etc.
Extensions

Other types of problems

- **Optimization Problems**: during the bottom-up traversal, keep track of the intermediate value of the target function.

- **Counting Problems**: compute the number of solutions in abduction, the number of minimal models, the number of answer sets, etc.

- **Enumeration Problems**: compute all solutions, all minimal models, all answer sets, etc. with fixed-parameter linear delay:
  - during bottom-up traversal: keep track of the relationship between the data structure at the child node(s) and the parent node
  - compute the output by one top-down traversal per solution, minimal model, answer set, etc.
Extensions

Other types of problems

- **Optimization Problems**: during the bottom-up traversal, keep track of the intermediate value of the target function.

- **Counting Problems**: compute the number of solutions in abduction, the number of minimal models, the number of answer sets, etc.

- **Enumeration Problems**: compute all solutions, all minimal models, all answer sets, etc. with fixed-parameter linear delay:
  - during bottom-up traversal: keep track of the relationship between the data structure at the child node(s) and the parent node
  - compute the output by **one top-down traversal per solution**, minimal model, answer set, etc.

- (Special Case) Enumeration Problem defined by a **unary predicate**: For instance, compute all relevant resp. necessary hypotheses in an abduction problem. A **single top-down traversal** suffices.
Experimental Results

### Implementation

Some of the algorithms presented here have been implemented in

- **Datalog** (SAT),
- **C** (propositional abduction), or
- **Haskell** (answer set programming).

### Experience

- **Datalog**: reasonable performance for instances of low treewidth, FPL-behaviour not achievable by using datalog engine as black box.
- **C**: FPL-behaviour achievable, e.g.: abduction for treewidth 3.
- **Haskell**: almost as fast as C-programme; formal description of dynamic programming algorithm is very close to programme code, e.g.: answer set programming with treewidth \( \leq 7 \).

**Conclusion**: Functional programming language seems to be best suited.
Outline

1. Motivation

2. Treewidth of Finite Structures

3. Tractable Reasoning via Courcelle’s Theorem

4. FPT-Algorithms via Dynamic Programming

5. Conclusion
Conclusion

Main Results

- Application of treewidth to non-monotonic reasoning
- Many FPT-results via Courcelle’s Theorem
- Efficient computation with dynamic programming algorithms
Conclusion

### Main Results

- Application of treewidth to non-monotonic reasoning
- Many FPT-results via Courcelle’s Theorem
- Efficient computation with dynamic programming algorithms

### Future Work

- Dynamic programming algorithms for further problems
- Other structural decomposition methods like clique-width
- Further extensions of Courcelle’s Theorem