Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
KR in Planning and Scheduling

Pruning with Belief Traps in Multi-agent Epistemic Planning

Biqing Fang , Fangzhen Lin
The Hong Kong University of Science and Technology, Hong Kong, China
biqing.fang@connect.ust.hk, flin@cs.ust.hk

Abstract

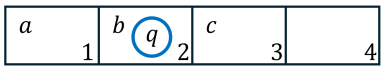
Multi-agent epistemic planning (MEP) addresses planning problems involving multiple agents with epistemic reasoning, often requiring the consideration of nested beliefs. In this paper, we extend the notion of traps in classical planning to MEP, and call them belief traps, which are epistemic formulas that once entailed by an epistemic state, remain entailed by all successor states. Identifying belief traps can sometimes improve MEP solving significantly. Here, we consider two methods for identifying and using belief traps to improve planning efficiency. Our first method adapts a classical preprocessing algorithm with integration into an MEP planner, simple-form construction of traps, and a novel use of beneficial traps to guide search. The second method systematically generalizes the belief lock strategy by formalizing its underlying preservation condition. Our experiments show that the new pruning techniques can accelerate problem-solving in the domains with irreversible beliefs.

1 Introduction

Multi-agent epistemic planning (MEP) is a form of automated planning in which agents reason about knowledge and belief (Baral et al. 2017; Belle et al. 2023). In contrast to classical planning, MEP deals with epistemic goals and epistemic action preconditions and effects, often requiring agents to reason about what other agents know or believe, and how those beliefs evolve as a result of actions. For example, the goal may be for agent a to learn a secret q, while ensuring that agent b neither knows q nor knows that agent a knows q. MEP is central to applications in multi-agent coordination, social robotics, and human-robot interaction (Thielscher 2017; Dissing and Bolander 2020; Bolander, Dissing, and Herrmann 2021).

To illustrate the MEP problem, we use an example derived from the Selective-communication (Kominis and Geffner 2015) domain: There are several rooms in a corridor. The agents can move from one room to a neighboring room. When agent i gives some information, all other agents in the same room or in a neighboring room can hear what was said. Initially, each agent is in one of the rooms. The goal is for some agents to learn certain information while some other agents do not.

Example 1. *Consider an instance with four rooms and three agents a, b, and c. Only agent a has the ability to move, $sense$ and communicate the information q at room i using $tell_p(i)$ or its negation $\neg q$ using $tell_n(i)$. Initially, the agents a, b and c are in room 1, 2, and 3, respectively, while the information q is located in room 2. The goal is to ensure that agents a and c know q while b not.*



Recently, Wan, Fang, and Liu (2021) proposed MEPK (MEP with **K**nowledge bases), a planner using alternating cover disjunctive formulas (ACDFs) to represent epistemic states. These formulas offer complete expressive power with respect to epistemic logic while maintaining tractability under certain operations. MEPK supports the generation of conditional policies, making it suitable for contingent planning in complex multi-agent settings. To improve the computational efficiency, two heuristic strategies are proposed for MEPK (Fang and Lin 2024)

In this paper, we build on the heuristic search for MEPK (Fang and Lin 2024) and introduce a general pruning mechanism based on belief traps, which are epistemic formulas that, once entailed by an epistemic state, remain entailed by all its successor states. We present two methods to detect such traps and show how they can be used to recognize dead-end states during planning. This enables our planner to prune large portions of the search space, thereby improving both efficiency and scalability.

The first method builds on a recent preprocessing algorithm that identifies traps in classical planning (Lipovetzky, Muise, and Geffner 2016). We adopt it for epistemic planning with three non-trivial adaptations: (1) integration with the state-of-the-art MEPK planner, (2) efficient construction of belief traps using simple-form expressions, and (3) the introduction of beneficial traps, a special type of traps that help guide the search toward the goal.

The second method constitutes a systematic reformulation and generalization of the belief lock strategy proposed by (Fang and Lin 2024). While the original strategy assumes that no action can reverse a certain belief, our approach formalizes this idea using a general preservation condition. This enables the detection of more complex belief traps by examining the structure of preconditions and conditional effects in action models.

Finally, we embed the proposed preprocessing algorithm in MEPK, one of the state-of-the-art syntactic MEP planners, and evaluate its performance against existing algorithms. The results demonstrate that the new pruning techniques significantly accelerate problem-solving in those domains with irreversible beliefs. We also extend the scale of instances and demonstrate that our strategies achieve better performance on these extended benchmarks.

2 Related Work

Theoretical Work. Initial work by Bolander and Andersen (2011) and Löwe, Pacuit, and Witzel (2011) introduced dynamic epistemic logic (DEL) (van Ditmarsch, van der Hoek, and Kooi 2007) as a formalism for MEP. In DEL-based MEP, states are represented by Kripke models and actions as event models. However, Aucher and Bolander (2013) demonstrated that MEP is undecidable in general. This led to the identification of decidable fragments (Yu, Wen, and Liu 2013; Cooper et al. 2016a), including the well-known "gossip problem" (Cooper et al. 2016b), where complexity ranges from tractable to NP-complete depending on the goal structure. Recently, two works (Buckingham et al. 2024; Engesser, Herzig, and Perrotin 2024) have proposed novel semantics and formalisms for reasoning about second-order false beliefs, although these remain theoretical and lack implemented planners.

Implementations. Numerous systems have been developed to implement MEP solvers, each reflecting different trade-offs between expressiveness and efficiency. Muise et al. (2015; 2022) compiled restricted MEP problems into classical planning using proper epistemic knowledge bases, enabling the use of off-the-shelf planners for belief-level planning. Similarly, the EL-O framework (Cooper et al. 2021) imposes syntactic restrictions to enable compilation to classical planning. The EFP family of planners (Le et al. 2018; Fabiano et al. 2020) leverages the $m\mathcal{A}$ action language and finitary S5-theories to describe MEP problems semantically and manage reasoning about Kripke structures. More recently, Hu, Miller, and Lipovetzky (2022; 2023) proposed a modular approach to epistemic planning, in which the reasoning about epistemic formulas is delegated to an external solver. Their planner supports disjunctive preconditions and goals, and is particularly notable for its applications in continuous domains.

3 Preliminaries

We follow the concepts and notations used in (Fang and Lin 2024). So in this section, we briefly review them here.

3.1 Multi-agent Modal Logic KD45$_n$

Consider a finite set of agents $\mathcal{A}$ and a finite set of atoms $\mathcal{P}$. We use ϕ and ψ for formulas, Φ and Ψ for sets of formulas.

Definition 1. The language $\mathcal{L}_{KC}$ of multi-agent modal logic with common knowledge is generated by the BNF:

$$\phi ::= p \mid \neg\phi \mid (\phi \wedge \phi) \mid B_a\phi \mid C\phi, \text{ where}$$

$p \in \mathcal{P}$, $a \in \mathcal{A}$, $\phi \in \mathcal{L}_{KC}$. $\mathcal{L}_K$ is used for the language without the C operator, and $\mathcal{L}_0$ for the propositional language.

Intuitively, $B_a\phi$ means that agent a believes ϕ holds, and $C\phi$ means ϕ is *common belief* among all agents.

We let $\top$ and $\bot$ represent *true* and *false* respectively. We let $\bigvee \Phi$ (resp. $\bigwedge \Phi$) denote the disjunction (resp. conjunction) of members of Φ. The *modal depth* of a formula ϕ in $\mathcal{L}_K$ is the depth of nesting of modal operators in ϕ.

Definition 2. A *frame* is a pair (W, R), where W is a nonempty set of possible worlds; for each agent $a \in \mathcal{A}$, $R : \mathcal{A} \to 2^{W\times W}$ assigns to a an accessibility relation R_a.

We say R_a is *serial* if for any $w \in W$, there is $w' \in W$ s.t. wR_aw'; R_a is *transitive* if whenever wR_aw_1 and $w_1R_aw_2$, we get wR_aw_2; R_a is *Euclidean* if whenever wR_aw_1 and wR_aw_2, we get $w_1R_aw_2$. A KD45$_n$ frame is a frame whose accessibility relations are serial, transitive, and Euclidean.

Definition 3. A Kripke model is a triple $M = (W, R, V)$, where (W, R) is a frame, and $V : W \to 2^{\mathcal{P}}$ a valuation map. A pointed Kripke model is a pair $s = (M, w)$, where M is a Kripke model and w a world of M, called the *actual world*.

Definition 4. Let $s = (M, w)$ be a pointed Kripke model where $M =(W, R, V)$. We interpret formulas in $\mathcal{L}_{KC}$ by induction: $M, w \models p$ iff $p \in V(w)$; $M, w \models \neg\phi$ iff $M, w \not\models \phi$; $M, w \models \phi \wedge \psi$ iff $M, w \models \phi$ and $M, w \models \psi$; $M, w \models B_a\phi$ iff for all v s.t. wR_av, $M, v \models \phi$; $M, w \models C\phi$ iff for all v s.t. $wR_{\mathcal{A}}v$, $M, v \models \phi$, where $R_{\mathcal{A}}$ is the transitive closure of the union of R_a for $a \in \mathcal{A}$.

A model of ϕ is a KD45$_n$ Kripke model (M, w) s.t. $M, w \models \phi$. We say ϕ is *satisfiable* if ϕ has a model. We say ϕ *entails* ψ, written $\phi \models \psi$, if any model of ϕ is also a model of ψ. We use $C^*\phi$ to denote $\phi \wedge C\phi$, and say that ϕ entails ψ w.r.t. constraint γ, written $\phi \models_\gamma \psi$, if $\phi \wedge C^*\gamma \models \psi \wedge C^*\gamma$. The notation of entailment under constraint γ is used in MEPK to support reasoning with propositional constraint, which is introduced later.

3.2 Modeling Framework of MEPK

We briefly introduce the MEPK modeling framework. Please refer to (Wan, Fang, and Liu 2021) for more details.

Definition 5. A multi-agent epistemic planning problem $\mathcal{Q}$ is a tuple $\langle \mathcal{A}, \mathcal{P}, \mathcal{D}, \mathcal{S}, \mathcal{I}, \mathcal{G}, \gamma\rangle$, where $\mathcal{A}$ is a set of agents; $\mathcal{P}$ is a set of atoms; $\mathcal{D}$ is a set of deterministic actions; $\mathcal{S}$ is a set of sensing actions; $\mathcal{I} \in \mathcal{L}_{\mathcal{K}}$ is the initial knowledge base; $\mathcal{G} \in \mathcal{L}_{\mathcal{K}}$ is the goal; and $\gamma \in \mathcal{L}_0$ is the constraint.

The propositional constraint γ is similar to the domain closure axiom in classical planning domains (Lin 2004). For example, in a domain with four rooms, the constraint $\bigvee_{i=1}^{4}\big(at(a,i) \wedge \bigwedge_{j=1,j\neq i}^{4}(\neg at(a,j)\big)$ indicates that the agent can be in exactly one room at a time.

There are three types of actions: ontic, communication, and sensing actions. The first two types share the same representation, called deterministic actions.

Definition 6. A deterministic action is a pair $\langle pre, \mathit{eff}\rangle$, where $pre \in \mathcal{L}_K$ is the precondition; *eff* is a set of conditional effects, each of which is a pair $\langle c, e\rangle$, where c, $e \in \mathcal{L}_K$ indicate the condition and effect, respectively.

Definition 7. A sensing action is a triple $\langle pre, pos, neg\rangle$ of $\mathcal{L}_K$ formulas, where pre, pos, and neg indicate the precondition, the positive result, and the negative result.

Sensing actions are used to gather information from the environment. For example, agents can sense the status of the lighting in a room (denoted by q), which leads to two possible results: the light is on (pos) or "not on" (neg). In contingent planning, the search graph branches on sensing actions according to these two outcomes.

An action a is *executable* w.r.t. a knowledge base (KB) $\phi \in \mathcal{L}_K$ if $\phi \models_\gamma pre(a)$. This means that a is executable in each model of ϕ. Suppose a is executable w.r.t. ϕ. The progression of ϕ w.r.t. a is defined by resorting to a higher-order revision operator $\circ_\gamma$ and a higher-order update operator $\diamond_\gamma$, where γ is a constraint.

The basic idea of these two operators is to reduce the change of higher-order epistemic formulas to that of lower-order epistemic formulas, and as a basis, it resorts to a change of propositional formulas. For example, $B_a(\phi \wedge B_b\psi) \circ_\gamma B_a(\phi' \wedge B_b\psi') = B_a((\phi \circ_s \phi') \wedge B_b(\psi \circ_\gamma \psi'))$, where $\phi, \phi' \in \mathcal{L}_0$ are propositional formulas; $\psi, \psi' \in \mathcal{L}_K$, and $\circ_s$ is Satoh's revision operator (Satoh 1988), see Appendix A for formal introduction of $\circ_\gamma$. Higher-order belief update operator $\diamond_\gamma$ is similar except that it is reduced to Winslett's update operator $\diamond_w$ (Winslett 1988). Intuitively, the distinction between revision and update lies in how model closeness is evaluated: revision, $\psi \circ \mu$ selects from the models of μ those that are closest to models of ψ as a whole, whereas update, $\psi \diamond \mu$ considers each model M of ψ individually and selects the models of μ that are closest to M.

The following is the progression for the sensing action.

Definition 8. Let $\phi \in \mathcal{L}_K$, and a a sensing action. If $\phi \wedge pos(a)$ is propositionally unsatisfiable w.r.t. γ, then the progression of ϕ w.r.t. a with positive result is $\bot$. Otherwise, the result is $\phi \circ_\gamma pos(a)$. The progression of ϕ w.r.t. a with negative result $neg(a)$ is defined similarly.

We use the following progression for deterministic action.

Definition 9. Let $\phi \in \mathcal{L}_K$, and a a deterministic action where $\mathit{eff}(a) = \{\langle c_1, e_1\rangle, \cdots, \langle c_n, e_n\rangle\}$. Suppose $c_{i_1}, \cdots, c_{i_m}$ are all the c_i's s.t. $\phi \models c_i$. Then the progression of ϕ w.r.t. a is defined as $((\phi \diamond_\gamma e_{i_1}) \cdots) \diamond_\gamma e_{i_m}$.

We now introduce the concepts of *action tree*, which is essential for defining a solution.

Definition 10. The progression of ϕ w.r.t. a sequence of actions is inductively defined as follows: $prog(\phi, \epsilon) = \phi$, where ϵ represents the empty sequence; $prog(\phi, (a; \sigma)) = prog(prog(\phi, a), \sigma)$ if $\phi \models pre(a)$, and undefined otherwise.

Definition 11. Let $\mathcal{Q}$ be an MEP problem $\langle \mathcal{A}, \mathcal{P}, \mathcal{D}, \mathcal{S}, \mathcal{I}, \mathcal{G}, \gamma\rangle$. The set $\mathcal{T}$ of action trees is inductively defined:

1. ϵ is in $\mathcal{T}$, here ϵ represents the empty tree;
2. if $a_d \in \mathcal{D}$ and $T \in \mathcal{T}$, then $a_d; T$ is in $\mathcal{T}$;
3. if $a_s \in \mathcal{S}, T^+, T^- \in \mathcal{T}$, then $a_s; (T^+ \mid T^-)$ is in $\mathcal{T}$.

Definition 12. Let $\mathcal{Q}$ be an MEP problem $\langle \mathcal{A}, \mathcal{P}, \mathcal{D}, \mathcal{S}, \mathcal{I}, \mathcal{G}, \gamma\rangle$. Let T be an action tree. We say a branch σ of T achieves the goal if $prog(\mathcal{I}, \sigma)$ is defined, and $prog(\mathcal{I}, \sigma) \models_\gamma \mathcal{G}$; and if $prog(\mathcal{I}, \sigma)$ is not $\bot$, we say σ properly achieves the goal. We say T is a solution of $\mathcal{Q}$ if each branch of T achieves the goal, and at least one branch properly achieves the goal.

Algorithm 1: Planning

Input: An MEP problem $\mathcal{Q} = \langle \mathcal{A}, \mathcal{P}, \mathcal{D}, \mathcal{S}, \mathcal{I}, \mathcal{G}, \gamma\rangle$.
Output: A solution or null.

```
if I ⊨ G then
    return an empty tree
while there are unexplored nodes do
    Choose the next node n with some method
    explore(n)
    if initial node is labeled with goal then
        return build_plan
    if initial state is labeled with dead then
        return null
return null
```

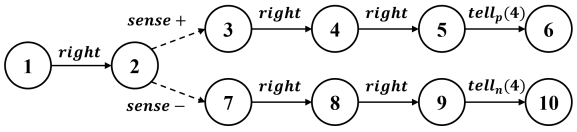


Figure 1: A solution to an instance of Selective-communication.

To illustrate the above concepts, we continue the example in the Introduction.

Example 1. ***(continued)*** *We formally model the example:*

- *The atoms are: $at(i, r)$, meaning agent i is in room r; and q, meaning the secret information.*
- *The sensing action is $sense = \langle pre, pos, neg\rangle$: the agent checks whether q or $\neg q$, where $pre = at(a, 2)$, meaning that agent i knows she is at the room 2; $pos = B_a(q)$, meaning that agent i observes the positive result; $neg = B_a(\neg q)$, meaning that agent i observes the negative result.*
- *The communication actions are $tell_p(r) = \langle at(a, r), \{\mathit{eff}_n \mid n \in \{1, 2, 3, 4\}\}\rangle$: agent a share what she knows about q to agents in the same and neighboring rooms, e.g., $\mathit{eff}_3 = \langle at(a, 3), B_b q \wedge B_c q\rangle$; Another communication action $tell_n(r)$ which shares the negative information $\neg q$ is similarly defined.*
- *The initial KB is $at(a, 1) \wedge \bigwedge_{i \in \mathcal{A}}(\neg B_i q \wedge \neg B_i \neg q)$, indicating that agent a is initially in room 1, and no agent has any beliefs about q or its negation.*
- *The goal is $(B_c q \vee B_c \neg q) \wedge (\neg B_b q) \wedge (\neg B_b \neg q)$, indicating that agent c believes q or its negation, while agent b remains unaware.*
- *The constraint is $\bigvee_{i=1}^{4} \big(at(a, i) \wedge \bigwedge_{j=1, j \neq i}^{4} (\neg at(a, j))\big)$, indicating that the agent can be in exactly one room at a time.*

Figure 1 is a feasible solution for Example 1. The action tree is a solution because both the sequences of $(right; sense+; right; right; tell_p(4))$ and $(right; sense-; right; right; tell_n(4))$ achieve the goal. Node 1 is the initial node. The sensing action, $sense$, produces two possible results: q and $\neg q$. In the planning process, nodes 6 and 10 are found to entail the goal.

The planner uses a knowledge base to model an epistemic state. When an action is performed, the current KB is progressed with the action's effects via belief change. MEPK adopts a general algorithm for contingent planning (To, Son, and Pontelli 2011) as the planning algorithm.

Algorithm 1 outlines the framework. The algorithm takes an MEP problem as input and finds a solution or returns "null" if no solution is found. Each node n is a knowledge base. Within the loop, it selects an unexplored node n using a specific method such as breadth-first search (BFS) or heuristic search (Heu). The selected node n is then explored. During exploration, the search graph is expanded and the labels $goal$ and $dead$ are propagated throughout the graph to indicate whether a node can lead to the goal or a dead-end. If, after this exploration, the initial node becomes goal-reachable, the algorithm *build_plan* simply constructs the plan with the nodes labeled $goal$ from the expanded graph and returns this plan as the solution (may be suboptimal in terms of solution size). If the initial state is determined to be $dead$, indicating that no action tree can route from the initial node while ensuring all its terminals entail the goal, the algorithm returns "null". Intuitively, this framework performs forward search by iteratively expanding the search graph and verifying whether an AND/OR tree can be constructed that leads from the initial node to the goal.

4 Our Methods

We present two approaches to identify traps for pruning in MEP. The first focuses on simple beliefs by extending a pre-processing algorithm from classical planning (Lipovetzky, Muise, and Geffner 2016). We also introduce a new notion, the beneficial trap, which captures belief formulas that are semantically aligned with the goal and can guide the search toward it. The second targets complex beliefs by generalizing a recent belief lock strategy (Fang and Lin 2024).

4.1 Belief Trap Graph

Below are definitions of belief traps and dead-end formulas.

Definition 13. A *belief trap* is a formula $\phi \in \mathcal{L}_K$ such that if a state $\psi \models \phi$, any successor state $\psi' \models \phi$.

Definition 14. A *dead-end* formula is a formula $\phi \in \mathcal{L}_K$ such that if a state $\psi \models \phi$, any successor state $\psi' \not\models goal$.

We now present *MEPTrapper* in Algorithm 2, designed to detect dead-end formulas. The main idea is to construct a belief trap graph, where the vertices represent simple belief expressions that conflict with the goal. The algorithm then traverses the graph, tagging each vertex to determine whether it is a trap, and ultimately returns the trap formulas. Intuitively, the algorithm starts from a candidate set of states and iteratively prunes those that can transition outside the set. If no such transitions remain, the formulas in this set are confirmed as belief traps.

The Algorithm 2 *MEPTrapper* takes an MEP problem as input and outputs a set of dead-end formulas. The algorithm consists of two primary stages: building the belief trap graph and extracting traps.

In the graph construction stage, the algorithm initializes a graph starting with a placeholder node, denoted as $\perp$ (line 2). For each agent and each grounded proposition p, it generates simple belief expressions such as $B_i q$ and $\neg B_i q$, as well as their logical combinations (line 6). Here, we limit the enumeration to simple-form belief expressions for efficiency, and this decision is based on two key considerations: Reasoning and progression are computationally lightweight via simple set operations in classical planning, but become time-consuming in MEP due to the higher-order belief change. Unlike classical planning, where the number of states is finite, the state-transition systems induced by MEP problems can be infinite (Bolander 2017). For instance, MEP allows formulas like $B_a B_b q$, $B_a B_b B_a q$, and so on. If a belief expression is unsatisfiable with the goal w.r.t. γ (line 8), it is added to the vertex set.

The edges of the graph are then constructed by iterating over each vertex v and each action a. If the precondition is consistent with v, the algorithm computes the progression, and add an edge from v to all vertices that are entailed by the progression (lines 14-16). We call those vertices entailed by the progression a-children of v.

To illustrate the above graph construction step, let us continue Example 1. The algorithm begins by constructing a graph where the vertices represent belief expressions that

Algorithm 2: *MEPTrapper*

Input: An MEP problem $\mathcal{Q} = \langle \mathcal{A}, \mathcal{P}, \mathcal{D}, \mathcal{S}, \mathcal{I}, \mathcal{G}, \gamma \rangle$.
Output: A set of dead-end formulas Φ.

```
   /* Building the belief trap graph                    */
1  Initialize a labeled acyclic directed graph G = (V, E)
2  V ← {⊥} /* ⊥ is a placeholder node                    */
3  E ← {}
4  foreach agent i ∈ A do
5      foreach atom p ∈ P do
6          foreach belief ψ ∈ {B_i p, B_i ¬p, ¬B_i p, ¬B_i ¬p,
7          ¬B_i p ∨ ¬B_i ¬p, ¬B_i p ∧ ¬B_i ¬p} do
8              if G ∧ ψ is unsatisfiable w.r.t. γ then
9                  V ← V ∪ {ψ}
10 foreach vertex v ∈ V do
11     foreach action a ∈ D ∪ S do
12         if pre(a) is consistent with v then
13             compute the progression of v: prog(v, a)
14             foreach vertex v' ∈ V do
15                 if prog(v, a) ⊨ v then
16                     E ← E ∪ {(v, a, v')}
   /* Extracting the trap                                */
17 Initialize tags t_v ← true for each v ∈ V
18 t_⊥ ← false;
19 v* ← ⊥
20 TrapTagger(v*)
21 foreach vertex v ∈ V do
22     if t_v = true then
23         Φ ← Φ ∪ v
24 return Φ
```

Algorithm 3: *TrapTagger*

Input: A vertex v^*.

```
1  Let V_p be all the predecessors of v*
2  if V_p = ∅ then
3      return
4  foreach v ∈ V_p do
5      foreach action a ∈ D ∪ S do
6          if ∀(v, a, v') ∈ E, t_{v'} = false then
7              t_v ← false
8              break
9  foreach v ∈ V_p do
10     TrapTagger(v)
```

conflict with the goal: $\bot$ (the placeholder node), B_bq, $B_b\neg q$, and $\neg B_cq \wedge \neg B_c\neg q$, while the edges represent actions. A partial belief trap graph is shown in Figure 2. The ontic actions *left* and *right* are applicable to every vertex, and their effects do not change the beliefs. Therefore, these actions result in self-loop edges. Consider the vertex B_bq. The applicable communication actions are $tell_p(i)$ and $tell_n(i)$. The progression through $tell_n(2)$ is entailed by $B_b\neg q$, resulting in an edge from B_bq to $B_b\neg q$. Now, consider the vertex $\neg B_cq \wedge \neg B_c\neg q$. An edge from this vertex to B_bq (resp. $B_b\neg q$) is added through the progression of the action $tell_p(3)$ (resp. $tell_n(3)$). However, the progressions of $\neg B_cq \wedge \neg B_c\neg q$ w.r.t. $tell_p(3)$ and $tell_n(3)$ do not entail any other vertices. As a result, two edges pointing to the placeholder node are added.

After the graph is constructed, the trap extraction stage begins. All vertices are initialized as potential traps except for the placeholder node $\bot$, which is tagged as not a trap (lines 17-18). The *TrapTagger* procedure then propagates trap information through the graph in reverse, starting from $\bot$ (line 20).

For example, in the graph shown in Figure 2, the vertex $\neg B_cq \wedge \neg B_c\neg q$ has a $tell_p(4)$-child and a $tell_n(4)$-child, both pointing to $\bot$. Since all successors under these actions are already tagged as non-traps, this vertex is also marked as not a trap. In contrast, the vertices B_bq and $B_b\neg q$ have no action a such that all a-children are non-traps. Therefore, their tags remain *true* and are returned as part of the final dead-end formula.

Algorithm 3 describes the *TrapTagger* procedure in detail. Given a vertex v^*, the algorithm identifies all of its predecessors. For each predecessor v, it checks whether there exists an action a such that all a-successors of v have been tagged as non-traps. If so, v is also tagged as not a trap, and the procedure recursively continues the propagation to its predecessors.

Now, we show the correctness of the algorithms.

Theorem 1. *Let $\Phi = \{\phi_1, \phi_2, \ldots\}$ be the set of formulas returned by the algorithm MEPTrapper. Then each $\phi_i \in \Phi$ is a dead-end formula.*

Proof. Let v be a vertex where $t_v = \textit{true}$, and the corresponding formula $\phi_i = v$ is included in the output set Φ. To establish that ϕ_i is a dead-end formula, we need to demonstrate that for any state ψ where $\psi \models \phi_i$, the progression of ψ under any applicable action a does not entail the goal. The trap tagging algorithm ensures that $t_v = \textit{true}$ iff for every applicable action a, at least one of its a-child (*i.e.*, vertex v' where $(v, a, v') \in E$) must also be a trap because its tag will be turned into *false* otherwise (lines 6-7, Algorithm 3). This guarantees that from any state ψ that entails ϕ_i, its progression under any applicable action a necessarily results in a state that entails some formula $\phi_j \in \Phi$.

Since all such $\phi_j = v'$ are, by construction, mutually exclusive with the goal (lines 8-9, Algorithm 2), it follows that no successor state entails the goal. Therefore, ϕ_i is a dead-end formula. □

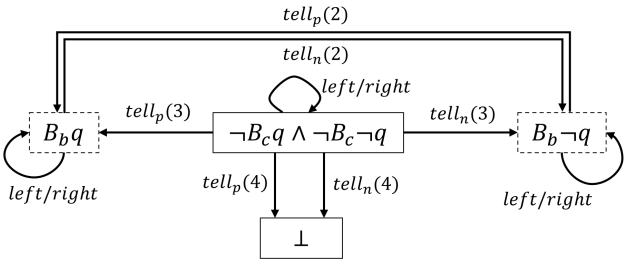


Figure 2: Partial belief trap graph for dead-end formulas.

The intuition behind is as follows. Each belief expression ϕ_i selected into the final output set Φ is known to be inconsistent with the goal. No matter which action is applied, the resulting successor state remains within the trap set, that is, also satisfies a vertex in Φ, and therefore are also inconsistent with the goal. Informally, we identify belief conditions that form closed "pockets" in the belief space from which it is impossible to escape toward a goal-satisfying state. These pockets are traps, and because they are also goal-inconsistent, they are dead-ends.

The original preprocessing algorithm for classical planning has a runtime that is exponential in parameter k, the number of terms present in the DNF formula representing the trap (Lipovetzky, Muise, and Geffner 2016). In our algorithms, considering only a candidate set of beliefs (line 6) amounts to set k to 1, and it runs only in time $O(|\mathcal{A}|^2 \cdot |\mathcal{P}|^2 \cdot |\mathcal{D} \cup \mathcal{S}|)$ (lines 10-16), where $|\mathcal{A}| \cdot |\mathcal{P}|$ limits the maximum number of vertices of the belief trap graph (lines 4-9). Since we focus on simple belief expressions and the modal depth of the formulas is limited to 1 to build the graph, the preprocessing process remains efficient.

Furthermore, because the belief trap graph is constructed solely based on the domain's action theory and the goal formula, one advantage is that it only needs to be run once per domain-goal pair. The resulting traps can then be reused across multiple problem instances that share the same domain and goal, even if their initial KBs are different.

The application of the *MEPTrapper* algorithm to an MEP problem is straightforward: whenever a new node (a new KB ϕ) is generated, it is marked as a dead-end if ϕ is a dead-end formula, which means it will not be explored anymore.

4.2 Beneficial Trap

We now introduce beneficial traps, a novel concept aimed at improving search efficiency.

Definition 15. A *beneficial trap* is a *trap* such that $goal \models \phi$.

Intuitively, a beneficial trap can potentially progress to a state that entails the goal. Consider an example from the Selective-communication domain with four agents, where the goal is $(\neg B_b q) \wedge (\neg B_b \neg q) \wedge (B_c q \vee B_c \neg q) \wedge (B_d q \vee B_d \neg q)$. In this case, $(B_d q \vee B_d \neg q)$ is a beneficial trap, as it can lead to the goal after agent a doing a single action that tells the information to agent c's room but not to b's.

The idea is similar to the use of subgoals in classical planning, but with important differences. In our setting, goals are arbitrary KD45 formulas and are defined via entailment. Moreover, unlike classical subgoals, beneficial traps are epistemic formulas that are preserved across all successor states, making them more robust and reliable as intermediate targets during search.

The computation of beneficial traps is similar to the computation of dead-end formulas. To build belief trap graph for beneficial traps, we modify the Algorithm 2 by replacing the condition in line 8 with $(\mathcal{G} \models \psi)$. Figure 3 is a partial belief trap graph for identifying beneficial traps. The tagging process begins from the vertex $\top$, and then concludes with the identification of the beneficial trap $B_c q \vee B_c \neg q$.

The application of beneficial trap to above algorithms is similar to dead-end formulas: whenever a new node (a new KB ϕ) is generated, its priority is promoted if ϕ is identified as a beneficial trap. Here, we only increase the priority rather than ensuring it is searched first, as a beneficial trap can also lead the search to a dead-end. Consider the same example above: $(B_d q \vee B_d \neg q)$. If agent a shares q with agent c while agent b stays in a neighboring room, the successor state becomes a dead-end state.

4.3 General Belief Lock

To efficiently construct the belief trap graph, we focused on simple belief expressions. However, there are traps in more complex forms that cannot be captured by the previously defined simple belief expressions. For example, in the Gossip domain (formalized later), where the goal could be $B_a s_b$ or a more complex formula such as $B_a(s_b \wedge s_c \vee s_d)$, leading to an exponential number of possibilities w.r.t. the number of agents. To address such cases, we extend the belief lock strategy to handle these more complex scenarios.

We introduce two definitions for the original belief lock.

Definition 16. Given an MEP problem $\mathcal{Q}$ with constraint γ. A formula ψ is *unreachable* from ϕ if $prog(\phi, \tau) = \bot$ or $prog(\phi, \tau) \not\models_\gamma \psi$ for any action sequence τ.

Definition 17. Given an MEP problem $\mathcal{Q}$ with deterministic actions $\mathcal{D}$ and sensing actions $\mathcal{S}$. The formula ϕ is said to be *strongly preserved* if the following hold:

1. $e_i \not\models_\gamma \neg\phi$, where $\langle c_i, e_i \rangle \in \mathit{eff}(d)$, for any $d \in \mathcal{D}$;
2. $pos(s) \not\models_\gamma \neg\phi$ and $neg(s) \not\models_\gamma \neg\phi$ for any $s \in \mathcal{S}$.

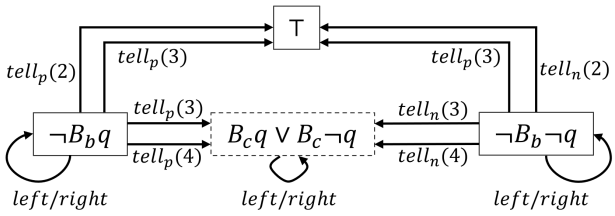


Figure 3: Partial belief trap graph for beneficial traps.

The above two conditions imply that the action model of the domain restricts agents from entering certain states. For example, the formula $B_a at(a, 1)$, which denotes that agent a knows she is in room 1, is not strongly preserved because the effects of the action $right$ entail the negation of $B_a at(a, 1)$. In contrast, the formula $B_a s_b$, which means agent a knows the secret of agent b, is strongly preserved since no actions in the Gossip domain can alter this belief.

The original proposition of belief lock (Fang and Lin 2024) can be rewritten as the following corollary and proposition.

Corollary 1. *If formula $\phi \in \mathcal{L}_K$ is strongly preserved, then ϕ is a belief trap.*

Proposition 1. Given an MEP problem $\mathcal{Q}$ with goal $\mathcal{G}$ and constraint γ. Let ϕ and ψ be two KBs. Then $\mathcal{G}$ is unreachable from ψ if (1) $\mathcal{G} \models_\gamma \neg\phi$; (2) $\psi \models_\gamma \phi$; and (3) ϕ is strongly preserved.

Proposition 1 can be applied to find dead-end formulas before planning. Intuitively, the conditions indicate that once the KB progresses into a specific belief state, and the domain cannot alter the situation further, then the KB indicates a trap. This property allows the planner to identify dead-end formulas in advance.

However, some dead-ends may be overlooked because the conditions of the proposition are strong.

Example 2 (Gossip)**.** *There are three agents a, b, and c. Each of them has her own secret s_a, s_b, and s_c. They can call each other and exchange all the secrets they know.*

The communication actions, $share_p(i, j)$ and $share_n(i, j)$ allow agent i to call agent j to exchange all the secrets or their negations that they know. For instance, let one of the conditional effects of $share_n(c, a)$ be $\langle c, e \rangle$, where $c = B_c \neg s_c \wedge \neg B_a s_c \wedge \neg B_a \neg s_c$, and $e = B_a \neg s_c$. This conditional effect means that if agent c believes the negation of s_c and a knows nothing about s_c, then a receives $\neg s_c$.

Here is an instance for this domain: The initial KB is $B_a s_a \wedge B_b s_b \wedge B_c \neg s_c$, meaning that initially each agent knows her own secret or its negation.

The goal is $B_a s_b \wedge B_a \neg s_c \wedge \neg B_c s_a$, meaning that agent a knows the secret of agent b and the negation of agent c' secret, while agent c does not know the secret of a.

The constraint is $\top$ as there are no domain-specific properties, such as agents' locations.

A solution to this example is for agents b and c to communicate first, followed by communication between agents a and b, ensuring that agent c does not learn agent a's secret.

According to the Definition 17, $B_c s_a$ is not strongly preserved, as the conditional effect $e \models_\gamma \neg B_a s_c$. However it is indeed a dead-end formula because once the KB progresses into the state that entails $B_a \neg s_c$, the precondition is no longer satisfied, causing the KB to become "stuck" in a trap.

To address this limitation, we introduce a general belief lock strategy to identify additional traps that are not covered by the original belief lock strategy.

Definition 18. Given an MEP problem $\mathcal{Q}$ with deterministic actions $\mathcal{D}$ and sensing actions $\mathcal{S}$. The formula $\phi \in \mathcal{L}_K$ is said to be *generally preserved* if the following hold:

1. For any $d \in \mathcal{D}$ and $\langle c_i, e_i \rangle \in \mathit{eff}(d)$: Either $e_i \not\models_\gamma \neg\phi$, or $\phi \wedge \mathit{pre}(d)$ or $\phi \wedge c_i$ is unsatisfiable;
2. For any $s \in \mathcal{S}$: Either $\mathit{pos}(s) \not\models_\gamma \neg\phi$ and $\mathit{neg}(s) \not\models_\gamma \neg\phi$, or $\phi \wedge \mathit{pre}(s)$ is unsatisfiable.

Technically, our approach extends the original belief lock strategy by replacing the strong preservation condition with a general one. Instead of requiring that no action can contradict a belief, the general preservation condition only requires that the conditions of all potential effects that could alter the belief become unsatisfied after it is established. For instance, in a variant of the Gossip problem from Example 2, agent a can obtain either s_c or its negation $\neg s_c$. However, once one of these is obtained, the condition of the corresponding conditional effect, $\neg B_a s_c \wedge \neg B_a \neg s_c$, becomes unsatisfiable.

Corollary 2. *If the formula $\phi \in \mathcal{L}_K$ is generally preserved, then ϕ is a belief trap.*

proof sketch. Given a state (KB) ψ and a generally preserved formula ϕ we show that if $\psi \models_\gamma \phi$, then any progression of ψ w.r.t. action sequences τ entails ϕ. This can be proved by induction on arbitrary action sequences τ. For the base case where τ consists of only one action t, we show that the single-step progression ψ w.r.t. action t entails ϕ. For the induction step where τ consists of two or more actions. We recursively apply the result from the base case and obtain that any progression will entail ϕ. The complete proof follows and generalizes the structure of Proposition 1's proof in (Fang and Lin 2024), accommodating the weaker conditions of general preservation. □

Then the following corollary is straightforward.

Corollary 3. *Given an MEP problem $\mathcal{Q}$ with goal $\mathcal{G}$ and constraint γ. If the formula $\phi \in \mathcal{L}_K$ is generally preserved and $\mathcal{G} \models_\gamma \phi$, then ϕ is a beneficial trap.*

Based on Corollary 2, we have following proposition.

Proposition 2 (General Belief Lock)**.** Given an MEP problem $\mathcal{Q}$ with goal $\mathcal{G}$ and constraint γ. Let ϕ and ψ be two KBs. Then $\mathcal{G}$ is unreachable from ψ if (1) $\mathcal{G} \models_\gamma \neg\phi$; (2) $\psi \models_\gamma \phi$; and (3) ϕ is generally preserved.

Beyond identifying dead-end formulas using Proposition 2, we can also leverage Corollary 3 to identify beneficial traps. Compared to the simple belief expressions in the *MEPTrapper* algorithm, this approach enables the identification of more complex forms of belief traps, such as formulas like $B_a(s_b \wedge s_c \vee s_d)$.

To apply the general belief lock strategy, we analyze the goal prior to planning. For each term in goal formula, we verify whether it satisfies the conditions in Corollary 3 and Proposition 2, categorizing it as either a beneficial trap or a dead-end formula. This process is similar to the *MEPTrapper* algorithm. All traps identified during preprocessing are subsequently leveraged for pruning during the planning phase, enhancing the overall efficiency of the planner.

5 Experimentation

All the experiments are run on Linux with a 2.90GHz CPU and 16GB RAM. The time limit for each run is set to 2 hours.

We conduct two experiments. First, we consider only the hard benchmark since the previous algorithms have already get optimal results on the easy benchmark. Second, we extend the scale of instances on some domains to further demonstrate the efficiency in these new instances.

In addition to the Selective-communication and Gossip domains introduced earlier, the existing hard benchmark (Fang and Lin 2024) also includes three other domains: Collaboration-and-communication (Kominis and Geffner 2015). Finding-the-truth (Wan, Fang, and Liu 2021), and Hexa Game (van Ditmarsch 2001). Below we give the description of each domain.

Collaboration-and-communication: CC(n). There is a corridor of four rooms. n boxes are located in some of the rooms. Two agents can move back and forth along this corridor. When an agent enters a room, she can see if a box is in the room. An agent can communicate information to another agent. Furthermore, there is a variant with a cheat action, denoted as †CC(n). The cheat action means agent i can mislead agent j about whether box b is in room r.

Finding-the-truth: FT(n). There are two boxes located in n rooms. Two agents start with wrong beliefs about the positions of the boxes. The agents can move between the rooms and check if a box is in a room. The goal is for the agents to find out the true locations of the boxes.

Hexa Game: HG(n). There are n agents and n cards, each with a unique color. Initially, everyone is holding a card, and can only see the color of her own card. A player can ask a question to another player whether her card is of a certain color. The goal is for some agents to know others' cards. HG(n)* denotes a variant with different goals.

Grapevine: Grap(n, m). A few agents meet in a villa with n rooms. Each agent has her own secret to share with others. Each agent can move between the rooms, and broadcast her secret to the agents in the same room. The goal is that some get to know the secrets of some agents while some other do not. m indicate the modal depth for each instance.

5.1 Results on Hard Benchmark

Table 1[1] present the results on the hard benchmark. The comparison includes the baseline algorithms BFS and Heu (Wan, Fang, and Liu 2021), their enhanced versions (BFS$^+$ and Heu$^+$), variants further improved with the belief lock strategy (BFS$^+$K and Heu$^+$K), and our proposed versions incorporating the trap-based strategy ($^+$T).

[1]Code & data be available at https://github.com/sysulic/MEPK

Instance	BFS	BFS$^+$	BFS$^+$K	BFS$^+$T	Heu	Heu$^+$	Heu$^+$K	Heu$^+$T
CC(3)	176 (17 / 1590)	49 (28 / 845)	47 (28 / 845)	46 (28 / 845)	34 (16 / 766)	**3 (15 / 167)**	**2 (15 / 167)**	**2 (15 / 167)**
CC(4)	5864 (33 / 15780)	163 (58 / 2934)	161 (58 / 2934)	152 (58 / 2934)	250 (17 / 3927)	**2 (24 / 268)**	**2 (24 / 268)**	**2 (24 / 268)**
CC(5)	–	–	–	–	–	**458 (84 / 2487)**	**364 (84 / 2487)**	**351 (84 / 2487)**
†CC(4)	49 (7 / 1115)	65 (19 / 1472)	64 (19 / 1472)	60 (19 / 1472)	17 (9 / 829)	**7 (14 / 466)**	**6 (14 / 466)**	**6 (14 / 466)**
†CC(5)	478 (7 / 1701)	1547 (14 / 3912)	1484 (14 / 3912)	1227 (14 / 3912)	**101 (12 / 748)**	**114 (12 / 748)**	**99 (12 / 748)**	**88 (12 / 748)**
†CC(6)	2456 (4 / 1709)	2309 (4 / 1709)	2123 (4 / 1709)	1764 (4 / 1709)	**6 (4 / 42)**	**6 (4 / 42)**	**7 (4 / 42)**	**6 (4 / 42)**
FT(4)	–	1 (20 / 192)	1 (20 / 192)	1 (20 / 192)	6 (22 / 625)	**0 (25 / 126)**	**0 (25 / 126)**	**0 (25 / 126)**
FT(5)	–	**1 (23 / 157)**	**1 (23 / 157)**	**1 (23 / 157)**	13 (19 / 486)	3 (35 / 199)	2 (35 / 199)	2 (35 / 199)
FT(6)	–	**2 (23 / 157)**	**3 (23 / 157)**	**2 (23 / 157)**	83 (22 / 892)	6 (29 / 195)	5 (29 / 195)	5 (29 / 195)
FT(7)	–	**5 (23 / 157)**	**5 (23 / 157)**	**5 (23 / 157)**	276 (55 / 1158)	11 (29 / 195)	12 (29 / 195)	10 (29 / 195)
FT(8)	–	**11 (23 / 157)**	**10 (23 / 157)**	**11 (23 / 157)**	447 (41 / 1016)	28 (29 / 195)	25 (29 / 195)	23 (29 / 195)
HG(4)	14 (35 / 2128)	12 (59 / 2076)	8 (59 / 2076)	10 (59 / 2076)	2167 (45 / 35508)	**6 (35 / 1459)**	**5 (35 / 1459)**	**4 (35 / 1459)**
HG(4)*	**77 (71 / 4774)**	73 (71 / 4810)	72 (71 / 4810)	66 (71 / 4810)	1836 (71 / 32843)	73 (71 / 4892)	63 (71 / 4892)	56 (71 / 4892)
HG(5)	9 (23 / 1447)	18 (23 / 2029)	16 (23 / 2029)	15 (23 / 2029)	–	**2 (27 / 678)**	**2 (27 / 678)**	**2 (27 / 678)**
HG(6)	**167 (47 / 4092)**	682 (47 / 10744)	665 (47 / 10744)	572 (47 / 10744)	–	825 (63 / 11264)	674 (63 / 11264)	682 (63 / 11264)
HG(7)	713 (47 / 7598)	**130 (49 / 2967)**	**125 (49 / 2967)**	**118 (49 / 2967)**	–	5365 (69 / 26400)	4881 (69 / 26400)	4880 (69 / 26400)
Gossip(6)	–	–	–	162 (9 / 1764)	4 (10 / 224)	5 (10 / 224)	**2 (10 / 190)**	**3 (10 / 190)**
Gossip(7)	–	–	–	–	**32 (10 / 425)**	**37 (10 / 425)**	38 (10 / 453)	35 (10 / 453)
Gossip(8)	–	–	–	779 (10 / 2082)	**73 (10 / 593)**	**83 (10 / 593)**	77 (10 / 615)	1095 (10 / 2728)
Gossip(9)	1252 (4 / 1779)	1416 (4 / 1779)	1307 (4 / 1751)	6 (4 / 108)	**4 (4 / 73)**	**5 (4 / 73)**	**4 (4 / 73)**	**4 (4 / 73)**
Gossip(10)	1437 (4 / 1402)	1565 (4 / 1402)	1373 (4 / 1369)	**12 (5 / 135)**	211 (6 / 591)	236 (6 / 591)	179 (6 / 591)	**12 (5 / 135)**
SC(16,7)	59 (50 / 1404)	46 (58 / 1189)	8 (58 / 446)	**1 (54 / 106)**	41 (63 / 1063)	46 (64 / 1077)	6 (56 / 382)	**1 (54 / 106)**
SC(16,8)	190 (54 / 2323)	119 (62 / 1861)	8 (62 / 471)	**1 (58 / 114)**	63 (54 / 1258)	104 (56 / 1617)	3 (60 / 218)	**1 (58 / 114)**
SC(16,14)	–	–	78 (70 / 1166)	**3 (64 / 128)**	1126 (69 / 4121)	2600 (72 / 7397)	26 (82 / 590)	**3 (64 / 128)**
SC(24,7)	260 (72 / 2053)	259 (84 / 2082)	34 (84 / 704)	**4 (78 / 161)**	122 (80 / 1282)	94 (72 / 1069)	13 (97 / 361)	7 (78 / 228)
SC(24,8)	639 (74 / 3069)	675 (87 / 3160)	44 (87 / 742)	**5 (82 / 170)**	569 (89 / 2656)	126 (81 / 1147)	16 (93 / 382)	8 (85 / 222)
SC(24,14)	–	–	793 (93 / 2617)	**11 (90 / 184)**	–	–	67 (104 / 582)	35 (90 / 343)
Grap(4,1)	11 (6 / 1206)	10 (6 / 1206)	7 (6 / 832)	1 (8 / 188)	1 **(10 / 179)**	**2 (10 / 179)**	1 **(10 / 179)**	5 (19 / 604)
Grap(4,2)	66 (7 / 3939)	64 (7 / 3939)	31 (7 / 2341)	2 (10 / 248)	1 (10 / 211)	1 (10 / 211)	2 (10 / 247)	**1 (10 / 148)**
Grap(4,3)	797 (8 / 12315)	730 (8 / 12315)	349 (8 / 8472)	**2 (13 / 127)**	46 (15 / 2199)	56 (15 / 2199)	45 (19 / 2280)	1 (14 / 165)
Grap(5,1)	171 (7 / 2767)	160 (7 / 2767)	79 (7 / 1721)	14 (9 / 416)	28 (18 / 909)	32 (18 / 909)	15 (13 / 475)	**7 (14 / 159)**
Grap(5,2)	563 (8 / 5359)	555 (8 / 5359)	342 (8 / 4146)	**8 (11 / 203)**	10 (12 / 433)	12 (12 / 433)	19 (16 / 659)	16 (16 / 587)
Grap(5,3)	–	–	3135 (9 / 13234)	14 (14 / 232)	–	–	105 (20 / 1926)	**6 (15 / 162)**

Table 1: Experimental results on hard benchmark. First column indicate the name of instance, followed by subsequent columns presenting the results from various algorithms. Each result is represented as $T(A/B)$, indicating T seconds of run time, A nodes for the size of plan, B nodes expanded across the entire search graph. "−" indicates timeout. The results with **smallest number of expanded nodes** are in boldface.

By importing the new pruning techniques, the improved results are underlined. Specifically, BFS$^+$T expands fewer nodes than BFS$^+$K in 16 instances, and Heu$^+$T surpasses Heu$^+$K in 12 cases, demonstrating the efficiency of the algorithms. Overall, BFS$^+$T and Heu$^+$T together achieve the lowest node expansion in 28 out of 33 instances. Turning to runtime, BFS$^+$T achieves a time reduction of more than 80% compared to BFS$^+$K in 16 instances, and Heu$^+$T achieves over 45% time savings compared to Heu$^+$K in 10 instances. These results collectively show the practical effectiveness of the trap-based strategy in significantly reducing both the search space and the computational cost.

One reason our pruning approaches bring improvements in the bottom three domains is that epistemic planning problems exhibit certain characteristics. Specifically, not all agent beliefs are reversible. For example, in the Gossip domain, agents share and receive secrets without any retraction. However, in domains where agents can reverse their beliefs about all propositions, such as the domain of FT where the agents can change their beliefs about box positions, and the variant of Collaboration-and-communication with the $cheat$ action, where agents can mislead others by providing incorrect box positions, BFS$^+$T and Heu$^+$T offer no improvements over BFS$^+$(K) and Heu$^+$(K).

5.2 Results on Extended Benchmark

To further showcase the effectiveness of the proposed pruning techniques in domains specifically tailored to highlight their advantages, we extend the scale (*e.g.*, the number of agents and rooms) of three domains: Gossip, Selective-communication, and Grapevine, resulting in a total of 20 instances for each domain (60 instances in total).

Domain	BFS$^+$T	Heu$^+$	Heu$^+$K	Heu$^+$T
Gossip	3 (4095)	6 (1004)	8 (705)	**17** (591)
SC	**13** (3)	0 (682)	0 (14)	10 (4)
Grapevine	0 (4959)	0 (7012)	0 (6648)	**20** (604)

Table 2: Experimental results on extended benchmark.

Table 2 summarizes the results of four best-performing algorithms for each domain. Similar to Table 1, we define the best result as the run with the smallest number of expanded nodes during search. For each result $N(T)$, N denotes the number of instances (out of 20) in which the algorithm achieved the best result, and T is the average runtime (in seconds) across all 20 instances.

As shown in the table, Heu$^+$T outperforms the other algorithms in the Gossip and Grapevine domains, achieving the best result in 17 and 20 instances, respectively, with significantly lower average runtimes. In contrast, BFS$^+$T demonstrates superior performance in the SC domain, achieving the best result in 13 out of 20 instances. These results highlight the effectiveness of the proposed trap-based pruning techniques, particularly in domains where the search space is large.

6 Discussion

There are two main paradigms for modeling and solving MEP problems: the semantic approach, which represents states as epistemic models (*e.g.*, DEL), and the syntactic approach, where states are represented as knowledge bases, *i.e.*, sets of formulas known to hold. In this work, we adopt the syntactic approach, following the design of MEPK, for two main reasons. First, MEPK provides an expressive framework capable of modeling a broad range of MEP problems, including those involving disjunctive beliefs, which are unsupported by some other planners such as RP-MEP (Muise et al. 2022). Second, MEPK demonstrates strong empirical scalability in the context of MEP, handling planning problems with up to 24 agents and plan size reaching 73 steps among the largest scales reported to date. For an overview of the connections between semantic and syntactic approach, we refer readers to a nice introduction by Bolander (2017).

MEP is still an emerging area of research. One of the current challenges lies in the lack of standardized benchmarks for evaluating epistemic planners. Existing planners often differ in their input languages and the logical fragments they target, making direct comparisons difficult. The planner most closely related to our work is RP-MEP, whose performance is comparable to the baseline (BFS and Heu) of MEPK, which was analyzed in previous work on MEPK (Wan, Fang, and Liu 2021). Although cross-planner comparisons are inherently difficult, two prior works (Wan, Fang, and Liu 2021; Muise et al. 2022) have provided limited yet insightful comparisons across different approaches.

Despite their effectiveness, the proposed pruning techniques have limitations, which suggest directions for future work. For example, the belief trap graph currently considers only simple-form belief expressions with limited modal depth. In addition, even when combined with the general belief lock strategy, the overall preprocessing procedure is sound but not complete, meaning that some traps may still be missed. Developing an algorithm that is both complete and similarly efficient would be an interesting direction for future research. Another promising direction is the development of benchmarks that better reflect real-world scenarios. For instance, one could model household robots that help resolve conflicts and handle false-belief tasks in complex family environments.

7 Conclusions

This paper proposed two approaches to improve the efficiency of multi-agent epistemic planning by identifying and utilizing belief traps for pruning. The first approach adapts classical planning techniques to identify simple-form expressions for belief traps, and introduces beneficial traps to guide search toward goal state. The second approach generalizes the belief lock strategy to detect complex traps by analyzing action models. Our experiments on existing and extended benchmarks show that the new pruning techniques can accelerate problem-solving compared to the previous heuristic search algorithms for MEPK.

A Higher-order Belief Revision

The higher-order revision operator of MEPK is defined on ACDF, a normal form for $\mathrm{KD45}_n$ to support efficient reasoning and progression. First, we introduce the cover modality and ACDF. Let $\widehat{B}_a\phi$ stand for $\neg B_a \neg\phi$, and $\widehat{B}_a\Phi$ to represent the conjunction of $\widehat{B}_a\phi$ where $\phi \in \Phi$.

Definition 19. Let $a \in \mathcal{A}$, and Φ a finite set of formulas. The cover modality is defined as: $\nabla_a\Phi \doteq B_a(\bigvee \Phi) \wedge \widehat{B}_a\Phi$.

Intuitively, $\nabla_a\Phi$ means that each world considered possible by agent a satisfies an element of Φ, and each element of Φ is satisfied by some world considered possible by agent a.

Definition 20. The set of *cover disjunctive formulas* (CDFs) is inductively defined as: 1. A propositional term, *i.e.*, a conjunction of propositional literals, is a CDF; 2. If ϕ_0 is a propositional CDF, and for each $a \in \mathcal{B} \subseteq \mathcal{A}$, Φ_a is a finite set of CDFs, then $\phi_0 \wedge \bigwedge_{a\in\mathcal{B}} \nabla_a\Phi_a$ is a CDF, called a *CDF term*; 3. If Φ is a non-empty finite set of CDF terms, then $\bigvee \Phi$ is a CDF, called a *disjunctive CDF*.

Definition 21. An *alternating CDF* (ACDF) is a CDF with no modal operators of an agent directly nested within those of the same agent.

For example, $\nabla_b\{\nabla_a\{\neg q\}\}$ is an ACDF; but the CDF $\nabla_a\{\nabla_a\{\neg q\}\}$ is not since it happens that ∇_a directly appears after ∇_a. Hales, French, and Davies (2012) introduced the notion of ACDFs, and showed that in $\mathrm{KD45}_n$, every formula in $\mathcal{L}_K$ is equivalent to such a formula.

Let $\Phi *_\gamma \Phi'$ denote $\{(\phi,\phi') \mid \phi \in \Phi, \phi' \in \Phi', \phi \wedge \phi' \text{ is satisfiable w.r.t. } \gamma\}$. Next, we introduce the higher-order revision operator.

Definition 22. Let ϕ and ϕ' be ACDFs, γ a DNF formula. The revision of ϕ with ϕ' under γ, denoted $\phi \circ_\gamma \phi'$, is recursively defined as follows:

1. When ϕ and ϕ' are propositional, the result is $\phi \circ_s (\phi' \wedge \gamma)$, where $\circ_s$ is Satoh's revision operator (Satoh 1988).
2. When $\phi = \phi_0 \wedge \bigwedge_{a\in\mathcal{B}} \nabla_a\Phi_a$, $\phi' = \phi'_0 \wedge \bigwedge_{a\in\mathcal{B}'} \nabla_a\Phi'_a$, and $\phi \wedge \phi'$ is satisfiable w.r.t. γ, $\phi \circ_\gamma \phi'$ is defined as:

$$(\phi_0 \circ_\gamma \phi'_0) \wedge \bigwedge_{a\in\mathcal{B}-\mathcal{B}'} \nabla_a\Phi_a \wedge \bigwedge_{a\in\mathcal{B}'-\mathcal{B}} \nabla_a\Phi'_a \wedge \pi$$

   where $\pi = \bigwedge_{a\in\mathcal{B}\cap\mathcal{B}'} \nabla_a[(\Phi_a \circ_\gamma \bigvee \Phi'_a) \cup (\Phi'_a \circ_\gamma \bigvee \Phi_a)]$ if $\phi \wedge \phi'$ is propositionally satisfiable w.r.t. γ; Otherwise, $\pi = \bigwedge_{a\in\mathcal{B}\cap\mathcal{B}'} \nabla_a[\Phi^*_a \cup (\Phi'_a - \Phi''_a)]$, where $\Phi^*_a = \{\phi \circ_\gamma \phi' \mid (\phi,\phi') \in \Phi_a *_\gamma \{\bigvee \Phi'_a\}\}$, and $\Phi''_a = \{\phi' \in \Phi'_a \mid \text{there exists a } \phi \in \Phi^*_a \text{ s.t. } \phi \mapsto_\gamma \phi'\}$.
3. $(\bigvee \Phi) \circ_\gamma (\bigvee \Phi') = \bigvee\{\phi \circ_\gamma \phi' \mid (\phi,\phi') \in \Phi *_\gamma \Phi'\}$.

For Rule 2, the first case of π is for the purpose of the conjunction property: $\nabla_a\Phi \wedge \nabla_a\Phi' \Leftrightarrow \nabla_a[\Phi \wedge (\bigvee \Phi') \cup \Phi' \wedge (\bigvee \Phi)]$. For second case of π, we explain Φ^*_a and $\Phi'_a - \Phi''_a$. Recall that $\bigvee \Phi_a$ is the belief of agent a, and each $\phi \in \Phi_a$ is a possibility for agent a. When there exist possibilities of Φ'_a that are consistent with the new belief $\bigvee \Phi'_a$, we revise them with the new belief. Also, among all the new possibilities, we remove those which are strongly entailed by an element of Φ^*_a, getting $\Phi'_a - \Phi''_a$.

Acknowledgements

We are grateful to anonymous reviewers whose comments and suggestions have helped us improve the paper.

References

Aucher, G., and Bolander, T. 2013. Undecidability in epistemic planning. In *IJCAI*, 27–33.

Baral, C.; Bolander, T.; van Ditmarsch, H.; and McIlrath, S. 2017. Epistemic planning (dagstuhl seminar 17231). *Dagstuhl Reports* 7(6).

Belle, V.; Bolander, T.; Herzig, A.; and Nebel, B. 2023. Epistemic planning: Perspectives on the special issue. *Artificial Intelligence* 316:103842.

Bolander, T., and Andersen, M. B. 2011. Epistemic planning for single and multi-agent systems. *Journal of Applied Non-Classical Logics* 21(1):9–34.

Bolander, T.; Dissing, L.; and Herrmann, N. 2021. Del-based epistemic planning for human-robot collaboration: Theory and implementation. In *KR*, 120–129.

Bolander, T. 2017. A gentle introduction to epistemic planning: The del approach. *arXiv preprint arXiv:1703.02192*.

Buckingham, D.; Scheutz, M.; Son, T. C.; and Fabiano, F. 2024. Action language ma* with higher-order action observability. In *KR-2024*, 210–220.

Cooper, M. C.; Herzig, A.; Maffre, F.; Maris, F.; and Régnier, P. 2016a. A simple account of multi-agent epistemic planning. In *ECAI*, 193–201.

Cooper, M. C.; Herzig, A.; Maffre, F.; Maris, F.; and Régnier, P. 2016b. Simple epistemic planning: Generalised gossiping. In *ECAI*, 1563–1564.

Cooper, M. C.; Herzig, A.; Maffre, F.; Maris, F.; Perrotin, E.; and Régnier, P. 2021. A lightweight epistemic logic and its application to planning. *Artificial Intelligence* 298:103437.

Dissing, L., and Bolander, T. 2020. Implementing theory of mind on a robot using dynamic epistemic logic. In *IJCAI*, 1615–1621.

Engesser, T.; Herzig, A.; and Perrotin, E. 2024. Towards epistemic-doxastic planning with observation and revision. In *AAAI-2024*, 10501–10508.

Fabiano, F.; Burigana, A.; Dovier, A.; and Pontelli, E. 2020. Efp 2.0: A multi-agent epistemic solver with multiple e-state representations. In *ICAPS*, volume 30, 101–109.

Fang, B., and Lin, F. 2024. Heuristic strategies for accelerating multi-agent epistemic planning. In *KR-2024*, 1912–1920.

Hales, J.; French, T.; and Davies, R. 2012. Refinement quantified logics of knowledge and belief for multiple agents. In *Advances in Modal Logic*, volume 9, 317–338.

Hu, G.; Miller, T.; and Lipovetzky, N. 2022. Planning with perspectives–decomposing epistemic planning using functional strips. *Journal of Artificial Intelligence Research* 75:489–539.

Hu, G.; Miller, T.; and Lipovetzky, N. 2023. Planning with multi-agent belief using justified perspectives. In *ICAPS*, volume 33, 180–188.

Kominis, F., and Geffner, H. 2015. Beliefs in multiagent planning: From one agent to many. In *ICAPS*, 147–155.

Le, T.; Fabiano, F.; Son, T. C.; and Pontelli, E. 2018. EFP and PG-EFP: epistemic forward search planners in multi-agent domains. In *ICAPS*, 161–170.

Lin, F. 2004. Discovering state invariants. *KR* 4:536–544.

Lipovetzky, N.; Muise, C.; and Geffner, H. 2016. Traps, invariants, and dead-ends. In *ICAPS*, volume 26, 211–215.

Löwe, B.; Pacuit, E.; and Witzel, A. 2011. DEL planning and some tractable cases. In *Proceedings of the Third International Workshop on Logic, Rationality, and Interaction*, 179–192.

Muise, C. J.; Belle, V.; Felli, P.; McIlraith, S. A.; Miller, T.; Pearce, A. R.; and Sonenberg, L. 2015. Planning over multi-agent epistemic states: A classical planning approach. In *AAAI*, 3327–3334.

Muise, C.; Belle, V.; Felli, P.; McIlraith, S.; Miller, T.; Pearce, A. R.; and Sonenberg, L. 2022. Efficient multi-agent epistemic planning: Teaching planners about nested belief. *Artificial Intelligence* 302:103605.

Satoh, K. 1988. Nonmonotonic reasoning by minimal belief revision. In *Proc. the First International Conference on Fifth Generation Computer Systems*.

Thielscher, M. 2017. Gdl-iii: A description language for epistemic general game playing. In *IJCAI*, 1276–1282.

To, S. T.; Son, T. C.; and Pontelli, E. 2011. Contingent planning as and/or forward search with disjunctive representation. In *ICAPS*, 258–265.

van Ditmarsch, H.; van der Hoek, W.; and Kooi, B. P. 2007. *Dynamic epistemic logic*. Springer.

van Ditmarsch, H. 2001. Knowledge games. *Bulletin of Economic Research* 249–273.

Wan, H.; Fang, B.; and Liu, Y. 2021. A general multi-agent epistemic planner based on higher-order belief change. *Artificial Intelligence* 301:103562.

Winslett, M. 1988. Reasoning about action using a possible models approach. In *AAAI*.

Yu, Q.; Wen, X.; and Liu, Y. 2013. Multi-agent epistemic explanatory diagnosis via reasoning about actions. In *IJCAI*, 1183–1190.