A Complete Epistemic Planner without the Epistemic Closed World Assumption

Hai Wan*, Rui Yangbps, Liangda Fangb, Yongmei Liub, and Huada Xua
bSchool of Software, Sun Yat-sen University, Guangzhou, China
aDept. of Computer Science, Sun Yat-sen University, Guangzhou, China
{wanhai,ymliu}@mail.sysu.edu.cn, {yangr7,fangld,xuhuada}@mail2.sysu.edu.cn

Abstract
Planning with epistemic goals has received attention from both the dynamic logic and planning communities. In the single-agent case, under the epistemic closed-world assumption (ECWA), epistemic planning can be reduced to contingent planning. However, it is inappropriate to make the ECWA in some epistemic planning scenarios, for example, when the agent is not fully introspective, or when the agent wants to devise a generic plan that applies to a wide range of situations. In this paper, we propose a complete single-agent epistemic planner without the ECWA. We identify two normal forms of epistemic formulas: weak minimal epistemic DNF and weak minimal epistemic CNF, and present the progression and entailment algorithms based on these normal forms. We adapt the PrAO algorithm for contingent planning from the literature as the main planning algorithm and develop a complete epistemic planner called EPK. Our experimental results show that EPK can generate solutions effectively for most of the epistemic planning problems we have considered including those without the ECWA.

1 Introduction
Planning with epistemic goals has received attention from both the dynamic logic and planning communities. Bolander and Andersen [2011] formalized multi-agent epistemic planning based on dynamic epistemic logic [van Ditmarsch et al., 2007] and showed that it is undecidable in general. Further, Aucher and Bolander [2013] showed that multi-agent epistemic planning is undecidable in the presence of only purely epistemic actions. Meanwhile, Yu et al. [2013] identified two important decidable fragments of multi-agent epistemic planning. Recently, Kominis and Geffner [2015], and Muise et al. [2015] showed how to exploit classical planning to solve restricted versions of multi-agent epistemic planning problems. There are earlier works on single-agent epistemic planning. Herzig et al. [2003] proposed a framework for epistemic planning where knowledge bases (KBs) are expressed as positive epistemic formulas, and showed how progression, regression and plan generation can be achieved in their framework. Bienvenu et al. [2010] identified two normal forms of epistemic formulas called $S_5$-CNF and $S_5$-DNF, and showed that progression is tractable when the KB is in $S_5$-DNF, and entailment is tractable when the KB is in $S_5$-CNF. However, the above works are only theoretical studies. Petrick and Bacchus [2002; 2004] presented a first-order epistemic planning system PKS based on the idea of progression. The kinds of knowledge they consider include literal knowledge, knowing-whether knowledge, and exclusive-or knowledge. However, their progression and reasoning algorithms are both incomplete, and hence their planner is incomplete.

In the single-agent case, under the epistemic closed-world assumption (ECWA) [Herzig et al., 2003], that is, if I cannot prove that I know $\phi$, then I don’t know $\phi$, epistemic planning can be reduced to contingent planning [Peot and Smith, 1992]. Bertoli et al. [2006] used binary decision diagrams [Bryant, 1986] to represent knowledge bases and proposed a planner MBP based on symbolic model checking. To et al. [2011] used minimal DNF formulas to represent knowledge bases, and proposed an algorithm PrAO with pruning techniques used in AND/OR forward search to solve contingent planning problems.

However, in some epistemic planning scenarios, it is inappropriate to make the ECWA. To illustrate this, consider the following modified wolf-sheep-cabbage example. Every Monday, a farmer has to take at least two objects of a wolf, a sheep and a basket of cabbages to cross a river. The wolf will eat the sheep, and the sheep will eat the cabbages, if the farmer is not with them. What is a plan for the farmer to cross the river? The initial knowledge base of the planning problem can be represented as $\mathbf{K}(wi \land si) \lor \mathbf{K}(si \land ci) \lor \mathbf{K}(wi \land ci)$, where $wi$ denotes that a wolf is on the left bank of the river initially. Also, $si$ and $ci$ are similar. Although the initial KB does not entail any of $\neg \mathbf{K}si$, $\neg \mathbf{K}ci$, and $\neg \mathbf{K}ci$, we cannot draw the conclusions $\neg \mathbf{K}ci$, $\neg \mathbf{K}si$, and $\neg \mathbf{K}ci$.

In general, the following are some scenarios where it is inappropriate to make the ECWA. Firstly, the agent is not fully introspective; hence the agent is not able to infer what she does not know from an explicit representation of what she knows. Secondly, the agent wants to devise a generic plan
that can be used for a wide range of situations. Thirdly, the agent is doing planning for the purpose of intention recognition; hence the agent is not sure about the observed agent’s mental state and can only make reasonable assumptions about what the observed agent knows and does not know.

In this paper, we propose a complete single-agent epistemic planner without the epistemic closed-world assumption. We begin with a theoretical formulation of single-agent epistemic planning without the ECWA. Then we identify two normal forms of epistemic formulas: weak minimal epistemic DNF, in short WM-EDNF, and weak minimal epistemic CNF, in short WM-ECNF. We present the proposition and entailment algorithms based on the above representations, and show that the entailment algorithm is tractable when the KB is in WM-EDNF and the query is in WM-ECNF. Next we adapt the PrAO algorithm [To et al., 2011] as the main planning algorithm and develop a complete epistemic planner called EPK. Our experimental results show that EPK can generate solutions effectively for most of the epistemic planning problems we have considered including those without the ECWA.

2 Preliminaries

In this section, we first introduce propositional logic and two notions: minimal DNF/CNF and strong entailment. Then, we extend the above notions to epistemic logic, and identify two forms of epistemic formulas: WM-EDNF which represents the KBs and WM-ECNF which represents the preconditions of actions and the goals of problems. Finally, we give the definitions of ontic actions and sensing actions, and the model-theoretic definition of progression wrt actions.

2.1 Propositional logic

Throughout this paper, we fix \( \mathcal{X} = \{x_1, \ldots, x_n\} \) be a finite set of propositions. The propositional language \( L_{prop} \) is generated from \( \mathcal{X} \), the usual connectives and two boolean constants \( \top \) and \( \bot \). A state is a subset of \( \mathcal{X} \), e.g., \( \{x_1, x_2\} \) is a state where both \( x_1 \) and \( x_2 \) hold and other propositions do not. A literal is either a proposition \( x \) or its negation \( \neg x \). \( \overline{l} \) denotes the complement of a literal \( l \). For a set of literals \( L, \overline{L} = \{l \mid l \in L\} \). We call \( \varphi \) a term (resp. clause) if it is in the form of \( \bigwedge_i l_i \) (resp. \( \bigvee_i l_i \)) where every \( l_i \) is a literal. Term (resp. clause) is often represented as a set of literals. So the union of two terms and the difference of two terms can be considered as terms. For example, suppose that \( t_1 = x_1 \land \neg x_2 \) and \( t_2 = \neg x_2 \land x_3 \). Then \( t_1 \cup t_2 = x_1 \land \neg x_2 \land x_3 \) and \( t_1 \setminus t_2 = x_1 \). A DNF (resp. CNF) \( \varphi \) is form of \( \bigvee_i \psi_i \) (resp. \( \bigwedge_i \psi_i \)) where every \( \psi_i \) is a term (resp. clause).

Definition 2.1 (Minimal DNF/CNF) A DNF (resp. CNF) \( \varphi \) whose form is \( \bigvee_i \psi_i \) (resp. \( \bigwedge_i \psi_i \)) is minimal if for any two distinct terms (resp. clauses) \( \psi_i \) and \( \psi_j \), \( \psi_i \notin \psi_j \) and \( \psi_j \notin \psi_i \).

Obviously, we can transform a DNF (resp. CNF) \( \varphi \) to an equivalent minimal one, denoted by \( min(\varphi) \), by removing the redundant terms (resp. clauses).

Definition 2.2 Let \( \varphi \) and \( \varphi' \) be two DNFs where \( \varphi = \bigvee_i \psi_i \) and \( \varphi' = \bigvee_j \psi'_j \). We say that \( \varphi \) strongly entails \( \varphi' \), denoted by \( \varphi \implies \varphi' \), if for all \( \psi_i \), there exists \( \psi'_j \) s.t. \( \psi_i \models \psi'_j \).

Proposition 2.1 Let \( \phi \) and \( \phi' \) be two DNFs. If \( \phi \implies \phi' \), then \( \phi \models \phi' \).

The above proposition says that if \( \phi \) strongly entails \( \phi' \), then \( \phi \) entails \( \phi' \), but not vice versa. For example, let \( \phi = a \land \neg a \) and \( \phi' = b \lor \neg b \). Since \( \phi \equiv \phi' \equiv \top \), we get that \( \phi \models \phi' \). But \( \phi \not\models \phi' \) because \( a \not\models b \) and \( a \not\models \neg b \). Similarly, we can extend the concept to CNFs.

Definition 2.3 Let \( \phi \) and \( \phi' \) be two CNFs where \( \phi = \bigwedge_i \psi_i \) and \( \phi' = \bigwedge_j \psi'_j \). We say that \( \phi \) strongly entails \( \phi' \), denoted by \( \phi \implies \phi' \), if for all \( \psi'_j \), there exists \( \psi_i \) s.t. \( \psi_i \models \psi'_j \).

Proposition 2.2 Let \( \phi \) and \( \phi' \) be two CNFs. If \( \phi \implies \phi' \), then \( \phi \models \phi' \).

2.2 Epistemic logic

Objective facts can be represented as propositional formulas. In epistemic planning, we model what the agent knows about the current state, which is an subjective formula, in epistemic logic. We will illustrate the syntax and semantics of epistemic logic \( L_K \) as follows.

Definition 2.4 The language of epistemic logic \( L_K \) is defined as follows using BNF notation:

\[
\Phi ::= K\phi \mid \Phi \land \Psi \mid \neg \Phi,
\]

where \( \phi \in L_{prop} \), and \( \Phi, \Psi \in L_K \).

For distinguishing two kinds of formulas, we use capital letters to denote the epistemic formulas, i.e., \( \Phi, \Psi \), etc. The propositional formulas are denoted by lowercase letter, i.e., \( \phi, \psi \), etc.

Definition 2.5 (Epistemic state) An epistemic state, in short e-state, is a finite non-empty set of states.

Intuitively, the agent considers every state in an epistemic state is possible.

Definition 2.6 Let \( M \) be an e-state. We interpret formulas in \( L_K \) as follows:

- for \( \phi \in L_{prop} \), \( M \models K\phi \) if for every \( s \in M \), \( s \models \phi \);
- \( M \models \Phi \land \Psi \) if \( M \models \Phi \) and \( M \models \Psi \);
- \( M \models \neg \Phi \) if \( M \not\models \Phi \).

We let \( Mod(\Phi) = \{M \subseteq 2^X \mid M \neq \emptyset \text{ and } M \models \Phi\} \), which is the set of e-states satisfying \( \Phi \). We remark that \( Mod(\bot) = \emptyset \) since there are no e-states satisfying \( \bot \).

Let \( \Phi \) be an epistemic formula. We say that \( \Phi \) is an epistemic literal if it is in the form of \( K\phi \) or \( K\neg \phi \) where \( \phi \in L_{prop} \).

We also say that \( \Phi \) is an epistemic term (resp. clause) if it is in the form of \( K\psi \land \bigwedge_i K\eta_i \) (resp. \( K\psi \lor \bigvee_i K\eta_i \)). Throughout this paper, we require \( \psi \) and all \( \eta_i \)’s are DNF (resp. CNF) for an epistemic term (resp. clause).

Definition 2.7 (Epistemic DNF/CNF) An epistemic formula \( \Phi \) is an epistemic DNF (resp. CNF), in short EDNF (resp. ECNF), if \( \Phi \) is in the form of \( \bigvee_i \Psi_i \) (resp. \( \bigwedge_i \Psi_i \)) where \( \Psi_i \) is an epistemic term (resp. clause).

Definition 2.8 (Separable)

- An epistemic term \( K\psi \land \bigwedge_i K\eta_i \) is separable if \( \eta_i \models \psi \) for each \( i \).
We use

For space limitations, we omit a similar notion weak minimal

to a WM-EDNF (resp. WM-ECNF) equivalently. Every epistemic formula can be trans-

Definition 2.9 (Strong entailment) Let Φ and Φ′ be two separable epistemic terms where

Definition 2.10 (Weak minimal EDNF) A separable EDNF

Definition 2.11 (Ontic action) An ontic action a_o is a pair ⟨pre, eff⟩, where

Proposition 2.3 Let Φ and Φ′ be two separable EDNFs. If

Now, we can define weak minimal EDNFs.

Definition 2.12 (Progression wrt ontic actions) Let Φ be an epistemic formula, a_o an action executable in Φ. Then a pre, eff, denoted by prog(Φ, a_o), is a progression of Φ for a_o if

We give the definition of sensing actions below.

Definition 2.13 (Sensing action) A sensing action a_s is a pair ⟨pre, res⟩, where

3 Epistemic planning

In this section, we introduce the definitions of epistemic planning problem and its solution.

Definition 3.1 (Epistemic planning problem) An epistemic planning problem P is a tuple ⟨O, S, I, G⟩, where

- O: a set of ontic actions;
- S: a set of sensing actions;
- I: an epistemic formula specifying the initial KB;
- G: an epistemic formula specifying the goal formula.

O and S are separate, i.e., O ∩ S = ∅.

Throughout this paper, we require I to be a WM-EDNF and G to be a WM-ECNF.

Example 1 We now formalize the example from the introduction as follows. Here, we only give several ontic actions about that the farmer moving from the right bank to the left bank. The other ontic actions are similar. We let so denote that the sheep is alive, ce denote that the cabbages are not eaten, d denote that the boat can load two objects (excluding the farmer), and fl, wl, sl and cl denote that the farmer, the wolf, the sheep or the cabbages are on the left bank of the river respectively.

1. left: the farmer moves to the left bank without taking anything.
   - pre(left) ≡ K¬fl
   - eff(left) = {K¬fl, {wi, si, sa, ¬wl, ¬sl}, {¬sa}}, {si, sa, ci, cc, sl, cl}, {¬ce}}, {wi, sa, ci, cc, ¬sl, ¬cl, ¬ce}}

2. wLeft: the farmer tries to take the wolf to the left bank.


Let follows:

\[ \text{inductively as follows:} \]

\[ \text{problem where} \ P \ \text{as follows.} \]

\[ \text{senseD} = 4. \]

\[ \text{wrt if trees.} \]

\[ \text{undefined} = \text{pre} = (\Phi \ \text{and} \ \Phi) = \text{res} \text{si} \]

\[ \text{si} \rightarrow \text{sa} \land \text{sl} \land (\text{wi} \rightarrow \text{wl}) \land \text{Kd} \land \text{Kd} = \text{d}. \]

\[ \text{G} = \text{K}((\text{ci} \rightarrow \text{ce} \land \neg\text{cl}) \land (\text{si} \rightarrow \text{sa} \land \neg\text{sl}) \land (\text{wi} \rightarrow \neg\text{wl})). \]

We formalize the notion of conditional plans through action trees as follows.

**Definition 3.2 (Action tree)** Let \( P \) be an epistemic planning problem where \( P = \langle O, S, I, G \rangle \). An action tree \( T \) is defined inductively as follows:

- \( e \): the empty tree
- \( a_o; T' \), where \( a_o \in O \), and \( T' \) is an action tree
- \( a_s; (T^+ | T^-) \), where \( a_s \in S \) and \( T^+ \) and \( T^- \) are action trees.

Let us denote undefined by \textit{undef}. Now, we define the progression wrt action trees.

**Definition 3.3 (Progression wrt action trees)** Let \( \Phi \) be an epistemic formula, and \( T \) an action tree. The progression of \( \Phi \) wrt \( T \), denoted by \( \text{prog}(\Phi, T) \), is defined inductively as follows:

- \( \text{prog}(\Phi, e) = \{ \Phi \} \), if \( \Phi \neq \bot \);
- \( \text{prog}(\Phi, e) = \text{undefined} \), otherwise;
- \( \text{prog}(\Phi, a_o; T') = \{ \text{prog}(\text{prog}(\Phi, a_o), T') \} \), if \( \Phi \models \text{pre}(a_o) \) and \( \Phi \neq \bot \);
- \( \text{prog}(\Phi, a_s; T') = \text{undefined} \), otherwise;
- \( \text{prog}(\Phi, a_s; (T^+ | T^-)) = \text{prog}(\text{prog}(\Phi, a_s; T^+)) \cup \text{prog}(\text{prog}(\Phi, a_s; T^-)) \), if \( \Phi \models \text{pre}(a_s) \) and \( \Phi \neq \bot \) and \( \Phi \neq \text{Kres}(a_s) \) and \( \Phi \neq \bot \);
- \( \text{prog}(\Phi, a_s; (T^+ | T^-)) = \text{undefined} \), otherwise.

We define solutions to epistemic planning problems below, which are typically action trees.

**Definition 3.4 (Epistemic planning solution)** Let \( P \) be an epistemic planning problem where \( P = \langle O, S, I, G \rangle \). An action tree \( T \) is a solution to \( P \) if \( T \) is finite, \( \text{prog}(I, T) \) is defined, and for every epistemic formulas \( \Phi \) of \( \text{prog}(I, T) \), \( \Phi \) entails the goal formula \( G \).

![Figure 1: A solution to the example](Image 336x487 to 534x665)

**Example 2 (Example 1 continued)** The action tree, depicted in Figure 1, is a solution to this example. It states as follows. Firstly, the farmer senses whether a boat can load two objects. If the boat can load two objects, then the farmer will try to take the sheep and the cabbages to cross the river. Because the cabbages are eaten by the sheep if the farmer is not around, he tries to take the cabbages back to the left bank. Finally, he tries to take the wolf and the cabbages to the right bank. Otherwise, the farmer should obey the “eat rule”, and has to take only one object to cross the river until all the objects are on the right bank of the river.

### 4 Progression and Reasoning

In this section, we introduce three core parts of EPK: (1) progression wrt ontic actions and sensing actions; (2) determining whether a KB entails the precondition of an action; (3) checking whether two KBs are strongly equivalent.

#### 4.1 Progression

To update KBs according to the effects of ontic actions, we firstly transform a DNF \( \phi \) to an equivalent one \( \phi' \) such that for every conditional effect and every term of \( \phi' \), either the term entails the condition \text{con} of the conditional effect, or it entails \text{con}.

**Definition 4.1** Let \( t \) and \( t' \) be two terms. The extension of \( t \) wrt \( t' \), denoted by \( \text{ext}(t, t') \), is defined as

\[
\text{ext}(t, t') = \begin{cases} 
    t, & \text{if } t \models t' \text{ or } t \land t' \equiv \bot; \\
    t \land t' \lor \bigvee_{l \neq t \text{ and } t' \models l} (t \land \neg l), & \text{otherwise.}
\end{cases}
\]

Intuitively, we keep \( t \) if it entails \( t' \) or it entails the negation of \( t' \). Otherwise, we transform \( t \) to a DNF \( \text{ext}(t, t') \) such that for each term of \( \text{ext}(t, t') \), either it entails \( t' \) or it entails the negation of \( t' \).
Proposition 4.1 Let $t$ and $t'$ be two terms, and $\phi = \min(ext(t, t'))$. Then $\phi \equiv t$ and for every term $t^*$ of $\phi$, either $t^* \models t$ or $t^* \models \neg t'$ holds.

For a DNF $\phi = \bigvee_i t_i$, we can define a similar operation: $\text{ext}(\phi, t') = \min(\bigvee_i \text{ext}(t_i, t'))$.

Definition 4.2 Let $a_o$ an ontic action. We say a term $t$ is enabling for $a_o$ if for every conditional effect $\langle \text{con}, \text{lit} \rangle \in \text{eff}(a_o)$, either $t \models \text{con}$ or $t \models \neg \text{con}$ holds. We say a DNF $\phi$ is enabling for $a_o$ if every term of $\phi$ is enabling for $a_o$.

Proposition 4.2 Let $\phi$ be a DNF, $a_o$ an ontic action. We say a term $t$ is enabling for $a_o$ if for every conditional effect $\langle \text{con}, \text{lit} \rangle \in \text{eff}(a_o)$, either $t \models \text{con}$ or $t \models \neg \text{con}$ holds. We say a DNF $\phi$ is enabling for $a_o$ if every term of $\phi$ is enabling for $a_o$.

Proposition 4.3 Let $\psi = K\phi \land \bigwedge_i K\eta_i$ be a separate epistemic term, and $a_o$ an ontic action executable in $\Psi$, and $\psi' = K\psi' \land \bigwedge_i K\eta_i'$ where $\psi' = \text{update}(\psi, a_o)$ and $\eta_i' = \text{update}(\eta_i, a_o)$ for each $i$. $\psi'$ is a progression of $\psi$ wrt $a_o$.

Proposition 4.4 Let $\Phi = \bigvee_i \Psi_i$ be a separable epistemic executable in $\Phi$. Then $\text{prog}(\Phi, a_o) \equiv \bigvee_i \text{prog}(\Psi_i, a_o)$.

Even if $\Phi$ is a WM-EDNF, the progressed KB $\text{prog}(\Phi, a_o)$ may not be a WM-EDNF. But we can transform it to an equivalent WM-EDNF.

Next, we introduce the progression of separable epistemic terms wrt observation.

Definition 4.5 Let $\Psi = K\phi \land \bigwedge_i K\eta_i$ be a separate epistemic term, and $\phi \in L_{\text{prop}}$. The update of $\Psi$ wrt $\phi$, denoted by $\text{obs}(\Psi, \phi)$, is $K(\psi \land \phi) \land \bigwedge_i K\eta_i$.

After observing $\phi$, an agent remains her knowledge of $\psi$ and keeps considering $\eta_i$ possible if $\eta_i \models \phi$. The latter is counterintuitive, but it is valid. This is because the possible state satisfying $\eta_i$ may not satisfy $\phi$ when we do not make the ECWA.

Proposition 4.5 Let $\Phi = \bigvee_i \Psi_i$ be an EDNF, and $a_s$ a sensing action executable in $\Phi$. A pair $(\Phi^+, \Phi^-)$ is a progression of $\Phi$ wrt $a_s$ where $\Phi^+ = \bigvee_i \text{obs}(\Psi_i, \text{res}(a_s))$ and $\Phi^- = \bigvee_i \text{obs}(\Psi_i, \neg \text{res}(a_s))$.

By Proposition 4.5, we can implement progression wrt sensing actions. Similarly, we transform $\Phi^+$ (resp. $\Phi^-$) to a WM-EDNF when it is not a WM-EDNF.

4.2 Reasoning

Firstly, we introduce a simple property of propositional logic.

Proposition 4.6 Let $\phi = \bigvee_i \psi_i$ be a DNF and $\phi' = \bigwedge_i \psi'_i$ a CNF. Then $\phi \models \phi'$ iff for every $i$, we have $\psi_i \models \psi'_i$.

Proposition 4.7 Let $\Psi$ be a separable epistemic term whose form is $K\psi \land \bigwedge_i K\eta_i$ and $\Psi'$ a separable epistemic clause whose form is $K\psi' \lor \bigvee_i K\eta_i'$. $\Psi \models \Psi'$ iff one of the following conditions holds

1. $\psi \land \neg \psi'$ is inconsistent;
2. there exists $i$ s.t. $\neg \psi' \land \eta_i$ is inconsistent;
3. there exists $j$ s.t. $\psi \land \neg \eta_j'$ is inconsistent.

Proposition 4.8 Let $\Phi = \bigvee_i \Psi_i$ be an EDNF and $\Phi' = \bigwedge_j \Psi_j$ an ECNF. $\Phi \models \Phi'$ iff for every $i$, $j$, $\Psi_i \models \Psi_j$.

By Proposition 4.8, we can determine whether an action is executable in the current KB in polynomial time.

Theorem 4.1 Let $\Phi$ be an EDNF and $\Psi$ be an ECNF. Deciding whether $\Phi \models \Psi$ is tractable.

Next, we show how to judge the equivalence of two KBs.

Definition 4.6 (Strong equivalence) Let $\Phi = \bigvee_i \Psi_i$ and $\Phi' = \bigwedge_j \Psi_j'$ be two WM-EDNFs. We say $\Phi \equiv \Phi'$ if the following two conditions all hold:

1. for every $i$, there exists $j$ s.t. $\Psi_i \hookrightarrow \Psi'_j$;
2. for every $j$, there exists $i$ s.t. $\Psi'_j \hookrightarrow \Psi_i$.

Proposition 4.9 Let $\Phi$ and $\Phi'$ be two WM-EDNFs. If $\Phi \equiv \Phi'$, then $\Phi \equiv \Phi'$.

By Proposition 4.9, we can determine whether two WM-EDNFs are strongly equivalent. Here, we only give a sufficient condition, but not a sufficient and necessary condition. So, sometimes, two WM-EDNFs are equivalent, but not strongly equivalent.

5 Planning Algorithm

In this section, we briefly describe PrAO algorithm [To et al., 2011], which is the main planning algorithm of EPK. The algorithm is based on progress and entailment which have been introduced in the last section.

A node $n$ in a search graph is a knowledge base $\Phi_n$ with a state $\text{state}(n)$ including four tags: $\text{goal}$, $\text{dead}$, $\text{unexplored}$, $\text{explored}$ and an additional tag: $\text{connected}$. As depicted in Algorithm 1, Lines 1 to 5 check whether $\mathcal{Z}$ entails $\mathcal{G}$, if so, then return the empty plan; otherwise, initialize the search graph. Lines 6 to 15 construct the search graph until it finds a solution or guarantees that there are no solutions. Lines 7 to 10 select an $\text{unexplored}$ and $\text{connected}$ node $n$ in $\mathcal{N}$ for exploration. If no such node exists then return no solution. Line 11 expands the selected node and updates the graph accordingly. Lines 12 to 15 check the state of the initial node. If its state is $\text{goal}$, return $\text{BuildPlan}(\mathcal{N}, T, n_0)$ to generate a solution; if $\text{dead}$, return no solution. Otherwise, continue the loop.

For space limitations, we briefly introduce subroutine $\text{Explore}$ and the heuristic function used in Algorithm 1.
Algorithm 1: Plan(\(S, I, G\))

Input: An epistemic problem \(P = (\mathcal{O}, S, I, G)\)

Output: A solution if it guarantees goal achievement or NULL otherwise.

1. if \(I \models G\) then
   return an empty plan \(\epsilon\)
2. else
   Let \(n_0 = I\), state\((n_0) = \text{unexplored}, \text{connected}(n_0) = \text{true}\), \(T = \emptyset\) and \(N = \{n_0\}\)
3. while true do
   4. if \(\{n \in N\mid \text{state}(n) = \text{unexplored} \land \text{connected}(n) = \text{true}\} = \emptyset\) then
      return NULL
   5. Let \(n\) be the unexplored and connected node with a heuristic function in \(N\)
   6. Start \(\text{Explore}(n)\)
   7. if state\((n_0) = \text{goal}\) then
      return BuildPlan\((N, T, n_0)\)
   8. if state\((n_0) = \text{dead}\) then
      return NULL

\(\text{Explore}(n)\) can generate the children of \(n\) for actions executable in \(\Phi_n\) with loop detection. The application of an ontic action starting from \(n\) results in a new node, while applying sensing actions produces two new nodes, i.e., \(n\) is split amongst the observations. It is a simple idea that applying sensing actions produces two new nodes, \(I_n\) can generate the children of \(n\), while \(I\) is a solution tree for \(\mathcal{N}\) and \(T\) is a solution tree for \(\mathcal{P}\); otherwise, \(T\) is NULL.

6 Implementation and Experiments

Our planner EPK is implemented in C++, and our experiments were done on a PC with Intel i7-4700MQ (2.4GHz) CPU and 4GB RAM on Linux.\(^1\) Results are in the form of \(t(d/s)\), where \(t\), \(d\) and \(s\) denote the overall execution time, the depth of the solution tree and the number of nodes in the solution respectively. Usually, \(d\) and \(s\) are criteria for evaluation of the quality of a solution. “...” denotes out-of-memory or time-out (20 minutes limit). Heuristic, BFS, DFS and IDS stand for search with heuristic function, breadth-first search, depth-first search, and iterative-deepening search respectively.

We experimentally compare EPK with PKS, and use a testbed of five domains, including two domains taken from PKS, other domains taken from contingent planning. Some domains in PKS contain functions and numerical representation, which EPK can not handle. Two domains in PKS are as follows: bdts and unix series. In addition, other domains, doors, push and dispose are from classical contingent planning benchmarks.

Table 1 reports the comparative experimental results. We can observe that, generally speaking, our planner EPK are better than PKS in the performance except bdts. In detail, the heuristic function is not suitable for these domains, since this search method is slower than BFS and DFS in EPK. For domains bdts, unix series, PKS is faster than EPK, while the quality of epistemic planning solutions of EPK are better than PKS using BFS or DFS. For domains such as push, doors, dispose series, EPK is faster than PKS using BFS and PKS even can not solve push and doors series using DFS.

Table 2 shows that domains without the ECWA. Although heuristic search in domains such as block, farmer, medpks-2 and medpks-3 is slower than BFS and DFS, it is obvious that heuristic function is very important especially for cube series and ebts-2-5.

\(^1\) http://ss.sysu.edu.cn/%7ewh/epk.html

---

Table 1: Experimental results between PKS and EPK

<table>
<thead>
<tr>
<th>Problem</th>
<th>Heuristic</th>
<th>BFS</th>
<th>DFS</th>
<th>DFS</th>
<th>IDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>bdts-20</td>
<td>0.549</td>
<td>0.033</td>
<td>0.005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bdts-30</td>
<td>3.976</td>
<td>0.169</td>
<td>0.008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bdts-50</td>
<td>(59/60)</td>
<td>(59/60)</td>
<td>0.61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doors-2</td>
<td>0.810</td>
<td>0.010</td>
<td>0.028</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doors-5</td>
<td>34.518</td>
<td>0.049</td>
<td>0.027</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push-2</td>
<td>0.969</td>
<td>0.049</td>
<td>0.100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push-3</td>
<td>163.478</td>
<td>0.009</td>
<td>0.002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dispose-2-1</td>
<td>0.001</td>
<td>0.000</td>
<td>0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dispose-2-2</td>
<td>0.094</td>
<td>0.176</td>
<td>0.091</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dispose-3-1</td>
<td>1.213</td>
<td>0.099</td>
<td>0.002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cube-3</td>
<td>0.714</td>
<td>0.086</td>
<td>0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cube-5</td>
<td>8.424</td>
<td>0.086</td>
<td>0.001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Experimental results in EPK without the ECWA

<table>
<thead>
<tr>
<th>Problem</th>
<th>Heuristic</th>
<th>BFS</th>
<th>DFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>farmer</td>
<td>0.073(8/13)</td>
<td>0.043(8/13)</td>
<td>0.084(8/13)</td>
</tr>
<tr>
<td>block-3</td>
<td>1.777(12/28)</td>
<td>0.000(9/79)</td>
<td>0.011(9/79)</td>
</tr>
<tr>
<td>doors-2</td>
<td>0.132(5/6)</td>
<td>0.991(5/6)</td>
<td>0.001(5/6)</td>
</tr>
<tr>
<td>medpks-2</td>
<td>0.000(2/3)</td>
<td>0.000(2/3)</td>
<td>0.000(2/3)</td>
</tr>
<tr>
<td>medpks-3</td>
<td>0.001(2/3)</td>
<td>0.002(2/3)</td>
<td>0.001(2/3)</td>
</tr>
<tr>
<td>medpks-6</td>
<td>0.000(2/3)</td>
<td>0.000(2/3)</td>
<td>0.000(2/3)</td>
</tr>
<tr>
<td>ebts-2-2</td>
<td>0.014(3/3)</td>
<td>0.013(3/3)</td>
<td>0.013(3/3)</td>
</tr>
<tr>
<td>ebts-2-6</td>
<td>8.424(49/51)</td>
<td>28.671(5/7)</td>
<td>0.137(6/8)</td>
</tr>
</tbody>
</table>

---

1. http://ss.sysu.edu.cn/%7ewh/epk.html
7 Conclusions

We have introduced a general and effective approach to single-agent epistemic planning with deterministic actions without making the ECWA. We identify two compact representations: WM-EDNF and WM-ECNF. Based on the representations, we give the progression and entailment algorithms which are core modules in planning. Moreover, entailment is tractable when the KB is in WM-EDNF and the query is in WM-ECNF. We adapt the PrAO algorithm as the main planning algorithm, and our planner is complete. Our empirical evaluation shows that our planner is better than PKS in most of the epistemic planning problems we have tested. Also, EPK generates solutions effectively for most of the epistemic planning problems we have considered, especially those without the ECWA.

There are several possible directions for future study. Firstly, we want to improve the efficiency of our planner via designing a compact yet efficient and scalable representation and developing better heuristic functions, such as graph heuristics. Secondly, we want to extend the definition of ontic actions so that conditional effects can contain not only propositional literals but also epistemic literals, and give the semantics for these actions. Thirdly, we would also like to extend this work to multi-agent cases.

8 Acknowledgments

The authors would like to thank Andreas Herzig for his useful suggestions. Hai Wan thanks Guangzhou Science and Technology Project (No. 2013J4100058) for the support of this research. Liangda Fang and Yongmei Liu acknowledge the support of the National Natural Science Foundation of China under grant No. 61073053.

References


[To et al., 2011] Son Thanh To, Tran Cao Son, and Enrico Pontelli. Contingent planning as and/or forward search with disjunctive representation. In Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS-2011), pages 258–265, 2011.
