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)

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)

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)

Motivation

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!

Motivation

- There exists a plethora of different inconsistency measures in the literature

Motivation

- There exists a plethora of different inconsistency measures in the literature
- Only few works consider the topic of inconsistency measurement from an *algorithmic* perspective

Motivation

- 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 *contension inconsistency measure*

Contributions

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

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)

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 $\mathbb{K}$ be the set of all (propositional) knowledge bases.
An *inconsistency measure* $\mathcal{I}$ is a function $\mathcal{I} : \mathbb{K} \to \mathbb{R}^{\infty}_{\geq 0}$ that satisfies $\mathcal{I}(\mathcal{K}) = 0$ iff $\mathcal{K}$ is consistent, for all $\mathcal{K} \in \mathbb{K}$.

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* ω^3 is a function that assigns one of the three truth values to each atom in a given knowledge base:

$$\omega^3 : \mathrm{At}(\mathcal{K}) \mapsto \{t, f, b\}$$

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* ω^3 is a function that assigns one of the three truth values to each atom in a given knowledge base:

$$\omega^3 : \mathrm{At}(\mathcal{K}) \mapsto \{t, f, b\}$$

- A three-valued *model* is an interpretation where each formula $\alpha \in \mathcal{K}$ is assigned either t or b.

Definition

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

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

Definition

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

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

We can divide the domain of an interpretation ω^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 ω^3 into two sets:

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

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 ω^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

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

We can divide the domain of an interpretation ω^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

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

Intuition

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

Intuition

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

Definition

$$\mathcal{I}_c(\mathcal{K}) = \min\{|\mathsf{Conflictbase}(\omega^3)| \mid \omega^3 \in \mathsf{Models}(\mathcal{K})\}.$$

Contention Inconsistency Measure

Intuition

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

Definition

$$\mathcal{I}_\mathrm{c}(\mathcal{K}) = \min\{|\mathsf{Conflictbase}(\omega^3)| \mid \omega^3 \in \mathsf{Models}(\mathcal{K})\}.$$

Example: $\mathcal{K}_1 = \{x \wedge y, \neg x, y \vee z\}$

Intuition

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

Definition

$$\mathcal{I}_c(\mathcal{K}) = \min\{|\mathsf{Conflictbase}(\omega^3)| \mid \omega^3 \in \mathsf{Models}(\mathcal{K})\}.$$

Example: $\mathcal{K}_1 = \{x \wedge y, \neg x, y \vee z\}$

- Let ω_1^3 be an interpretation with $\omega_1^3(y) = \omega_1^3(z) = t$ and $\omega_1^3(x) = b$

Intuition

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

Definition

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

Example: $\mathcal{K}_1 = \{x \wedge y, \neg x, y \vee z\}$

- Let ω_1^3 be an interpretation with $\omega_1^3(y) = \omega_1^3(z) = t$ and $\omega_1^3(x) = b$
- ω_1^3 is a model of $\mathcal{K}_1$

Intuition

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

Definition

$$\mathcal{I}_\mathrm{c}(\mathcal{K}) = \min\{|\mathsf{Conflictbase}(\omega^3)| \mid \omega^3 \in \mathsf{Models}(\mathcal{K})\}.$$

Example: $\mathcal{K}_1 = \{x \wedge y, \neg x, y \vee z\}$

- Let ω_1^3 be an interpretation with $\omega_1^3(y) = \omega_1^3(z) = t$ and $\omega_1^3(x) = b$
- ω_1^3 is a model of $\mathcal{K}_1$
- $\mathsf{Conflictbase}(\omega_1^3) = \{x\}$

Intuition

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

Definition

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

Example: $\mathcal{K}_1 = \{x \wedge y, \neg x, y \vee z\}$

- Let ω_1^3 be an interpretation with $\omega_1^3(y) = \omega_1^3(z) = t$ and $\omega_1^3(x) = b$
- ω_1^3 is a model of $\mathcal{K}_1$
- $\text{Conflictbase}(\omega_1^3) = \{x\}$
- $\mathcal{I}_c(\mathcal{K}_1) = |\text{Conflictbase}(\omega_1^3)| = |\{x\}| = 1$

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

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)

Satisfiability Problem

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 $\mathcal{I}_{\mathrm{c}}(\mathcal{K})$ wrt. a given knowledge base $\mathcal{K}$. ($\textsc{Value}_{\mathcal{I}_{\mathrm{c}}}$)

Algorithm Based on SAT

Goal

Find the value of $\mathcal{I}_\mathrm{c}(\mathcal{K})$ wrt. a given knowledge base $\mathcal{K}$. ($\textsc{Value}_{\mathcal{I}_\mathrm{c}}$)

- We cannot encode $\textsc{Value}_{\mathcal{I}_\mathrm{c}}$ directly in SAT
  - We encode the problem of deciding whether a given value u is an upper bound of $\mathcal{I}_\mathrm{c}(\mathcal{K})$ ($\textsc{Upper}_{\mathcal{I}_\mathrm{c}}$)

Goal

Find the value of $\mathcal{I}_{\mathrm{c}}(\mathcal{K})$ wrt. a given knowledge base $\mathcal{K}$. ($\textsc{Value}_{\mathcal{I}_{\mathrm{c}}}$)

- We cannot encode $\textsc{Value}_{\mathcal{I}_{\mathrm{c}}}$ directly in SAT
  - We encode the problem of deciding whether a given value u is an upper bound of $\mathcal{I}_{\mathrm{c}}(\mathcal{K})$ ($\textsc{Upper}_{\mathcal{I}_{\mathrm{c}}}$)
- Using a **binary search procedure** which includes iterative calls to a SAT solver, we can ultimately determine $\textsc{Value}_{\mathcal{I}_{\mathrm{c}}}$:

Algorithm Based on SAT

Goal

Find the value of $\mathcal{I}_{\mathrm{c}}(\mathcal{K})$ wrt. a given knowledge base $\mathcal{K}$. ($\textsc{Value}_{\mathcal{I}_{\mathrm{c}}}$)

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

Goal

Find the value of $\mathcal{I}_{\mathrm{c}}(\mathcal{K})$ wrt. a given knowledge base $\mathcal{K}$. ($\textsc{Value}_{\mathcal{I}_{\mathrm{c}}}$)

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

Algorithm Based on SAT

Goal

Find the value of $\mathcal{I}_{\mathrm{c}}(\mathcal{K})$ wrt. a given knowledge base $\mathcal{K}$. ($\textsc{Value}_{\mathcal{I}_{\mathrm{c}}}$)

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

Algorithm Based on SAT

Goal

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

- We cannot encode $\textsc{Value}_{\mathcal{I}_c}$ directly in SAT
  - We encode the problem of deciding whether a given value u is an upper bound of $\mathcal{I}_c(\mathcal{K})$ ($\textsc{Upper}_{\mathcal{I}_c}$)
- Using a **binary search procedure** which includes iterative calls to a SAT solver, we can ultimately determine $\textsc{Value}_{\mathcal{I}_c}$:
  - We start with $u = \lfloor |\mathsf{At}(\mathcal{K})|/2 \rfloor$ 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 $\log_2(|\mathsf{At}(\mathcal{K})|)$ calls[1], we know $\textsc{Value}_{\mathcal{I}_c}$

[1] $\mathsf{At}(\mathcal{K})$ refers to the signature size of $\mathcal{K}$

SAT encoding for $\textsc{Upper}_{\mathcal{I}_c}$:

SAT encoding for $\textsc{Upper}_{\mathcal{I}_c}$:

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

SAT encoding for $\textsc{Upper}_{\mathcal{I}_c}$:

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

$$(x_t \vee x_f \vee x_b) \wedge (\neg x_t \vee \neg x_f) \wedge (\neg x_t \vee \neg x_b) \wedge (\neg x_b \vee \neg x_f)$$

SAT encoding for $\textsc{Upper}_{\mathcal{I}_c}$:

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

$$(x_t \vee x_f \vee x_b) \wedge (\neg x_t \vee \neg x_f) \wedge (\neg x_t \vee \neg x_b) \wedge (\neg x_b \vee \neg x_f)$$

- For every (sub-)formula ϕ we introduce $v_\phi^t, v_\phi^b, v_\phi^f$

SAT encoding for $\textsc{Upper}_{\mathcal{I}_c}$:

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

$$(x_t \vee x_f \vee x_b) \wedge (\neg x_t \vee \neg x_f) \wedge (\neg x_t \vee \neg x_b) \wedge (\neg x_b \vee \neg x_f)$$

- For every (sub-)formula ϕ we introduce $v_\phi^t, v_\phi^b, v_\phi^f$
- We encode $\wedge$, $\vee$, and $\neg$ in Priest's three-valued logic

SAT encoding for $\textsc{Upper}_{\mathcal{I}_c}$:

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

$$(x_t \vee x_f \vee x_b) \wedge (\neg x_t \vee \neg x_f) \wedge (\neg x_t \vee \neg x_b) \wedge (\neg x_b \vee \neg x_f)$$

- For every (sub-)formula ϕ we introduce $v_\phi^t, v_\phi^b, v_\phi^f$
- We encode $\wedge$, $\vee$, and $\neg$ in Priest's three-valued logic
  - *Example:* A conjunction $\phi = \psi_1 \wedge \psi_2$ is true if both conjuncts are true:

$$v_\phi^t \leftrightarrow v_{\psi_1}^t \wedge v_{\psi_2}^t$$

SAT encoding for $\textsc{Upper}_{\mathcal{I}_c}$:

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

$$(x_t \vee x_f \vee x_b) \wedge (\neg x_t \vee \neg x_f) \wedge (\neg x_t \vee \neg x_b) \wedge (\neg x_b \vee \neg x_f)$$

- For every (sub-)formula ϕ we introduce $v_\phi^t, v_\phi^b, v_\phi^f$
- We encode $\wedge$, $\vee$, and $\neg$ in Priest's three-valued logic
  - *Example:* A conjunction $\phi = \psi_1 \wedge \psi_2$ is true if both conjuncts are true:

$$v_\phi^t \leftrightarrow v_{\psi_1}^t \wedge v_{\psi_2}^t$$

- If a formula ϕ consists of an individual atom x: $v_\phi^t \leftrightarrow x_t, v_\phi^b \leftrightarrow x_b, v_\phi^f \leftrightarrow x_f$

SAT encoding for $\textsc{Upper}_{\mathcal{I}_c}$:

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

$$(x_t \vee x_f \vee x_b) \wedge (\neg x_t \vee \neg x_f) \wedge (\neg x_t \vee \neg x_b) \wedge (\neg x_b \vee \neg x_f)$$

- For every (sub-)formula ϕ we introduce $v_\phi^t, v_\phi^b, v_\phi^f$
- We encode $\wedge$, $\vee$, and $\neg$ in Priest's three-valued logic
  - *Example:* A conjunction $\phi = \psi_1 \wedge \psi_2$ is true if both conjuncts are true:

$$v_\phi^t \leftrightarrow v_{\psi_1}^t \wedge v_{\psi_2}^t$$

- If a formula ϕ consists of an individual atom x: $v_\phi^t \leftrightarrow x_t, v_\phi^b \leftrightarrow x_b, v_\phi^f \leftrightarrow x_f$
- We ensure that each formula $\alpha \in \mathcal{K}$ evaluates to t or b: $v_\alpha^t \vee v_\alpha^b$

SAT encoding for $\textsc{Upper}_{\mathcal{I}_c}$:

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

$$(x_t \vee x_f \vee x_b) \wedge (\neg x_t \vee \neg x_f) \wedge (\neg x_t \vee \neg x_b) \wedge (\neg x_b \vee \neg x_f)$$

- For every (sub-)formula ϕ we introduce $v_\phi^t, v_\phi^b, v_\phi^f$
- We encode $\wedge$, $\vee$, and $\neg$ in Priest's three-valued logic
  - *Example:* A conjunction $\phi = \psi_1 \wedge \psi_2$ is true if both conjuncts are true:

$$v_\phi^t \leftrightarrow v_{\psi_1}^t \wedge v_{\psi_2}^t$$

- If a formula ϕ consists of an individual atom x: $v_\phi^t \leftrightarrow x_t, v_\phi^b \leftrightarrow x_b, v_\phi^f \leftrightarrow x_f$
- We ensure that each formula $\alpha \in \mathcal{K}$ evaluates to t or b: $v_\alpha^t \vee v_\alpha^b$
- Cardinality constraint representing that at most u of the b-atoms are true:

$$\texttt{at\_most\_u}(\mathsf{At}_b)$$

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

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

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 $\textsc{Value}_{\mathcal{I}_c}$ directly by using a *minimize statement*.

Observation

In ASP, we can encode $\textsc{Value}_{\mathcal{I}_c}$ 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 $\textsc{Value}_{\mathcal{I}_c}$ 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 $\textsc{Value}_{\mathcal{I}_c}$ 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

ASP encoding for $\textsc{Value}_{\mathcal{I}_c}$**:**

ASP encoding for $\textsc{Value}_{\mathcal{I}_c}$:

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

ASP encoding for $\textsc{Value}_{\mathcal{I}_c}$:

- Every atom $x \in \mathsf{At}(\mathcal{K})$ is represented as `atom(`x`)`, every formula $\alpha \in \mathcal{K}$ as `kbMember(`α`)`, and the truth values as `tv(`t`)`, `tv(`b`)`, `tv(`f`)`
- We represent conjunctions, disjunctions, negations, and formulas consisting of individual atoms as such

ASP encoding for $\textsc{Value}_{\mathcal{I}_c}$:

- Every atom $x \in \mathrm{At}(\mathcal{K})$ is represented as `atom(`x`)`, every formula $\alpha \in \mathcal{K}$ as `kbMember(`α`)`, 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 $\phi = \psi_1 \wedge \psi_2$ is represented as `conjunction(`ϕ`,`ψ_1`,`ψ_2`)`

ASP encoding for $\textsc{Value}_{\mathcal{I}_c}$:

- Every atom $x \in \mathsf{At}(\mathcal{K})$ is represented as `atom(`x`)`, every formula $\alpha \in \mathcal{K}$ as `kbMember(`α`)`, 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 $\phi = \psi_1 \wedge \psi_2$ is represented as `conjunction(`ϕ`,`ψ_1`,`ψ_2`)`
- "Guess" an interpretation: `1{truthValue(A,T): tv(T)}1:- atom(A).`

ASP encoding for $\textsc{Value}_{\mathcal{I}_c}$:

- Every atom $x \in \mathsf{At}(\mathcal{K})$ is represented as `atom(`x`)`, every formula $\alpha \in \mathcal{K}$ as `kbMember(`α`)`, 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 $\phi = \psi_1 \wedge \psi_2$ is represented as `conjunction(`ϕ`,`ψ_1`,`ψ_2`)`
- "Guess" an interpretation: `1{truthValue(A,T): tv(T)}1:- atom(A).`
- We encode $\wedge$, $\vee$, $\neg$, and formulas consisting of individual atoms:

ASP encoding for $\textsc{Value}_{\mathcal{I}_c}$:

- Every atom $x \in \mathsf{At}(\mathcal{K})$ is represented as `atom(`x`)`, every formula $\alpha \in \mathcal{K}$ as `kbMember(`α`)`, 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 $\phi = \psi_1 \wedge \psi_2$ is represented as `conjunction(`ϕ`,`ψ_1`,`ψ_2`)`
- "Guess" an interpretation: `1{truthValue(A,T): tv(T)}1:- atom(A).`
- We encode $\wedge$, $\vee$, $\neg$, and formulas consisting of individual atoms:
  - *Example:* A conjunction $\phi = \psi_1 \wedge \psi_2$ is true if both conjuncts are true:

```
truthValue(F,t) :- conjunction(F,G,H),
                   truthValue(G,t), truthValue(H,t).
```

ASP encoding for $\textsc{Value}_{\mathcal{I}_c}$:

- Every atom $x \in \mathsf{At}(\mathcal{K})$ is represented as `atom(`x`)`, every formula $\alpha \in \mathcal{K}$ as `kbMember(`α`)`, 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 $\phi = \psi_1 \wedge \psi_2$ is represented as `conjunction(`ϕ`,`ψ_1`,`ψ_2`)`
- "Guess" an interpretation: `1{truthValue(A,T): tv(T)}1:- atom(A).`
- We encode $\wedge$, $\vee$, $\neg$, and formulas consisting of individual atoms:
  - *Example:* A conjunction $\phi = \psi_1 \wedge \psi_2$ is true if both conjuncts are true:

```
truthValue(F,t) :- conjunction(F,G,H),
                   truthValue(G,t), truthValue(H,t).
```

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

ASP encoding for $\textsc{Value}_{\mathcal{I}_c}$:

- Every atom $x \in \mathsf{At}(\mathcal{K})$ is represented as `atom(`x`)`, every formula $\alpha \in \mathcal{K}$ as `kbMember(`α`)`, 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 $\phi = \psi_1 \wedge \psi_2$ is represented as `conjunction(`ϕ`,`ψ_1`,`ψ_2`)`
- "Guess" an interpretation: `1{truthValue(A,T): tv(T)}1:- atom(A).`
- We encode $\wedge$, $\vee$, $\neg$, and formulas consisting of individual atoms:
  - *Example:* A conjunction $\phi = \psi_1 \wedge \psi_2$ is true if both conjuncts are true:

```
truthValue(F,t) :- conjunction(F,G,H),
                   truthValue(G,t), truthValue(H,t).
```

- Every $\alpha \in \mathcal{K}$ must evaluate to t or b: `:- truthValue(F,f), kbMember(F).`
- Minimize statement: `#minimize{1,A: truthValue(A,b), 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

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

[2] http://tweetyproject.org/api/1.14/net/sf/tweety/logics/pl/util/SyntacticRandomSampler.html

Implementation details:

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/ContensionInconsistencyMeasure.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

Implementation details:

- SAT-based and ASP-based approach are implemented in C++
- Naive method: provided by TweetyProject[3] (Java)
- SAT solver: CaDiCal sc2021
- ASP solver: Clingo 5.5.1

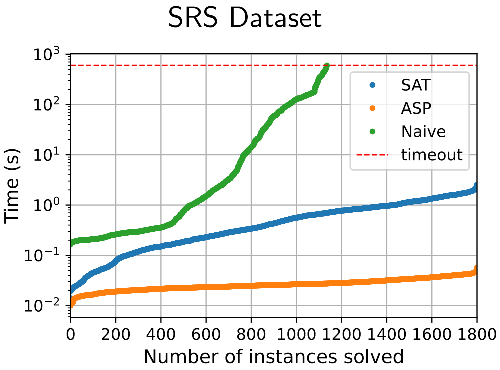
[3] http://tweetyproject.org/api/1.14/net/sf/tweety/logics/pl/analysis/ContensionInconsistencyMeasure.html

Experiment

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

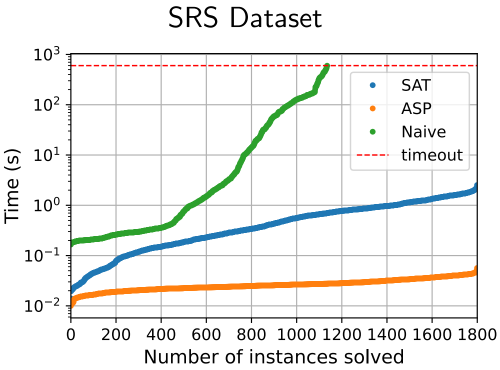
Experiment

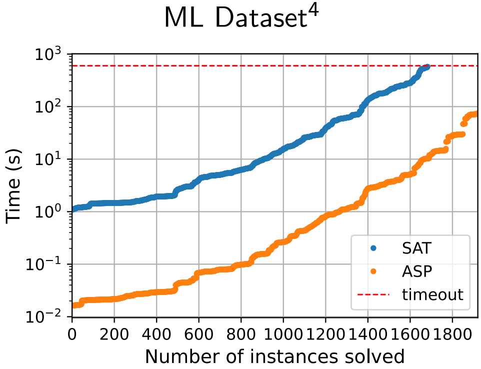
As a first step, we measured the runtime of each approach wrt. each knowledge base.



Experiment

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





[4] Note that we excluded the naive method here.

Question

Why did the ASP approach outperform the SAT approach?

Question

Why did the ASP approach outperform the SAT approach?

Experiment

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

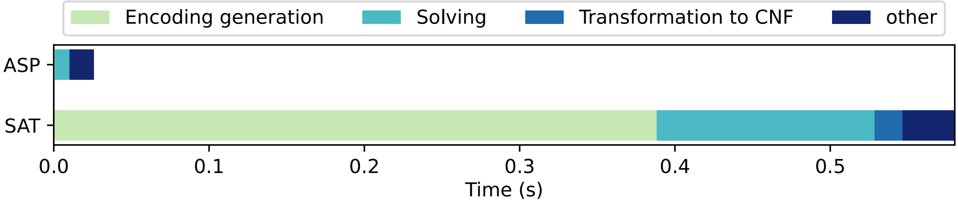
Results

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?

Results

Question

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

Experiment

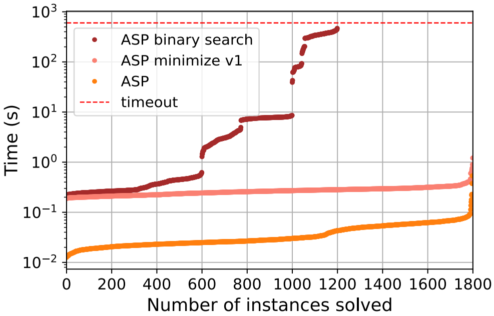
We compare the two previous ASP approaches with the newly proposed revision.

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

Summary:

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

Summary:

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

Conclusion

Summary:

- We presented a SAT-based, and a revised ASP-based approach for computing the contension 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 contension 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

Summary:

- We presented a SAT-based, and a revised ASP-based approach for computing the contension 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

Conclusion

Summary:

- We presented a SAT-based, and a revised ASP-based approach for computing the contension 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 contension 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!

Baseline Implementation

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