

DEPTH-BOUNDED EPISTEMIC PLANNING

Thomas Bolander

Denmark Technical University

Alessandro Burigana

Free University of Bozen-Bolzano

Marco Montali

Free University of Bozen-Bolzano

November 14th, 2025

22nd International Conference on Principles
of Knowledge Representation and Reasoning

Melbourne, Victoria, Australia

What Does Make a Problem *Hard*?

Different types of problems, **different sources of complexity**:

What Does Make a Problem *Hard*?

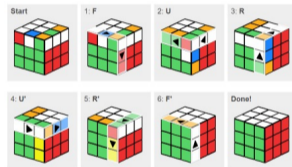
Different types of problems, **different sources of complexity**:



What Does Make a Problem *Hard*?

Different types of problems, **different sources of complexity**:

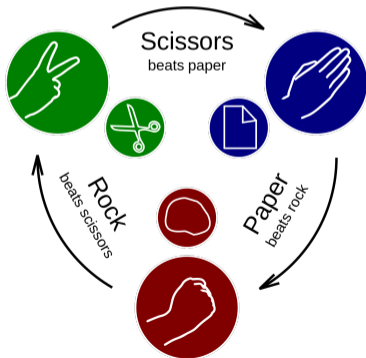
1 Combinatorial reasoning.



What Does Make a Problem *Hard*?

Different types of problems, **different sources of complexity**:

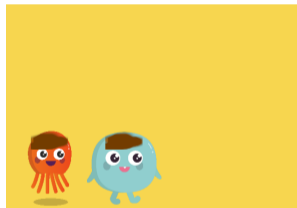
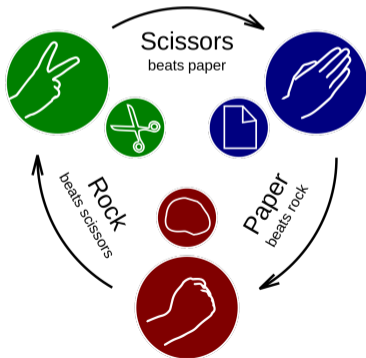
1 Combinatorial reasoning.



What Does Make a Problem *Hard*?

Different types of problems, **different sources of complexity**:

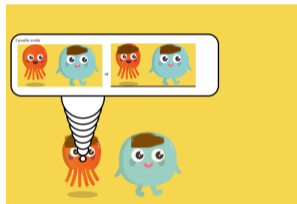
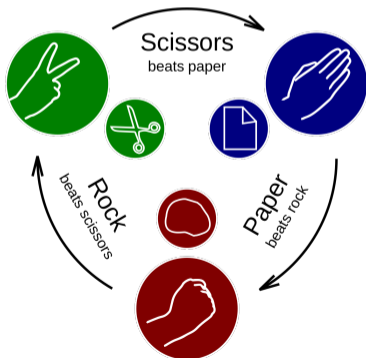
1 Combinatorial reasoning.



What Does Make a Problem *Hard*?

Different types of problems, **different sources of complexity**:

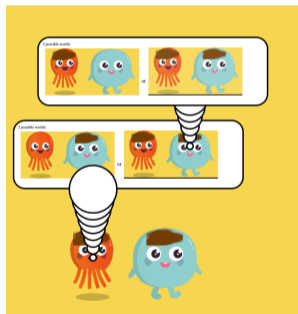
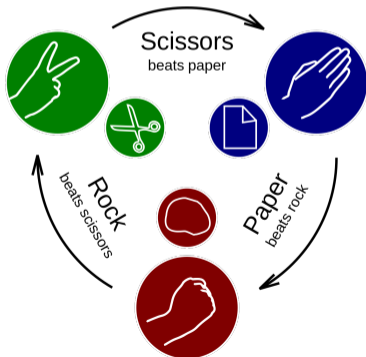
1 Combinatorial reasoning.



What Does Make a Problem *Hard*?

Different types of problems, **different sources of complexity**:

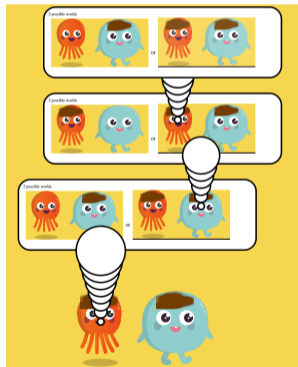
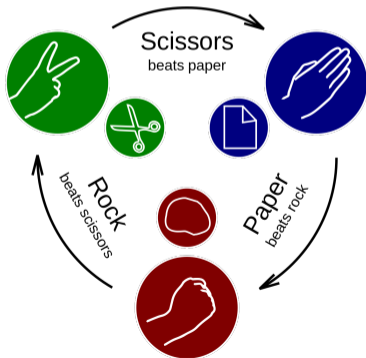
1 Combinatorial reasoning.



What Does Make a Problem *Hard*?

Different types of problems, **different sources of complexity**:

- 1 **Combinatorial** reasoning.
- 2 Reasoning about **knowledge/belief**.



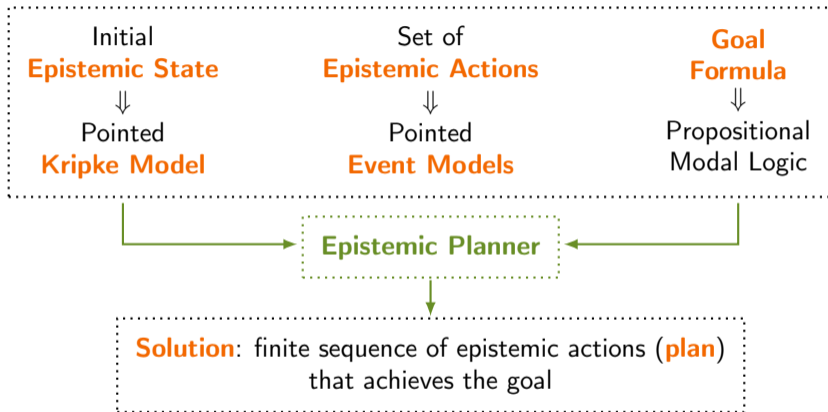
Epistemic planning: enrichment of **classical planning** with notions of **knowledge** and **belief**.

- **Epistemic states** represent what the agents **know/believe** about the world and others' perspective of the world.
- **Epistemic actions** can change both the world and the **knowledge/belief** of the agents.
- Agents have to reason about each others' (higher-order) **knowledge/beliefs** to reach a shared **goal**.

DEL-based Epistemic Planning

(Epistemic)
Planning Task

Dynamic
Epistemic Logic



Language of Multi-Agent Epistemic Logic

Let P be a finite set of **propositional atoms** and $Ag = \{1, \dots, n\}$ a finite set of **agents**. The **language $\mathcal{L}_{P,Ag}$ of Multi-Agent Epistemic Logic** is given by the BNF:

Definition (Language of Epistemic Logic)

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \Box_i \phi,$$

- Operator \Box_i : depending on the context, describes what agent i **knows** or **believes**.
- Dual operator \Diamond_i ($\equiv \neg\Box_i\neg$): describes what agent i **considers to be possible** or **compatible**.

The Consecutive Numbers Puzzle

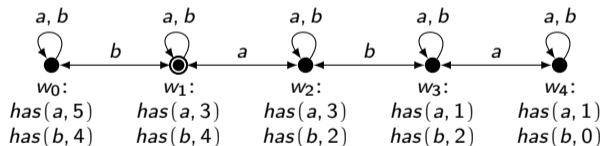
Example

- Two agents, **Anne** and **Bob**, are assigned a number between 0 and 5.
- The numbers are whispered to them, so they only get to know their number.
- It is common knowledge of **Anne** and **Bob** that the numbers are **consecutive**.
- Suppose **Anne** has number 3 and **Bob** has 4 (world w_1).

The Consecutive Numbers Puzzle

Example

- Two agents, **Anne** and **Bob**, are assigned a number between 0 and 5.
- The numbers are whispered to them, so they only get to know their number.
- It is common knowledge of **Anne** and **Bob** that the numbers are **consecutive**.
- Suppose **Anne** has number 3 and **Bob** has 4 (world w_1).



💡 When does i know/believe that ϕ ?

$(M, w) \models \Box_i \phi$ iff ϕ holds in all worlds v accessible by i from w .

Epistemic Actions and Product Update

💡 In a Nutshell: Epistemic Actions

An **epistemic action** is a Kripke-like structure built on a finite set of **possible events**:

- **Events** represent different **possible outcomes of the action**.
- Each event has its own **precondition** and **effects** (postconditions).
- **Accessibility relations** specify the perspectives of agents on which events take place.

💡 In a Nutshell: Product Update

The **product update** $s \otimes \alpha$ formalizes the application of α in s :

- 💡 **Cross product**: we apply each event to all worlds where the event is applicable.
- 💡 **Postconditions** specify how **atoms are updated**.
- 💡 New perspectives of agents: combine what agents **previously knew** and what they **observe during the action**.

The Consecutive Number Puzzle (Cont.)

Continuing our example:

- Agents can publicly announce that they don't know their partner's number.
- The goal is for Bob to know that Anne knows that he has 4: $\Box_b \Box_a has(b, 4)$.

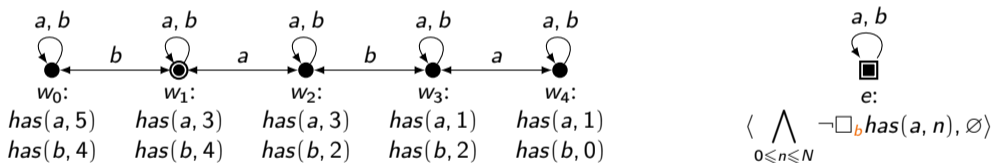
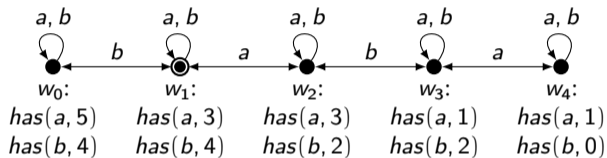
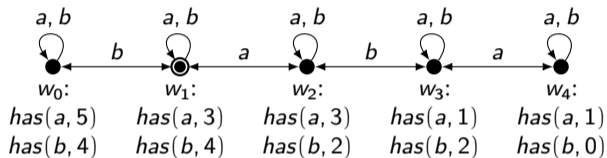


Figure: Initial state (left) and public announcement of the fact that “Bob doesn't know Anne's number” (right).

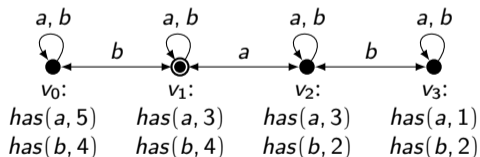
The Consecutive Number Puzzle (Cont.)



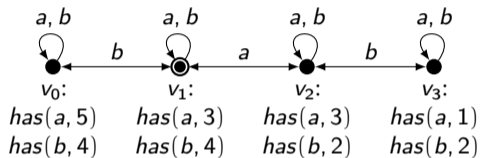
The Consecutive Number Puzzle (Cont.)



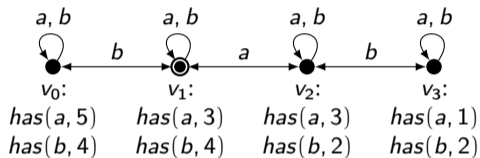
\otimes



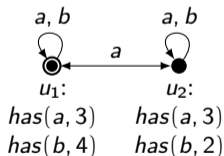
The Consecutive Number Puzzle (Cont.)



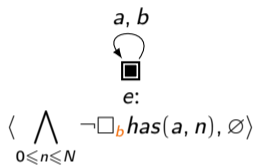
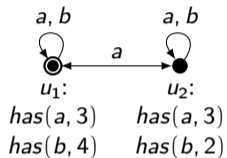
The Consecutive Number Puzzle (Cont.)



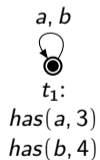
\otimes



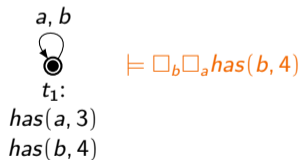
The Consecutive Number Puzzle (Cont.)



The Consecutive Number Puzzle (Cont.)



The Consecutive Number Puzzle (Cont.)



The Consecutive Number Puzzle (Cont.)

A **solution** to the puzzle is the following **plan**:

- 1 Bob announces that he **doesn't know Anne's** number;
- 2 Anne announces that she **doesn't know Bob's** number;
- 3 Bob announces that he **doesn't know Anne's** number.

BISIMULATION CONTRACTIONS

A Simple Breadth-First Search

BFS

```
1: function BFS(( $s_0$ ,  $Act$ ,  $\phi_g$ ))  
2:    $frontier \leftarrow \langle s_0 \rangle$   
3:    $visited \leftarrow \emptyset$   
4:   while  $\neg frontier.empty()$  do  
5:      $s \leftarrow frontier.pop()$   
6:      $visited.push(s)$   
7:     if  $s \models \phi_g$  then return plan to  $s$   
8:     for all  $\alpha \in Act$  applicable in  $s$  do  
9:        $s' \leftarrow s \otimes \alpha$   
10:      If  $s'$  is not visited, push it to  $frontier$   
11: return fail
```

A Main Issue in Epistemic Planning

Epistemic models can get **very large**:

- **Higher uncertainty** of agents means **bigger models**.
- Worst-case **exponential blowup** of size of states after product update.

A common approach to solve this: **bisimulation contractions**.

- 💡 Identify a notion of modal equivalence of epistemic states: **bisimulations**.
- 💡 Bisimilar states **satisfy the same formulas** [BRV01].
- 💡 Product update **preserves bisimilarity** [DHK07].
- 💡 **Bisimulation contraction** of a state s : smallest state bisimilar to s .

💡 **Replace any visited state with its contraction**

Smaller states mean lighter computation and faster performances.

Bisimulation Contractions

💡 **Bisimulations** capture epistemic states that “behave” in the same way:

Proposition ([BRV01])

Two states are **bisimilar** iff they **satisfy the same formulas in $\mathcal{L}_{P,Ag}$** .

Example (Two bisimilar states)



Definition (Bisimulation Contraction)

The **(bisimulation) contraction** of s is the **quotient structure $[s]_{\approx}$** of s induced by the bisimilarity relation.

Proposition ([BRV01])

$[s]_{\approx}$ is a **minimal state** (smallest number of worlds and edges) **bisimilar** to s .

BFS with Bisimulation Contractions

BFS

```
1: function BFS(( $s_0$ ,  $Act$ ,  $\phi_g$ ))
2:    $frontier \leftarrow \langle [s_0]_{\approx} \rangle$ 
3:    $visited \leftarrow \emptyset$ 
4:   while  $\neg frontier.empty()$  do
5:      $s \leftarrow frontier.pop()$ 
6:      $visited.push(s)$ 
7:     if  $s \models \phi_g$  then return plan to  $s$ 
8:     for all  $\alpha \in Act$  applicable in  $s$  do
9:        $s' \leftarrow [s \otimes \alpha]_{\approx}$ 
10:      If  $s'$  is not visited, push it to  $frontier$ 
11:  return fail
```

💡 Key Idea

We can **replace any visited state s with** its bisimulation contraction $[s]_{\approx}$.

Proposition ([BRV01])

Two states are **bisimilar** iff they **satisfy the same formulas in $\mathcal{L}_{P,Ag}$** .

Proposition ([DHK07])

If $s \approx s'$ and α is applicable in both, then $s \otimes \alpha \approx s' \otimes \alpha$.

DEPTH-BOUNDED EPISTEMIC PLANNING

Depth-Bounded Epistemic Planning

In DEL-based epistemic planning agents can reason unboundedly about each other's knowledge.

- This leads to **undecidability** of the plan existence problem.
- Often unrealistic in many practical scenarios.

Depth-Bounded Epistemic Planning

In DEL-based epistemic planning agents can reason unboundedly about each other's knowledge.

- This leads to **undecidability** of the plan existence problem.
- Often unrealistic in many practical scenarios.

What if we **restricted the reasoning depth** of the planning agent to some bound b ?

- 💡 Reduce the size of epistemic states: **bounded bisimulation contractions**.
- 💡 Look for plans requiring the lowest bound: **iterative bound-deepening search**.

Bounded Bisimulations

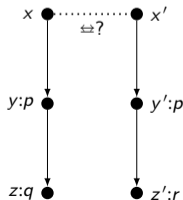
💡 In a Nutshell: b -bisimilarity

- $x \Leftrightarrow_0 x'$ iff they agree on all propositional atoms.
- $x \Leftrightarrow_{b+1} x'$ iff $x \xrightarrow{i} y$ implies $x' \xrightarrow{i} y'$ and $x' \Leftrightarrow_b y'$ for some y' (and vice versa).

Proposition ([BRV01])

Two states are **b -bisimilar** iff they **satisfy the same formulas up to modal depth b** .

Example (Are x and x' bisimilar?)



Bounded Bisimulations

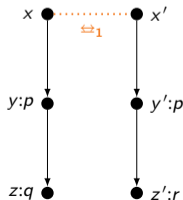
💡 In a Nutshell: b -bisimilarity

- $x \Leftrightarrow_0 x'$ iff they agree on all propositional atoms.
- $x \Leftrightarrow_{b+1} x'$ iff $x \xrightarrow{i} y$ implies $x' \xrightarrow{i} y'$ and $x' \Leftrightarrow_b y'$ for some y' (and vice versa).

Proposition ([BRV01])

Two states are **b -bisimilar** iff they **satisfy the same formulas up to modal depth b** .

Example (Are x and x' bisimilar? No, but they are 1-bisimilar!)

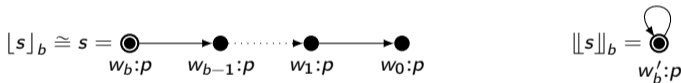


Rooted b -Contractions

Early definitions of bounded contractions in the literature did not behave as expected:

- **Standard b -contractions** [YWL13] do not generally yield minimal b -bisimilar models.
- Bolander and Burigana improved the definition: **rooted b -contractions** [BB24].

Example (Standard (left) and rooted (right) b -contraction of a chain model)



Theorem ([BB24])

The rooted b -contraction $\llbracket s \rrbracket_b$ is a **minimal state b -bisimilar** to s .

Canonical b -Contractions

A problem remains: **rooted b -contractions** of b -bisimilar states may be **non-isomorphic!**

→ Checking for visited states is **inefficient**.

Canonical b -Contractions

A problem remains: **rooted b -contractions** of b -bisimilar states may be **non-isomorphic**!

→ Checking for visited states is **inefficient**.



Improved definition called **canonical b -contractions**, based on the following notion:

Definition (h -signature)

The **h -signature** of a world w is a pair $\sigma_h(w) = (L(w), \Sigma_h(w))$, where:

$$\Sigma_h(w, i) = \begin{cases} \emptyset & \text{if } h = 0 \\ \{\sigma_{h-1}(v) \mid wR_i v\} & \text{otherwise} \end{cases}$$

Canonical b -Contractions

A problem remains: rooted b -contractions of b -bisimilar states may be **non-isomorphic!**

→ Checking for visited states is **inefficient**.



Improved definition called **canonical b -contractions**, based on the following notion:

Definition (h -signature)

The **h -signature** of a world w is a pair $\sigma_h(w) = (L(w), \Sigma_h(w))$, where:

$$\Sigma_h(w, i) = \begin{cases} \emptyset & \text{if } h = 0 \\ \{\sigma_{h-1}(v) \mid wR_i v\} & \text{otherwise} \end{cases}$$

Proposition (Unique identifiers of h -bisimilar worlds)

Two worlds are **h -bisimilar** iff they have the **same h -signature**.

Theorem (Identity [BBM25])

The canonical b -contractions of b -bisimilar states are **identical**.

From Breadth-First Search...

Consider our BFS with standard bisimulation contractions and check for visited states:

BFS

```
1: function BFS(( $s_0$ ,  $Act$ ,  $\phi_g$ ))
2:    $frontier \leftarrow \langle \lfloor s_0 \rfloor_{\Leftrightarrow} \rangle$ 
3:    $visited \leftarrow \emptyset$ 
4:   while  $\neg frontier.empty()$  do
5:      $s \leftarrow frontier.pop()$ 
6:      $visited.push(s)$ 
7:     if  $s \models \phi_g$  then return plan to  $s$ 
8:     for all  $\alpha \in Act$  applicable in  $s$  do
9:        $s' \leftarrow \lfloor s \otimes \alpha \rfloor_{\Leftrightarrow}$ 
10:      If  $s'$  is not visited, push it to  $frontier$ 
11:   return fail
```

Proposition ([BRV01])

Two states are *bisimilar* iff they *satisfy the same formulas in* $\mathcal{L}_{P,Ag}$.

Proposition ([DHK07])

If $s \Leftrightarrow s'$ and α is applicable in both, then $s \otimes \alpha \Leftrightarrow s' \otimes \alpha$.

...To Bounded Search

Let b_0 be the **reasoning depth bound** of the planning agent (*i.e.*, the agent can reason to formulas with **modal depth at most** b_0).

BoundedSearch

```
1: function BoundedSearch( $(s_0, Act, \phi_g), b_0$ )
2:    $frontier \leftarrow \langle \lfloor s_0 \rfloor_{\Leftrightarrow} \rangle$ 
3:    $visited \leftarrow \emptyset$ 
4:   while  $\neg frontier.empty()$  do
5:      $s \leftarrow frontier.pop()$ 
6:      $visited.push(s)$ 
7:     if  $s \models \phi_g$  then return plan to  $s$ 
8:     for all  $\alpha \in Act$  applicable in  $s$  do
9:        $s' \leftarrow \lfloor s \otimes \alpha \rfloor_{\Leftrightarrow}$ 
10:      If  $s'$  is not visited, push it to  $frontier$ 
11:   return fail
```

...To Bounded Search

Let b_0 be the **reasoning depth bound** of the planning agent (i.e., the agent can reason to formulas with **modal depth at most** b_0).

BoundedSearch

```
1: function BoundedSearch( $(s_0, Act, \phi_g), b_0$ )
2:    $frontier \leftarrow \langle \llbracket s_0 \rrbracket_{b_0}^* \rangle$ 
3:    $visited \leftarrow \emptyset$ 
4:   while  $\neg frontier.empty()$  do
5:      $s \leftarrow frontier.pop()$ 
6:      $visited.push(s)$ 
7:     if  $s \models \phi_g$  then return plan to  $s$ 
8:     for all  $\alpha \in Act$  applicable in  $s$  do
9:        $s' \leftarrow \lfloor s \otimes \alpha \rfloor_{\Leftrightarrow}$ 
10:      If  $s'$  is not visited, push it to  $frontier$ 
11:   return fail
```

Proposition ([BRV01])

Two states are **b -bisimilar** iff they **satisfy the same formulas up to modal depth b** .

...To Bounded Search

Let b_0 be the **reasoning depth bound** of the planning agent (i.e., the agent can reason to formulas with **modal depth at most b_0**).

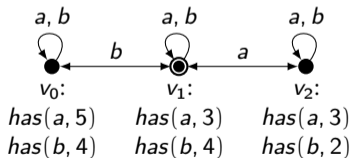
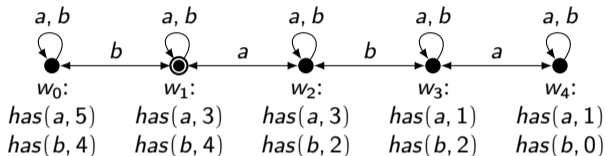
BoundedSearch

```
1: function BoundedSearch( $((s_0, Act, \phi_g), b_0)$ )
2:    $frontier \leftarrow \langle \llbracket s_0 \rrbracket_{b_0}^* \rangle$ 
3:    $visited \leftarrow \emptyset$ 
4:   while  $\neg frontier.empty()$  do
5:      $s \leftarrow frontier.pop()$ 
6:      $visited.push(s)$ 
7:     if  $s \models \phi_g$  then return plan to  $s$ 
8:     for all  $\alpha \in Act$  applicable in  $s$  do
9:        $s' \leftarrow \llbracket s \otimes \alpha \rrbracket_{b_0}^* \quad (?)$ 
10:      If  $s'$  is not visited, push it to  $frontier$ 
11:   return fail
```

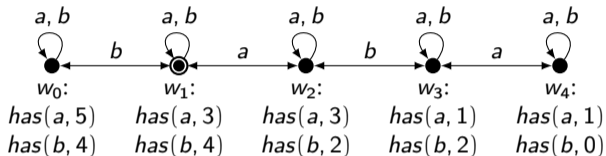
Proposition ([BRV01])

Two states are **b -bisimilar** iff they **satisfy the same formulas up to modal depth b** .

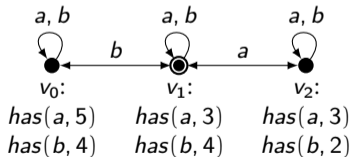
Does Product Update Preserve b -Bisimilarity?



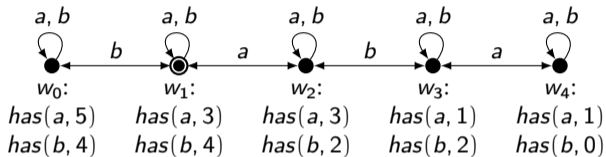
Does Product Update Preserve b -Bisimilarity?



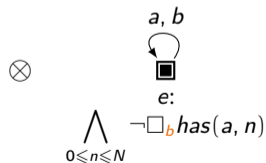
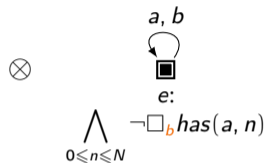
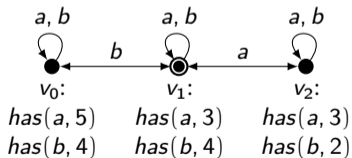
\Leftrightarrow_1



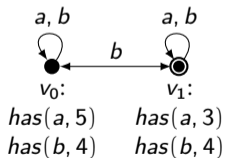
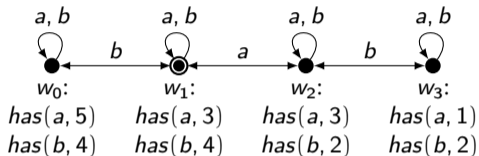
Does Product Update Preserve b -Bisimilarity?



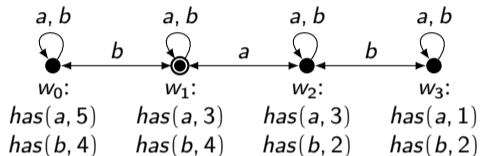
\Leftrightarrow_1



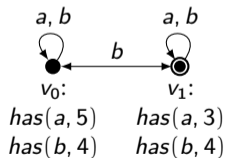
Does Product Update Preserve b -Bisimilarity?



Does Product Update Preserve b -Bisimilarity?



$\not\equiv_1$



...To Bounded Search

Let b_0 be the **reasoning depth bound** of the planning agent (i.e., the agent can reason to formulas with **modal depth at most** b_0).

BoundedSearch

```
1: function BoundedSearch( $(s_0, Act, \phi_g), b_0$ )
2:    $frontier \leftarrow \langle \llbracket s_0 \rrbracket_{b_0}^* \rangle$ 
3:    $visited \leftarrow \emptyset$ 
4:   while  $\neg frontier.empty()$  do
5:      $s \leftarrow frontier.pop()$ 
6:      $visited.push(s)$ 
7:     if  $s \models \phi_g$  then return plan to  $s$ 
8:     for all  $\alpha \in Act$  applicable in  $s$  do
9:        $s' \leftarrow \llbracket s \otimes \alpha \rrbracket_{b_0}^* \quad (?)$ 
10:      If  $s'$  is not visited, push it to  $frontier$ 
11:   return fail
```

Proposition ([BL22])

Let $s \rightleftharpoons_b s'$ and let α be an action with $md(\alpha) \leq b$. Then, $s \otimes \alpha \rightleftharpoons_{b-md(\alpha)} s' \otimes \alpha$.

Where $md(\alpha)$ denotes the **maximal modal depth** of all pre- and postconditions in α .

...To Bounded Search

Let b_0 be the **reasoning depth bound** of the planning agent (i.e., the agent can reason to formulas with **modal depth at most** b_0).

BoundedSearch

```
1: function BoundedSearch( $(s_0, Act, \phi_g), b_0$ )
2:    $frontier \leftarrow \langle \llbracket s_0 \rrbracket_{b_0}^* \rangle$ 
3:    $visited \leftarrow \emptyset$ 
4:   while  $\neg frontier.empty()$  do
5:      $s \leftarrow frontier.pop()$ 
6:      $visited.push(s)$ 
7:     if  $s \models \phi_g$  then return plan to  $s$ 
8:     for all  $\alpha \in Act$  applicable in  $s$  do
9:        $s' \leftarrow \llbracket s \otimes \alpha \rrbracket_{b_0}^*$ 
10:      If  $s'$  is not visited, push it to  $frontier$ 
11:   return fail
```

Proposition ([BL22])

Let $s \rightleftharpoons_b s'$ and let α be an action with $md(\alpha) \leq b$. Then, $s \otimes \alpha \rightleftharpoons_{b-md(\alpha)} s' \otimes \alpha$.

Where $md(\alpha)$ denotes the **maximal modal depth** of all pre- and postconditions in α .

→ We need to **update** the bound value after an update.

Updating Bounds Value After Updates

We let a **node** of the search space be a pair $n = (s, b)$, where:

- 1 s is the **state** of n (denoted $n.state$).
- 2 b is the **(depth) bound** (denoted $n.bound$) \rightarrow **maximum modal depth** of formulas we can safely evaluate in s .

Updating Bounds Value After Updates

We let a **node** of the search space be a pair $n = (s, b)$, where:

- 1 s is the **state** of n (denoted $n.state$).
- 2 b is the **(depth) bound** (denoted $n.bound$) \rightarrow **maximum modal depth** of formulas we can safely evaluate in s .

In general, s will be a **b -contracted state**: **approximation to the “true” state**.

\rightarrow We are always guaranteed that s is **at least b -bisimilar to the true state**.

Putting Everything Together

BoundedSearch

```
1: function BoundedSearch( $(s_0, Act, \phi_g), b_0$ )
2:    $frontier \leftarrow \langle \llbracket s_0 \rrbracket_{b_0}^*, b_0 \rangle$ 
3:    $visited \leftarrow \emptyset$ 
4:   while  $\neg frontier.empty()$  do
5:      $(s, b) \leftarrow frontier.pop()$ 
6:      $visited.push(s)$ 
7:     if  $s \models \phi_g$  then return plan to  $s$ 
8:     for all  $\alpha \in Act \mid b \geq md(\alpha)$  do
9:       if  $\alpha$  is applicable in  $s$  then
10:         $s' \leftarrow \llbracket s \otimes \alpha \rrbracket_{b-md(\alpha)}^*$ 
11:         $n' \leftarrow (s', b - md(\alpha))$ 
12:        If  $s'$  is not visited, push  $n'$  to  $frontier$ 
13:   return fail
```

Proposition ([BRV01])

Two states are *b-bisimilar* iff they *satisfy the same formulas up to modal depth b*.

Proposition ([BL22])

Let $s \rightleftharpoons_b s'$ and let α be an action with $md(\alpha) \leq b$. Then, $s \otimes \alpha \rightleftharpoons_{b-md(\alpha)} s' \otimes \alpha$.

Putting Everything Together

BoundedSearch

```
1: function BoundedSearch( $(s_0, Act, \phi_g), b_0$ )
2:    $frontier \leftarrow \langle (\llbracket s_0 \rrbracket_{b_0}^*, b_0) \rangle$ 
3:    $visited \leftarrow \emptyset$ 
4:   while  $\neg frontier.empty()$  do
5:      $(s, b) \leftarrow frontier.pop()$ 
6:      $visited.push(s)$ 
7:     if  $s \models \phi_g$  then return plan to  $s$ 
8:     for all  $\alpha \in Act \mid b \geq md(\alpha) + md(\phi_g)$  do
9:       if  $\alpha$  is applicable in  $s$  then
10:         $s' \leftarrow \llbracket s \otimes \alpha \rrbracket_{b-md(\alpha)}^*$ 
11:         $n' \leftarrow (s', b - md(\alpha))$ 
12:        if  $s'$  is not visited, push  $n'$  to  $frontier$ 
13:   return fail
```

Proposition ([BRV01])

Two states are *b-bisimilar* iff they *satisfy the same formulas up to modal depth b*.

Proposition ([BL22])

Let $s \Leftrightarrow_b s'$ and let α be an action with $md(\alpha) \leq b$. Then, $s \otimes \alpha \Leftrightarrow_{b-md(\alpha)} s' \otimes \alpha$.

Putting Everything Together

BoundedSearch

```
1: function BoundedSearch( $(s_0, Act, \phi_g), b_0$ )
2:   frontier  $\leftarrow \langle (\llbracket s_0 \rrbracket_{b_0}^*, b_0) \rangle$ 
3:   visited  $\leftarrow \emptyset$ 
4:   while  $\neg \text{frontier.empty}()$  do
5:      $(s, b) \leftarrow \text{frontier.pop}()$ 
6:     visited.push( $s$ )
7:     if  $s \models \phi_g$  then return plan to  $s$ 
8:     for all  $\alpha \in Act \mid b \geq md(\alpha) + md(\phi_g)$  do
9:       if  $\alpha$  is applicable in  $s$  then
10:         $s' \leftarrow \llbracket s \otimes \alpha \rrbracket_{b-md(\alpha)}^*$ 
11:         $n' \leftarrow (s', b - md(\alpha))$ 
12:        if  $s' \notin \text{visited}$  then frontier.push( $n'$ )
13:   return fail
```

Proposition ([BRV01])

Two states are *b-bisimilar* iff they *satisfy the same formulas up to modal depth b*.

Proposition ([BL22])

Let $s \rightleftharpoons_b s'$ and let α be an action with $md(\alpha) \leq b$. Then, $s \otimes \alpha \rightleftharpoons_{b-md(\alpha)} s' \otimes \alpha$.

Theorem ([BBM25])

The canonical b -contractions of b -bisimilar states are *identical*.

Iterative Bound-Deepening Search

Iterative Bound-Deepening Search

```
1: function IBDS( $T = (s_0, Act, \phi_g)$ )  
2:   for  $b \leftarrow md(\phi_g)$  to  $\infty$  do  
3:      $\pi \leftarrow$  BoundedSearch( $T, b$ )  
4:     if  $\pi \neq fail$  then return  $\pi$ 
```

We call **BoundedSearch** over increasing values of b :

- If $b < md(\phi_g)$, then the **bound is too low** to safely evaluate the goal formula.
- So initially we let $b = md(\phi_g)$.
- If no goal is found with bound b , we **increment the bound and try again**.

Improving Bounded Search

In a node $n = (s, b)$, the state s can be considered as an **approximation to modal depth b** of some “true state” t (namely, we are guaranteed that $s \rightleftharpoons_b t$). However:

Improving Bounded Search

In a node $n = (s, b)$, the state s can be considered as an **approximation to modal depth b** of some “true state” t (namely, we are guaranteed that $s \rightleftharpoons_b t$). However:

- In general it could be that $s \not\rightleftharpoons t$!

Improving Bounded Search

In a node $n = (s, b)$, the state s can be considered as an **approximation to modal depth b** of some “true state” t (namely, we are guaranteed that $s \rightleftharpoons_b t$). However:

- In general it could be that $s \not\rightleftharpoons t$!
- In this case, when we update s with an action α , we don't have to decrease the bound.
 - Recall that **bisimilarity is preserved** after product update!

Improving Bounded Search

In a node $n = (s, b)$, the state s can be considered as an **approximation to modal depth b** of some “true state” t (namely, we are guaranteed that $s \rightleftharpoons_b t$). However:

- In general it could be that $s \not\rightleftharpoons t$!
- In this case, when we update s with an action α , we don't have to decrease the bound.
 - Recall that **bisimilarity is preserved** after product update!

We can use this idea to include the following **optimizations** in BoundedSearch:

- We add a **third parameter** called *is_bisim* to our nodes, representing **whether the state of a node is bisimilar to its corresponding true state**.
- Depending on whether *is_bisim* holds, we **update a node with the appropriate bound value**.

Improving Bounded Search

In a node $n = (s, b)$, the state s can be considered as an **approximation to modal depth b** of some “true state” t (namely, we are guaranteed that $s \rightleftharpoons_b t$). However:

- In general it could be that $s \not\rightleftharpoons t$!
- In this case, when we update s with an action α , we don't have to decrease the bound.
 - Recall that **bisimilarity is preserved** after product update!

We can use this idea to include the following **optimizations** in BoundedSearch:

- We add a **third parameter** called *is_bisim* to our nodes, representing **whether the state of a node is bisimilar to its corresponding true state**.
- Depending on whether *is_bisim* holds, we **update a node with the appropriate bound value**.
- Across different iterations of IBDS, we **preserve all nodes having *is_bisim* true**.
 - They would otherwise be **recomputed** in the next iteration!

Soundness, Completeness, Complexity

Let $T = (s_0, Act, \phi_g)$ be a planning task and let $b \geq md(\phi_g)$ be a constant.

Theorem (Soundness)

If **BoundedSearch**(T, b) returns an action sequence π , then π is a solution to T .

Theorem (Completeness)

If T has a solution of length ℓ , then **BoundedSearch**($T, c \cdot \ell + md(\phi_g)$) will find a solution to it, where $c = \max\{md(\alpha) \mid \alpha \in Act\}$.

Theorem (Complexity)

BoundedSearch runs in $(b+1)$ -ExpTime.

The Epistemic Planner *DAEDALUS*

We implemented our iterative bound-deepening search algorithm in the novel **epistemic planner DAEDALUS**: **D**yn**A**mic **E**pistemic and **D**ox**A**stic **L**ogic **U**niversal **S**olver.

→ Entirely implemented in C++17.

→ **Publicly available** at: <https://github.com/a-burigana/daedalus>.

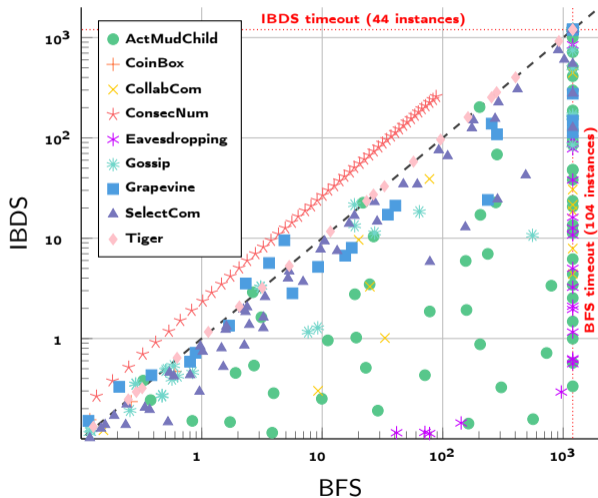
Experimental Setup

We collected different **benchmarks** from the epistemic planning literature:

- Over **400 instances** from **8 different domains** (timeout of 20 minutes).
- Tested several configurations of IBDS.
- Comparison with the DEL-based solver **EFP 2.0** [Fab+20].

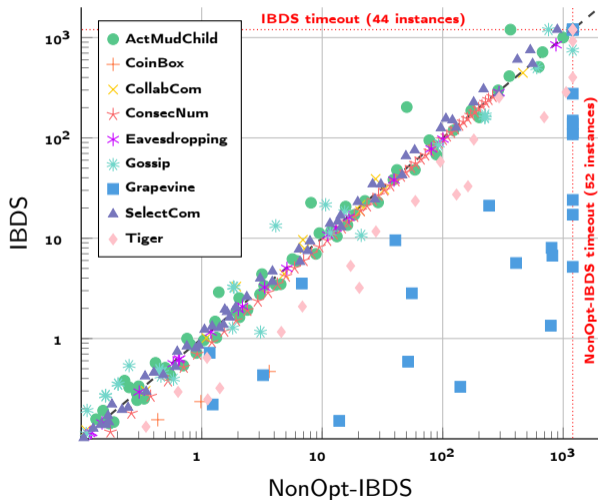
	Active Muddy Child	Coin in the Box	Collab. through Communication	Consec. Numbers	Gossip	Grpevine	Select. Comm.	Tiger
<i>Multi-Agent</i>	✓	✓	✓	✓	✓	✓	✓	
<i>Knowledge</i>	✓			✓				✓
<i>Belief</i>		✓	✓		✓	✓	✓	
<i>Ontic</i>		✓	✓			✓	✓	✓
<i>Sensing</i>		✓	✓					✓
<i>Announcement</i>	✓	✓	✓	✓	✓	✓	✓	
<i>Public</i>	✓	✓	✓	✓		✓	✓	✓
<i>Private</i>		✓	✓		✓	✓	✓	
<i>Semi-private</i>		✓	✓					
<i>Epistemic Goals</i>	✓	✓	✓	✓	✓	✓	✓	✓

IBDS vs. BFS



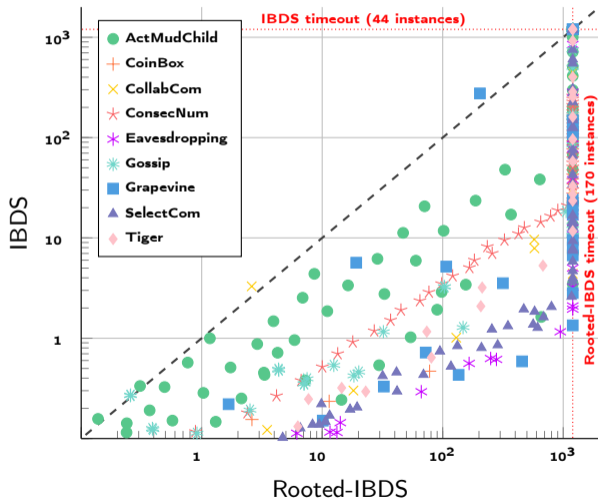
- **Bounded contractions yield smaller states** than standard contractions.
- Improved performances on vast majority of domains.
- **Solved instances:** +15%.
- **Average speedup:** $\sim 88\times$.

IBDS vs. Non-Optimized IBDS



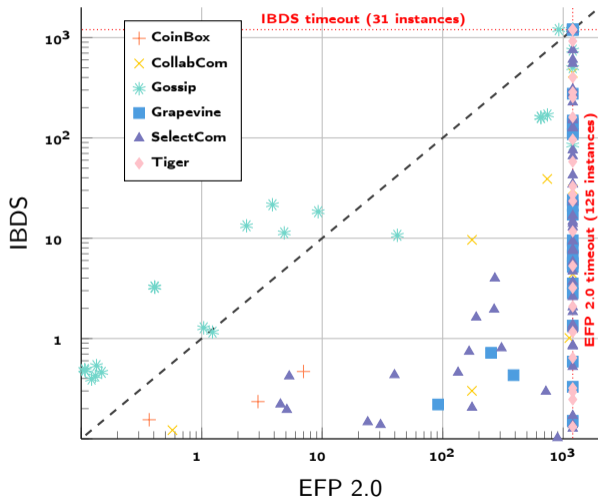
- **Fewer states** are computed during search.
- Generally **smaller bounds** for contractions.
- **Solved instances:** +2%.
- **Average speedup:** $\sim 6\times$.

Canonical vs. Rooted Contractions



- Checking for visited states is very inefficient with rooted contractions.
- We implemented it in **amortized constant time with canonical contractions**.
- Solved instances: +31%.
- Average speedup: $\sim 57\times$.

IBDS vs. EFP 2.0



- EFP 2.0 uses standard bisimulation contractions, IBDS uses bounded contractions, yielding **smaller states**.
- EFP 2.0 uses a bisimulation-based check for visited states: **quite expensive**.
- **Solved instances:** +44%.
- **Average speedup:** $\sim 280\times$.

WRAPPING UP

Conclusions

Take-home message: **bounding the reasoning depth works!**

- Effective performance improvements over state-of-the-art DEL-based planner EFP 2.0.
- The **average required reasoning bound** was 6.9, and drops to **2.4** when omitting the Consecutive Number domain, which forces the planner to use maximal reasoning depth.
- Many state-of-the-art benchmarks can be solved with a **quite limited reasoning bound**, which naturally matches our approach.
- Fully general DEL planners **can be efficient!**

Future works:

- Reducing the number of iterations of IBDS.
- Further optimizing the algorithm with **heuristics**.
- Generalize to **multi-pointed models**.

Conclusions

Take-home message: **bounding the reasoning depth works!**

- Effective performance improvements over state-of-the-art DEL-based planner EFP 2.0.
- The **average required reasoning bound** was 6.9, and drops to **2.4** when omitting the Consecutive Number domain, which forces the planner to use maximal reasoning depth.
- Many state-of-the-art benchmarks can be solved with a **quite limited reasoning bound**, which naturally matches our approach.
- Fully general DEL planners **can be efficient!**

Future works:

- Reducing the number of iterations of IBDS.
- Further optimizing the algorithm with **heuristics**.
- Generalize to **multi-pointed models**.

Thank you! Questions?