A Comparison of ASP-Based and SAT-Based Algorithms for the Contension Inconsistency Measure

Isabelle Kuhlmann, Anna Gessler, Vivien Laszlo and Matthias Thimm

Artificial Intelligence Group,
University of Hagen, Germany

October 17, 2022
Motivation

In Artificial Intelligence, we cannot avoid the occurrence of conflicting (inconsistent) information.
In Artificial Intelligence, we cannot avoid the occurrence of conflicting (*inconsistent*) information.

**Examples**

- Different expert opinions or assessments
- Noisy/distorted sensor data
In Artificial Intelligence, we cannot avoid the occurrence of conflicting (*inconsistent*) information.

**Examples**

- Different expert opinions or assessments
- Noisy/distorted sensor data

Hence, the handling of inconsistent information is a crucial problem.
Motivation

The field of *inconsistency measurement* provides an analytical perspective on this matter.
The field of *inconsistency measurement* provides an analytical perspective on this matter.

- **Goal:** quantitatively assess the *severity* of inconsistency
The field of *inconsistency measurement* provides an analytical perspective on this matter.

**Goal:** quantitatively assess the *severity* of inconsistency

### Application Examples

- Analysis of inconsistencies in news reports (Hunter, 2006)

There is clearly a need for practical working solutions!
Motivation

The field of *inconsistency measurement* provides an analytical perspective on this matter.

▶ **Goal:** quantitatively assess the *severity* of inconsistency

<table>
<thead>
<tr>
<th>Application Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Analysis of inconsistencies in news reports (Hunter, 2006)</td>
</tr>
<tr>
<td>▶ Support of collaborative software requirements specifications (Martinez et al., 2004)</td>
</tr>
</tbody>
</table>
The field of inconsistency measurement provides an analytical perspective on this matter.

**Goal:** quantitatively assess the severity of inconsistency

### Application Examples

- Analysis of inconsistencies in news reports (Hunter, 2006)
- Support of collaborative software requirements specifications (Martinez et al., 2004)
- Monitoring and maintenance of quality in database settings (Bertossi, 2018)
The field of *inconsistency measurement* provides an analytical perspective on this matter.

▶ **Goal:** quantitatively assess the *severity* of inconsistency

### Application Examples

▶ Analysis of inconsistencies in news reports (Hunter, 2006)
▶ Support of collaborative software requirements specifications (Martinez et al., 2004)
▶ Monitoring and maintenance of quality in database settings (Bertossi, 2018)

There is clearly a need for practical working solutions!
There exists a plethora of different inconsistency measures in the literature.
There exists a plethora of different inconsistency measures in the literature. Only few works consider the topic of inconsistency measurement from an *algorithmic* perspective.
There exists a plethora of different inconsistency measures in the literature.

Only few works consider the topic of inconsistency measurement from an algorithmic perspective.

Complexity study by Thimm and Wallner (2019):

- Inconsistency measurement is computationally hard in general.
- The most suitable candidates for practical applications are on the first level of the polynomial hierarchy.
- In this work, we consider the contention inconsistency measure.
Motivation

Contributions

- We propose an algorithm for the contention inconsistency measure based on satisfiability problem (SAT) solving

- We propose a revised version of an algorithm based on answer set programming (ASP)

- Experimental evaluation: We compare the two methods to each other, and to a naive baseline method
We propose an algorithm for the contension inconsistency measure based on satisfiability problem (SAT) solving.

We propose a revised version of an algorithm based on answer set programming (ASP).
Motivation

Contributions

- We propose an algorithm for the contension inconsistency measure based on satisfiability problem (SAT) solving
- We propose a revised version of an algorithm based on answer set programming (ASP)
- Experimental evaluation: We compare the two methods to each other, and to a naive baseline method
Overview

1. Preliminaries
2. Algorithm Based on SAT
3. Algorithm Based on ASP
4. Experimental Analysis
5. Conclusion
Overview

1 Preliminaries

2 Algorithm Based on SAT

3 Algorithm Based on ASP

4 Experimental Analysis

5 Conclusion
Intuition

An inconsistency measure assigns a value to a (propositional) knowledge base.

- The larger the value, the more severe the inconsistency
- Consistent knowledge bases have the value 0
Intuition
An inconsistency measure assigns a value to a (propositional) knowledge base.
- The larger the value, the more severe the inconsistency
- Consistent knowledge bases have the value 0

Definition
Let $K$ be the set of all (propositional) knowledge bases.
An inconsistency measure $\mathcal{I}$ is a function $\mathcal{I}: K \rightarrow \mathbb{R}_{\geq 0}^\infty$ that satisfies $\mathcal{I}(K) = 0$ iff $K$ is consistent, for all $K \in K$. 
The contension inconsistency measure is based on Priest’s *three-valued logic*:

- In addition to true (t) and false (f), this logic includes a third value which indicates *paradoxical*, or *both true and false* (b)
The contension inconsistency measure is based on Priest’s *three-valued logic*:

- In addition to true \((t)\) and false \((f)\), this logic includes a third value which indicates *paradoxical*, or *both true and false* \((b)\)
- A *three-valued interpretation* \(\omega^3\) is a function that assigns one of the three truth values to each atom in a given knowledge base:

\[
\omega^3 : At(\mathcal{K}) \rightarrow \{ t, f, b \}
\]
The contention inconsistency measure is based on Priest’s *three-valued logic*:

- In addition to true (\(t\)) and false (\(f\)), this logic includes a third value which indicates *paradoxical*, or both true and false (\(b\)).
- A *three-valued interpretation* \(\omega^3\) is a function that assigns one of the three truth values to each atom in a given knowledge base:
  \[
  \omega^3 : \text{At}(\mathcal{K}) \rightarrow \{t, f, b\}
  \]
- A three-valued *model* is an interpretation where each formula \(\alpha \in \mathcal{K}\) is assigned either \(t\) or \(b\).
The set of models wrt. $\mathcal{K}$ is defined as

$$\text{Models}(\mathcal{K}) = \{\omega^3 | \forall \alpha \in \mathcal{K}, \omega^3(\alpha) = t \text{ or } \omega^3(\alpha) = b\}.$$
Definition

The set of models wrt. $\mathcal{K}$ is defined as

$$\text{Models}(\mathcal{K}) = \{\omega^3 \mid \forall \alpha \in \mathcal{K}, \omega^3(\alpha) = t \text{ or } \omega^3(\alpha) = b\}.$$ 

We can divide the domain of an interpretation $\omega^3$ into two sets:
Definition

The set of models wrt. $\mathcal{K}$ is defined as

$$\text{Models}(\mathcal{K}) = \{ \omega^3 \mid \forall \alpha \in \mathcal{K}, \omega^3(\alpha) = t \text{ or } \omega^3(\alpha) = b \}.$$ 

We can divide the domain of an interpretation $\omega^3$ into two sets:

- One contains those atoms that are assigned a classical truth value $(t, f)$

An interpretation $\omega^3$ is consistent if and only if $\text{Conflictbase}(\omega^3) = \emptyset$. If $\text{Conflictbase}(\omega^3) \neq \emptyset$, then there exist two conflicting clauses in $\mathcal{K}$.

We can define $\text{Conflictbase}(\omega^3)$ as follows:

$$\text{Conflictbase}(\omega^3) = \{ x \in \text{At}(\mathcal{K}) \mid \omega^3(x) = b \}.$$
Definition

The set of models wrt. $\mathcal{K}$ is defined as

$$\text{Models}(\mathcal{K}) = \{ \omega^3 \mid \forall \alpha \in \mathcal{K}, \omega^3(\alpha) = t \text{ or } \omega^3(\alpha) = b \}.$$ 

We can divide the domain of an interpretation $\omega^3$ into two sets:

- One contains those atoms that are assigned a classical truth value ($t$, $f$)
- One contains those atoms that are assigned $b$
Definition

The set of models wrt. $\mathcal{K}$ is defined as

$$\text{Models}(\mathcal{K}) = \{ \omega^3 | \forall \alpha \in \mathcal{K}, \omega^3(\alpha) = t \text{ or } \omega^3(\alpha) = b \}.$$  

We can divide the domain of an interpretation $\omega^3$ into two sets:

- One contains those atoms that are assigned a classical truth value ($t, f$)
- One contains those atoms that are assigned $b$

Definition

$$\text{Conflictbase}(\omega^3) = \{ x \in \text{At}(\mathcal{K}) | \omega^3(x) = b \}$$
Intuition

The contension inconsistency measure $I_c$ describes the minimum number of atoms in $\mathcal{K}$ that are assigned truth value $b$. 
The contension inconsistency measure $I_c$ describes the minimum number of atoms in $\mathcal{K}$ that are assigned truth value $b$.

**Definition**

$$I_c(\mathcal{K}) = \min\{|\text{Conflictbase}(\omega^3)| \mid \omega^3 \in \text{Models}(\mathcal{K})\}.$$
**Intuition**

The contension inconsistency measure $I_c$ describes the minimum number of atoms in $\mathcal{K}$ that are assigned truth value $b$.

**Definition**

$$I_c(\mathcal{K}) = \min\{|\text{Conflictbase}(\omega^3)| \mid \omega^3 \in \text{Models}(\mathcal{K})\}.$$ 

*Example:* $\mathcal{K}_1 = \{x \land y, \neg x, y \lor z\}$
The contension inconsistency measure $I_c$ describes the minimum number of atoms in $\mathcal{K}$ that are assigned truth value $b$.

**Definition**

$$I_c(\mathcal{K}) = \min \{|\text{Conflictbase}(\omega^3)| \mid \omega^3 \in \text{Models}(\mathcal{K})\}.$$  

**Example:** $\mathcal{K}_1 = \{x \land y, \neg x, y \lor z\}$

- Let $\omega^3_1$ be an interpretation with $\omega^3_1(y) = \omega^3_1(z) = t$ and $\omega^3_1(x) = b$
Intuition
The contension inconsistency measure $I_c$ describes the minimum number of atoms in $\mathcal{K}$ that are assigned truth value $b$.

Definition
$$I_c(\mathcal{K}) = \min\{|\text{Conflictbase}(\omega^3)| \mid \omega^3 \in \text{Models}(\mathcal{K})\}.$$  

Example: $\mathcal{K}_1 = \{x \land y, \neg x, y \lor z\}$
- Let $\omega^3_1$ be an interpretation with $\omega^3_1(y) = \omega^3_1(z) = t$ and $\omega^3_1(x) = b$
- $\omega^3_1$ is a model of $\mathcal{K}_1$
**Intuition**

The contension inconsistency measure $I_c$ describes the minimum number of atoms in $\mathcal{K}$ that are assigned truth value $b$.

**Definition**

$$I_c(\mathcal{K}) = \min \{|\text{Conflictbase}(\omega^3)| \mid \omega^3 \in \text{Models}(\mathcal{K})\}.$$ 

**Example:** $\mathcal{K}_1 = \{x \land y, \neg x, y \lor z\}$

- Let $\omega_1^3$ be an interpretation with $\omega_1^3(y) = \omega_1^3(z) = t$ and $\omega_1^3(x) = b$
- $\omega_1^3$ is a model of $\mathcal{K}_1$
- $\text{Conflictbase}(\omega_1^3) = \{x\}$
Intuition

The contension inconsistency measure $I_c$ describes the minimum number of atoms in $\mathcal{K}$ that are assigned truth value $b$.

Definition

$$I_c(\mathcal{K}) = \min \{|\text{Conflictbase}(\omega^3)| \mid \omega^3 \in \text{Models}(\mathcal{K})\}.$$

Example: $\mathcal{K}_1 = \{x \land y, \neg x, y \lor z\}$

- Let $\omega_1^3$ be an interpretation with $\omega_1^3(y) = \omega_1^3(z) = t$ and $\omega_1^3(x) = b$
- $\omega_1^3$ is a model of $\mathcal{K}_1$
- $\text{Conflictbase}(\omega_1^3) = \{x\}$
- $I_c(\mathcal{K}_1) = |\text{Conflictbase}(\omega_1^3)| = |\{x\}| = 1$
Overview

1 Preliminaries

2 Algorithm Based on SAT

3 Algorithm Based on ASP

4 Experimental Analysis

5 Conclusion
Definition

The *Boolean Satisfiability Problem* (SAT) is the problem of deciding if there exists an interpretation that satisfies a given propositional formula.
Definition

The *Boolean Satisfiability Problem* (SAT) is the problem of deciding if there exists an interpretation that satisfies a given propositional formula.

- A *SAT solver* is a program that solves SAT for a given formula.
Definition

The *Boolean Satisfiability Problem* (SAT) is the problem of deciding if there exists an interpretation that satisfies a given propositional formula.

- A *SAT solver* is a program that solves SAT for a given formula
- There exist high-performance SAT solvers
  - Annual SAT competition

The problem of deciding if there exists an interpretation that satisfies a given propositional formula.
### Definition

The *Boolean Satisfiability Problem* (SAT) is the problem of deciding if there exists an interpretation that satisfies a given propositional formula.

- A *SAT solver* is a program that solves SAT for a given formula
- There exist high-performance SAT solvers
  - Annual SAT competition
- Input formulas must be in *Conjunctive Normal Form* (CNF)
Definition

The Boolean Satisfiability Problem (SAT) is the problem of deciding if there exists an interpretation that satisfies a given propositional formula.

- A SAT solver is a program that solves SAT for a given formula
- There exist high-performance SAT solvers
  - Annual SAT competition
- Input formulas must be in Conjunctive Normal Form (CNF)
- Cardinality constraints (here: at-most-k constraints): $a_1 + \ldots + a_n \leq k$
  - + operator: for every true atom, add 1
Algorithm Based on SAT

Goal

Find the value of $I_c(K)$ wrt. a given knowledge base $K$. ($\text{VALUE}_{I_c}$)
Algorithm Based on SAT

Goal

Find the value of $I_c(\mathcal{K})$ wrt. a given knowledge base $\mathcal{K}$. ($\text{VALUE}_{I_c}$)

- We cannot encode $\text{VALUE}_{I_c}$ directly in SAT
  - We encode the problem of deciding whether a given value $u$ is an upper bound of $I_c(\mathcal{K})$ ($\text{UPPER}_{I_c}$)
Algorithm Based on SAT

Goal

Find the value of $I_c(\mathcal{K})$ wrt. a given knowledge base $\mathcal{K}$. ($\text{VALUE}_{I_c}$)

- We cannot encode $\text{VALUE}_{I_c}$ directly in SAT
  - We encode the problem of deciding whether a given value $u$ is an upper bound of $I_c(\mathcal{K})$ ($\text{UPPER}_{I_c}$)
- Using a **binary search procedure** which includes iterative calls to a SAT solver, we can ultimately determine $\text{VALUE}_{I_c}$:
Algorithm Based on SAT

Goal

Find the value of $I_c(K)$ wrt. a given knowledge base $K$. ($\text{VALUE}_{I_c}$)

- We cannot encode $\text{VALUE}_{I_c}$ directly in SAT
  - We encode the problem of deciding whether a given value $u$ is an upper bound of $I_c(K)$ ($\text{UPPER}_{I_c}$)
- Using a **binary search procedure** which includes iterative calls to a SAT solver, we can ultimately determine $\text{VALUE}_{I_c}$:
  - We start with $u = \lfloor |\text{At}(K)|/2 \rfloor$ and determine the corresponding SAT encoding
Algorithm Based on SAT

Goal

Find the value of $\mathcal{I}_c(\mathcal{K})$ wrt. a given knowledge base $\mathcal{K}$. ($\text{VALUE}_{\mathcal{I}_c}$)

- We cannot encode $\text{VALUE}_{\mathcal{I}_c}$ directly in SAT
  - We encode the problem of deciding whether a given value $u$ is an upper bound of $\mathcal{I}_c(\mathcal{K})$ ($\text{UPPER}_{\mathcal{I}_c}$)
- Using a **binary search procedure** which includes iterative calls to a SAT solver, we can ultimately determine $\text{VALUE}_{\mathcal{I}_c}$:
  - We start with $u = \lfloor |\text{At}(\mathcal{K})|/2 \rfloor$ and determine the corresponding SAT encoding
  - If $u$ is an upper bound of $\mathcal{I}_c(\mathcal{K})$, we continue the search in the upper interval
Goal

Find the value of $I_c(K)$ wrt. a given knowledge base $K$. ($\text{VALUE}_{I_c}$)

- We cannot encode $\text{VALUE}_{I_c}$ directly in SAT
  - We encode the problem of deciding whether a given value $u$ is an upper bound of $I_c(K)$ ($\text{UPPER}_{I_c}$)
- Using a **binary search procedure** which includes iterative calls to a SAT solver, we can ultimately determine $\text{VALUE}_{I_c}$:
  - We start with $u = \lfloor |\text{At}(K)|/2 \rfloor$ and determine the corresponding SAT encoding
  - If $u$ is an upper bound of $I_c(K)$, we continue the search in the upper interval
  - If $u$ is *not* an upper bound, we continue the search in the lower interval
Algorithm Based on SAT

Goal

Find the value of $\mathcal{I}_c(\mathcal{K})$ wrt. a given knowledge base $\mathcal{K}$. ($\text{VALUE}_{\mathcal{I}_c}$)

- We cannot encode $\text{VALUE}_{\mathcal{I}_c}$ directly in SAT
  - We encode the problem of deciding whether a given value $u$ is an upper bound of $\mathcal{I}_c(\mathcal{K})$ ($\text{UPPER}_{\mathcal{I}_c}$)
- Using a binary search procedure which includes iterative calls to a SAT solver, we can ultimately determine $\text{VALUE}_{\mathcal{I}_c}$:
  - We start with $u = \lfloor |\text{At}(\mathcal{K})|/2 \rfloor$ and determine the corresponding SAT encoding
  - If $u$ is an upper bound of $\mathcal{I}_c(\mathcal{K})$, we continue the search in the upper interval
  - If $u$ is not an upper bound, we continue the search in the lower interval
  - After $\log_2(|\text{At}(\mathcal{K})|)$ calls\(^1\), we know $\text{VALUE}_{\mathcal{I}_c}$

\(^1\) $\text{At}(\mathcal{K})$ refers to the signature size of $\mathcal{K}$
Algorithm Based on SAT

SAT encoding for $\text{UPPER}_{\mathcal{L}_c}$:

- For every atom $x \in \text{At}(K)$ we introduce $x_t, x_b, x_f$:
  - We ensure that only one of these atoms is true: 
    \[(x_t \lor x_f \lor x_b) \land (
eg x_t \lor \neg x_f) \land (\neg x_t \lor \neg x_b) \land (\neg x_b \lor \neg x_f)\]
- For every (sub-)formula $\phi$ we introduce $v_t \phi, v_b \phi, v_f \phi$:
  - We encode $\land$, $\lor$, and $\neg$ in Priest’s three-valued logic:
    - Example: A conjunction $\phi = \psi_1 \land \psi_2$ is true if both conjuncts are true:
      \[v_t \phi \leftrightarrow v_t \psi_1 \land v_t \psi_2\]
  - If a formula $\phi$ consists of an individual atom $x$:
    \[v_t \phi \leftrightarrow x_t, \quad v_b \phi \leftrightarrow x_b, \quad v_f \phi \leftrightarrow x_f\]
- We ensure that each formula $\alpha \in K$ evaluates to $t$ or $b$:
  \[v_t \alpha \lor v_b \alpha\]
- Cardinality constraint representing that at most $u$ of the $b$-atoms are true:
  \[\text{at most } u \cdot \text{At}_b\]
Algorithm Based on SAT

SAT encoding for $\text{UPPER}_{T_c}$:

- For every atom $x \in \text{At}(\mathcal{K})$ we introduce $x_t, x_b, x_f$

- We ensure that only one of these atoms is true:
  \[(x_t \lor x_f \lor x_b) \land (\neg x_t \lor \neg x_f) \land (\neg x_t \lor \neg x_b) \land (\neg x_b \lor \neg x_f)\]

- For every (sub-)formula $\phi$ we introduce $v_t \phi, v_b \phi, v_f \phi$

- We encode $\land$, $\lor$, and $\neg$ in Priest’s three-valued logic

- Example: A conjunction $\phi = \psi_1 \land \psi_2$ is true if both conjuncts are true:
  \[v_t \phi \leftrightarrow v_t \psi_1 \land v_t \psi_2\]

- If a formula $\phi$ consists of an individual atom $x$:
  \[v_t \phi \leftrightarrow x_t, v_b \phi \leftrightarrow x_b, v_f \phi \leftrightarrow x_f\]

- We ensure that each formula $\alpha \in K$ evaluates to $t$ or $b$:
  \[v_t \alpha \lor v_b \alpha\]

- Cardinality constraint representing that at most $u$ of the $b$-atoms are true:
  \[\text{at most } u(\text{At } b)\]
**SAT encoding for** \( \text{UPPER}_{\mathcal{I}_c} \):

- For every atom \( x \in \text{At}(\mathcal{K}) \) we introduce \( x_t, x_b, x_f \)
- We ensure that only one of these atoms is true:

\[
(x_t \lor x_f \lor x_b) \land (\neg x_t \lor \neg x_f) \land (\neg x_t \lor \neg x_b) \land (\neg x_b \lor \neg x_f)
\]
SAT encoding for $\text{UPPER}_{I_c}$:

- For every atom $x \in \text{At}(\mathcal{K})$ we introduce $x_t, x_b, x_f$

- We ensure that only one of these atoms is true:

$$\begin{align*}
(x_t \lor x_f \lor x_b) \land (\neg x_t \lor \neg x_f) \land (\neg x_t \lor \neg x_b) \land (\neg x_b \lor \neg x_f)
\end{align*}$$

- For every (sub-)formula $\phi$ we introduce $v^t_{\phi}, v^b_{\phi}, v^f_{\phi}$

- We ensure that each formula $\alpha \in K$ evaluates to $t$ or $b$:

$$v^t_\alpha \lor v^b_\alpha$$

- Cardinality constraint representing that at most $u$ of the $b$-atoms are true:

$$\text{at most } u (\text{At}_b)$$
Algorithm Based on SAT

**SAT encoding for** $\text{UPPER}_{T_c}$:

- For every atom $x \in \text{At}(\mathcal{K})$ we introduce $x_t, x_b, x_f$
  
  - We ensure that only one of these atoms is true:
    
    $$ (x_t \lor x_f \lor x_b) \land (\neg x_t \lor \neg x_f) \land (\neg x_t \lor \neg x_b) \land (\neg x_b \lor \neg x_f) $$
  
- For every (sub-)formula $\phi$ we introduce $v^t_\phi, v^b_\phi, v^f_\phi$
  
- We encode $\land, \lor, \neg$ in Priest's three-valued logic
Algorithm Based on SAT

**SAT encoding for $\text{UPPER}_{I_c}$:**

- For every atom $x \in \text{At}(K)$ we introduce $x_t, x_b, x_f$
  - We ensure that only one of these atoms is true:
    $$(x_t \lor x_f \lor x_b) \land (\neg x_t \lor \neg x_f) \land (\neg x_t \lor \neg x_b) \land (\neg x_b \lor \neg x_f)$$

- For every (sub-)formula $\phi$ we introduce $v^t_\phi, v^b_\phi, v^f_\phi$

- We encode $\land$, $\lor$, and $\neg$ in Priest’s three-valued logic
  - Example: A conjunction $\phi = \psi_1 \land \psi_2$ is true if both conjuncts are true:
    $$v^t_\phi \leftrightarrow v^t_{\psi_1} \land v^t_{\psi_2}$$
Algorithm Based on SAT

**SAT encoding for \( \text{UPPER}_{I_c} \):**

- For every atom \( x \in \text{At}(K) \) we introduce \( x_t, x_b, x_f \)
  - We ensure that only one of these atoms is true:
    \[
    (x_t \lor x_f \lor x_b) \land (\neg x_t \lor \neg x_f) \land (\neg x_t \lor \neg x_b) \land (\neg x_b \lor \neg x_f)
    \]

- For every (sub-)formula \( \phi \) we introduce \( v^t_\phi, v^b_\phi, v^f_\phi \)

- We encode \( \land, \lor, \text{ and } \neg \) in Priest’s three-valued logic
  - *Example:* A conjunction \( \phi = \psi_1 \land \psi_2 \) is true if both conjuncts are true:
    \[
    v^t_\phi \iff v^t_{\psi_1} \land v^t_{\psi_2}
    \]

- If a formula \( \phi \) consists of an individual atom \( x \):
  \[
  v^t_\phi \iff x_t, \quad v^b_\phi \iff x_b, \quad v^f_\phi \iff x_f
  \]
Algorithm Based on SAT

**SAT encoding for** $\text{UPPER}_{Ic}$:

- For every atom $x \in \text{At}(\mathcal{K})$ we introduce $x_t, x_b, x_f$
  - We ensure that only one of these atoms is true:
    $$(x_t \lor x_f \lor x_b) \land (\neg x_t \lor \neg x_f) \land (\neg x_t \lor \neg x_b) \land (\neg x_b \lor \neg x_f)$$

- For every (sub-)formula $\phi$ we introduce $v^t_\phi, v^b_\phi, v^f_\phi$

- We encode $\land$, $\lor$, and $\neg$ in Priest’s three-valued logic
  - *Example:* A conjunction $\phi = \psi_1 \land \psi_2$ is true if both conjuncts are true:
    $$v^t_\phi \leftrightarrow v^t_{\psi_1} \land v^t_{\psi_2}$$

- If a formula $\phi$ consists of an individual atom $x$:
  $$v^t_\phi \leftrightarrow x_t, v^b_\phi \leftrightarrow x_b, v^f_\phi \leftrightarrow x_f$$

- We ensure that each formula $\alpha \in \mathcal{K}$ evaluates to $t$ or $b$:
  $$v^t_\alpha \lor v^b_\alpha$$
Algorithm Based on SAT

**SAT encoding for** \( \text{UPPER}_{I_c} \):

- For every atom \( x \in \text{At}(\mathcal{K}) \) we introduce \( x_t, x_b, x_f \)
  - We ensure that only one of these atoms is true:
    \[
    (x_t \lor x_f \lor x_b) \land (\neg x_t \lor \neg x_f) \land (\neg x_t \lor \neg x_b) \land (\neg x_b \lor \neg x_f)
    \]

- For every (sub-)formula \( \phi \) we introduce \( v^t_\phi, v^b_\phi, v^f_\phi \)
- We encode \( \land, \lor, \) and \( \neg \) in Priest’s three-valued logic
  - *Example:* A conjunction \( \phi = \psi_1 \land \psi_2 \) is true if both conjuncts are true:
    \[
    v^t_\phi \leftrightarrow v^t_{\psi_1} \land v^t_{\psi_2}
    \]

- If a formula \( \phi \) consists of an individual atom \( x \):
  \[
  v^t_\phi \leftrightarrow x_t, v^b_\phi \leftrightarrow x_b, v^f_\phi \leftrightarrow x_f
  \]

- We ensure that each formula \( \alpha \in \mathcal{K} \) evaluates to \( t \) or \( b \):
  \[
  v^t_\alpha \lor v^b_\alpha
  \]

- Cardinality constraint representing that at most \( u \) of the \( b \)-atoms are true:
  \[
  \text{at most } u(\text{At}_b)
  \]
Overview

1 Preliminaries

2 Algorithm Based on SAT

3 Algorithm Based on ASP

4 Experimental Analysis

5 Conclusion
Answer Set Programming (ASP) is a declarative programming paradigm.

- Targeted at difficult search problems
*Answer Set Programming* (ASP) is a declarative programming paradigm.

- Targeted at difficult search problems

**Intuition**

Rather than modeling instructions on how to solve a problem, a *representation of the problem itself* is modeled.
Answer Set Programming (ASP) is a declarative programming paradigm.

- Targeted at difficult search problems

**Intuition**

Rather than modeling instructions on how to solve a problem, a *representation of the problem itself* is modeled.

- Problem is represented in a logical format (*extended logic program*)
Answer Set Programming (ASP) is a declarative programming paradigm.

- Targeted at difficult search problems

**Intuition**

Rather than modeling instructions on how to solve a problem, a *representation of the problem itself* is modeled.

- Problem is represented in a logical format (*extended logic program*)
- The models of this representation describes the solution of the original problem
  - These models are called *answer sets*
Answer Set Programming (ASP) is a declarative programming paradigm.

- Targeted at difficult search problems

Intuition

Rather than modeling instructions on how to solve a problem, a representation of the problem itself is modeled.

- Problem is represented in a logical format (extended logic program)
- The models of this representation describes the solution of the original problem
  - These models are called answer sets
- An extended logic program consists of rules
  - In addition, we use cardinality constraints, and optimize statements
Observation

In ASP, we can encode $\text{VALUE}_{I_c}$ directly by using a minimize statement.
Observation

In ASP, we can encode $\text{VALUE}_{I_c}$ directly by using a *minimize statement*.

- There are two previous versions of the ASP-based approach in the literature.
Observation

In ASP, we can encode $\text{VALUE}_{I_c}$ directly by using a *minimize statement*.

- There are two previous versions of the ASP-based approach in the literature
- Our new revision is very similar to the second approach (Kuhlmann and Thimm, 2021), however it uses first-order concepts for ASP rules
Observation

In ASP, we can encode $\text{VALUE}_{I_c}$ directly by using a *minimize statement*. 

- There are two previous versions of the ASP-based approach in the literature.
- Our new revision is very similar to the second approach (Kuhlmann and Thimm, 2021), however it uses first-order concepts for ASP rules.
  - This eases readability and
  - allows for an automated, internally optimized, grounding procedure.
Algorithm Based on ASP

**ASP encoding for** $\text{VALUE}_{I_c}$:

- Every atom $x \in \text{At}(K)$ is represented as $\text{atom}(x)$.
- Every formula $\alpha \in K$ as $\text{kbMember}(\alpha)$.
- The truth values as $\text{tv}(t)$, $\text{tv}(b)$, $\text{tv}(f)$.

- We represent conjunctions, disjunctions, negations, and formulas consisting of individual atoms as such.

**Example:**
- A conjunction $\phi = \psi_1 \land \psi_2$ is represented as $\text{conjunction}(\phi, \psi_1, \psi_2)$.

- "Guess" an interpretation:
  $$\{\text{truthValue}(A, T) : \text{tv}(T)\} :- \text{atom}(A).$$

- We encode $\land$, $\lor$, $\neg$, and formulas consisting of individual atoms:
  **Example:**
  - A conjunction $\phi = \psi_1 \land \psi_2$ is true if both conjuncts are true:
    $$\text{truthValue}(F, t) :- \text{conjunction}(F, G, H), \text{truthValue}(G, t), \text{truthValue}(H, t).$$

- Every $\alpha \in K$ must evaluate to $t$ or $b$:
  $$:- \text{truthValue}(F, f), \text{kbMember}(F).$$

- Minimize statement:
  $$\#\text{minimize} \{1, A : \text{truthValue}(A, b), \text{atom}(A)\}.$$
Algorithm Based on ASP

**ASP encoding for $\text{VALUE}_{\mathcal{I}_c}$:**

- Every atom $x \in \text{At}(\mathcal{K})$ is represented as $\text{atom}(x)$, every formula $\alpha \in \mathcal{K}$ as $\text{kbMember}(\alpha)$, and the truth values as $\text{tv}(\mathit{t}), \text{tv}(\mathit{b}), \text{tv}(\mathit{f})$.
Algorithm Based on ASP

**ASP encoding for** \( \text{VALUE}_{\mathcal{L}_c} \):

- Every atom \( x \in \text{At}(\mathcal{K}) \) is represented as \( \text{atom}(x) \), every formula \( \alpha \in \mathcal{K} \) as \( \text{kbMember}(\alpha) \), and the truth values as \( \text{tv}(t), \text{tv}(b), \text{tv}(f) \).

- We represent conjunctions, disjunctions, negations, and formulas consisting of individual atoms as such:

\[
\begin{align*}
\text{conjunction}(F,G,H) & \iff \text{truthValue}(G,t) \land \text{truthValue}(H,t) \\
\text{disjunction}(F,G,H) & \iff \text{truthValue}(G,t) \lor \text{truthValue}(H,t) \\
\text{negation}(F) & \iff \neg \text{truthValue}(F,t)
\end{align*}
\]

- “Guess” an interpretation:

\[
\begin{align*}
\{ \text{truthValue}(A,T) : \text{tv}(T) \} & :- \text{atom}(A).
\end{align*}
\]

- We encode \( \land, \lor, \neg \), and formulas consisting of individual atoms:

\[
\begin{align*}
\text{truthValue}(F,t) & :- \text{conjunction}(F,G,H), \text{truthValue}(G,t), \text{truthValue}(H,t).
\end{align*}
\]

- Every \( \alpha \in \mathcal{K} \) must evaluate to \( t \) or \( b \):

\[
\begin{align*}
&:- \text{truthValue}(F,f), \text{kbMember}(F).
\end{align*}
\]

- Minimize statement:

\[
\begin{align*}
&\#\text{minimize} \{1, A : \text{truthValue}(A,b), \text{atom}(A)\}.
\end{align*}
\]
Algorithm Based on ASP

**ASP encoding for \( \text{VALUE}_{I_c} \):**

- Every atom \( x \in \text{At}(\mathcal{K}) \) is represented as \( \text{atom}(x) \), every formula \( \alpha \in \mathcal{K} \) as \( \text{kbMember}(\alpha) \), and the truth values as \( \text{tv}(t), \text{tv}(b), \text{tv}(f) \).

- We represent conjunctions, disjunctions, negations, and formulas consisting of individual atoms as such.
  - **Example:** A conjunction \( \phi = \psi_1 \land \psi_2 \) is represented as \( \text{conjunction}(\phi, \psi_1, \psi_2) \).
Algorithm Based on ASP

**ASP encoding for \( \text{VALUE}_{I_c} \):**

- Every atom \( x \in \text{At}(\mathcal{K}) \) is represented as \( \text{atom}(x) \), every formula \( \alpha \in \mathcal{K} \) as \( \text{kbMember}(\alpha) \), and the truth values as \( \text{tv}(t), \text{tv}(b), \text{tv}(f) \).

- We represent conjunctions, disjunctions, negations, and formulas consisting of individual atoms as such.
  - Example: A conjunction \( \phi = \psi_1 \land \psi_2 \) is represented as \( \text{conjunction}(\phi, \psi_1, \psi_2) \).

- “Guess” an interpretation: \( 1\{\text{truthValue}(A,T): tv(T)\}1:- \text{atom}(A) \).
Algorithm Based on ASP

**ASP encoding for** $\text{VALUE}_{I_c}$:

- Every atom $x \in \text{At}(\mathcal{K})$ is represented as $\text{atom}(x)$, every formula $\alpha \in \mathcal{K}$ as $\text{kbMember}(\alpha)$, and the truth values as $\text{tv}(t), \text{tv}(b), \text{tv}(f)$.

- We represent conjunctions, disjunctions, negations, and formulas consisting of individual atoms as such.
  
  - *Example:* A conjunction $\phi = \psi_1 \land \psi_2$ is represented as $\text{conjunction}(\phi, \psi_1, \psi_2)$.

- “Guess” an interpretation: $1\{\text{truthValue}(A,T): \text{tv}(T)\}1:- \text{atom}(A)$.

- We encode $\land$, $\lor$, $\neg$, and formulas consisting of individual atoms:
Algorithm Based on ASP

**ASP encoding for \( \text{VALUE}_{I_c} \):**

- Every atom \( x \in \text{At}(\mathcal{K}) \) is represented as \( \text{atom}(x) \), every formula \( \alpha \in \mathcal{K} \) as \( \text{kbMember}(\alpha) \), and the truth values as \( \text{tv}(t), \text{tv}(b), \text{tv}(f) \).

- We represent conjunctions, disjunctions, negations, and formulas consisting of individual atoms as such.

  - **Example:** A conjunction \( \phi = \psi_1 \land \psi_2 \) is represented as \( \text{conjunction}(\phi, \psi_1, \psi_2) \).

- “Guess” an interpretation:

  \[
  1\{\text{truthValue}(A,T): \text{tv}(T)\}1:- \text{atom}(A).
  \]

- We encode \( \land, \lor, \neg \), and formulas consisting of individual atoms:

  - **Example:** A conjunction \( \phi = \psi_1 \land \psi_2 \) is true if both conjuncts are true:

    \[
    \text{truthValue}(F,t) :- \text{conjunction}(F,G,H), \text{truthValue}(G,t), \text{truthValue}(H,t).
    \]

  I. Kuhlmann et al. A Comparison of ASP-Based and SAT-Based Algorithms for the Contension Inconsistency Measure
Algorithm Based on ASP

ASP encoding for $\text{VALUE}_{I_c}$:

- Every atom $x \in \text{At}(K)$ is represented as $\text{atom}(x)$, every formula $\alpha \in K$ as $\text{kbMember}(\alpha)$, and the truth values as $tv(t), tv(b), tv(f)$.

- We represent conjunctions, disjunctions, negations, and formulas consisting of individual atoms as such.
  - Example: A conjunction $\phi = \psi_1 \land \psi_2$ is represented as $\text{conjunction}(\phi, \psi_1, \psi_2)$.

- “Guess” an interpretation: $1\{\text{truthValue}(A,T): tv(T)\}1:- \text{atom}(A)$.

- We encode $\land, \lor, \neg$, and formulas consisting of individual atoms:
  - Example: A conjunction $\phi = \psi_1 \land \psi_2$ is true if both conjuncts are true:
    $$\text{truthValue}(F,t) :- \text{conjunction}(F,G,H), \text{truthValue}(G,t), \text{truthValue}(H,t).$$

- Every $\alpha \in K$ must evaluate to $t$ or $b$: $:- \text{truthValue}(F,f), \text{kbMember}(F)$. 
Algorithm Based on ASP

**ASP encoding for \( \text{VALUE}_{I_c} \):**

- Every atom \( x \in \text{At}(\mathcal{K}) \) is represented as \( \text{atom}(x) \), every formula \( \alpha \in \mathcal{K} \) as \( \text{kbMember}(\alpha) \), and the truth values as \( \text{tv}(t), \text{tv}(b), \text{tv}(f) \).

- We represent conjunctions, disjunctions, negations, and formulas consisting of individual atoms as such.
  
  - **Example:** A conjunction \( \phi = \psi_1 \land \psi_2 \) is represented as \( \text{conjunction}(\phi, \psi_1, \psi_2) \).

- “Guess” an interpretation:

  \[
  1\{\text{truthValue}(A,T): \text{tv}(T)\}1:- \text{atom}(A).
  \]

- We encode \( \land, \lor, \neg \), and formulas consisting of individual atoms:

  - **Example:** A conjunction \( \phi = \psi_1 \land \psi_2 \) is true if both conjuncts are true:

    \[
    \text{truthValue}(F,t) :- \text{conjunction}(F,G,H), \text{truthValue}(G,t), \text{truthValue}(H,t).
    \]

- Every \( \alpha \in \mathcal{K} \) must evaluate to \( t \) or \( b \): \( :- \text{truthValue}(F,f), \text{kbMember}(F) \).

- Minimize statement: \( \#\text{minimize}\{1,A: \text{truthValue}(A,b), \text{atom}(A)\} \).
Overview

1 Preliminaries

2 Algorithm Based on SAT

3 Algorithm Based on ASP

4 Experimental Analysis

5 Conclusion
**SRS Dataset**

- Synthetic dataset
- Created using the *SyntacticRandomSampler*\(^2\)
- 1800 knowledge bases
- Smallest instances: signature size 3; 5–15 formulas
- Largest instances: signature size 30; 50–100 formulas
Experimental Setup — Datasets

### SRS Dataset
- Synthetic dataset
- Created using the `SyntacticRandomSampler`\(^2\)
- 1800 knowledge bases
- Smallest instances: signature size 3; 5–15 formulas
- Largest instances: signature size 30; 50–100 formulas

\(^2\)http://tweetyproject.org/api/1.14/net/sf/tweety/logics/pl/util/SyntacticRandomSampler.html

### ML Dataset
- “Translated” dataset
- Based on *Animals with Attributes*
  - Using the Apriori algorithm, we mined association rules
  - Rules were interpreted as propositional logic implications
- 1920 knowledge bases
- Mean signature size: 76
- Mean number of formulas: 11,767
Experimental Setup

Implementation details:

▶ SAT-based and ASP-based approach are implemented in C++
▶ Naive method: provided by TweetyProject (Java)
▶ SAT solver: CaDiCal sc2021
▶ ASP solver: Clingo 5.5.1

http://tweetyproject.org/api/1.14/net/sf/tweety/logics/pl/analysis/ContensionInconsistencyMeasure.html

I. Kuhlmann et al. A Comparison of ASP-Based and SAT-Based Algorithms for the Contension Inconsistency Measure 22 / 27
Implementation details:

- SAT-based and ASP-based approach are implemented in C++
Implementation details:

- SAT-based and ASP-based approach are implemented in C++
- Naive method: provided by TweetyProject\(^3\) (Java)

\(^3\)http://tweetyproject.org/api/1.14/net/sf/tweety/logics/pl/analysis/ContentionInconsistencyMeasure.html
Implementation details:

- SAT-based and ASP-based approach are implemented in C++
- Naive method: provided by TweetyProject\(^3\) (Java)
- SAT solver: CaDiCal sc2021

\(^3\)http://tweetyproject.org/api/1.14/net/sf/tweety/logics/pl/analysis/ContensionInconsistencyMeasure.html
Experimental Setup

Implementation details:

- SAT-based and ASP-based approach are implemented in C++
- Naive method: provided by TweetyProject³ (Java)
- SAT solver: CaDiCal sc2021
- ASP solver: Clingo 5.5.1

³http://tweetyproject.org/api/1.14/net/sf/tweety/logics/pl/analysis/ContensionInconsistencyMeasure.html
Experiment

As a first step, we measured the runtime of each approach wrt. each knowledge base.
As a first step, we measured the runtime of each approach wrt. each knowledge base.

SRS Dataset

![Graph showing runtime comparison between different approaches for the SRS Dataset. The graph plots the time (in seconds) on a logarithmic scale against the number of instances solved, with different lines representing SAT, ASP, Naive, and a timeout.]
Results

Experiment

As a first step, we measured the runtime of each approach wrt. each knowledge base.

Note that we excluded the naive method here.

---

4 Note that we excluded the naive method here.
Question

Why did the ASP approach outperform the SAT approach?
Why did the ASP approach outperform the SAT approach?

We examine the average runtime *composition* of each approach.
Question

Why did the ASP approach outperform the SAT approach?

Experiment

We examine the average runtime *composition* of each approach.
Results

Question

Is the newly revised ASP approach actually better than its predecessors?
**Question**

Is the newly revised ASP approach actually better than its predecessors?

**Experiment**

We compare the two previous ASP approaches with the newly proposed revision.
Results

Question

Is the newly revised ASP approach actually better than its predecessors?

Experiment

We compare the two previous ASP approaches with the newly proposed revision.
Overview

1 Preliminaries

2 Algorithm Based on SAT

3 Algorithm Based on ASP

4 Experimental Analysis

5 Conclusion
Conclusion

Summary:

▶ We presented a SAT-based, and a revised ASP-based approach for computing the contention inconsistency measure

Future Work:

▶ SAT: use different SAT solvers, and/or different methods for generating cardinality constraints; exploit approaches to MaxSAT
▶ Consider measures of higher complexity
▶ Explore other formalisms, such as Quantified Boolean Formulas

Thank you for your attention!
Conclusion

Summary:

▶ We presented a SAT-based, and a revised ASP-based approach for computing the contention inconsistency measure
▶ Our experimental analysis showed that both approaches are superior to a naive baseline method

Future Work:

▶ SAT: use different SAT solvers, and/or different methods for generating cardinality constraints; exploit approaches to MaxSAT
▶ Consider measures of higher complexity
▶ Explore other formalisms, such as Quantified Boolean Formulas
Summary:

- We presented a SAT-based, and a revised ASP-based approach for computing the contention inconsistency measure.
- Our experimental analysis showed that both approaches are superior to a naive baseline method.
- The ASP approach outperformed the SAT approach, as well as its predecessors from the literature (new state of the art).
Conclusion

Summary:
▶ We presented a SAT-based, and a revised ASP-based approach for computing the contention inconsistency measure
▶ Our experimental analysis showed that both approaches are superior to a naive baseline method
▶ The ASP approach outperformed the SAT approach, as well as its predecessors from the literature (new state of the art)

Future Work:
▶ SAT: use different SAT solvers, and/or different methods for generating cardinality constraints; exploit approaches to MaxSAT

Thank you for your attention!
Summary:

▶ We presented a SAT-based, and a revised ASP-based approach for computing the contention inconsistency measure

▶ Our experimental analysis showed that both approaches are superior to a naive baseline method

▶ The ASP approach outperformed the SAT approach, as well as its predecessors from the literature (new state of the art)

Future Work:

▶ SAT: use different SAT solvers, and/or different methods for generating cardinality constraints; exploit approaches to MaxSAT

▶ Consider measures of higher complexity
Summary:

- We presented a SAT-based, and a revised ASP-based approach for computing the contention inconsistency measure.
- Our experimental analysis showed that both approaches are superior to a naive baseline method.
- The ASP approach outperformed the SAT approach, as well as its predecessors from the literature (new state of the art).

Future Work:

- SAT: use different SAT solvers, and/or different methods for generating cardinality constraints; exploit approaches to MaxSAT.
- Consider measures of higher complexity.
- Explore other formalisms, such as *Quantified Boolean Formulas*. 
Conclusion

Summary:
- We presented a SAT-based, and a revised ASP-based approach for computing the contention inconsistency measure.
- Our experimental analysis showed that both approaches are superior to a naive baseline method.
- The ASP approach outperformed the SAT approach, as well as its predecessors from the literature (new state of the art).

Future Work:
- SAT: use different SAT solvers, and/or different methods for generating cardinality constraints; exploit approaches to MaxSAT.
- Consider measures of higher complexity.
- Explore other formalisms, such as Quantified Boolean Formulas.

Thank you for your attention!
We compare the SAT-based and ASP-based approach with a *naive baseline implementation*:

- The given knowledge base is first converted to CNF and checked for consistency
- If consistent: return 0
- Else: for each proposition $x$, remove each clause containing $x$ and check for consistency again
  - This is equivalent to setting $x$ to $b$
- If one of the new knowledge bases is consistent, return 1
- Otherwise: repeat the process with each pair of propositions, then with each triple, and so forth