EFP AND PG-EFP:

EPISTEMIC FORWARD PLANNERS IN MULTI-AGENT DOMAINS

BY

FRANCESCO FABIANO, B.S.

Master Project Report

for the degree

Master of Science

Major Subject: Computer Science

New Mexico State University

Las Cruces New Mexico

April 2018

DEDICATION

I dedicate this work to my family.

ACKNOWLEDGMENTS

I would like to thank, Enrico Pontelli, Son Tran and Tiep Le for their support and patience. I am grateful to them for sharing their knowledge and for helping me to enrich my study in Artificial Intelligence.

VITA

January 5, 1994	Born in Parma, Italy
2013-2016	B.S., Università degli Studi di Parma, Parma, Italy

FIELD OF STUDY

Major Field: Artificial Intelligence

ABSTRACT

EFP AND PG-EFP:

EPISTEMIC FORWARD PLANNERS IN MULTI-AGENT DOMAINS

BY

FRANCESCO FABIANO, B.S.,

Master of Science

New Mexico State University

Las Cruces, New Mexico, 2018

Dr. Enrico Pontelli, Chair

In recent years, multi-agent epistemic planning has received attention from both dynamic logic and planning communities. This thesis presents two prototype epistemic forward planners, called Efp and PG-Efp, for generating plans in multiagent environments that differ from other epistemic planners recently developed. In particular Efp and PG-Efp can deal with unlimited nested beliefs, common knowledge, and are capable of generating plans with both knowledge and belief goals. Efp is simply a breadth first search planner while PG-Efp is an heuristic search based system.

The thesis also introduces the notion of epistemic planning graph (EPG) that could play an important role in the epistemic planning environment, as the planning graph in classical planning. The EPG is used to generate heuristics in PG-EFP. Moreover the thesis presents experimental evaluation that proves the usefulness of epistemic planning graphs.

Finally an evaluation of the planners, using benchmarks collected from the literature, and a discussion of the issues that affect the scalability and efficiency of the planners, thus identifies potentially directions for future work, are discussed.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

1 PLANNING

We now introduce the concept of *planning*, one of the most important branch of *Artificial Intelligence*.

As said in [39] AI[1] is the study of rational action which makes planning—i.e., the problem of finding a sequence of actions that will achieve one's goal—one of its most critical part.

There are different types of planning[2] but all of them share the same objective: given an initial configuration of the environment, find a sequence of permitted actions to reach the desired configuration of the same environment.

1.1 Basic Concepts

Since the planning problem is a really important topic it is been long studied and researched.

In the following paragraphs we will introduce the basic terminology and concepts related to the planning environment. The *Block World* domain, a common used example, will be used to explain in a more clear way all the features of the planning field.

Block World Domain. The Block World, due its simplicity, is one of the most used domain to explain the planning area baseline.

[1] Where "AI" stands for "artificial intelligence".

[2] e.g. classical, conformant, non-deterministic, epistemic etc.

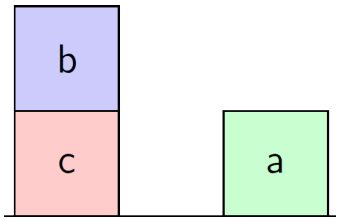


Figure 1: The World Block domain.

This domain consists of few simple elements:

- *Blocks* of the same size that can be placed either on the table or in the top of another block;
- a *mechanical arm* that can move the blocks and can tell if is holding or not one block.

Moreover there are some constraints that regulate the Block World:

- Two blocks cannot be placed on top of one single block;
- the mechanical arm can only hold, and therefore move, one block at the same time;
- a block can only be placed on top of a *clear* block—a block with no other block on top of it and that is not held by the mechanical arm—or on the table.

In particular this domain will be used to describe the key features of planning: *states*, *actions*, *planning problem*, *transition function*, *agents*, and *solutions*.

Definition 1.1 (State)**.** A *state* of the domain is a configuration of the *world*[3] represented as conjunction of *ground fluents*.

Example 1.1. A state in the Block World domain, shown in Figure 1, is defined as follow: {`onTable_A`, `onTable_C`, `onC_B`, `Clear_Arm`}. In this description some fluents as ¬`onTable_B` or ¬`onA_C` are omitted for the sake of readability.

Definition 1.2 (Action)**.** An *action* in planning is an operation, made by some agent, that changes the actual world or its perception. Actions can have *executability conditions* that express when an action is, as the name suggests, executable and when is not.

Example 1.2. Given the state in Figure 1 we can give an example of executable and not-executable action:

- As executable action is `take`(`B`). This action states that mechanical arm takes the block `B` and keeps it (As shown in the left configuration in Figure 2). The executability conditions of this action are {`ClearArm`, ¬`onB_A`, ¬`onB_C`} which are respected in the current state.

- Instead as example of not-executable action we can use `take`(`C`) which means that mechanical arm takes the block `C`. This action is not executable because the block `C` is not clear, more formally the executability conditions of this

[3]We refer to world as the environment described by the domain.

action are {ClearArm, ¬onC_A, ¬onC_B}. Since ¬onC_B is not respected the action cannot be executed.

Definition 1.3 (Planning Problem)**.** A *planning problem* is a tuple $\langle D, I, G \rangle$ where:

- D is an *action domain* expressed in some language;
- I is a set of state (Definition 1.1) of the domain—called *Initial state*—that describe the (possible) starting configurations of the world. The example in Figure 2 is described as {onTable_A, onTable_C, holding_B, ¬ClearArm, ¬onA_B, ¬onA_C, ¬onB_A, ¬onB_C, ¬onC_A, ¬onC_B, ¬holding_A, ¬holding_C}. From now on we will omit, when possible, the false fluents from the states representation. We will assume that what is not specify in the state is false in the world.
- G is a set of state of the domain, called *Goal state*, that describe the desired configurations of the domain. The example in Figure 2 is described as {onTable_A, onA_B, onB_C, ClearArm}.

Definition 1.4 (Transition Function)**.** A *transition function* Φ is a function that, given a starting state and an action, returns a set of states[4] in which the world can be after the execution of the action in the starting state. More formally

[4] A set and not a single state because the actions sometime can be non-deterministic.

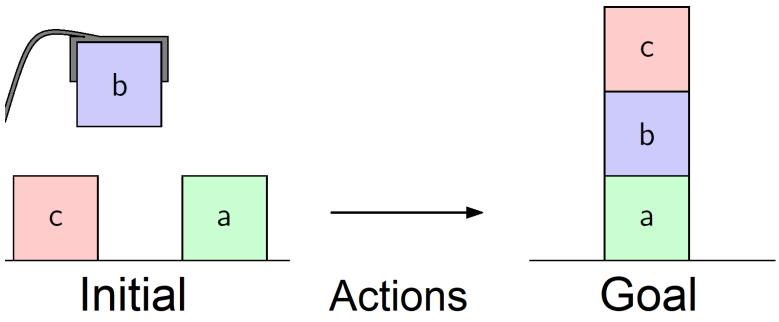


Figure 2: The Planning Problem in the World Block domain.

$\Phi : \Sigma \times A \to 2^{\Sigma}$ where Σ is the set of all the possible states and A the set of all the possible actions in the domain. If an action a is not executable in a state s then we will have $\Phi(a, s) = \emptyset$.

In [41] an *agent* is described as a persistent software entity dedicated to a specific purpose. We can translate this definition in the planning field just defining the purpose. In the planning area the purpose of an agent will be reaching the goal. Consequently an agent in planning is the element of the domain that execute[5] the actions and change the states trying to reach the goal.

Example 1.3. In the Block World domain the agent is the simulation of the mechanical arm that moves the blocks to find the desired configuration.

Definition 1.5 (Solution)**.** Finally a *solution* for the planning problem $\langle D, I, G \rangle$ is a sequence of action $[a_1, \ldots, a_n]$ such that:

[5]Execution in a software environment is just a simulation of the actions.

- a_1 is executable in every state s belonging to I,
- a_i is executable in every state s belonging to $\Phi_D(a_{i-1}, \ldots, \Phi_D(a_1, I))$ for $i = 2, \ldots, n$
- G is true in every state s belonging to $\Phi_D(a_n, \ldots, \Phi_D(a_1, I))$.

Even if these terminology represents the basis for the entire planning area, as said before there are different types of planning problems. A brief explanation of all the most common category of planning problems will be presented next.

1.2 Planning Problem Domains

1.2.1 Classical Planning

The most known and studied "family" of domain for planning problems is the one that is called *classical*. This category is often seen as basis for all the other kinds of planning that usually bring more complexity with them. In fact is not unusual to reduce, when it is possible, problems from more complex domains to the classical ones.

To reduce one problem into a classical planning means to elaborate the problem itself so it can be solved by a classical *planner*.

Definition 1.6 (Planner)**.** A planner, or sometimes a *solver*, is a computer program that computes the solution of any given planning problem with compatible

domain[6].

For most of its early life in the 60's and 70's, the field of automated planning was concerned with ways in which the problem of creating long-term plans for achieving goals could be formulated, such that solving problems of non-trivial size, would be computationally feasible. The types of planning that arose from this early work, is what is known today as classical planning [7].

Classical planning, as defined in [22], imposes several restrictions on the planning problem, namely the domain has to be *fully observable*, *deterministic* and *static*.

A *static* problem is represented by a domain that is not modified by elements that are external to the domain itself.

A domain is *deterministic* when for each state s and action a $\Phi(a, s)$ has at most one element, that is there is no ambiguity in which state will be the world after the execution of the action. Determinism also implies that the plan will be in one and only one state at each step of computation of the planning. This condition implies that as soon a goal state is found this state is the solution of the problem (this is not true in conformant planning as we will see in §1.2.2).

Fully observable means that the agent that is in the domain knows the com-

[6]e.g. a single-agent planner cannot solve a multi-agents problem.

plete description of the world, that is she knows the state of every fluent in the domain.

Most of the time to maintain its simplicity the classical planning domains are single-agent: domains where only one agent can perform the actions and has to reach the goal.

A good example of classical planning can be, once again, the World Block domain described in Section §1.1. Here the single agent, the mechanical arm, knows everything about the world (if a block or itself is clear or not for example) and every action that the arm does has only one possible outcome.

There are a lot of different approaches, and therefor planners, to solve classical planning problems, each one useful for specific situations. To improve the performance of these planners different representation of the domain, like *trees* (Figure 3), have been developed. With them other optimizations, like heuristics, have been studied. We will discuss this in more detail in Section §1.3.

Since the classical planning problems are fairly easy in respect to other type of planning problems, the existing solver are the most efficient. For this reason sometimes is better elaborate one problem's domain to be compatible with a classical solvers.

Anyway sometimes reducing a domain can cause loss of expressiveness. De-

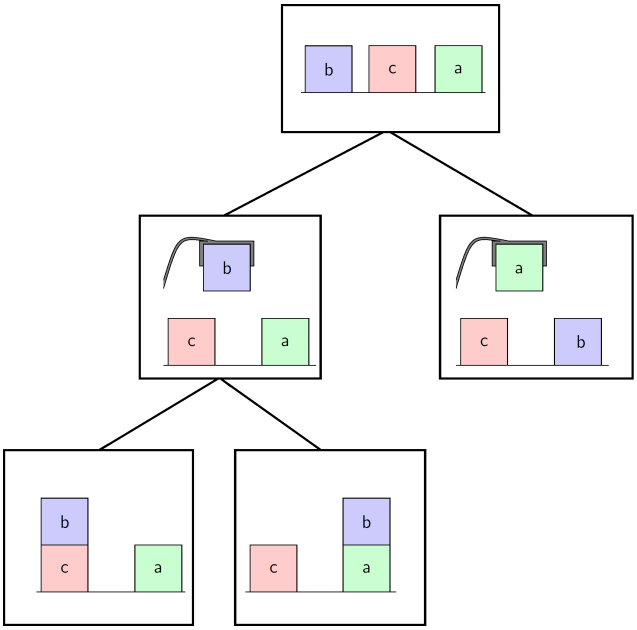


Figure 3: A planning tree for World Block domain.

pending on the domain, these losses can be acceptable or not. So when is not possible to reduce the problem to classical planning it is necessary to come up with a new planner.

1.2.2 Conformant Planning

Unfortunately the problems that we want to solve with the help of the planning method are not always as easy as the ones solvable by the classical planners. In particular sometimes the agent (or agents)[7] is not able to have complete information about the real world and neither is able to retrieve this information through *sensing* actions. This type of domain falls under the *conformant planning* prob-

[7] As we will see in §1.2.3.

lem. This brings to a difference between classical planning when it comes to the transition function Φ. In fact while in classical planning applying an action produces only one successor here we have a set of possible states.

As said in Definition 1.5 the solution for a problem must be true in all the reachable states. This means that the plan must be valid for all the possible configuration of the initial state.

A clarification about the action results is required. When it comes to conformant planning the uncertainty of the initial state is carried on with the actions. This means that the actions do not generate non-determinism, for example through *if-else* conditions, but just produce multiple possible states because they carry on the uncertainty of the state where they were executed. On the other hand when actions generate non-determinism we talk about *contingent* planning. To explain better this difference we will use two different description of the same domain, the well known *Vacuum Cleaner* [39].

Vacuum Cleaner Domain. In this domain (represented in Figure 4) an agent, the vacuum cleaner, has to clean two rooms from the dirt using a sequence of four actions "`Left`", "`Right`", "`Suck`", "`NoOperation`" that do what their names suggest (described in [39], Chapter 2).

- As example of domain that is contingent we use the domain with a slight

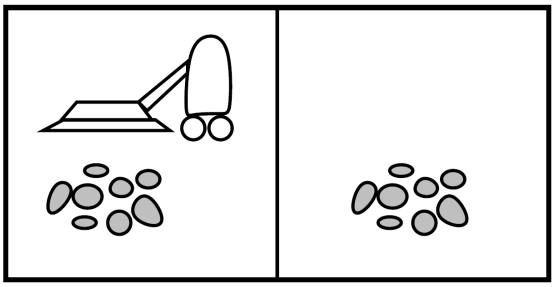


Figure 4: The Vacuum Cleaner Domain with both the room dirty.

modification of the action "*Suck*". When this action is applied to a room with dirt it cleans the room but sometimes clean the other room too and when applied to a clean room it sometimes deposits dirt on the carpet. A plan for the initial state as describe in Figure 4 is described by the *AND-OR tree* in Figure 5.

- Instead the example of conformant planning uses the normal interpretation of the actions. For this example though the vacuum cleaner cannot percept whether the rooms are dirty or clean. This implies that all the states where the agent is in the left room (accordingly to Figure 4) is a possible initial states and the successor state of the action "`Suck`" is shown in Figure 6. A solution for this problem is {`Suck`, `Right`, `Suck`}; in fact this sequence of action reach the goal from every possible initial state.

The planner presented in Section §3 admits not-complete initial description and consequently solves problems even in conformant domain. Even so as due the special data structure used in the planner the initial state without complete

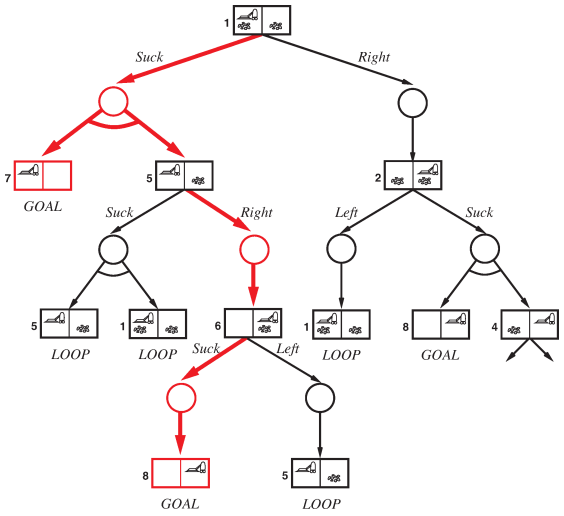


Figure 5: AND-OR tree for the Vacuum Cleaner with non-deterministic actions.

information brings a really incisive overhead.

1.2.3 Multi agents Planning

When we think about the real world it is clear that single-agent domains lacks in expressiveness. A lot of problems in reality require more than one agent in their domain. This family of planning problems is named *multi-agents planning* and deals with domains that contain more than a single independent agent.

Having more agents in the same domain can initially seem only a more efficient way to solve the problem, and sometimes is true (for example if we think as multi-agents as parallelization), but most of the time the concept of multiple agents adds complexity to the solver.

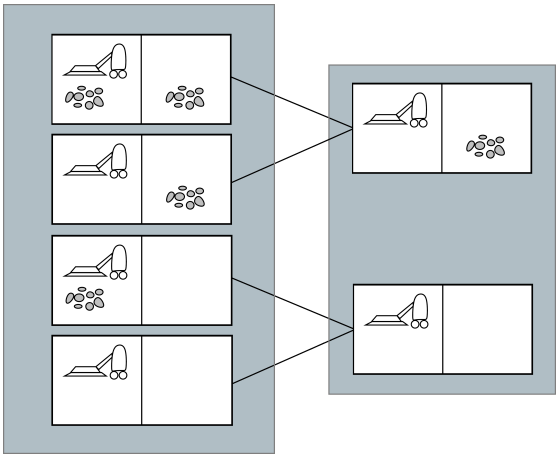


Figure 6: The execution of the action "`Suck`" in a conformant domain.

Inside the family of multi-agents problems different configurations are possible. Next we will list briefly some of these configurations that are mostly famous in the literature.

First of all different planning problems can be defined by the relation of the agents-goal. A simplified, but accurate enough for this thesis, subdivision is given in [9]

- **Not-deterministic**: Here each agent doesn't know which action and when the other agents will perform. This implies that she will not know in which state the world will be in the future.

- **Cooperative**: In this case the agents try to cooperate to reach the same goal.

- **Adversary**: Under this section of problems we can find competitive *games*.

Agents have opposite goals and try to reach their own trying to penalize the others'.

- **Overlapping Goals**: In this configuration the agents just happen, without willing it, to help each other to reach their own goal.

Another distinction is based on the type of communications between agents. We can simplify the subdivision described in [20, 27] in two different categories of communication:

- **Free**: In this type of planning problems the agents are allowed to share all their knowledge about the world to help each other in solving problem.
- **Privacy limited**: When a limit is inserted, usually for security reasons, the agents can share only certain information with the other. It also possible that some agent get to share specific types of information with only a subset of the other agents.

Finally in [15, 20] another distinction, based on the solving process is introduced; a planning system can therefore be:

- **Centralized**: In this type of planner a *master* agent coordinates the action of the others.
- **Decentralized**: Instead in this approach each agent act independently.

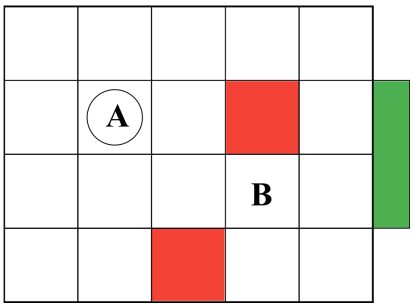


Figure 7: The Soccer domain.

Example 1.4. An example of multi-agents planning domain with cooperatives agents and without restriction about sharing information about the state of the world, is a variation of the *Soccer domain* presented in [31]. In this game the agents are placed in a single cell of the grid (Figure 7) and they can move following the compass directions (N, S, E, and W) or wait.

Agent `A` starts conventionally with the ball but every time that an agent try to move in already occupied she has to "pass" the ball to the agent that is in that cell.

The goal of this domain is to bring the ball inside the goal zone (green in Figure 7) without hitting the obstacles (red in Figure 7) in the fastest way possible.

So the agent are cooperating to reach the goal and they can share information about their next move to reach the goal easily.

1.3 Search Space Exploration

Previously we have defined what is a *planning system* and consequently a planner. This branch of AI usually explores problems and returns the list of actions that permits to modify the world in the desired way.

Since the problems that a planner is required to solve are usually derived by the real world, their *search space* can be too big or expensive to be explored. That is why different ways to represent the search space—i.e., *search tree*—and to explore it—e.g.,*Breadth-first search*, *Depth-first search* and *Best-first search* algorithms—have been largely studied and implemented.

We will explain the first two exploration techniques briefly since they are general approaches and do not need to be fully investigated for the comprehension of this thesis. We will then describe Best-first search more precisely, as such method is the one used for our planner optimization and it requires more details for its explanation.

1.3.1 Search Tree

When we talk about planning we are referring to a procedure that permits to simulate a real world problem.

This means that a planner should explore all the domain configurations and find the one, called the goal state (Definition §1.3), that satisfies certain proprieties.

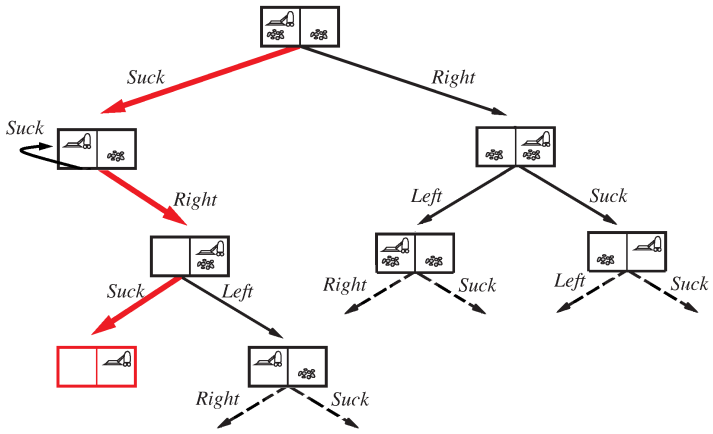


Figure 8: The search tree in the deterministic Vacuum Cleaner domain

The most common representation of a planning problem, and the one we will use from now on if not specified otherwise, is the *search tree*.

In this representation each possible state (Definition 1.1) is represented by a *node* of the tree and the solving process is the procedure that permits to explore the set of the states to find the goal.

Each node generates its children executing all the executable actions by applying the *transition function* (Definition 1.4) to generate new states. In Figure 8 we have a planning tree for the deterministic Vacuum Cleaner domain where given the initial state is {`inLeft`, ¬`cleanRight`, ¬`cleanLeft`} and the goal is {`inLeft`, `cleanRight`, `cleanLeft`}. A solution is found following the path marked in red.

1.3.2 Uninformed Search

As said previously the techniques to explore the search tree have been largely studied. Two of the most famous approach to explore the tree are Breadth-first search (*BFS*) and Depth-first search (*DFS*).

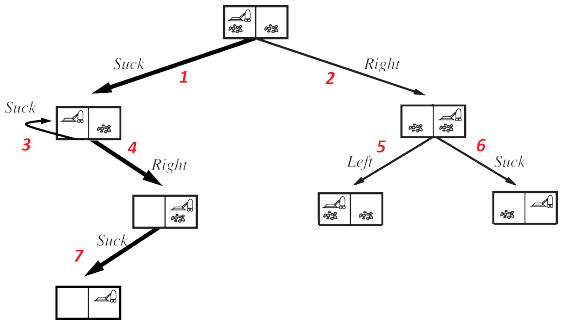
Both the algorithms explore the tree using patterns that rely on the structure of the tree rather then the information contained in the nodes, performing what is know as *uninformed search*.

Our planner makes use of BFS when it compute the solution without any optimization.

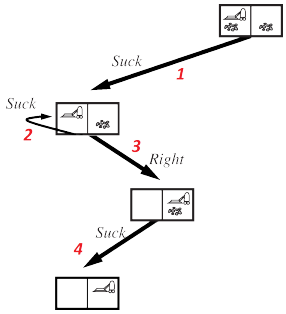
As the name suggests, Breadth-first explores the tree one layer at time, assuring to find always the shallowest solution. On the other side Depth-first explores the tree in depth. This means that the children of a node have priority respect to the other nodes in the same layer. A comparison of this two methods is given in Figure 9 where the tree in Figure 8 is explored with the two different approaches.

1.3.3 Informed Search

Unfortunately the search space for the majority of the planning problems is not small enough to be explored with uninformed search techniques. In fact most of the time this type of exploration on real domain requires a really large amount of time and memory. Moreover even when the exploration of the tree is not impossible generally using the uninformed techniques leads to a waste of resources.



(a) Execution of BFS.



(b) Execution of DFS.

Figure 9: Comparison of BFS and DFS[8] on the Vacuum Cleaner domain.

For these reasons a different approach, when it is possible[9], is really common in planning: the *informed* search.

This type of search uses information that are retrievable from the domain to score the states given them a priority of expansion. This approach performs the exploration in a "smart" way, i.e. it expands the node of the tree with the greatest[10] value. This means that the algorithm, called *Best-first*, will always expand the node that it thinks to be the closest to the solution.

It is clear that for Best-first to work well a good scoring technique is required. In fact, if the score given to the nodes does not actually reflects the quality of the states in the domain, this algorithm will perform even worst than the uninformed techniques. In the worst case scenario a bad scoring will lead Best-first to chose always the farther state from the solution.

[9]When the domain permits it.

[10]Or lowest depending how the priority order is specified.

We will refer at the "score" of the state as *heuristic* where heuristics stand for *strategies using readily accessible information to control problem-solving processes* [37], in our case planning problems.

1.4 Heuristics

The planning branch of AI is constantly under development and along with the introduction of new and more powerful solvers the domains are getting bigger and more complex.

That is because the aim of planning is to solve "real-world" problems that usually bring with them a huge amount of information and scenarios. Moreover as said in Section §1.2.1 also the type of domain can differ from problem to problem bringing additional complications.

That is why nowadays it is really important the study of heuristics in the planning field.

Definition 1.7 (Heuristic)**.** In the tree or graph setting, heuristics are methods for estimating the distance from a node to some goal node [28].

Heuristics—that can be extracted directly from the study of the domain or, more commonly, following general procedures—bring with them a significant decrease in resources needing. In fact solving a problem knowing the quality of each state can reduce the space of search significantly.

Of course this is true only when the heuristic can actually score the states of the domain in a meaningful way, assuring that the informed search is going "in the right direction".

To have a better idea of how an heuristic works we can look at the example, on the World Block domain (Section § 1.1), described in [40].

Example 1.5. First of all we introduce three concepts which are not included in the descriptions of the World Block

1. A block is in *final position* if it is on the table and it should be on the table in the goal configuration, or if it is on a block it should be on in the goal configuration and that block is in final position.

2. A block is in tower-deadlock position if it is not in final position and it is above a block it should be above in the goal configuration.

3. A block is in next-final position if it should be on the table, or if it should be on another block that it is currently clear and in final position.

The heuristics used for selecting the next action to apply will be:

- If a block can be moved to its final position, this should be done right away.

- If a block is in tower-deadlock position, then put it on the table.

- Suppose a block A is not in its final position, and it is on a block B that is in final position and should have a third block C on it. Suppose a block

D is on a block E that is not in its final position or should be clear in goal configuration. Then it is better to move A than to move D.

- Suppose a block A is on a block B that is in next-final position. Suppose a block C is on a block D that is not in next-final position, then it is better to move A than to move C.

- If a block is in its final position, do not move it.

- If a block is neither on the table nor in its final position and cannot be moved to its final position, do not move it to any place different from the table.

In this example the states are not directly scored but the proprieties described can be the base to create a scoring model.

1.4.1 Proprieties of Heuristics

Along with the idea that an heuristic must encode real and useful information, in relation to the problem, of the states there are other aspects that should be satisfied. In particular two proprieties are commonly required from an heuristic.

These two proprieties permit to general algorithms, that work using the "scoring" of the states, to have good performances.

Definition 1.8 (Admissibility)**.** An heuristic is *admissible* when for each node n

its approximated value is smaller than the actual cost from n to the goal. In a more formal way $h(n) \leq w(n, \text{goal})$ where $w(n, m)$ is the real cost from m to n.

Definition 1.9 (Consistency)**.** An heuristic is consistent when for each node n and its parent m the relation $(h(m) - h(n)) \leq w(n, m)$ is satisfied.

While the property 1.8, admissibility, is usually a mandatory requirement, to ensure the correct working of the algorithms that perform informed search, the consistency is in most case desirable since it does not affect the correctness of the algorithms.

1.4.2 Relaxed Domains

There is not only one way to retrieve heuristics from a problem, any intuition on the domain can be used as base for a different scoring method. There is though a common approach used to find heuristics.

In fact generally heuristics are derived from the *relaxed version* of the domain. This means that, given a domain and a problem, solving the same problem on a version of the domain with less restrictions can generate useful information about the quality of the states.

The generation of the relaxed domain can be done reasoning about the domain itself, as shown in Example 1.6 or using more general techniques as explained in Section §1.5.

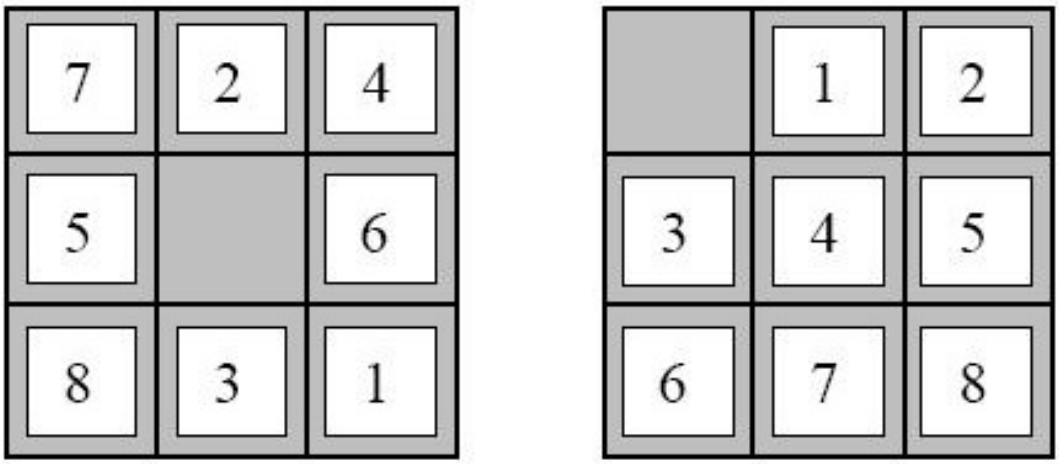


Figure 10: Examples of initial (left) and goal (right) states for Eight Puzzle.

Example 1.6. Given the famous domain *Eight-Puzzle*, where it is required to reach the desired configuration moving only one tile in the empty space per time, we can describe two heuristics from two different versions of the relaxed version of the problem. The initial and the goal configurations are shown in Figure 10.

- The first heuristic is obtained simply by considering the number of misplaced tiles. This simulates the possibility to move the tiles without any rule. Each misplaced tile will add one to the heuristic value of a certain configuration, in the initial state $h = 6$. The state that will be expanded in this case will be the one with the lowest heuristic value.

- The other possible heuristic is derived using the *Manhattan Distance*[11] of the tiles. This simulates the possibility to move the tiles in the grid, on cell

[11] Given two points M and P $manhattan(M, P) \equiv |M_x - P_x| + |M_y - P_y|$.

per time, without considering the other tiles. Even in this case the node with lowest h value is the best node to expand.

1.5 Planning Graph

Unfortunately not in all domains it is clear what is a good or what is a bad action and consequently what is a good heuristic. Moreover, when a planner is built, it is not usually used to solve problems only related to one domain. So even if the idea of heuristic is based on the quality of the actions, and therefore intuitively an heuristic should be derived looking into the domain details, this is not how the planners usually work. In fact the solvers commonly use general techniques to find an heuristic.

Our planner uses one of these techniques, called *planning graph*. This type of techniques, as we will see, are based on the *relaxed version of the problem*. This type of approach, even if different, is really similar to the one used in Example 1.6 where a relaxed version of the domain is used.

1.5.1 Planing Graph Structure

As said all of the heuristics based on the domain definition can suffer from inaccuracies. Instead the *Planning Graph* is a special data structure that can generate better heuristics. These heuristics can be applied to any search algorithm. Alternatively, we can search for a solution over the space formed by the planning graph,

using an algorithm called *GRAPHPLAN*. Intuitively a planning graph (Figure 11) is a directed graph organized into *levels*: first a level S_0 for the initial state, consisting of nodes representing each fluent that holds in S_0; then a level A_0 consisting of node for each ground action that might be applicable in S_0; then alternating levels S_i followed by A_i; until we reach a termination condition. More formally:

Definition 1.10 (Planning Graph)**.** Given a planning problem $P = \langle D, I, G \rangle$, the planning graph of P is an alternative sequence of state levels and action levels $L_0, A_0, \ldots, L_k, A_k, \ldots$ where

- $_0$ represents I;
- for $i \geq 0$,
  - A_i is the set of actions executable in L_i; and
  - $L_{i+1} = L_i \cup \left(\bigcup_{a \in A_i} \Phi(L_i, a) \right)$. Where Φ is the transition function in P.

The idea is that, despite the possible errors, the level j at which a *literal* first appears is a good estimate of how difficult it is to achieve that literal from the initial state. The idea behind the planning graph works only for propositional planning, the ones with no variables.

To keep track of the conflicts that appear at each level of the planning graph are used the *mutex links*. These links (in gray in Figure 11b) indicate whenever in the planning graph appear:

- *Inconsistent effects*: one action negates an effect of the other.
- *Interference*: one of the effects of one action is the negation of a precondition of the other.
- *Competing needs*: one of the preconditions of one action i mutually exclusive with a precondition of the other.

1.5.2 Planning Graph Proprieties

What make a planning graph useful is that it gives important information about the problem in *polynomial* time.

Given the search tree of the problem is of course possible[12] to understand if a state G is reachable from the initial state S_0. The problem is that trees are exponential in size, so most of the time this approach is impractical. A planning graph is a polynomial-size approximation to the search tree and can be constructed quickly.

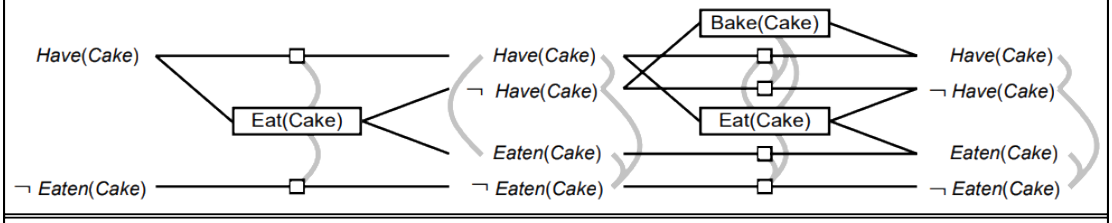
Even though the planning graph cannot answer definitively whether G is reachable from S_0, it can estimate how many steps it takes to reach G. Other important properties of the planning graph is that the estimation is always correct when it reports that the goal is not reachable, and it never overestimates the number of steps, so it generates an admissible heuristic.

[12] When the search space if finite.

$Init(Have(Cake))$
$Goal(Have(Cake) \land Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg\, Have(Cake) \land Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg\, Have(Cake)$
EFFECT: $Have(Cake)$

The "have cake and eat cake too" problem.

(a) Problem description.



The planning graph for the "have cake and eat cake too" problem up to level S_2. Rectangles indicate actions (small squares indicate persistence actions) and straight lines indicate preconditions and effects. Mutex links are shown as curved gray lines.

(b) Planning graph of the problem in Figure 11a.

Figure 11: Example of Planning graph [39].

The termination of the construction of the planning graph[12] is also guaranteed through saturation.

2 EPISTEMIC PLANNING

In the previous chapters we have introduced the main concepts related to the branch of AI called "planning". We have especially investigated the basis of this field and the domain family called "classical planning". While there certainly are computational benefits to the restrictions imposed by classical planning (§1.2.1), it is also clear that such planning domains are much easier to construct theoretically sound planning algorithms for. In other words, the reason that classical planning became so dominant was not only due to limited computational resources, but also a limited understanding of the intricacies of how, for instance, to take the actions of other agents into account when planning, or how to naturally represent incomplete information about the world state—the complexity of automated planning is not solely computational.

In this chapter we will describe a different type of planning that is the one related to the main contribution of this thesis: *epistemic planning*.

Epistemic planning [7] refers to the generation of plans for multiple agents to achieve goals which can refer to the state of the world, the beliefs of agents, and/or the knowledge of agents. It has recently attracted the attention of researchers from various communities such as planning, dynamic epistemic logic, and knowledge representation.

2.1 Epistemic Planners and Action Languages

With the introduction of the classical planning problem, in the early days of artificial intelligence, also languages for representing actions and their effects were proposed [19].

Over the years, several action languages (e.g., $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$) have been developed [21] and have also provided the foundations for several successful approaches to automated planning; for example, the language $\mathcal{C}$ is used in the planner C-PLAN [11] and the language $\mathcal{B}$ is used in CPA [44].

However, the main focus of these research efforts has been about reasoning within single agent domains. In single agent domains, reasoning about actions and change mainly involves reasoning about what is true in the world, what the agent knows about the world, how the agent can manipulate the world (using world-changing actions) to reach particular states, and how the agent (using sensing actions) can learn unknown aspects of the world.

In multi-agent domains an agent's action may not just change the world and the agent's knowledge about the world, but also may change other agents' knowledge about the world and their knowledge about other agents' knowledge about the world. Similarly, goals of an agent in a multi-agent world may involve manipulating the knowledge of other agents—in particular, this may involve not just their knowledge about the world, but also their knowledge about other agents'

knowledge about the world.

As said before epistemic planning is not only interested in the state of the world but also in the beliefs and the knowledge of the agents and although there is a large body of research on multi-agent planning [16, 14, 15, 1, 6, 23, 24, 34, 38], very few efforts address the above aspects of multi-agent domains which pose a number of new research challenges in representing and reasoning about actions and change. The following example illustrates some of these issues.

Example 2.1 (**Three Agents and the Coin Box**)**.** Three agents A, B, and C, are in a room. In the middle of the room there is a box containing a coin. It is common knowledge that:

- None of the agents knows whether the coin lies heads or tails up;
- The box is locked and one needs to have a key to open it; only agent A has the key of the box;
- In order to learn whether the coin lies heads or tails up, an agent can peek into the box. This, however, requires the box to be open;
- If an agent is looking at the box and someone peeks into the box, then the first agent will notice this fact and will be able to conclude that the second agent knows the status of the coin; yet, the first agent's knowledge about which face of the coin is up does not change;

- Distracting an agent causes that agent to not look at the box;
- Signaling an agent to look at the box causes this agent to look at the box;
- Announcing that the coin lies heads or tails up will make this a common knowledge among the agents that are listening.

Suppose that the agent A would like to know whether the coin lies heads or tails up. She would also like to let the agent B know that she knows this fact. However, she would like to keep this information secret from C. (Note that the last two sentences express goals that are about agents' knowledge about other agents' knowledge.) Intuitively, she could achieve her goal by:

1. Distracting C from looking at the box;
2. Signaling B to look at the box;
3. Opening the box; and
4. Peeking into the box

This simple story presents a number of challenges for research in representing and reasoning about actions and their effects in multi-agent domains. In particular:

- The domain contains several types of actions:

- Actions that allow the agents to change the state of the world (e.g., opening the box);
- Actions that change the knowledge of the agents (e.g, peeking into the box, announcing head/tail);
- Actions that manipulate the beliefs of other agents (e.g., peeking while other agents are looking); and
- Actions that change the observability of agents with respect to awareness about future actions (e.g., distract and signal actions before peeking into the box).

We observe that the third and fourth types of actions are not considered in single agent systems.

- The reasoning process that helps A to realize that steps (1)-(4) will achieve her goal requires A's ability to reason about the effects of her actions on several entities:
  - *The state of the world*—e.g., opening the box causes the box to become open;
  - *The agents' awareness of the environment and of other agents' actions*—e.g., distracting or signaling an agent causes this agent not to look or to look at the box; and

– *The knowledge of other agents about her own knowledge*—e.g., someone following her actions would know what she knows.

While the first requirement is the same as for an agent in single agent domains, the last two are specific to multi-agent domains.

With respect to planning, the above specifics of multi-agent systems raise an interesting question: "How can one generate a plan for the agent A to achieve her goal, given the description in Example 2.1?" To the best of our knowledge only a few epistemic automated planners[13], and even less action languages, exist that can address the complete range of issues as exemplified in Example 2.1.

2.2 The Action Language m$\mathcal{A}$*

The action language that will the base for the planner developed as this thesis contribution is called m$\mathcal{A}$*. This action language is an extension of the language m$\mathcal{A}$+ introduced in [5][14]. m$\mathcal{A}$* is a high-level action language for epistemic planning in multi-agent domains. The language can play a role, in a multi-agent context, similar to the role of the *Planning Domain Description Language (PDDL)* in single-agent planning domains. The language has a declarative, English-like syntax and an event model based semantics that can deal with false beliefs of agents.

[13]Discussed in §3 and §4.

[14]We will discuss the differences between the two languages in the last paragraph of §2.4.3

We will introduce the basic notions from the literature on formalizing knowledge and reasoning about effects of actions in multi-agent systems without describing the language in depth. All the remaining details about the language can be found in [5].

2.2.1 Belief Formulae

Let us consider an environment with a set $\mathcal{AG}$ of n agents. The *real state of the world* (or *real state*, for brevity) is described by a set $\mathcal{F}$ of propositional variables, called *fluents*. We are concerned with the beliefs of agents about the environment and about the beliefs of other agents. For this purpose, we adapt the logic of knowledge and the notations used in [18, 48]. We associate to each agent $i \in \mathcal{AG}$ a modal operator $\mathbf{B}_i$ and represent the beliefs of an agent as belief formulae in a logic extended with these operators. Formally, we define belief formulae as follows:

- *Fluent formulae:* a fluent formula is a propositional formula built using the propositional variables in $\mathcal{F}$ and the traditional propositional operators $\vee, \wedge, \Rightarrow, \neg$, etc. In particular, a *fluent atom* is a formula containing just an element $f \in \mathcal{F}$, while a *fluent literal* is either a fluent atom $f \in \mathcal{F}$ or its negation $\neg f$. We will use $\top$ and $\bot$ to denote $true$ and $false$, respectively.

- *Belief formulae*: a belief formula is a formula in one of the following forms:

  – a fluent formula;

- a formula of the form $\mathbf{B}_i\varphi$ where φ is a belief formula;
- a formula of the form $\varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2, \varphi_1 \Rightarrow \varphi_2$, or $\neg\varphi_1$ where φ_1, φ_2 are belief formulae;
- a formula of the form $\mathbf{E}_\alpha\varphi$ or $\mathbf{C}_\alpha\varphi$, where φ is a formula and $\emptyset \neq \alpha \subseteq \mathcal{AG}$.

Formulae of the form $\mathbf{E}_\alpha\varphi$ and $\mathbf{C}_\alpha\varphi$ are referred to as *group formulae*. Whenever $\alpha = \mathcal{AG}$, we simply write $\mathbf{E}\varphi$ and $\mathbf{C}\varphi$ to denote $\mathbf{E}_\alpha\varphi$ and $\mathbf{C}_\alpha\varphi$, respectively. Let us denote with $\mathcal{L}_{\mathcal{AG}}$ the language of the belief formulae over $\mathcal{F}$ and $\mathcal{AG}$.

Intuitively, belief formulae are used to describe the beliefs of one agent concerning the state of the world as well as about the beliefs of other agents. For example, the formula $\mathbf{B}_1\mathbf{B}_2 p$ expresses the fact that "*Agent* 1 *believes that agent* 2 *believes that p is true,*" while $\mathbf{B}_1 f$ states that "*Agent* 1 *believes that f is true.*"

While the beliefs of the agent i are represented with $\mathbf{B}_i$ her knowledge is represented with K_i. This means that $K_i\varphi$ reads "agent i knows φ." The two representation are related, in fact we can use the equivalence $K\varphi \equiv B\varphi \wedge \varphi$ to link them.

In what follows, we will simply talk about "formulae" instead of "belief formulae," whenever there is no risk of confusion. The notion of a Kripke structure is defined next.

2.2.2 Kripke Structures

Definition 2.1 (Kripke Structure)**.** A *Kripke structure* (Figure 12), also referred to as *epistemic model* (*e-model*), is a tuple $\langle S, \pi, \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$, where

- S is a set of worlds,
- $\pi : S \mapsto 2^{\mathcal{F}}$ is a function that associates an interpretation of $\mathcal{F}$ to each element of S,
- For $1 \leq i \leq n$, $\mathcal{B}_i \subseteq S \times S$ is a binary relation[15] over S.

Definition 2.2 (Pointed Kripke Structure)**.** A *Pointed Kripke structure*, also referred to as *pointed epistemic model* (*pe-model*), is a pair (M, s) where $M = \langle S, \pi, \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$ is a Kripke structure and $s \in S$. In a pointed Kripke structure (M, s), we refer to s as the *real* (or actual) *world*[16].

Intuitively, a Kripke structure describes the possible worlds envisioned by the agents—and the presence of multiple worlds identifies uncertainty and presence of different beliefs. The relation $(s_1, s_2) \in \mathcal{B}_i$ denotes that the belief of agent i about the real state of the world is insufficient for her to distinguish between the world described by s_1 and the one described by s_2. The world s in the state (M, s) identifies the world in $M[S]$ that corresponds to the actual world. We will often

[15] Sometimes also defined as $\mathcal{B} : \mathcal{AG} \to 2^{W \times W}$ that assigns an accessibility relation $\mathcal{B}(i) = \mathcal{B}_i$ to each agent $i \in \mathcal{AG}$.

[16] Represented by the double circle in Figure 12

view a Kripke structure $M = \langle S, \pi, \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$ as a directed labeled graph, whose set of nodes is S and whose set of edges contains (s, i, t)[17] if and only if $(s, t) \in \mathcal{B}_i$. (s, i, t) is referred to as an edge coming out of (resp. into) the world s (resp. t).

For the sake of readability, we use $M[S]$, $M[\pi]$, and $M[i]$ to denote the components S, π, and $\mathcal{B}_i$, of M, respectively. We write $M[\pi](u)$ to denote the interpretation associated to u via π and $M[\pi](u)(\varphi)$ to denote the truth value of a fluent formula φ with respect to the interpretation $M[\pi](u)$.

In keeping with the tradition of action languages, we will often refer to $\pi(u)$ as the set of fluent literals defined by

$$\{f \mid f \in \mathcal{F}, M[\pi](u)(f) = \top\} \cup \{\neg f \mid f \in \mathcal{F}, M[\pi](u)(f) = \bot\}.$$

Given a consistent and complete set of literals S, i.e., $|f, \neg f \cap S| = 1$ for every $f \in \mathcal{F}$, we write $\pi(u) = S$ or $M[\pi](u) = S$ to indicate that the interpretation $M[\pi](u)$ is defined in such a way that $\pi(u) = S$.

The satisfaction relation between belief formulae and a state is defined as next.

Definition 2.3 (Satisfaction Relation)**.** Given a formula φ, a Kripke structure $M = \langle S, \pi, \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$, and a world $s \in S$:

(i) $(M, s) \models \varphi$ if φ is a fluent formula and $\pi(s) \models \varphi$;

(ii) $(M, s) \models \mathbf{B}_i\varphi$ if for each t such that $(s, t) \in \mathbf{B}_i$, $(M, t) \models \varphi$;

[17] (s, i, t) denotes the edge from node s to node t, labeled by i.

(iii) $(M, s) \models \neg\varphi$ if $(M, s) \not\models \varphi$;

(iv) $(M, s) \models \varphi_1 \vee \varphi_2$ if $(M, s) \models \varphi_1$ or $(M, s) \models \varphi_2$;

(v) $(M, s) \models \varphi_1 \wedge \varphi_2$ if $(M, s) \models \varphi_1$ and $(M, s) \models \varphi_2$.

(vi) $(M, s) \models E_\alpha\varphi$ if $(M, s) \models \mathbf{B}_i\varphi$ for every $i \in \alpha$.

(vii) $(M, s) \models C_\alpha\varphi$ if $(M, s) \models E^k_\alpha\varphi$ for every $k \geq 0$, where $E^0_\alpha\varphi = \varphi$ and $E^{k+1}_\alpha = E_\alpha(E^k_\alpha\varphi)$.

For a Kripke structure M and a formula φ, $M \models \varphi$ denotes the fact that $(M, s) \models \varphi$ for each $s \in M[S]$, while $\models \varphi$ denotes the fact that $M \models \varphi$ for every Kripke structure M.

Example 2.2. Let us consider a simplified version of Example 2.1 in which the agents are concerned only with the status of the coin. The three agents A, B, C do not know whether the coin has '*heads*' or '*tails*' up and this is a common belief. Let us assume that the coin is heads up. The beliefs of the agents about the world and about the beliefs of other agents can be captured by the state of Figure 12.

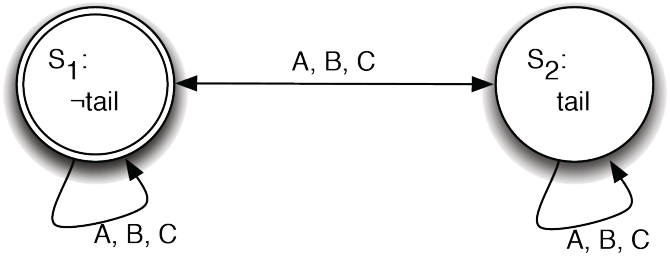


Figure 12: An example of Kripke Structure

In the figure, a circle represents a world. The name and interpretation of the world are written in the circle. Labeled edges between worlds denote the belief relations of the structure. A double circle identifies the real world.

We will no further describe the properties of the Kripke structures since those are not strictly needed to describe the contribution of this thesis. Moreover these are extensively defined in [5].

Anyway is necessary to point out another common, and equivalent, representation for a Kripke structure of the language $\mathcal{L}_{\mathcal{AG}}$.

2.2.3 Actions Types

m$\mathcal{A}^*$ describes three types of actions that an agent can perform: *world-altering* actions (also known as *ontic* actions), *sensing* actions, and *announcement* actions. Intuitively,

- A world-altering action is used to explicitly modify certain properties of the world—e.g., in Example 2.1, the agent A opens the box, or the agent A distracts the agent B so that B does not look at the box;

- A sensing action is used by an agent to refine its beliefs about the world, by making direct observations—e.g., an agent peeks into the box; the effect of the sensing action is to reduce the amount of uncertainty of the agent (in Example 2.1);

- An announcement action is used by an agent to affect the beliefs of the agents receiving the communication—e.g., still in Example 2.1, announcing that the coin lies heads or tails up. We operate under the assumption that agents receiving an announcement always believe what is being announced.

In general, as for other types of planning, an action can be executed only under when the executability conditions are respected. For example, the statement "to open the box, an agent must have its key" in Example 2.1 describes the executability condition of the action of opening a box.

Example 2.3. Following the *syntax* described in [5] the actions of domain of

Example 2.1 can be specified by the following statements:

executable $open(X)$ **if** $has\_key(X)$

executable $peek(X)$ **if** $opened, looking(X)$

executable $shout\_tail(X)$ **if** $K_X(tail), tail$

executable $signal(X, Y)$ **if** $looking(X), \neg looking(Y)$

executable $distract(X, Y)$ **if** $looking(X), looking(Y)$

$open(X)$ **causes** $opened$

$signal(X, Y)$ **causes** $looking(Y)$

$distract(X, Y)$ **causes** $\neg looking(Y)$

$peek(X)$ **determines** $tail$

$shout\_tail(X)$ **announces** $tail$

where X and Y are different agents in $\{A, B, C\}$. The first five statements encode the executability conditions of the five actions in the domain. The next three statements describe the effects of the three world-altering actions. $peek(X)$ is an example of a sensing action. Finally, $shout\_tail(X)$ is an example of an announcement action.

The general cases for the actions that are presented in Example 2.1 are described by:

executable a **if** ψ (1)

a **causes** l **if** ψ (2)

a **determines** f (3)

a **announces** l (4)

where a in (1) is any of the three types of action and in (2), (3) (4) is an, ontic, sensing, or announcement action, respectively, f is a fluent, l is a fluent literal, and ψ is a belief formula. (2) states that a is an ontic action and its execution will cause l to be true in the actual world state if ψ is true. (3) indicates that a is a sensing action and executing a will reveal the truth value of f. (4) says that a is an announcement action and its execution will inform aware agents that l is true.

2.2.4 Actions Observability

One of the key differences between single-agent and multi-agent domains lies in how the execution of an action changes the beliefs of agents. This is because, in multi-agent domains, an agent can be oblivious about the occurrence of an action or unable to observe the effect of an action. For example, watching another agent open the box allows the agent to know that the box is open after the execution of the action; however, the agent would still believe that the box is closed if she is

not aware of the action occurrence. On the other hand, watching another agent peeking into the box does not help the observer in learning whether the coin lies heads or tails up; the only thing she would learn is that the agent who is peeking into the box has knowledge of the status of the coin.

To represent the fact that not all the agents may be completely aware of the presence of actions being executed, depending on the action and the current situation, m$\mathcal{A}^*$ categorizes agents in three classes:

- *Full observers*—i.e., the agents are aware of the occurrence of the action and of its results;
- *Partial observers*—i.e., the agents have knowledge that the full observers have performed a sensing action or an announcement but without knowledge of the result of the observation or without knowing what has been announced; and
- *Oblivious* (or others)—i.e., the agents are unaware of the occurrence of the action.

This categorization is dynamic: changes in the state of the world may lead to changes to the agent's category w.r.t. each action. The possible observability of agents for different action types is showed in Table 1.

The dynamic nature of the agents observability can be manipulated using agent observability statements of the following form:

action type	full observers	partial observers	oblivious/others
world-altering actions	✓		✓
sensing actions	✓	✓	✓
announcement actions	✓	✓	✓

Table 1: Action types and agent observability

$$X \textbf{ observes } a \textbf{ if } \psi \tag{5}$$

$$X \textbf{ aware\_of } a \textbf{ if } \psi \tag{6}$$

Where X represent an agent, a represent an action and ψ a formula. (5) states that agent X is a full observer of the action a. Instead (6) says that X is partially observant of the occurrence of a.

Definition 2.4 (m$\mathcal{A}^*$ **Domain**)**.** An m$\mathcal{A}^*$ *domain* is a collection of statements of the forms (1)-(6).

2.3 Epistemic Planning Problem

An epistemic planning problem [7] is defined over a multi-agent epistemic logic language $\mathcal{L}_{\mathcal{AG}}$ (in our case m$\mathcal{A}^*$), with a set of agents $\mathcal{AG}$ and a set of *fluents* $\mathcal{F}$.

The satisfaction of a formula of the language $\mathcal{L}_{\mathcal{AG}}$, respect to the real world s, is expressed in §2.3

Epistemic planning relies on the notion of an *event model* (also called *update model* in the literature) for modeling changes to *epistemic states*. We use the

formalization of update model introduced in [3, 46]. Let us start with some preliminary definitions.

Definition 2.5 (Epistemic State)**.** An epistemic state (or *e-state*) is a pair $(\mathcal{M}, W_d)$ where $\mathcal{M} = (M, \mathcal{B}, \pi)$ is an epistemic model and $W_d \subseteq W$. A truth value of a formula φ with respect to an epistemic state $(\mathcal{M}, W_d)$ is defined by

$$(\mathcal{M}, W_d) \models \varphi \qquad \text{iff} \qquad \forall w \in W_d.[(\mathcal{M}, w) \models \varphi]$$

Definition 2.6 ($\mathcal{L}_{\mathcal{AG}}$-substitution)**.** An *$\mathcal{L}_{\mathcal{AG}}$-substitution* is a set $\{p_1 \to \varphi_1, \ldots, p_k \to \varphi_k\}$, where each p_i is a distinct proposition in $\mathcal{F}$ and each $\varphi_i \in \mathcal{L}_{\mathcal{AG}}$. We will implicitly assume that for each $p \in \mathcal{F} \setminus \{p_1, \ldots, p_k\}$, the substitution contains $p \to p$. $SUB_{\mathcal{F},\mathcal{AG}}$ denotes the set of all $\mathcal{L}_{\mathcal{AG}}$-substitutions.

Definition 2.7 (Event Model)**.** Given a set $\mathcal{AG}$ of n agents, an *event model* $\mathbf{\Sigma}$ is a tuple $\langle E, Q, pre, sub\rangle$ where

(i) E is a set, whose elements are called *events*;

(ii) $Q : \mathcal{AG} \to 2^{E \times E}$ assigns an accessibility relation to each agent $i \in \mathcal{AG}$;

(iii) $pre : E \to \mathcal{L}_{\mathcal{AG}}$ is a function mapping each event $e \in E$ to a formula in $\mathcal{L}_{\mathcal{AG}}$; and

(iv) $sub : E \to SUB_{\mathcal{F},\mathcal{AG}}$ is a function mapping each event $e \in E$ to a substitution in $SUB_{\mathcal{F},\mathcal{AG}}$.

Definition 2.8 (Epistemic Action)**.** An *epistemic action* is a pair $(\mathcal{E}, E_d)$, consisting of an event model $\mathcal{E} = (E, Q, pre, sub)$ and a non-empty set of designated events $E_d \subseteq E$.

Given an epistemic action $(\mathcal{E}, E_d)$ and an epistemic state $(\mathcal{M}, W_d)$, we say that $(\mathcal{E}, E_d)$ is executable in $(\mathcal{M}, W_d)$ if, for each $w \in W_d$, there exists at least one $e \in E_d$ such that $(\mathcal{M}, w) \models pre(e)$. The execution of $(\mathcal{E}, E_d)$ in $(\mathcal{M}, W_d)$ results in an epistemic state $(\mathcal{M}, W_d) \otimes (\mathcal{E}, E_d) = ((W', \mathcal{B}', \pi'), W'_d)$ where

- $W' = \{(w, e) \in W \times E \mid (\mathcal{M}, w) \models pre(e)\}$
- $\mathcal{B}'_i = \{((w, e), (v, f)) \in W' \times W' \mid w \mathcal{B}_i v \wedge e Q_i f\}$
- For each $(w, e) \in W'$ and $p \in \mathcal{F}$, $\pi'((w, e))(p)$ is true iff $p \to \varphi \in sub(e)$ and $(\mathcal{M}, w) \models \varphi$
- $W'_d = \{(w, e) \in W' \mid w \in W_d \, and \, e \in E_d\}$

Definition 2.9 (Epistemic Planning Domain)**.** An *epistemic planning domain* is then defined as a state-transition system $\Sigma = (S, A, \gamma)$, where S is a set of epistemic states of $\mathcal{L}_{\mathcal{AG}}$, A is a finite set of epistemic actions of $\mathcal{L}_{\mathcal{AG}}$, and $\gamma(s, a) = s \otimes a$ if $s \otimes a$ is defined and $\otimes$ is the above operation.

Definition 2.10 (Epistemic Planning Problem)**.** An *epistemic planning problem* is a triple (Σ, s_0, ϕ_g) where $\Sigma = (S, A, \gamma)$ is an epistemic planning domain on $(\mathcal{F}, \mathcal{AG})$, $s_0 \in S$ is the initial state, and ϕ_g is a formula in $\mathcal{L}_{\mathcal{AG}}$.

An action sequence $a_1, \ldots, a_n$ such that $s_0 \otimes a_1 \otimes a_2 \otimes \ldots \otimes a_n \models \phi_g$ is a *solution*

of the problem.

2.4 Problem Specification

The above formalization of epistemic planning problem leaves open the question of *how to compactly*[18] *specify an epistemic planning problem?*—i.e., how to compactly specify an epistemic planning domain $\Sigma = (S, A, \gamma)$ on $(\mathcal{F}, \mathcal{AG})$ and how to specify the initial state s_0. In this thesis, we will use finitary **S5**-theory [42] (described next) and the language m$\mathcal{A}^*$ to specify the initial state and a planning domain, respectively.

2.4.1 Specification of the Initial State

A set of formulae s_0 in $\mathcal{L}_{\mathcal{AG}}$ is said to be a finitary **S5**-theory if it a **S5**-theory [48] and contains only formulae of the following form:

(i) φ;

(ii) $C(\mathbf{B}_i\varphi)$;

(iii) $C(\mathbf{B}_i\varphi \vee \mathbf{B}_i\neg\varphi)$;

(iv) $C(\neg\mathbf{B}_i\varphi \wedge \neg\mathbf{B}_i\neg\varphi)$

where φ is a fluent formula.

[18] By a "compact specification" we mean a specification that *does not explicitly list* all possible e-states of the problem.

Intuitively, formulae of type (**i**) indicate formulas that are true in the actual world; (**ii**)-(**iii**) indicate that all agents know that i knows the truth value of φ; formulae (**iv**) say that all agents know that i does not know whether φ is true or false.

It is shown that if the initial state is a finitary **S5**-theory then there exists a unique e-model $\mathcal{M} = (W, \mathcal{B}, \pi)$ such that

- W is finite and for every $w \in W$, $(\mathcal{M}, w) \models s_0$; and
- every pe-model $(\mathcal{M}', w')$ such that $(\mathcal{M}', w') \models s_0$ can be reduced by bisimulation to some $(\mathcal{M}, w)$.

This means that $(\mathcal{M}, W)$ is a unique e-state satisfying s_0.

There are two reasons supporting the choice of finitary **S5**-theory as the specification of initial state. First, it is computable. Second, the majority of benchmarks in epistemic planning have this property.

Example 2.4. The initial state of the problem in Example 2.1 is encoded by

$$\textbf{initially}\ \ \mathbf{C}(has\_key(A)) \qquad \textbf{initially}\ \ \mathbf{C}(\neg has\_key(B))$$

$$\textbf{initially}\ \ \mathbf{C}(\neg has\_key(C)) \quad \textbf{initially}\ \ \mathbf{C}(\neg opened)$$

$$\textbf{initially}\ \ \mathbf{C}(\neg K_x tail \wedge \neg K_x \neg tail) \qquad \text{for}\ \ x \in \{A, B, C\}$$

$$\textbf{initially}\ \ \mathbf{C}(looking(A) \wedge looking(C) \wedge \neg looking(B))$$

2.4.2 Specification of the planning domain

Using m$\mathcal{A}^*$, a multi-agent planning domain D can be specified by a tuple $\langle \mathcal{AG}, \mathcal{F}, A, O \rangle$ where $\mathcal{AG}$ is the set of agents and $\mathcal{F}$ is the set of fluents; A is the set of actions, where each action is either an ontic action, a sensing action, or an announcement action. It is assumed that each action $a \in A$ is of a unique type. A also contains the description of preconditions and effects of actions; and O are observability statements, describing the frames of reference of agents with respect to action occurrences.

The semantics of a multi-agent domain is defined by a transition function Φ that maps pairs of actions and e-states to sets of e-states. Let a be an action in $\mathcal{A}$ and $(\mathcal{M}, w)$ be an e-model. a is executable $(\mathcal{M}, w)$ if $(\mathcal{M}, w) \models pre(a)$. Let

$$F(a, \mathcal{M}, w) = \{X \in \mathcal{AG} \mid [Y \textbf{ observes } a \textbf{ if } \varphi] \in A \text{ such that } (\mathcal{M}, w) \models \varphi \wedge X \in Y\}$$

$$P(a, \mathcal{M}, w) = \{X \in \mathcal{AG} \mid [Y \textbf{ aware\_of } a \textbf{ if } \varphi] \in A \text{ such that } (\mathcal{M}, w) \models \varphi \wedge X \in Y\}$$

$$O(a, \mathcal{M}, w) = \mathcal{AG} \setminus (F(a, \mathcal{M}, w) \cup P(a, \mathcal{M}, w))$$

These sets represent the set of agents who are fully aware, partially aware, and oblivious of the execution of a in $(\mathcal{M}, w)$, respectively. m$\mathcal{A}^*$ specifies how the corresponding epistemic action $(\mathcal{E}_{(a,\mathcal{M},w)}, E_{(a,\mathcal{M},w)})$ of an occurrence of a in $(\mathcal{M}, w)$ is defined given a and $(\mathcal{M}, w)$. The precise definition of $(\mathcal{E}_{(a,\mathcal{M},w)}, E_{(a,\mathcal{M},w)})$ is presented next.

2.4.3 The Transition Function

Let $(\mathcal{M}, w)$ be a pe-model and a be an action executable in $(\mathcal{M}, w)$. The epistemic action corresponding to the occurrence of a in $(\mathcal{M}, w)$ is defined as follows

Definition 2.11 (Ontic Action Occurrence)**.** Let a be an ontic action with the precondition ψ and $(\mathcal{M}, w)$ a pointed epistemic model. The epistemic action representing the occurrence of a in $(\mathcal{M}, w)$ is the pair $(\Delta(a, \mathcal{M}, w), \{\sigma\})$ where $\Delta(a, \mathcal{M}, w) = \langle E, Q, pre, sub\rangle$ with

- $E = \{\sigma, \epsilon\}$ if $O_D(a, \mathcal{M}, w) \neq \emptyset$ and $E = \{\sigma\}$ otherwise;
- $Q(i) = \mathcal{B}_i$ where[19]
  - $\mathcal{B}_i = \{(\sigma, \sigma), (\epsilon, \epsilon)\}$ for $i \in F_D(a, \mathcal{M}, w)$ and
  - $\mathcal{B}_i = \{(\sigma, \epsilon), (\epsilon, \epsilon)\}$ for $i \in O_D(a, \mathcal{M}, w)$;
- $pre(\sigma) = \psi$ and $pre(\epsilon) = \top$; and
- $sub(\epsilon) = \emptyset$ and $sub(\sigma) = \{p \to \Psi^+(p, a) \vee (p \wedge \neg\Psi^-(p, a)) \mid p \in \mathcal{P}\}$, where $\Psi^+(p, a) = \bigvee\{\varphi | [a \textbf{ causes } p \textbf{ if } \varphi] \in D\}$ and $\Psi^-(p, a) = \bigvee\{\varphi | [a \textbf{ causes } \neg p \textbf{ if } \varphi] \in D\}$.

As an example, Figure 13 (Right) shows the action occurrence of the ontic action $open(A)$ in the e-model on the left.

Definition 2.12 (Sensing Action Occurrence)**.** Let a be a sensing action with the precondition ψ and the sensed fluent f, and $(\mathcal{M}, w)$ a pointed epistemic

[19] When ϵ does not belong to E, the links associated with ϵ are removed.

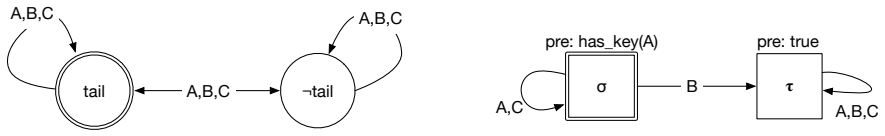


Figure 13: Epistemic action of $open(A)$

model. The epistemic action representing the execution of a in $(\mathcal{M}, w)$ is the pair $(\Delta(a, \mathcal{M}, w), \{\sigma, \tau\})$ where $\Delta(a, \mathcal{M}, w) = \langle E, Q, pre, sub\rangle$ with

- $E = \{\sigma, \tau, \epsilon\}$[20];
- $Q(i) = \mathcal{B}_i$ where
  - $\mathcal{B}_i = \{(\sigma, \sigma), (\tau, \tau), (\epsilon, \epsilon)\}$ for $i \in F_D(a, \mathcal{M}, w)$
  - $\mathcal{B}_i = \{(\sigma, \sigma), (\tau, \tau), (\epsilon, \epsilon), (\sigma, \tau), (\tau, \sigma)\}$ for $i \in P_D(a, \mathcal{M}, w)$
  - $\mathcal{B}_i = \{(\sigma, \epsilon), (\tau, \epsilon), (\epsilon, \epsilon)\}$ for $i \in O_D(a, \mathcal{M}, w)$
- $pre(\sigma) = \psi \wedge f$, $pre(\tau) = \psi \wedge \neg f$, and $pre(\epsilon) = \top$;
- $sub(x) = \emptyset$ for each $x \in E$.

Definition 2.13 (Announcement Action Occurrence)**.** Let a be an announcement action a with the precondition ψ and $(\mathcal{M}, w)$ a pointed epistemic model. The epistemic action representing the execution of a in $(\mathcal{M}, w)$ is the pair $(\Delta(a, \mathcal{M}, w), \{\sigma\})$ where $\Delta(a, \mathcal{M}, w) = \langle E, Q, pre, sub\rangle$ with

- $E = \{\sigma, \tau, \epsilon\}$ ($\epsilon \in E$ iff $O_D(a, \mathcal{M}, w) \neq \emptyset$);
- $Q(i) = \mathcal{B}_i$ where
  - $\mathcal{B}_i = \{(\sigma, \sigma), (\tau, \tau), (\epsilon, \epsilon)\}$ for $i \in F_D(a, \mathcal{M}, w)$

[20]Similar to Definition 2.11, $\epsilon \in E$ only if $O_D(a, \mathcal{M}, w) \neq \emptyset$.

- $\mathcal{B}_i = \{(\sigma,\sigma),(\tau,\tau),(\epsilon,\epsilon),(\sigma,\tau),(\tau,\sigma)\}$ for $i \in P_D(a,\mathcal{M},w)$
- $\mathcal{B}_i = \{(\sigma,\epsilon),(\tau,\epsilon),(\epsilon,\epsilon)\}$ for $i \in O_D(a,\mathcal{M},w)$

∘ $pre(\sigma) = \psi \wedge \varphi$, $pre(\tau) = \psi \wedge \neg\varphi$, and $pre(\epsilon) = \top$;

∘ $sub(x) = \emptyset$ for any $x \in E$.

We illustrate these definitions using Figure 14. $(\mathcal{M}, s_0)$ (Fig. 14(left)) with $\mathcal{M} = (\{s_0, s_1\}, \mathcal{B}, \pi)$, where[21] $s_0 = \{has\_key(A), \neg has\_key(B), \neg has\_key(C), tail, looking(A), \neg looking(B), looking(C)\}$ and $s_1 = \{has\_key(A), \neg has\_key(B), \neg has\_key(C), \neg tail, looking(A), \neg looking(B), looking(C)\}$, and $\mathcal{B}(A) = \mathcal{B}(B) = \mathcal{B}(C) = \{(u,v) \mid u,v \in \{s_0,s_1\}\}$, is an initial pe-model of the problem given in Example 2.1. Fig. 14(middle) and Fig. 14(right) depict the results of executing $distract(A,C)$ and $signal(A,B)$ in $(\mathcal{M}, s_0)$ respectively.

$\Phi(distract(A,C),(M,s_0))$=$\{(U,u_0)\}$ where the interpretations of s_0 and s_1 are the same as in M, $u_0 = s_0 \backslash \{looking(C)\} \cup \{\neg looking(C)\}$ and $u_1 = s_1 \backslash \{looking(C)\} \cup \{\neg looking(C)\}$. In (U,u_0), B does not know that C is not looking at the box since B is oblivious of the action occurrence (due the last statement in the set of statements of observability of agents).

$\Phi(signal(A,B),(M,s_0)) = \{(W,w_0)\}$ where $w_0 = s_0 \backslash \{\neg looking(B)\} \cup \{looking(B)\}$ and $w_1 = s_1 \setminus \{\neg looking(B)\} \cup \{looking(B)\}$. In (W,w_0), C knows that B is looking at the box since he is fully aware of the execution of the action.

Figure 15 shows the epistemic actions (on the right) correspond to the oc-

[21]The interpretation $\pi(s)$ is simply given by a complete set of fluent literals assigned to s.

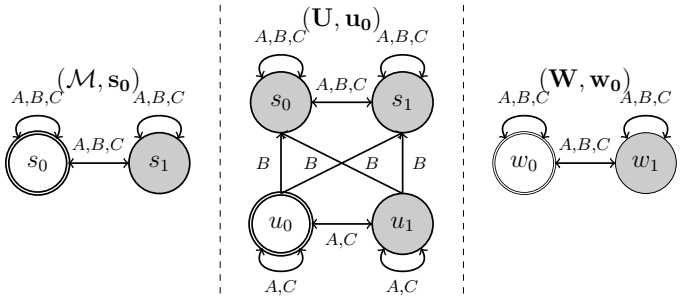


Figure 14: Initial e-state $(\mathcal{M}, s_0)$ and results of action executions

currence of *open*(A) in two e-models (left); in the e-model at the top, B is not looking and in the e-model at the bottom, B is looking. We use the conventional representation of e-models in the figure: nodes represent worlds, labeled links represents the accessibility relation, and π is given implicit (listing only important fluent literals in the nodes). In both models, we omit fluent literals encoding the facts that A and C are looking, A has the key, and the box is locked. Likewise, rectangles represent events, double lines rectangles represent designated events, etc.

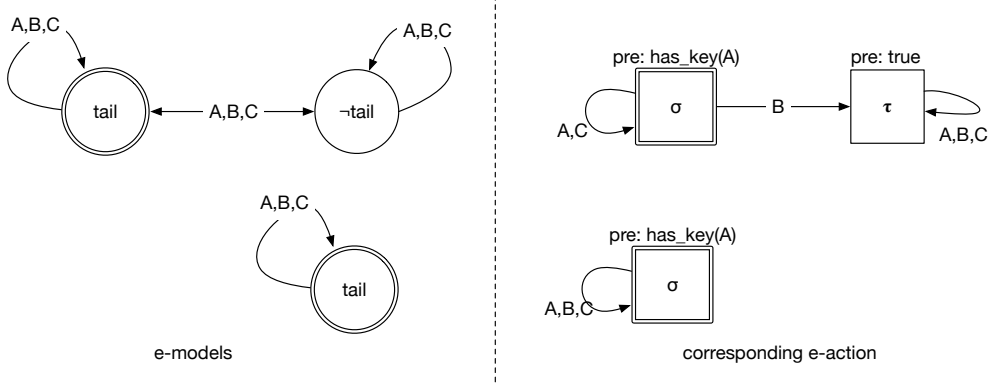


Figure 15: Epistemic action of *open*(A)

We note that the idea of constructing e-actions from the action description and action occurrences has recently been used in [17].

For each e-model $(\mathcal{M}, w)$ and an action a, $\Phi(a, (\mathcal{M}, w))$ denotes the result of executing in a in $(\mathcal{M}, w)$. $\Phi(a,(\mathrm{M,w})) = \emptyset$ if a is not executable in $(\mathcal{M}, w)$, i.e., $(\mathcal{M}, w) \not\models pre(a)$.

$$\Phi(a, (\mathcal{M}, w)) = (\mathcal{M}, w) \otimes (\mathcal{E}_{(a,\mathcal{M},w)}, E_{(a,\mathcal{M},w)})$$

If $(\mathcal{M}, W_d) \models pre(a)$, $\Phi(a, (\mathcal{M}, W_d)) = \bigcup_{w \in W_d} \Phi(a, (\mathcal{M}, w)$; otherwise,

$\Phi(a, (\mathcal{M}, W_d) = \emptyset$.

It is worth to noting that it has been discussed in [5] that, for certain situations, the semantics of m$\mathcal{A}$+ as defined in [5] is not intuitive (e.g., it does not allow agents to correct their knowledge). Fortunately, a recent proposal [49] provides a way to fix this. It involves two steps. The first step corrects the false beliefs of agents by creating $(\mathcal{M}', w)$ from $(\mathcal{M}, w)$ and the action occurrence. The second step is apply the operation $\otimes$ on $(\mathcal{M}', w)$ and $(\mathcal{E}_{(a,\mathcal{M},w)}, E_{(a,\mathcal{M},w)})$. The result of this correction is the language m$\mathcal{A}^*$ and both Efp and PG-Efp implement this modified semantics.

3 EFP AND PG-EFP

Epistemic planning in multi-agent domains has recently gained momentum in several research communities. Its complexity has been studied in [2, 8, 12]. Studies of epistemic planning can be found in [7, 47, 32, 33, 29, 30, 17, 13, 50, 26].

3.1 Other Epistemic Planners

Due to its complexity, the majority of search based epistemic planners (e.g., [13, 17, 29, 30, 26, 33, 50]) impose certain restrictions, such as the finiteness of the levels of nested beliefs. This restrictions permit, for instance, in [33, 26, 50] to solve the problem by translating it into classical planning. This is because an *epistemic state* (*e-state*) encodes the world state and the knowledge of the agents and transitions between e-states are often formalized using *update models* [3, 46, 48, 25]. This means that an epistemic planning problem can be viewed as a search problem over the e-states space, and known search algorithms used in classical planning can be used to solve epistemic planning problems.

In other approaches (e.g., [7, 47, 32]), the initial epistemic state is assumed to be known and finite or recursively enumerable.

To the best of our knowledge, there is no system that can deal with Example 2.1. We observe that this problem cannot be expressed by the formulation given in several epistemic planners (e.g., [29, 26, 50, 33]) since it requires the representation

and reasoning about common knowledge (defined as $\mathbf{C}_\alpha$ in §2.2.1).

Issues and research directions related to epistemic planning have been summarized nicely in the report of the 2017 Dagstuhl's meeting [4].

3.2 The Systems

The overall architecture of EFP and PG-EFP is given in Algorithm 1. It is not different from the standard algorithm implemented by heuristic search based planners. The key modules of EFP and PG-EFP are: (**i**) a pre-processor; (**ii**) initial e-state computation; and (**iii**) a search engine.

- *Pre-processor*: this module is responsible for parsing the planning problem description, setting up the planning domain, that includes the list of agents, the list of actions, the rules for computing frames of reference, and the list of fluents. This module is also responsible for the initialization of necessary data structures (e.g., e-state) and executes some transformations (e.g., the transformation of general planning problems to d-observable problems[22]).

- *Initial e-state computation*: this module is responsible for computing the set of initial e-states. Under the assumption that I encodes a finitary S5-theory[23], the set of initial e-states is finite (up to bi-simulation); EFP implements the algorithm given in [42] for computing the set of initial e-states.

[22] d-observable problems is defined later in Epistemic Planning Graph: Formal Definition

[23] Described in §2.4.1.

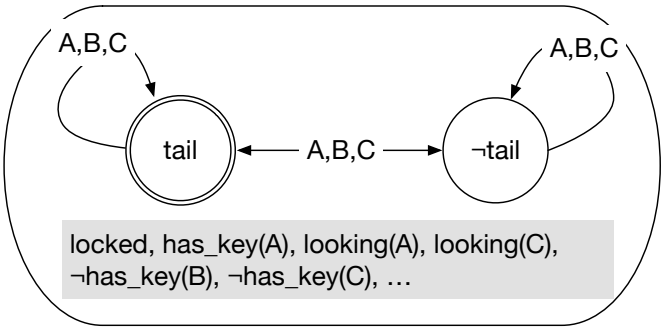


Figure 16: Initial e-state.

As an example, Figure 16 shows the e-state representing the initial state of the problem in Example 2.1. The fluent literals in the gray box of the figure are true in all worlds in the e-state.

- *Search engine*: this module is responsible for computing a solution. EFP and PG-EFP implement a best-first search (Algorithm 1). EFP is a breath-first search planner and PG-EFP is a heuristic search planner, using a heuristic defined using epistemic planning graph, described in the next section.

Both are modularly organized (e.g., the function Φ[24] is implemented in a separate module) which will facilitate future modifications and extensions.

3.3 Epistemic Planning Graph

In this section, we develop a means for deriving heuristics in the implementation of EFP. We note that epistemic planners in the literature do not focus on this

[24]Described in §2.4.3.

issue because the majority of them translate an epistemic planning problem into a classical planning problem and uses classical planners for computing the solutions. The exceptions to this trend are presented in [26, 50] where the planning systems described in these papers use a heuristic similar to the number of satisfaction subgoals in single agent planning. In this thesis, we approach this problem from a different angle. Specifically, we start with the fact that we search for plans using forward-search. It is easy to see that a generalization of the notion of a planning graph in single-agent to epistemic planning is a reasonable choice because it allows us to compactly represent the search tree using one single data structure and to extract different ways to extract heuristics. We therefore introduce the notion of an *Epistemic Planning Graph (EPG)* for epistemic planning.

As in single-agent domains, we expect that an epistemic planning graph will also consist of alternating *state levels* and *action levels*. While the content of action levels is obvious (i.e., it should be actions), it is not trivially clear *what should the state levels contain?* There are at least two possible answers: formulae in $\mathcal{L}(\mathcal{F}, \mathcal{AG})$ or epistemic states. The former would require the computation of all effects of an action, which could be an infinite set of formulae, and it is not desirable; this set could be approximated by finding an approximation of common knowledge but this not trivial (e.g., a public announcement of a fluent f makes f a common knowledge). For this reason, we use e-states in state levels. Intuitively, each e-state in a state level represents the updated knowledge/belief of the agents

about some fluents after an action is executed. Before we present the formal definition of this notion, we illustrate this idea in the following figure.

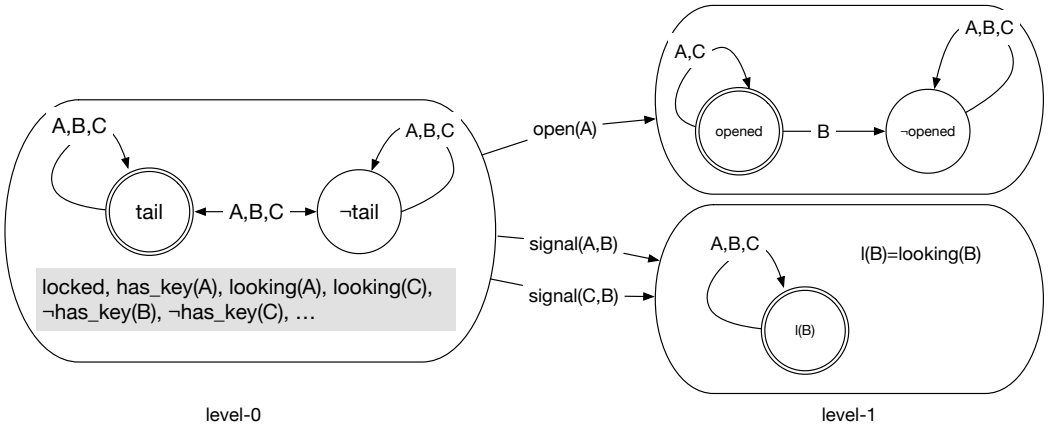


Figure 17: Epistemic planning graph: an illustration

In Figure 17, level 0 contains the unique e-state of the planning problem. In this e-state, the actions $open(A)$, $signal(A, B)$, $signal(C, B)$, etc. are executable. The execution of $open(A)$ in the pe-model of the initial e-state will result in A and C know that the box is open and B oblivious about this fact. This creates the top-right e-state of the figure (top of level-1). The execution of either $signal(A, B)$ or $signal(C, B)$ will result in B looking at the box and everyone knows that everyone is looking at the box; this creates the bottom-right e-state of the figure (bottom of level-1).

As it is shown in the application of planning graph for single-agent domains, planning graphs with mutexes provide better heuristics. Therefore, it is natural to expect that EPG should also include mutexes. By definition, a mutex represents

a pair of actions with conflicting effects. This means that computing mutexes requires checking whether two actions produce conflicting effects. As it turns out, this is computational cheap and straightforward in single-agent domains under PDDL-specification, computing effects of actions under m$\mathcal{A}^*$ is not straightforward. For these reasons, we leave the mutex definition for future work.

3.3.1 Formal Definitions

We will next formally define the notion of an epistemic planning graph. Given a planning problem $P = \langle \mathcal{F}, \mathcal{AG}, A, O, s_0, \phi_g \rangle$, we say that P is *deterministically observable* (or *d-observable*) if O contains only statements of the form "α **observes** a" and "α **aware_of** a." It is easy to see that every planning problem P can be translated[25] into an equivalent d-observable problem P', i.e., a solution to P is equivalent to a solution to P' and vice versa. Intuitively, in d-observable problems, $F(a, \mathcal{M}, w)$ (or $P(.)$ and $O(.)$) is independent from $(\mathcal{M}, w)$. For this reason, we will use F_a, P_a, and O_a to denote the sets of agents $F(a, \mathcal{M}, w)$, $P(a, \mathcal{M}, w)$, and $O(a, \mathcal{M}, w)$, respectively. This simplified the definition of an epistemic planning graph. For simplicity of the presentation, we will present the notion of an epistemic planning graph only for d-observable planning problems. Observe that, from now on, we assume that $P = \langle \mathcal{F}, \mathcal{AG}, A, O, s_0, \phi_g \rangle$ is given and

[25]This process is similar to the process of converting an action with conditional effects to a set of actions without conditional effects in classical planning.

is d-observable.

In the following, we will consider *incomplete* e-model $(W, \mathcal{B}, \pi)$ where W, $\mathcal{B}$, and π are defined similarly to the components of an e-model with the exception that $\pi(w)$ can be a partial interpretation of $\mathcal{F}$. An incomplete pe-model (or e-state) is of the form $(\mathcal{M}, w)$ (or $(\mathcal{M}, W)$) where $\mathcal{M}$ is an incomplete e-model. From now on, whenever we refer to an (p)e-model (or e-state), we mean a (possibly incomplete) (p)e-model (or e-state).

For a set of e-states $\mathcal{P} = \{(\mathcal{M}_1, W_1), \ldots, (\mathcal{M}_n, W_n)\}$ and a conjunction of fluent literals $\psi = \ell_1 \wedge \ell_2 \ldots \wedge \ell_k$, we say $\mathcal{P} \mid\!\sim \psi$ if for every $1 \leq i \leq k$, there exists some $1 \leq j \leq n$ such that $(\mathcal{M}_j, W_j) \models \ell_i$. For a fluent formula φ, let $\bigvee_{i=1}^{k} \varphi_k$ be its DNF, i.e., each φ_i is a conjunction of literals. We say $\mathcal{P} \mid\!\sim \varphi$ iff $\mathcal{P} \mid\!\sim \varphi_i$ for some $1 \leq i \leq k$. Intuitively, if $\mathcal{P}$ represents a state level, $\mathcal{P} \mid\!\sim \varphi$ means that the level, $\mathcal{P}$, "*possibly entails*" φ. $\mid\!\sim$ is used for checking whether or not some formula could be entailed from a state level. This is similar to the verification of the presence of a literal in a state level in a planning graph. $\mid\!\sim$ is generalized to arbitrary formula as follows.

Definition 3.1 (Entailment in Planning Graph)**.** Given a set of e-states $\mathcal{P} = \{(\mathcal{M}_1, W_1), \ldots, (\mathcal{M}_n, W_n)\}$ where $\mathcal{M}_j = (W_j, \mathcal{B}_j, \pi_j)$ for $1 \leq j \leq n$, and a belief formula φ, we say $\mathcal{P} \mid\!\sim \varphi$ if

- φ is a fluent formula and $\mathcal{P} \mid\!\sim \varphi$;

- $\varphi = K_i\psi$ and $\{(\mathcal{M}_1, W_1'), \ldots, (\mathcal{M}_n, W_n')\} \mid\sim \psi$ where for $1 \leq j \leq n$, $W_j' = \{w_j' \mid \exists w_j \in W_j \text{ such that } (w_j, w_j') \in \mathcal{B}_j(i)\}$;
- $\varphi = \neg\psi$ and ψ is not a fluent formula and $\mathcal{P} \not\mid\sim \psi$;
- $\varphi = \varphi_1 \vee \varphi_2$ and ($\mathcal{P} \mid\sim \varphi_1$ or $\mathcal{P} \mid\sim \varphi_2$);
- $\varphi = \varphi_1 \wedge \varphi_2$ and ($\mathcal{P} \mid\sim \varphi_1$ and $\mathcal{P} \mid\sim \varphi_2$);
- $\varphi = E_\alpha\psi$ and $\mathcal{P} \mid\sim K_i\psi$ for every $i \in \alpha$; and
- $\varphi = C_\alpha\varphi$ and $\mathcal{P} \mid\sim E_\alpha^k\varphi$ for every $k \geq 1$, where $E_\alpha^1\varphi = E_\alpha\varphi$ and $E_\alpha^{k+1}\varphi = E_\alpha(E_\alpha^k\varphi)$.

This allows us to define the notion of an action *potentially applicable* from a set of e-states, which allows us to compute the action level of an epistemic planning graph, as follows.

Definition 3.2 (Potentially Applicable Action)**.** An action $a \in A$ is *potentially applicable* in a state level $\mathcal{K}$ if there exists a set of e-states $\mathcal{PS} \subseteq \mathcal{K}$ such that $\mathcal{PS} \mid\sim pre(a)$.

Let $\mathcal{K}$ be a set of e-states and a be an action potentially applicable in $\mathcal{K}$. To complete the definition of an epistemic planning graph, we need to define the set of e-states that could be obtained given that a is executed. We first define a set of e-states $E(\mathcal{K}, a)$ as follows.

- *a is an ontic action*: For each [a **causes** ℓ **if** φ] in A such that $\mathcal{K} \mid\sim \varphi$, let $\mathcal{M} = (W, \mathcal{B}, \pi)$ where

  - $W = \{u, v\}$;
  - $\mathcal{B} : \mathcal{AG} \to 2^{W \times W}$ is defined as:
    - $\mathcal{B}(i) = \{(u, u), (v, v)\}$ for all $i \in F_a$; and
    - $\mathcal{B}(i) = \{(u, v), (v, v)\}$ for all $i \in O_a$.
  - $\pi(u) = \{\ell\}$ and $\pi(v) = \{\neg\ell\}$.

  $E(\mathcal{K}, a) = \{(\mathcal{M}, \{u\}) \mid [a$ **causes** ℓ **if** $\varphi] \in A$ such that $\mathcal{K} \mid\sim \varphi\ \}$.

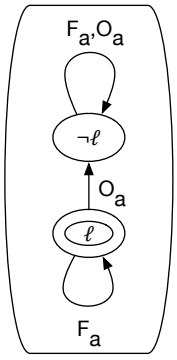
- *a is a sensing action*: $E(\mathcal{K}, a) = \{(\mathcal{M}, \{u^1\}) \mid [a$ **determines** $f] \in A$ *and* $\mathcal{K} \mid\sim f\} \cup \{(\mathcal{M}, \{u^2\}) \mid [a$ **determines** $f] \in A$ *and* $\mathcal{K} \mid\sim \neg f\}$ where $\mathcal{M} = (W, \mathcal{B}, \pi)$ and

  - $W = \{u^1, u^2, v^1, v^2\}$;
  - $\mathcal{B} : \mathcal{AG} \to 2^{W \times W}$ is defined as:
    - For $i \in F_a$: $\mathcal{B}(i) = \{(u^1, u^1), (u^2, u^2)\} \cup \{(v^1, v^1), (v^2, v^2), (v^1, v^2), (v^2, v^1)\}$;
    - For $i \in P_a$: $\mathcal{B}(i) = \{(u^1, u^1), (u^2, u^2), (u^1, u^2), (u^2, u^1)\}$
      $\cup \{(v^1, v^1), (v^2, v^2), (v^1, v^2), (v^2, v^1)\}$;
    - For $i \in O_a$: $\mathcal{B}(i) = \{(u^1, v^1), (u^1, v^2), (u^2, v^1), (u^2, v^2)\}$
      $\cup \{(v^1, v^1), (v^2, v^2), (v^1, v^2), (v^2, v^1)\}$;

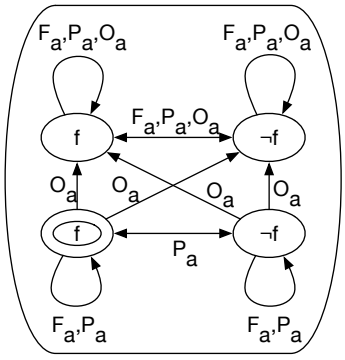
- $\pi(u^1)$=$\pi(v^1)$=$\{f\}$ and $\pi(u^2)$=$\pi(v^2)$=$\{\neg f\}$.

- *a is an announcement action*: assume that [a **announces** φ] is in A. Let W^1 and W^2 be two disjoint sets, each represents the set of all complete interpretations of the set of fluents occurring in φ. We define $E(\mathcal{K}, a) = \{(\mathcal{M}, \{w \mid w \in W^1, w \models \varphi\})\}$ where $\mathcal{M} = (W^1 \cup W^2)$ and

  - For $i \in F_a$: $\mathcal{B}(i) = \{(w, w') \mid w, w' \in W^1 \wedge w \models \varphi \wedge w' \models \varphi\} \cup \{(w, w') \mid w, w' \in W^1 \wedge w \not\models \varphi \wedge w' \not\models \varphi\} \cup \{(w, w') \mid w, w' \in W^2\}$;
  - For $i \in P_a$: $\mathcal{B}(i) = \{(w, w') \mid w, w' \in W^1\} \cup \{(w, w') \mid w, w' \in W^2\}$;
  - For $i \in O_a$: $\mathcal{B}(i) = \{(w, w') \mid w, w' \in W^2\} \cup \{(w, w') \mid w \in W^1 \wedge w' \in W^2\}$.



(a) Ontic action.



(b) Sensing or announcement action.

Figure 18: Examples of resulting e-states in $E(\mathcal{K}, a)$

Intuitively, $E(\mathcal{K}, a)$ is the set of e-states that partially represents the updated knowledge/belief of the agents after action a is fired. Each e-state in $E(\mathcal{K}, a)$

encodes that agents in F_a know the effects of a, agents in O_a are oblivious, and agents in P_a do not know the effects of a but are aware that those in F_a know the effects of a. Figure 18(a) presents the e-state resulting from the firing of a with the statement $[a$ **causes** ℓ **if** $\top] \in A$, and Figure 18(b) illustrates an e-state of firing a sensing action (resp. an announcement action) a where $[a$ **determines** $f] \in A \wedge \mathcal{K} \mathrel{|\!\sim} f$ (resp. $[a$ **announces** $f] \in A$).

Definition 3.3 (Epistemic Planning Graph)**.** Given a planning problem $P = \langle \mathcal{F}, \mathcal{AG}, A, O, s_0, \phi_g \rangle$, the *epistemic planning graph* of P is an alternative sequence of state levels and action levels $\mathcal{K}_0, A_0, \ldots, \mathcal{K}_k, A_k, \ldots$ where

- $\mathcal{K}_0$ is the set consisting of the unique initial e-state of P;
- for $i \geq 0$,
  - A_i is the set of actions potentially applicable in $\mathcal{K}_i$; and
  - $\mathcal{K}_{i+1} = \mathcal{K}_i \cup \big(\bigcup_{a \in A_i} E(\mathcal{K}_i, a)\big)$.

Algorithm 2 shows the computation of an epistemic planning graph of a planning problem given the e-state in the first state-level. At each iteration, the set of potentially applicable actions are computed and then the set of potential effects of these actions is added to the next state level. The following property guarantees that Algorithm 2 terminates.

Proposition 3.1. *For a planning problem with finite sets of actions A, Algorithm 2 terminates.*

The proof of this proposition relies on the following observations:

(i) $\mathcal{K}_{i-1} \subseteq \mathcal{K}_i$ (Line 19); and thus

(ii) if an action is potentially applicable at level i then it is also potentially applicable at level $i+1$ (Definition 3.2);

(iii) $E(\mathcal{K}_i, a) \subseteq E(\mathcal{K}_{i+1}, a)$ and $E(\mathcal{K}, a)$ is finite.

It is also easy to see that since $\mid\!\sim$ stands for "possibly entails," the following property holds.

Proposition 3.2. *Let $\langle \mathcal{K}_0, A_0, \ldots, \mathcal{K}_i, A_i \rangle$ be the epistemic planning graph returned by Algorithm 2 with the planning problem $P = \langle \mathcal{F}, \mathcal{AG}, A, O, s_0, \phi_g \rangle$ and $(\mathcal{M}, W)$ as input. Furthermore, let φ be a formula and j be the smallest index such that $\mathcal{K}_j \mid\!\sim \varphi$. Then, the shortest solution of the planning problem $\langle \mathcal{F}, \mathcal{AG}, A, O, \{(\mathcal{M}, W)\}, \varphi \rangle$ has at least j actions.*

In the following, we denote with $level(\varphi)$ the smallest state level index in the epistemic planning graph such that $\mathcal{K}_{level(\varphi)} \mid\!\sim \varphi$.

3.3.2 Heuristics from Epistemic Planning Graphs

It is well-known that there are several possible ways to extract heuristics from a planning graph in planning in single-agent domain [35]. Studying different

heuristics from epistemic planning graphs is an interesting topic of research of its own right, but it is not the focus of this thesis. For this reason, we will describe the heuristics that we used in the experiments in the next section.

We experimented with two heuristics derived from the epistemic planning graphs. We assume that ϕ_g is a conjunction of formulas $\phi_1 \wedge \ldots \wedge \phi_n$ where ϕ_i is either a fluent formula or a formula of the form $K_i\varphi$ or $C_X\varphi$. We define

$$h^{max}(\phi_g) = \max\{level(\phi_i) \mid i = 1, \ldots, n\} \tag{7}$$

and

$$h^{sum}(\phi_g) = \sum_{i=1}^{n} level(\phi_i) \tag{8}$$

Algorithm 1: EFP($\langle \mathcal{F}, \mathcal{AG}, A, O, s_0, \phi_g \rangle$)

Input : A planning problem $P = \langle \mathcal{F}, \mathcal{AG}, A, O, s_0, \phi_g \rangle$

Output: A solution for P if exists; failed otherwise

Compute the initial e-state given s_0: $(\mathcal{M}_i, W_i)$

Initialize a priority queue $q = [(\{(\mathcal{M}_i, W_i)\}, [])]$

while *q is not empty* **do**

  $(\Omega, plan) = dequeue(q)$

  if $(\mathcal{M}, W_d) \models \phi_g$ *for every* $(\mathcal{M}, W_d) \in \Omega$ **then**

    return *plan*

  end

  for *action a executable in every* $(\mathcal{M}, W_d)$ *in* Ω **do**

    Compute $\Omega' = \bigcup_{(\mathcal{M}, W_d) \in \Omega} \Phi(a, (\mathcal{M}, W_d))$

    Compute heuristics and insert $(\Omega', plan \circ a)$ into q

  end

end

return failed

Algorithm 2: EpistemicPlanningGraph(P, $(\mathcal{M}_0, W_0)$)

Input : a planning problem $P = \langle \mathcal{F}, \mathcal{AG}, A, O, s_0, \phi_g \rangle$ and an e-state $(\mathcal{M}_0, W_0)$

Output : An epistemic planning graph whose first state level is $\{(\mathcal{M}_0, W_0)\}$

Let $\mathcal{K}_0 = \{(\mathcal{M}_0, W_0)\}$

$i = 0$

while *true* **do**

  Compute $A_i = \{a \in A \mid a \textit{ is potentially applicable given } \mathcal{K}_i\}$

  $i = i + 1$

  Let $\mathcal{K}_i = \mathcal{K}_{i-1} \cup \left(\bigcup_{a \in A_i} E(\mathcal{K}_{i-1}, a)\right)$

  if $\mathcal{K}_i \equiv \mathcal{K}_{i-1}$ *and* $A_i \equiv A_{i-1}$ **then**

    return $\langle \mathcal{K}_0, A_0, \ldots, \mathcal{K}_i, A_i \rangle$

  end

end

4 EXPERIMENTAL EVALUATION

As we have mentioned earlier, we implement two planners–one that uses Epistemic Planning Graph as heuristics, which we call "PG-EFP", and one that does not use heuristic but performs *breadth first search* to search for the plan, which we call "EFP". We note that the heuristic used in PG-EFP is the h^{sum} heuristic defined in 8. We note that we did experiment with h^{max}, but its performance is generally worse than EFP with h^{sum}.

In our experiments, we compare EFP and PG-EFP against the two systems proposed in [33, 26]. The first system, which is called RP-MEP, supports the notion of planning with nested belief, and demonstrates how to automatically convert such planning problems into problems that can be solved using classical planning technology. The limitation of this system is that it limits the number of nested belief modality. The second system, which is called MEPK, supports efficient reasoning in the multi-agent KD45 logic by using alternating cover disjunctive formulas (ACDFs). Such ACDFs are subjected to proposed belief-revision and update algorithms that adapt the PrAO algorithm originally developed for contingent planning. We use publicly-available implementations of RP-MEP and MEPK in our experiments.[26] We did not compare our system with the system in [30] as it is an online planner. We also did not compare our system with the

[26]We retrieved RP-MEP from `https://bitbucket.org/haz/pdkb-planning` and MEPK from `https://github.com/sysulic/MEPK`.

system in [29] as this has been already compared with the system in [26] and its performance is not as good as that of MEPK.

We evaluate EFP and PG-EFP using the *Selective-communication* (SC), and *Collaboration-and-communication* domains that are adapted from [29],[27] as well as *Assemble Line* introduced in [26], and *Logistics* (LO) that was used in the Competition of Distributed and Multiagent Planners 2015 (CoDMAP'15).[28]

In the *Selective-communication* (SC(n,m)), there are n agents, each of them initially is in one of m rooms in a corridor. An agent, says a, can move from a room to a neighboring room by actions *left* and *right*. When the agent tells some information—which is the truth value of a fluent q—in a room i (i.e., executing an announcement action "*shout_i*"), all the other agents in the same room i or in a neighboring room of i can hear what was told. The goal is that some agents get to know q while some other agents do not. In this experiment, we will vary n, m, the depth of the knowledge d, and the length of the plan by varying the goal.

In the *Collaboration-and-communication* (CC(n,m,k)), there is a corridor of $k \geq 2$ rooms. m boxes are located in some rooms. n agents can move back and forth along this corridor. When an agent gets into a room, he can see if a box is in the room. An agent can communicate information to another agent. Initially,

[27]SC is called "Corridor" in [33]

[28]LO is retrieved from `http://agents.fel.cvut.cz/codmap` with *centralized* and *unfactored-privacy* setting.

we set all agents are in room 2 and the boxes are not there. The goals are varied in which some agents know the position of some boxes.

The *Grapevine* (GR(n)) problem is a modification of a similar domain from [29]: n agents are located in two rooms, and they share secrets with agents in the same room. The domain supports different goals, from sharing secrets with other agents to having misconceptions about agents' beliefs.

Assembly-line: AL(d). There are two agents, each responsible for processing a part of a product. It is possible that an agent fails in processing his part. An agent can inform the other agent of the status of his task. Two agents decide to assemble the product or restart, depending on their knowledge of the status of the agents' tasks. In order to compare with RP-MEP and MEPK, we vary the depth of the knowledge d in this experiment.

Logistics: LO. The logistics domain is recommended by one of the reviewers to evaluate the scalability of EFP and PG-EFP since it consists of a large number of fluents and actions. This benchmark is not originally developed for epistemic planning. As such, we use its slightly-modified version by modeling the public/privacy separation of fluents [10] using the "observes" statement of the form (5). In our experiment, the Logistics problem has 3 agents (i.e., airplane, truck1, and truck2), 2 packages, and 4 locations (i.e., "airport1", "airport2", "pos1", and "pos2"). The packages can be moved from one location to another location by being loaded/unloaded onto agents and then moving agents. We vary the length

Selective Communication: SC(3,4) $\lvert\mathcal{AG}\rvert = 3$, $\lvert\mathcal{F}\rvert = 5$, $\lvert A\rvert = 7$						Selective Communication: SC(5,6) $\lvert\mathcal{AG}\rvert = 5$, $\lvert\mathcal{F}\rvert = 7$, $\lvert A\rvert = 9$						Selective Communication: SC(7,8) $\lvert\mathcal{AG}\rvert = 7$, $\lvert\mathcal{F}\rvert = 9$, $\lvert A\rvert = 11$					
L	d	MEPK	RP-MEP	EFP	PG-EFP	L	d	MEPK	RP-MEP	EFP	PG-EFP	L	d	MEPK	RP-MEP	EFP	PG-EFP
2	1	.01	.1	.01	.02	2	1	.6	.2	.02	.04	5	1	35	.36	.22	TO
	3	.2	.5	.02	.07		3	TO	3.2	.02	.04		3	TO	10.7	.22	TO
	5	TO	28	.03	.08		4	TO	51.58	.03	.04		4	TO	292	.24	TO
3	1	.02	.1	.02	.06	4	1	.68	.2	.07	TO	7	1	35	.36	1.9	TO
	3	.2	.5	.02	.07		3	TO	3.18	.08	TO		3	TO	10.8	1.92	TO
	5	TO	30	.02	.06		4	TO	54.78	.08	TO		4	TO	300	1.9	TO
5	1	.05	.1	.08	TO	6	1	.81	.2	.51	.35	9	1	35.7	.32	23.7	1.86
	3	.21	.6	.09	TO		3	TO	3.21	.52	.36		3	TO	12.72	24	1.9
	5	TO	28	.1	TO		4	TO	51.81	.51	.34		4	TO	312	23.5	1.93

Table 2: Runtimes for Selective Communication Problems

L of the optimal plan by changing the desired locations of packages as goals.

All experiments are performed on a 2.8 GHz Intel Core i7 machine with 16GB of memory. We report the runtimes in second for all experiments. We set the timeout to 25 minutes. "TO" means one algorithm fails to solve one problem due to timeout. The results of our experiments are summarized in Tables 2–6. In the tables, L denotes the length of the shortest plan (optimal plan) and d the depth of knowledge. We make the following observations:

- EFP performs reasonably well comparing to other systems in the SC domain. When depth (d) of the knowledge increases, EFP's performance does not decrease but the performance of other systems (i.e., MEPK and RP-MEP) gets significantly worse. The reason is that the representation of the problem in EFP remains unchanged when d increases. In contrast, the size of the problem representation for MEPK and RP-MEP increases when d increases. This results in the worse performance of MEPK and RP-MEP

when d increases. In this domain, PG-EFP cannot solve some instances which contain goals with negation (e.g., goal of the form $\neg K_1\varphi$). Our analysis shows that the heuristic h^{max} does not work well for this situation. On the other hand, if PG-EFP can solve an instance; it is almost always the fastest, due to the heuristic h^{sum}.

- EFP performs well in the Grapevine domain against RP-MEP. It is not as fast as RP-MEP in the first configuration of this problem but is faster than RP-MEP when d increases in Configuration 2. We believe that this reflects the trade-off between the representation and the computation of the transition function. In general, computing Φ is more expensive than computing the transition function implemented in RP-MEP. This is the reason when the domain is small, EFP is not as good as RP-MEP. However, when the size of the problem increases (i.e., $|\mathcal{AG}|$, $|\mathcal{F}|$, $|A|$, and d increase in Configuration 2), the size of the representation used in RP-MEP increases and this affects its performance much more than the complexity of computing Φ.

  We were unable to run PG-EFP in this domain as our naive translation from general domain to deterministically-observable produces a significantly larger input. We believe that some optimization of the translation could help in this regard.

- For the AL domain (Table 4(Left)), we were unable to create the input for

Grapevine - Configuration 1 $\lvert\mathcal{AG}\rvert = 3$, $\lvert\mathcal{F}\rvert = 9$, $\lvert A\rvert = 24$				Grapevine - Configuration 2 $\lvert\mathcal{AG}\rvert = 5$, $\lvert\mathcal{F}\rvert = 15$, $\lvert A\rvert = 60$			
L	d	RP-MEP	EFP	L	d	RP-MEP	EFP
2	1	0.090	0.034	2	1	0.260	0.912
	2	0.255	0.036		2	1.970	0.908
	3	1.178	0.035		3	26.110	0.974
4	1	0.088	1.130		4	1499	0.954
	2	0.256	1.125		5	TO	0.982
	3	1.222	1.141	3	1	0.271	10.879
5	1	0.090	22.687		2	1.990	10.320
	2	0.267	22.692		3	26.407	10.568
	3	1.220	22.694		4	1575	10.836
					5	TO	10.967
				4	1	0.264	88.662
					2	2.035	90.128
					3	26.510	89.135
					4	1632	87.631
					5	TO	88.164

Table 3: RP-MEP vs. EFP in Grapevine

RP-MEP as the instances become too large to do the translation manually. Other than that, the performances of EFP and PG-EFP are similar to their performances in the SC domain and are independent of the number of depth of knowledge. MEPK's performance decreases when the depth of knowledge increases.

- The coin-in-the-box domain (Example 2.1) is not solvable using other planners. Table 4(Right) shows the comparison between EFP and PG-EFP to investigate the influence of the heuristic. In the last column of Table 4(Right),

(X) stands for the length X of the plan returned by PG-EFP, and (N/A) means that the respective problem has no plan. We change the goals to create different instances. We can observe that PG-EFP is often faster than EFP.

Assemble Line: AL(d) $\lvert\mathcal{AG}\rvert = 2$, $\lvert\mathcal{F}\rvert = 4$, $\lvert A\rvert = 6$			
d	MEPK	EFP	PG-EFP
2	.03	1.9	.9
5	.11		
10	5.47		
15	175		
20	TO		

Coin in the Box $\lvert\mathcal{AG}\rvert = 3$, $\lvert\mathcal{F}\rvert = 8$, $\lvert A\rvert = 31$		
L	EFP	PG-EFP
2	.04	.27(2)
3	.22	1.16(3)
5	3.58	1.44(6)
no plan	TO	.43(N/A)

Table 4: AL domain (left) and Coin-in-the-box domain (Right)

- Table 5 compares the two systems EFP and PG-EFP in the CC domain. In this domain, we did not run the experiment with MEPK as it produces solutions that are not an action sequence and only runs with $d = 1$. The experiments are done on four different configurations and different goals. We note that CC(3,3,3) has smaller number of $|\mathcal{F}|$ and $|A|$ than CC(3,2,3) because we experiment with a different representation of the problem. It is interesting to observe that PG-EFP is slower than EFP in less-complicated problems (i.e., problems with $L = 2$), but significantly faster than EFP (several magnitudes faster) in more-complicated problems (i.e., problems with $L > 2$). The reason for the former observation is that PG-EFP needs

to compute epistemic planning graphs while EFP does not. However, when the problems become more complex, the search space of PG-EFP seems to reduce significantly and much smaller than that of EFP due to the heuristic derivable from the EPG. These results show that the heuristic produced by the epistemic planning graph is indeed quite useful.

Problem	L	EFP	PG-EFP
CC(2,2,3) $\|\mathcal{AG}\| = 2$, $\|\mathcal{F}\| = 10$, $\|A\| = 16$	2	.61	.81
	5	48.6	2.5
	6	278.6	4.3
CC(2,2,4) $\|\mathcal{AG}\| = 2$, $\|\mathcal{F}\| = 14$, $\|A\| = 22$	2	22.2	27.5
	4	TO	70
	7	TO	160
CC(3,2,3) $\|\mathcal{AG}\| = 3$, $\|\mathcal{F}\| = 13$, $\|A\| = 24$	2	3.3	2.3
	5	257.9	7.9
	6	TO	10.3
CC(3,3,3) $\|\mathcal{AG}\| = 3$, $\|\mathcal{F}\| = 12$, $\|A\| = 21$	2	.42	.68
	5	115.7	2.27
	6	TO	3

Table 5: EFP vs. PG-EFP in CC domain

- The logistics domain cannot be modeled using the specification language m$\mathcal{A}^*$ in a straightforward manner due to the fact that the observability of agents is specified at the fluent levels (e.g, when an action is executed, some agents observe one effect and others observe another one), and m$\mathcal{A}^*$ does not consider this situation yet. For this reason, we used a slightly-modified representation of this domain in testing EFP and PG-EFP. Table 6

displays the runtime comparison between the two systems EFP and PG-EFP in the LO domain. In Table 6, (X) stands for the length X of the plan that is returned by PG-EFP. The results showed in Table 6 are consistent with those shown earlier, and exhibit that PG-EFP can scale up to solve problems with a large number of fluents and actions. This is clearly due to the heuristic produced by the epistemic planning graph.

Logistics $\|\mathcal{AG}\| = 3$, $\|\mathcal{F}\| = 66$, $\|A\| = 84$		
L	EFP	PG-EFP
3	10.32	16.63(3)
4	75.1	27.88(4)
30	TO	1355.6(31)
31	TO	1447.2(36)

Table 6: EFP vs. PG-EFP in LO domain

Solution quality: We now discuss about solution quality in terms of the length of the plans that are returned by EFP and PG-EFP. In our experiments, we observed that PG-EFP returns optimal plans in most problems, except for some in the coin-in-the-box domain (see the row of $L = 5$ in Table 4(Right)) and some in the logistics domain (see the rows of $L = 30, 31$ in Table 6). Thus, we report only the plan lengths for PG-EFP in the coin-in-the-box domain and the logistics domain. Theoretically, EFP returns the optimal plan whose length is smallest since it is a breadth-first search planner. In contrast, as PG-EFP is a heuristic search planner that uses heuristic derived from epistemic planning graph, its plan

is not guaranteed to be optimal.

5 CONCLUSIONS AND FUTURE WORKS

5.1 Conclusion

EFP (or PG-EFP) is closely related to RP-MEP [33] or the system (called K(P) hereafter) described in [29]. The two planners differ from the system MEPK in [26] in that they do not use a special search algorithm why MEPK uses a special search algorithm, called PrAO, from [45]. EFP (or PG-EFP) is an alternative to the other systems in two aspects: *(i)* the methods of searching for solution in EFP is different; *(ii)* EFP and K(P) can deal with common knowledge while other systems do not.

The proposed systems represent an ideal testbed for research in the area of epistemic planning in multi-agent settings. The advantages offered by the proposed platform are:

- The system is the first of its kind, supporting planning with complex forms of knowledge and beliefs; EFP does not require restrictions on the nesting of knowledge/beliefs, thus allowing us, e.g., to reason about common knowledge.
- The platform, even in its current prototype form, can solve problems for which RP-MEP or other systems have difficulty; for some problems, even for a small number of nested beliefs (e.g., 5), the generation of the planning instance for other planners takes more than an hour.
- The system serves as a research platform; its modular organization allows re-

searchers to experiment with different transitions, heuristics, and state representations.

- Finally, it should be noted that EFP and PG-EFP can deal with both knowledge and belief goals as [43] shows that epistemic actions of the form described in this paper, which are generated from an action a and an e-state $(\mathcal{M}, W)$, can maintain the $\mathbf{KD}_{45}$ properties of an e-state. As such, a goal with both K and B-modalities can be dealt with by (a) using the equivalence $K\varphi \equiv B\varphi \wedge \varphi$ to remove the occurrences of K in the goal; and then (b) planning with the new goal.

Although EFP performs reasonably well, there are a number of issues that require further study so that it can efficiently deal with larger problems. In our experiments, we observe the following:

- If the number of unknown fluents in the actual world of the initial e-state is high, then EFP does not work well. The reason for this is the size, in terms of the number of worlds and the number of edges, of the initial e-state. For example, for the CC(2, 3, 4) problems, there are 17 fluents whose values are unknown in the initial e-state. This leads to the initial state has 512 pe-models, and 524288 edges. This raises the question of how to represent and reason with pe-models and e-states with a high-degree of connectivity.

- An e-state is essentially a graph. Checking for graph isomorphism is not a

computational easy task. As such, we did not check for repeated element in the queue q of Algorithm 1. How best to check for repeated element in the queue and whether or not it improves the performance of EFP are the two questions left for the future work.

We describe two forward search epistemic planners, EFP and PG-EFP, and experimentally evaluate these planners with domains collected from the literature. We also introduce the notion of an epistemic planning graph and provide algorithm for computing epistemic planning graphs. The experimental evaluation of the two planners shows that both are working reasonable well comparing to other epistemic planners; and that heuristics derived from epistemic planning graphs are indeed useful.

5.2 Future Works

As mentioned in §3 computing the mutex definition for EPG is not straightforward. A deepen investigation of the mutex is though necessary to provide a more complete and efficient description of epistemic planning graph.

With the introduction of the notion of epistemic planning graph the possibility to retrieve several heuristic functions has become easier. In fact the EPG can play a role in epistemic planning that can be similar to the one played by the planning graph structure in classical planning. Therefore, since from the classical planning

graph numerous heuristics can be extracted [36], we can assume that the same can be true for EPG. While EFP and PG-EFP only use two heuristic derived by the planning graph it will be interesting to examine and compare other heuristics that can be estimated from the planning graph.

On the other hand other approaches, different from the planning graph, exist that can increase the planner performances. Reducing the search space could be one of these. As future study would then be interesting find techniques that can extract, from the problem domain, information that can reduce the search space itself. For example, in some domain, could happen that some fluents and/or some beliefs, after the execution of n[29] actions, will never change their interpretation. Finding this set of fluents or beliefs will permit to reduce the search space, removing from the possible outcomes all the states with a variations in this set.

Another study that we would like to elaborate in future is about the introduction of a classical planning component in EFP and PG-EFP. In fact, as is shown in [50, 26], when the domain problem respects determinate conditions this could be translate into classical planning and solved with well known and well performing techniques. It would be interesting finding a way to examine the domain and see if it respect these conditions and then, when it is possible, use the translation to

[29] Where n could also be equal to 0.

classical planning to improve the performance of the systems.

As a matter of fact the EFP and PG-EFP systems suffer a huge overhead due the large data structures used to represent the e-states. Along with the computational overhead these complex data structures bring with them other complications. As a future objective we have several optimizations for the planners systems and their data structures: In particular:

- Develop methods for comparing e-states so that we can eliminate repeated e-states from the queue;
- investigate alternative representation that can improve the performance (in time and space) of the planners;
- a more clean computation of the initial state from the problem description.

REFERENCES

[1] Martin Allen and Shlomo Zilberstein. Complexity of decentralized control: Special cases. In *Advances in Neural Information Processing Systems*, pages 19–27, 2009.

[2] Guillaume Aucher and Thomas Bolander. Undecidability in epistemic planning. In *IJCAI-International Joint Conference in Artificial Intelligence-2013*, 2013.

[3] Alexandru Baltag and Lawrence S Moss. Logics for epistemic programs. *Synthese*, 139(2):165–224, 2004.

[4] Chitta Baral, Thomas Bolander, Hans van Ditmarsch, Sheila McIlraith, Xiaojun Bi, Otmar Hilliges, Takeo Igarashi, Antti Oulasvirta, Paul W Goldberg, Yishay Mansour, et al. Dagstuhl reports, vol. 7, issue 6 issn 2192-5283. 2018.

[5] Chitta Baral, Gregory Gelfond, Enrico Pontelli, and Tran Cao Son. An action language for multi-agent domains: Foundations. *CoRR*, abs/1511.01960, 2015.

[6] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.

[7] Thomas Bolander and Mikkel Birkegaard Andersen. Epistemic planning for single-and multi-agent systems. *Journal of Applied Non-Classical Logics*, 21(1):9–34, 2011.

[8] Thomas Bolander, Martin Holm Jensen, Francois Schwarzentruber, and ENS Rennes. Complexity results in epistemic planning. In *IJCAI*, pages 2791–2797, 2015.

[9] Michael H Bowling, Rune M Jensen, and Manuela M Veloso. Multiagent planning in the presence of multiple goals. *Planning in Intelligent Systems: Aspects, Motivations and Methods, John Wiley and Sons, Inc*, 2005.

[10] Ronen I. Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *Proc. of ICAPS*, pages 28–35, 2008.

[11] Claudio Castellini, Enrico Giunchiglia, and Armando Tacchella. Improvements to sat-based conformant planning. In *Sixth European Conference on Planning*, 2014.

[12] Tristan Charrier, Bastien Maubert, and François Schwarzentruber. On the impact of modal depth in epistemic planning. In *IJCAI*, pages 1030–1036, 2016.

[13] Matthew Crosby, Anders Jonsson, and Michael Rovatsos. A single-agent approach to multiagent planning. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, pages 237–242. IOS Press, 2014.

[14] Mathijs De Weerdt, André Bos, Hans Tonino, and Cees Witteveen. A resource logic for multi-agent plan merging. *Annals of Mathematics and Artificial Intelligence*, 37(1-2):93–130, 2003.

[15] Mathijs De Weerdt and Brad Clement. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, 5(4):345–355, 2009.

[16] Edmund H Durfee. Distributed problem solving and planning, multiagent systems: a modern approach to distributed artificial intelligence, 1999.

[17] Thorsten Engesser, Thomas Bolander, Robert Mattmüller, and Bernhard Nebel. Cooperative epistemic multi-agent planning for implicit coordination. *arXiv preprint arXiv:1703.02196*, 2017.

[18] Ronald Fagin, Joseph Y Halpern, Yoram Moses, and Moshe Vardi. *Reasoning about knowledge*. MIT press, 2004.

[19] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[20] Nicoletta Fornara. *Interaction and communication among autonomous agents in multiagent systems*. PhD thesis, Università della Svizzera italiana, 2003.

[21] Michael Gelfond and Vladimir Lifschitz. Action languages. 1998.

[22] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[23] Claudia V Goldman and Shlomo Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *J. Artif. Intell. Res.(JAIR)*, 22:143–174, 2004.

[24] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored mdps. In *Advances in neural information processing systems*, pages 1523–1530, 2002.

[25] Andreas Herzig, Jérôme Lang, and Pierre Marquis. Action progression and revision in multiagent belief structures. In *Sixth Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC 2005)*. Citeseer, 2005.

[26] Xiao Huang, Biqing Fang, Hai Wan, and Yongmei Liu. A general multi-agent epistemic planner based on higher-order belief change. In *IJCAI 2017, Proceedings of the 26th International Joint Conference on Artificial Intelligence, Melbourne, Australia, Sydney*, 2017.

[27] Vaibhav Katewa. *Analysis and design of multi-agent systems under communication and privacy constraints*. University of Notre Dame, 2017.

[28] Emil Keyder and Héctor Geffner. Heuristics for planning with action costs revisited. In *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pages 588–592, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[29] Filippos Kominis and Hector Geffner. Beliefs in multiagent planning: From one agent to many. In *Proc. ICAPS*, pages 147–155, 2015.

[30] Filippos Kominis and Hector Geffner. Multiagent online planning with nested beliefs and dialogue. pages 186–194, 2017.

[31] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Elsevier, 1994.

[32] Benedikt Löwe, Eric Pacuit, and Andreas Witzel. Del planning and some tractable cases. In *International Workshop on Logic, Rationality and Interaction*, pages 179–192. Springer, 2011.

[33] Christian J. Muise, Vaishak Belle, Paolo Felli, Sheila A. McIlraith, Tim Miller, Adrian R. Pearce, and Liz Sonenberg. Planning over multi-agent epistemic states: A classical planning approach. In *Proc. of AAAI*, pages 3327–3334, 2015.

[34] Ranjit Nair, Milind Tambe, Makoto Yokoo, David Pynadath, and Stacy Marsella. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In *IJCAI*, volume 3, pages 705–711, 2003.

[35] X.L Nguyen, S. Kambhampati, and R. Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, 135(1-2):73–123, 2002.

[36] XuanLong Nguyen and Subbarao Kambhampati. Extracting effective and admissible state space heuristics from the planning graph. In *AAAI/IAAI*, pages 798–805, 2000.

[37] J. Pearl. Heuristics: Intelligent search strategies for computer problem solving.

[38] Leonid Peshkin and Virginia Savova. Reinforcement learning for adaptive routing. In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, volume 2, pages 1825–1830. IEEE, 2002.

[39] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[40] Josefina Sierra-Santibáñez. Heuristic planning: A declarative approach based on strategies for action selection. *Artificial Intelligence*, 153(1-2):307–337, 2004.

[41] David Canfield Smith, Allen Cypher, and Jim Spohrer. Kidsim: programming agents without a programming language. *Communications of the ACM*, 37(7):54–67, 1994.

[42] Tran Cao Son, Enrico Pontelli, Chitta Baral, and Gregory Gelfond. Finitary s5-theories. In *European Workshop on Logics in Artificial Intelligence*, pages 239–252. Springer, 2014.

[43] Tran Cao Son, Enrico Pontelli, Chitta Baral, and Gregory Gelfond. Exploring the KD45 property of a kripke model after the execution of an action sequence. In *Proc. of AAAI*, pages 1604–1610, 2015.

[44] Tran Cao Son, Phan Huy Tu, Michael Gelfond, and A Ricardo Morales. Conformant planning for domains with constraints-a new approach. In *AAAI*, volume 5, pages 1211–1216, 2005.

[45] Son Thanh To, Tran Cao Son, and Enrico Pontelli. Contingent planning as AND/OR forward search with disjunctive representation. In *Proc. of ICAPS*, 2011.

[46] Johan Van Benthem, Jan Van Eijck, and Barteld Kooi. Logics of communication and change. *Information and computation*, 204(11):1620–1662, 2006.

[47] Wiebe Van Der Hoek and Michael Wooldridge. Tractable multiagent planning for epistemic goals. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, pages 1167–1174. ACM, 2002.

[48] Hans Van Ditmarsch, Wiebe van Der Hoek, and Barteld Kooi. *Dynamic epistemic logic*, volume 337. Springer Science & Business Media, 2007.

[49] Jan van Eijck. Public announcements and public lies. Technical report, Lying Workshop, 2017.

[50] Hai Wan, Rui Yang, Liangda Fang, Yongmei Liu, and Huada Xu. A complete epistemic planner without the epistemic closed world assumption. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.