Multiagent Planning in the Presence of Multiple Goals

Michael H. Bowling, Rune M. Jensen, and Manuela M. Veloso
Computer Science Department, Carnegie Mellon University,
5000 Forbes Ave, Pittsburgh, PA 15213-3891, USA

1 Introduction

Traditionally, planning involves a single agent for which a planner needs to find a sequence of actions that can transform some initial state into some state where a given goal statement is satisfied. A good example of such a problem is how to solve Rubik's cube. The initial state is some configuration of the cube, and we need to find a sequence of rotations such that every tile on each side have the same color. Even though this problem is hard, the planning agent has full control over the situation. The outcome of a rotation action is completely known.

Real-world planning problems, however, seldom conform to this simple domain model. There may be uncontrollable actions of other agents in the domain interfering with the actions applied by the planning agent. Such uncertainty can be modeled by nondeterminism where actions may have several possible outcomes. One approach is to assume that transition probabilities are known and produce plans with a high likelihood to succeed (e.g., [12, 8]). The scaleability off such planners, however, is limited due to the overhead of reasoning about probabilities. In addition, it may be hard to gather enough statistical data to estimate the transition probabilities. In this chapter, we consider a simpler model of nondeterminism without transition probabilities. The effect of a nondeterminstic action is given as a set of possible next states. Recently, efficient planners have been developed for this class of nondeterministic domains [3]. These planners represents states and perform search implicitly in a space of Boolean functions represented efficiently with reduced Ordered Binary Decision Diagrams (OBDDs) [1]. The plans produced by these planners are encoded compactly with OBDDs and correspond to *universal plans* [18] or policies in Reinforcement Learning [13]. Hence, a *nondeterministic plan* is a state-action table mapping from states to actions relevant to execute in the state in order to reach a set of goal states. A plan is executed by iteratively observing the current state and applying one on the actions associated with that state.

In this chapter, we specifically examine nondeterminstic planning in domains where nondeterminism is caused by uncontrollable agents with specific goals of their own. Such problems have been considered in the AI literature on multiagent systems, in particular, concerning co-operation and negotiation (e.g.,[2, 19, 11, 7, 5, 9]). They have also been studied in game theory and formal verification under various forms (e.g.,[15, 4]).

The novelty of our work is two-fold. First, we introduce *adversarial planning problems* and a class of adversarial plans called *strong cyclic adversarial plans*. An adversarial planning domain has a single system agent that is controllable and a single environment agent that is uncontrollable and may be an opponent to the system agent. The task is to synthesize plans for the system agent that is robust to any plan of the environment agent. We go beyond theoretical work and present a OBDD-based planning algorithm that efficiently can generate strong cyclic adversarial plans. This algorithm has been fully implemented in the BDD-Based Informed Planning and controller Synthesis Tool (BIFROST) 0.7 [10]. Second, we formally define our concept of *multiagent planning equilibria* for multiagent domains inspired by the game theoretic notion of equilibria [16].

In Section 2, we begin by presenting the foundations of multiagent planning equilibria. First, we demonstrate intuitively through simple examples that plan solutions depend on the goals of all the agents. We also present the formal notion of a planning domain that we use throughout the chapter. In Section 3, we introduce adversarial planning. We present an algorithm for synthesizing adversarial plans and demonstrate theoretically and experimentally that the generated plans are robust to any strategy of the environment. In Section 6, we generalize the notion of accounting for an adversary with the solution concept of multiagent planning equilibria. We formalize this notion for planning environments where all the agents' goals are explicitly specified. We also analyze this notion of equilibria in a variety of domains.

2 Foundations for Multiagent Planning

Plans are contingent upon the agent's goals. Plans are usually evaluated as to whether they achieve the goals; sometimes considering how quickly, with what probability, or from what initial states. In addition, goals are often the driving mechanism for finding good plans through heuristics and Means-Ends Analysis [14, 17]. In multiagent domains plans are of course still contingent on goals. There is an additional dependence, though. Good plans also depend on the plans of the

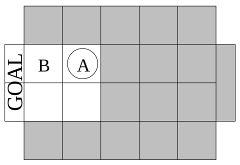

Figure 1: A soccer-like grid domain.

other agents, which as we have stated, depends heavily on their goals.

We assume that agents are synchronized and defined by a set of actions. The world consists of a finite set of states. In each time step, each agent applies exactly one action. The resulting action is called a *joint action*. Thus, a domain is a finite graph where vertices are states and edges are joint actions. A plan of an agent is a state-action table mapping states to actions relevant to execute in order to achieve the goals of the agent. We will illustrate our concept of multiagent planning through a small two-agent soccer domain.

2.1 The Soccer Domain

Consider the simple soccer-like grid domain simplified from Littman's two-player grid game [13], which is diagrammed in Figure 1. There are two agents A and B each of which solely occupies one of the four unshaded squares on the field. Agent A begins in possession of the ball. The shaded squares are out-of-bounds for our simplification of the domain. The agents have operators or actions associated with each of the compass directions (N, S, E, and W) or can wait, holding its position (H). The two agents in this domain select their actions simultaneously, but in execution there is an undetermined delay before these actions are carried out. So, the execution is serial but it is nondeterministic as to which agent's action is carried out first. The effect of an action is to simply move the agent in the specified direction as long as the target square is unoccupied. If the target is occupied then the agent does not move, but if the agent was carrying the ball it loses the ball. For agent A, losing the ball terminates execution as a failure. The goal for agent A is to move into either of the two labeled goal squares, which also terminates execution.

Figure 2 shows two examples of how the agents operators affect the state. From the initial state, if both agents choose their south operator (S,S) they will both simply move south. But if agent A selects south and agent B selects East (S,E), then there are two possible outcomes depending on their order of execution: (i) agent A moves south first, and then agent B moves east into the now unoccupied square; or (ii) agent B bumps into agent A first causing no position change, and then agent A moves south.

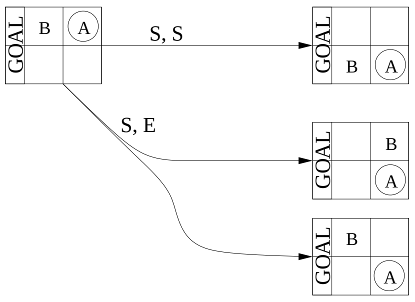

Figure 2: Two example effects of joint actions in the grid-soccer domain. The ordered pairs of actions represent the actions of agents A and B, respectively.

2.2 Possible Plans

There are a number of different possible plans for agent A to achieve its goals in this domain. The various plans depend on what actions we expect agent B to perform. We describe these plans without rigorous definitions or proofs, appealing to the reader's intuition as to what constitutes a "good" plan. The concept of a "good" plan will be formalized later in the chapter.

Nondeterministic. One very simple case is if we believe agent B has no goals in particular and will select its actions nondeterministically, i.e., randomly. A sensible plan would be to hold position until the agent's actions carry it into the bottom right state. From this position, regardless of the action of the other agent, a plan of two consecutive west actions is guaranteed to reach the goal. Since agent B selects actions nondeterministically it will eventually enter the bottom right state and so this plan is guaranteed to reach the goal. Other plans risk agent B's random actions causing it to move in the way, resulting in the loss of the ball and failure. Although this plan guarantees reaching the goal, it does not necessarily guarantee achievement in a finite time, as it requires waiting a possibly infinite number of cycles before agent B moves to the bottom right square.

Teammate. As we assume that most agents have goals, and often these goals are known, we go beyond the assumption that the other agent selects actions randomly. Their actions therefore are not likely to be nondeterministic but rather planned carefully to reach their own goals. Consider the case that agent B is actually agent A's teammate and therefore they have an identical set of goal states. Then there is a much more efficient plan. Agent A should simply hold at the initial state while its teammate moves south out of its way. Then move west into the goal, without fear that the other agent will move in front of it. As a teammate it can be sure that the agent will comply with these assumptions since its goals are identical.[1] This plan is guaranteed to reach the goal in a finite number of steps, as opposed to the plan for the nondeterministic agent.

Adversary. Neither of these plans though have any guarantees if agent B is in fact planning to stop agent A from succeeding. If agent B simply holds its ground, then neither the nondeterministic nor teammate plan for agent A would ever reach its goal. In this situation an adversarial plan, as described in Section 3, that can provide worst-case guarantees is more appropriate. One such plan is to nondeterministically select between holding and moving north or south until the other agent is not in front of it. Then, move west into the goal hoping its action gets executed first. This plan has no guarantee of success as the opponent may still move in front of it while it advances toward the goal causing the ball to be lost. It does, though, have some possibility of success. In fact, against a good plan by the opponent this is all that can be guaranteed.

Overlapping Goals. A whole new situation arises if we believe that agent B is not quite an opponent but not quite a teammate. Suppose its goal is to have agent A score, but only across the south square. In this case moving south from the initial state and then west would reach the goal without its interference. This plan, like the teammate plan, is guaranteed to succeed in a finite number of steps. Notice that the teammate-based plan and the nondeterministic-based plan would both fail in this case as both agents would hold indefinitely.

These four plans are all completely different, despite the fact that the conditions that generated the plans, the domain rules, and the agent's goal did not change from situation to situation. This demonstrates that multiagent planning solutions need to take into account the goals of the other agents.

2.3 A Formalization

We first begin by formalizing the notion of planning domain and agent behavior, which we will use throughout the chapter. The definitions parallel closely with Cimatti and colleagues' single-agent formalization [3].

Definition 1 (Multiagent Planning Domain)
A multiagent planning domain D is a tuple $\langle \mathcal{S}, n, \mathcal{A}_{i=1\ldots n}, \mathcal{R} \rangle$ *where,*

- $\mathcal{S}$ *is the set of states,*
- n *is the number of agents,*

[1]This admittedly does not address crucial issues of how this team compliance can be achieved, and may require planned communication and coordination strategies for distributed execution.

- *A_i is agent i's finite set of actions, and*
- *$\mathcal{R} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is a nondeterministic transition relation where $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_n$ and must satisfy the following condition. If $\langle s, a, s' \rangle \in \mathcal{R}$ and $\langle s, b, s'' \rangle \in \mathcal{R}$ then, $\forall i$ there exists $s''' \in \mathcal{S}$,*

  $$\langle s, \langle a_1, \ldots, a_{i-1}, b_i, a_{i+1}, \ldots, a_n \rangle, s''' \rangle \in \mathcal{R}.$$

  I.e., each agent's set of actions that can be executed from a state are independent.

In addition, let $\textsc{Act}_i(s) \subseteq \mathcal{A}_i$ be the set of applicable or executable actions in state s. Formally,

$$\textsc{Act}_i(s) = \{a_i \in \mathcal{A}_i \mid \exists \langle s, \langle \cdots, a_i, \cdots \rangle, \cdot \rangle \in \mathcal{R}\}.$$

The additional condition in the planning domain definition on $\mathcal{R}$ requires that each agent be capable of selecting actions independently. Formally this amounts to the following. For all states s and executable actions for the agents $a_i \in Act_i(s)$ there exists some transition $\langle s, \langle a_{i=1\ldots n} \rangle, s' \rangle$ that is in $\mathcal{R}$.

A Simple Example — The Narrow Doorway. Consider a two agent robot domain where both agents are in a hallway and want to move into the same room through a single doorway. The agents have an operator to go through the door (G) that only succeeds if the other agent is not also trying to go through the door. They also have the choice of waiting (W).

This domain can be defined using the formalization of a multiagent planning domain above. There are four states in the domain $\mathcal{S} = \{0, 1, 2, 3\}$ corresponding to the four possible configurations of the two agents in the room. n is two and $\mathcal{A}_{A,B}$ is the set of actions $\{G, W\}$. The transition relation $\mathcal{R}$ is defined by the rules described above. The complete enumeration of states and transitions is shown in Figure 3. Note that the domain satisfies the independent action condition on $\mathcal{R}$.

The behavior of agents is assumed to be governed by a *state-action table*. In each execution step, each agent chooses randomly between the actions associated with the current state in its state-action table. Thus, the agents have no memory of previously visited states.

Definition 2 (State-Action Table)
A state-action table π_i for agent i in domain $\mathcal{D}$ is a set of pairs $\{\langle s, a_i \rangle \,|\, s \in \mathcal{S}, a_i \in \textsc{Act}_i(s)\}$. A joint state-action table π constructed from state-action tables for each agent $\pi_{i=1\ldots n}$ is the set of pairs

$$\{\langle s, \langle a_1, \ldots, a_n \rangle \rangle \,|\, s \in \mathcal{S}, \langle s, a_i \rangle \in \pi_i\}$$

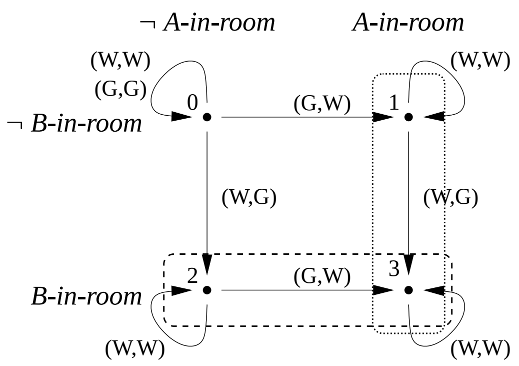

Figure 3: Doorway domain. ⬭ and ⬭ represent A's and B's goal states and will be discussed further in Section 6.

A (joint) state-action table is complete if and only if for any $s \in \mathcal{S}$ there exists some pair $\langle s, \cdot\rangle$ in the state-action table.

For the doorway domain, a state-action table (or plan) for each agent might be,

$$\begin{aligned}\pi_A &= \{\langle 0, G\rangle, \langle 1, W\rangle, \langle 2, G\rangle, \langle 2, W\rangle\, \langle 3, W\rangle\},\\ \pi_B &= \{\langle 0, G\rangle, \langle 0, W\rangle, \langle 1, G\rangle, \langle 2, W\rangle, \langle 3, W\rangle\}.\end{aligned}$$

These are also *complete* state-action tables since they specify at least one action for each state. We can combine these tables into a complete *joint* state-action table. In general, a joint state-action table together with a multiagent planning domain determines the entire execution of the system. In order to define what it means for a plan to be a solution to a planning problem we need to formalize the notion of reachability and paths of execution. We will do this by first defining the execution structure of the multiagent system.

Definition 3 (Induced Execution Structure)
Let π be a joint state-action table of a multiagent planning domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{S}, n, \mathcal{A}_i, \mathcal{R}\rangle$. The execution structure induced by π from the set of initial states $\mathcal{I} \subseteq \mathcal{S}$ is a tuple $K = \langle Q, T\rangle$ with $Q \subseteq S$ and $T \subseteq \mathcal{S} \times \mathcal{S}$ inductively defined as follows:

- *if $s \in \mathcal{I}$, then $s \in Q$, and*
- *if $s \in Q$ and there exists a state-action pair $\langle s, a\rangle \in \pi$ and transition $\langle s, a, s'\rangle \in \mathcal{R}$, then $s' \in Q$ and $\langle s, s'\rangle \in T$.*

A state $s \in Q$ is a terminal state of K if and only if there is no $s' \in Q$ such that $\langle s, s'\rangle \in T$.

Intuitively, Q is the set of states that the system could reach during execution of the plan π, and T is the set of transitions that the system could cross during execution. For our doorway domain the execution structure induced by our example joint state-action table is,

$$\begin{aligned} Q &= \{0, 1, 3\}, \\ T &= \{\langle 0, 1\rangle, \langle 0, 0\rangle, \langle 1, 3\rangle, \langle 1, 1\rangle, \langle 3, 3\rangle\}. \end{aligned}$$

We can now formalize an execution path.

Definition 4 (Execution Path)
Let $K = \langle Q, T\rangle$ be the execution structure induced by a state-action table π from $\mathcal{I}$. An execution path of K from $s_0 \in \mathcal{I}$ is a possibly infinite sequence $s_0, s_1, s_2, \ldots$ of states in Q such that, for all states s_i in the sequence:

- *either s_i is the last state of the sequence, in which case s_i is a terminal state of K, or*
- *$\langle s_i, s_{i+1}\rangle \in T$.*

A state s' is reachable from a state s if and only if there is an execution path with $s_0 = s$ and $s_i = s'$.

For our doorway domain and example joint state-action table one execution path from the initial state is,

$$0, 0, 0, 0, 1, 1, \ldots$$

Let $\textsc{Exec}(s, \pi)$ denote the execution paths starting at s. Let the length of a path $p = s_0 s_1 \cdots$ with respect to a set of states C be defined by

$$|p|_C = \begin{cases} i & : \text{ if } s_i \in C \text{ and } s_j \notin C \text{ for } 0 \leq j < i \\ \infty & : \text{ otherwise.} \end{cases} \tag{1}$$

We will say that an execution path p *reaches* a state s iff $|q|_{\{s\}} \neq \infty$. The definition of execution paths serves to formalize solution concepts in the remainder of the chapter.

In the next section we present a definition of the adversarial planning problem. We also present and analyze an algorithm for finding plans with strong guarantees in adversarial domains. In Section 6 we generalize adversarial planning problems to a more general multiagent planning problem where all agents' goals are made explicit. We then introduce the general solution concept of multiagent planning equilibria and analyze it in a variety of domains.

3 Adversarial Planning

An adversarial planning problem is a multiagent planning problem with two agents called *system* and *environment*. The system agent is controllable and its goal is to reach a set of goal states. The environment agent is uncontrollable. It might be an adversary to the system agent trying to prevent it from reaching its goals.

Definition 5 (Adversarial Planning Problem)
Let $\mathcal{D} = \langle \mathcal{S}, 2, \mathcal{A}_s, \mathcal{A}_e, \mathcal{R} \rangle$ be a multiagent planning domain with a system and environment agent and a deterministic transition relation. An adversarial planning problem P for $\mathcal{D}$ is a tuple $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{I} \subseteq \mathcal{S}$ is the set of possible initial states and $\mathcal{G} \subseteq \mathcal{S}$ is the set of goal states for the system agent.

An *adversarial plan* for an adversarial planning problem $\mathcal{P}$ is a state-action table for the system agent such that all execution paths starting in an initial state eventually will reach a goal state. Thus, an adversarial plan is total and can not reach dead ends or loop indefinitely.

As an example, consider the domain shown in Figure 4. The actions of the system and environment agents are $\mathcal{A}_s = \{+s, -s\}$ and $\mathcal{A}_e = \{+e, -e\}$, respectively. Transitions are labeled with the corresponding joint action. There are 5 states, namely I, F, D, U and G. I and G are initial and goal states. D is a dead end, since the goal is unreachable from D. This introduces an important difference between F and U that captures a main aspect of the adversarial planning problem. We can view the two states F and U as states in which the system and environment have different opportunities. Observe that the system "wins", i.e., reaches the goal, only if the sign of the two actions in the joint action are different. Otherwise it "loses" since there is no transition to the goal with a joint action where the actions have the same sign. The goal is reachable from both state F and U. However, the result of a "losing" joint action is different for F and U. In F, the system agent remains in F. Thus, the goal is still reachable. In U, however, the agent may transition to the dead end D which makes it impossible to reach the goal in subsequent steps.

Now consider how an adversarial environment can take advantage of the possibility for the system to reach a dead end from U. Since the system may end in D, when executing $-s$ in U, it is reasonable for the environment to assume that the system will always execute $+s$ in U. But now the environment can prevent the system from ever reaching the goal by always choosing action $+e$, so the system should completely avoid the path through U.

This example domain is important because it illustrates how an adversarial environment can act purposely to obstruct the goal achievement of the system. We

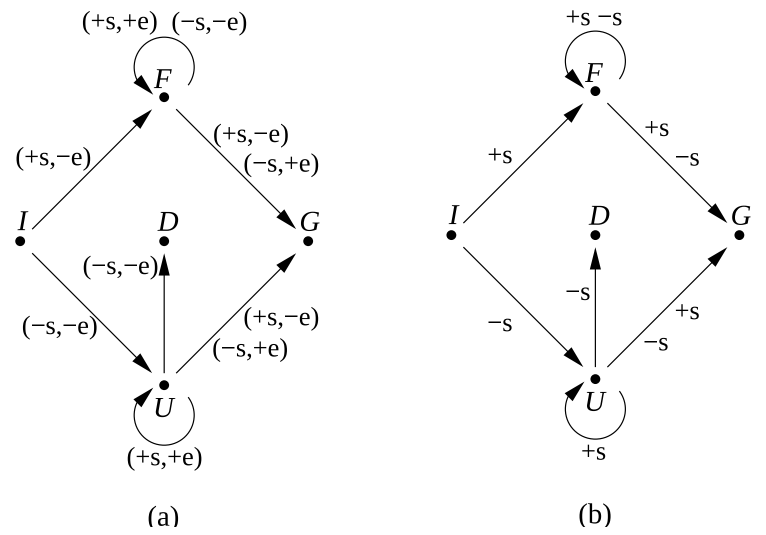

Figure 4: An adversarial planning domain example. (a) shows the explicit representation of environment, while (b) shows the implicit representation used by previous algorithms.

will use it in the following sections to explain our algorithm. A solution, guaranteeing that G is eventually reached, is $\{\langle I, +s\rangle, \langle F, +s\rangle, \langle F, -s\rangle\}$.

3.1 The Algorithm

We introduce a generic function PLAN($\mathcal{I}, \mathcal{G}$) for representing OBDD-based nondeterministic planning algorithms that produce state-action tables as solutions. The algorithms, including our, only differ by definition of the function computing the precomponent (PRECOMP(C)).

The generic function performs a backward breadth-first search from the goal states to the initial states. In each step, the precomponent SA_p of the set of states C covered by the plan is computed. The precomponent is a state-action table forming a partition of the final plan with relevant actions for reaching C.

```
function PLAN($\mathcal{I}, \mathcal{G}$)
    $SA \leftarrow \emptyset$; $C \leftarrow \mathcal{G}$
    while $\mathcal{I} \not\subseteq C$
        $SA_p \leftarrow$ PRECOMP($C$)
        if $SA_p = \emptyset$ then return failure
        else $SA \leftarrow SA \cup SA_p$
        $C \leftarrow C \cup \textsc{States}(SA_p)$
    **return** $SA$
```

If the precomponent is empty a fixed point of C has been reached that does not cover the initial states. Since this means that no plan can be generated that covers the initial states, the algorithm returns *failure*. Otherwise, the precomponent is added to the plan and the states in the precomponent are added to the set of covered states.

All sets and mappings in the algorithm are represented by OBDDs. An OBDD is a compact representation of Boolean functions. Thus, states and actions are represented by bit-vectors and sets of states and actions are encoded by OBDDs representing their *characteristic function*. The set operations intersection, union, and compliment translates into conjunction, disjunction, and negation on the corresponding characteristic functions. In the reminder we will not distinguish between set operations and Boolean operations.

This algorithm was introduced in the MBP [3] planning system for synthesizing *weak*, *strong cyclic*, and *strong plans*. An execution path of a strong plan is guaranteed to reach states covered by the plan until a goal state after a finite number of steps is reached. An execution of a strong cyclic plan is also guaranteed to reach states covered by the plan or a goal state. However, due to cycles, it may never reach a goal state. An execution of a weak plan may reach states not covered by the plan, it only guarantees that some execution exists that reaches the goal from each state covered by the plan. A limitation of these algorithms when nondeterminism is caused by uncontrollable actions of an adversarial environment is that they are naive in the sense that they assume the environment to be friendly. For instance, a valid strong cyclic plan for the example problem is

$$\{\langle I, +s\rangle, \langle I, -s\rangle, \langle U, +s\rangle, \langle F, +s\rangle, \langle F, -s\rangle\}.$$

This plan will only reach states covered by the plan. However, given an adversarial opponent, it may enter a live-lock in state F, since the environment may choose only to execute action $-e$. Thus, strong cyclic planning is insufficient in adversarial domains.

3.1.1 The Strong Cyclic Adversarial Precomponent

A valid adversarial plan ensures that the environment agent, even with complete knowledge of the domain and the plan, is unable to prevent the goal states to be reached. We formalize this idea in the definition of a *fair state*. A state s is fair with respect to a set of states C and a plan SA if, s is not already a member of C and for each applicable environment action, there exists a system action in SA such that the joint action leads into C.

Definition 6 (Fair State) *A state $s \notin C$ is fair with respect to a set of states C and a plan SA iff $\forall a_e \in \textsc{Act}_e(s) \,.\, \exists \langle s, a_s \rangle \in SA, s' \in C \,.\, \langle s, \langle a_s, a_e \rangle, s' \rangle \in \mathcal{R}$.*

For convenience, we define an *unfair* state to be a state that is not fair. The adversarial precomponent is a strong cyclic precomponent [3] pruned for unfair states. In order to use a precomponent for OBDD-based universal planning, we need to define it as a Boolean function. We first define a Boolean function representing the transition relation $T(s, a_s, a_e, s') = \langle s, \langle a_s, a_e \rangle, s' \rangle \in \mathcal{R}$. A core computation is to find all the state-action pairs where the action applied in the state can lead into a set of states C. This set of state-action pairs is called the *preimage* of C. The preimage of joint actions is

$$\textsc{JPreImg}(C)(s, a_s, a_e) = \exists s' \,.\, T(s, a_s, a_e, s') \wedge C(s').$$

By abstracting environment actions, we get the preimage of system actions

$$\textsc{PreImg}(C)(s, a_s) = \exists a_e \,.\, \textsc{JPreImg}(C)(s, a_s, a_e).$$

We can now define a Boolean function representing the state-action pairs of a plan SA for which the state is fair with respect to a set of states C

$$\begin{array}{rcl} \textsc{Fair}(SA, C)(s, a_s) & = & SA(s, a_s) \wedge \neg C(s) \wedge \\ \forall a_e \,.\, \textsc{Act}_e(s, a_e) & \Rightarrow & \exists a_s \,.\, SA(s, a_s) \wedge \textsc{JPreImg}(C)(s, a_s, a_e) \end{array}$$

where $\textsc{Act}_e(s, a_e) = \exists a_s, s' \,.\, T(s, a_s, a_e, s')$. The strong cyclic adversarial precomponent (SCAP) is computed by iteratively extending a set of candidate state-action pairs and in turn prune

- State-actions that can reach states not covered by the current plan or the states in the candidate;
- State-actions of states that are unfair with respect to the currently covered states.

The computation of the precomponent terminates either if the pruned candidate has reached a non-empty fixed point or if it is impossible to extend the candidate further. In the latter case, the returned precomponent is empty.

Definition 7 (SCAP) *The strong cyclic adversarial precomponent of a set of states C is the set of state-action pairs computed by function* $\textsc{SCAP}(C)$.

```
function SCAP($C$)
1  $wSA \leftarrow \emptyset$
2  repeat
3      $OldwSA \leftarrow wSA$
4      $wSA \leftarrow \textsc{PreImg}(C \cup \textsc{States}(wSA))$
5      $wSA \leftarrow \textsc{Prune}(wSA, C)$
6      $SCA \leftarrow SCAPlanAux(wSA, C)$
7  until $SCA \neq \emptyset \vee wSA = OldwSA$
8  return $SCA$

function SCAPlanAux($startSA, C$)
1  $SA \leftarrow startSA$
2  repeat
3      $OldSA \leftarrow SA$
4      $SA \leftarrow \textsc{PruneOutgoing}(SA, C)$
5      $SA \leftarrow \textsc{PruneUnfair}(SA, C)$
6  until $SA = OldSA$
7  return $SA$

function PruneOutgoing($SA, C$)
1  $NewSA \leftarrow SA \setminus \textsc{PreImg}(\overline{C \cup \textsc{States}(SA)})$
2  return $NewSA$

function PruneUnfair($SA, C$)
1  $NewSA \leftarrow \emptyset$
2  repeat
3      $OldSA \leftarrow NewSA$
4      $FairStates \leftarrow C \cup \textsc{States}(NewSA)$
5      $NewSA \leftarrow NewSA \cup \textsc{Fair}(SA, FairStates)$
6  until $NewSA = OldSA$
7  return $NewSA$
```

$$\begin{aligned}\textsc{Prune}(SA, C)(s, a_s) &= SA(s, a_s) \wedge \neg C(s) \\ \textsc{States}(SA)(s) &= \exists a_s \,.\, SA(s, a_s).\end{aligned}$$

For an illustration, consider the first candidate of SCAP(G) shown in Figure 5(a). Action $-s$ would have to be pruned from U since it has an outgoing

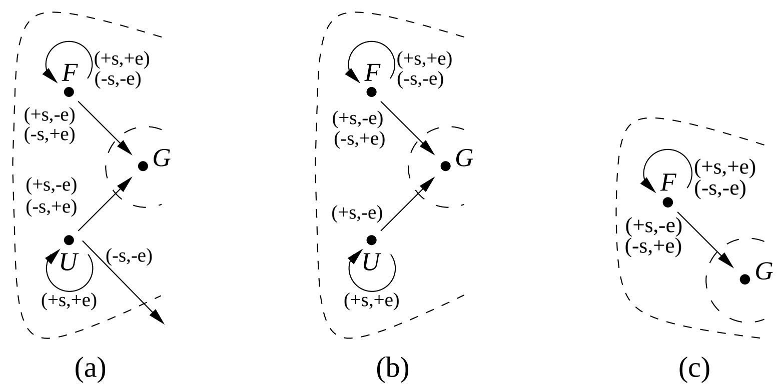

Figure 5: (a) The first candidate of SCAP(G), for the example shown in Figure 4; (b) The candidate pruned for actions with outgoing transitions; (c) The remaining candidate pruned for unfair states. Since no further state-action pairs are pruned, this is the strong cyclic adversarial precomponent returned by SCAP(G).

transition. The pruned candidate is shown in Figure 5(b). Now there is no action leading to G in U when the environment chooses $+e$. U has become unfair and must be pruned from the candidate. The resulting candidate is shown in Figure 5(c). Since the remaining candidate is non-empty and no further state-action pairs need to be pruned, a non-empty strong cyclic precomponent has been found.

The two major design goals of the strong cyclic adversarial planning algorithm is that it is correct and efficient. With respect to correctness, the strong cyclic adversarial algorithm has been chosen to closely match the strong cyclic algorithm such that a similar correctness proof can be applied. With respect to efficiency, three major choices have been made. First of all, we use an OBDD-based implementation. Second, as for the strong cyclic algorithm, we build up a strong cyclic adversarial plan incrementally from the goal states. Alternatively, the algorithm could iteratively prune a largest possible plan as the algorithm suggested in [4]. However, this approach seems less efficient and has to our knowledge never been implemented and experimentally evaluated.

4 Action Selection Strategies

A strong cyclic adversarial plan guarantees that no intelligent environment can choose a plan that forces executions to cycle forever without ever reaching a goal state. In principle, though, infinite paths never reaching a goal state can still be produced by a system that "keeps losing" to the environment. However, by assuming the system selects randomly between actions in its plan, we can show that the probability of producing such paths is zero.

Theorem 1 (Termination of Strong Cyclic Adversarial) *By choosing actions randomly from a strong cyclic adversarial plan π for the adversarial planning problem $\mathcal{P} = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$, any execution path will eventually reach a goal state.*

Proof. Since all unfair states and actions with transitions leading out of the states covered by π have been removed, all the visited states of an execution path will be fair and covered by the plan. Assume without loss of generality that n strong cyclic adversarial precomponents were computed in order to generate π. Due to the definition of precomponent functions, we can then partition the set of states covered by π into $n + 1$ ordered subsets $C_n, \cdots, C_0$ where $\mathcal{I} \subseteq C_n$, $C_0 = \mathcal{G}$, and C_i for $0 < i \leq n$ contains the states covered by precomponent i. Consider an arbitrary subset C_i. Assume that there were m iterations of the repeat loop in the last call to PRUNEUNFAIR when computing precomponent i. We can then subpartition C_i into m ordered subsets $C_{i,m}, \cdots, C_{i,1}$ where $C_{i,j}$ contains the states of the state-action pairs added to $NewSA$ in iteration j of PRUNEUNFAIR. Due to the definition of FAIR, we have that the states in $C_{i,j}$ are fair with respect to π and the states C given by

$$C = \bigcup_{k=1}^{j-1} C_{i,j} \cup \bigcup_{k=0}^{i-1} C_i.$$

By flattening the hierarchical ordering of the partitions $C_n, \cdots, C_0$ and their subpartitions, we can assume without loss of generality that we get the ordered partitioning $L_T, \cdots, L_0$ where $L_0 = C_0$. Given that actions are selected uniformly in π, the fairness between the states in the levels guarantees that there is a non-zero probability to transition to a state in $L_{i-1}, \cdots, L_0$ from any state in L_i. Consequently, an execution path only reaching states covered by π will eventually reach a state in L_0. $\square$

5 Experimental Evaluation

The performance of the strong cyclic adversarial planning algorithms has been evaluated in two domains. The first of these is a parameterized version of the example domain shown in Figure 4. The second is a grid world with a hunter and prey. All experiments are carried out using the BIFROST 0.7 search engine on a Pentium III Redhat Linux 7.1 PC with 500MHz CPU and 512 MB RAM. Total CPU time is measured in seconds and includes time spent on allocating memory for the OBDD package and parsing the problem description.

5.1 Parameterized Example Domain

The parameterized example domain considers a system and environment actions $\{+s, -s, l\}$ and $\{+e, -e\}$, respectively. The domain is shown in Figure 6. The initial state is $\mathcal{I} = \{I\}$ and the goal states are $\mathcal{G} = \{g_1, g_2\}$. Progress toward the goal states is made if the sign of the two actions in the joint action are different. At any time, the system can cause a switch from the lower to the upper row of states by executing l. In the upper row, the system can only execute $+s$. Thus, in these states an adversarial environment can prevent further progress by always executing $+e$. Figure 7 shows the total CPU time and the size of the produced plans of the ordinary strong cyclic algorithm compared to the strong cyclic adversarial algorithm.

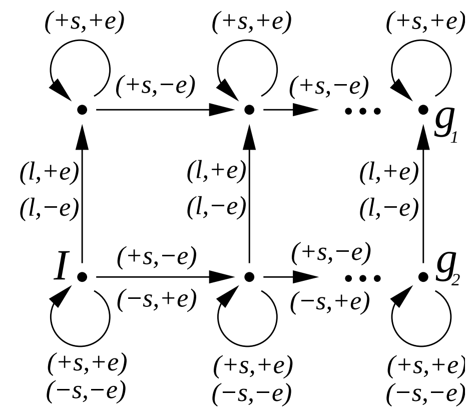

Figure 6: The generalized example domain shown in Figure 4(a).

Due to the structure of the domain, the length l of a shortest path between the initial state and one of the goal states grows linearly with the number of states. Since each of the four algorithms at least must compute l preimages, their complexity is at least exponential in the number of Boolean state variables. The experimental results seem to confirm this. In this domain, there only is a small overhead of generating adversarial plans compared to non-adversarial plans. The quality of the produced plans, however, is very different. For instance, the strong cyclic adversarial plans only consider executing $-s$ and $+s$, while the strong cyclic plans consider all applicable actions. The strong cyclic adversarial plan is guaranteed to achieve the goal. In contrast, the probability of achieving the goal in the worst case for the strong cyclic plan is less than $(\frac{2}{3})^{N/2-1}$, where N is the number of states in the domain. Thus, for an adversarial environment the probability of reaching the goal with a strong cyclic plan is practically zero, even for small instances of the problem.

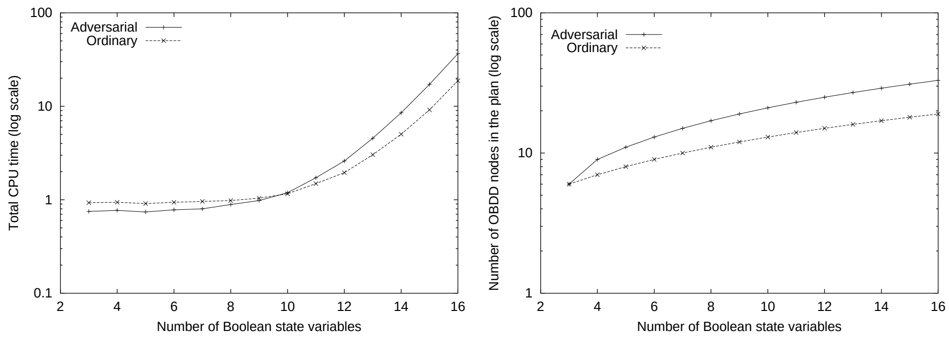

Figure 7: Results of the parameterized example domain.

5.2 Hunter and Prey Domain

The hunter and prey domain consists of a hunter and prey agent moving on a chess board. Initially, the hunter is at the lower left position of the board and the prey is at the upper right. The initial state of the game is shown in Figure 8. The task of

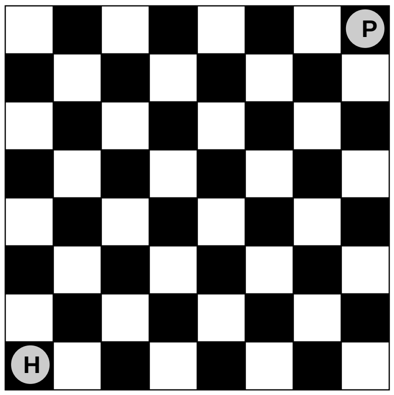

Figure 8: The hunter and prey domain.

the hunter is to catch the prey. This happens if the hunter and prey at some point are at the same position. The hunter and prey move simultaneously. They are not aware of each others moves before both moves are carried out. In each step, they can either stay at the spot or move like a king in chess. However, if the prey is at the lower left corner position, it may change the moves of the hunter to that of a bishop (making single step moves). This has a dramatic impact on the game, since the hunter then only can move on positions with the same color. Thus, to avoid the hunter, the prey just have to stay at positions with opposite color. A strong cyclic adversarial plan therefore only exits, if it is possible for the hunter to find a plan that guarantees that the prey never gets to the lower left corner. A strong cyclic plan, on the other hand, does not differentiate between whether the hunter moves like a king or a bishop. In both cases, a "friendly" prey can be catched.

We consider a parameterized version of the domain with the size of the chess board ranging from 8×8 to 512×512. Figure 9 shows the total CPU time and the size of the plans produced by the ordinary strong cyclic algorithm compared to the strong cyclic adversarial algorithm. In this domain strong cyclic adversarial plans

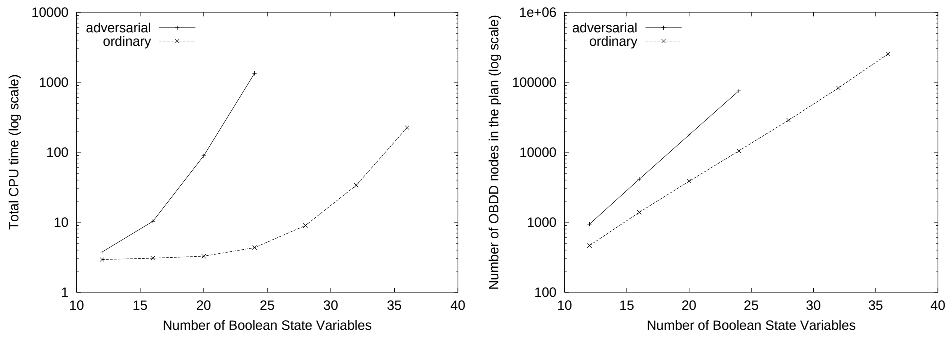

Figure 9: Results of the Hunter and Prey domain.

are larger and take substantially longer time to generate than ordinary strong cyclic plans. The strong cyclic adversarial algorithm spends more than 4000 seconds for problems with 28 Boolean state variables or more. However, as discussed above, it is non-trivial, if there exists a strategy of the hunter that guarantees that the prey never succeeds in reaching the lower left corner. Thus, we may expect these plans to be computationally harder than strong cyclic plans. This interpretation is supported by the size of the plans. The adversarial plans are substantially larger than the ordinary plans.

6 Equilibria in Multiagent Planning

In this section, we introduce a more general framework for multiagent planning. We explicitly specify all of the agents goals, and introduce the solution concept of multiagent planning equilibria that accounts for all of the agents goals. The definitions and concepts presented in this section are not bound to any particular planning algorithm or language. The reader may find it easier to look at some of the examples given in Section 6.2 before moving to the formalization given below.

6.1 The Formalization

We start by formalizing the notion of a multiagent planning problem that explicitly enumerates all of the agents' goals.

Definition 8 (Multiagent Planning Problem)
Let $\mathcal{D} = \langle \mathcal{S}, n, \mathcal{A}_{i=1\ldots n}, \mathcal{R} \rangle$ be a multiagent planning domain. A multiagent planning problem P for $\mathcal{D}$ is a tuple $\langle \mathcal{D}, \mathcal{I}, \mathcal{G}_{i=1\ldots n} \rangle$, where $\mathcal{I} \subseteq \mathcal{S}$ is the set of possible initial states and $\mathcal{G}_i \subseteq \mathcal{S}$ is the set of goal states for agent i.

Recall the doorway example shown in Figure 3. The goal states for agent A are $\mathcal{G}_A = \{1, 3\}$ and for agent B are $\mathcal{G}_B = \{2, 3\}$. The initial state set is the singular set $\{0\}$.

We can now formalize our notion of a plan as a state-action table. We actually define multiple concepts increasing in strength. These concepts formalize some of the intuitive discussion from the previous section about whether a plan has one or more of the following properties:

- the possibility of reaching the goal,
- a guarantee of reaching the goal, and
- a guarantee of reaching the goal in a finite number of steps.

These concepts and their formalization are inspired and highly related to Cimatti and colleagues' single-agent solution concepts [3]. They are also strongly related to the properties of the adversarial planning algorithm described in Section 3.1.

Definition 9 (Multiagent Planning Solutions)
Let $\mathcal{D}$ be a multiagent planning domain and $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G}_{i=1\ldots n} \rangle$ be a multiagent planning problem. Let π be a complete *joint state-action table for $\mathcal{D}$. Let $K = \langle Q, T \rangle$ be the execution structure induced by π from $\mathcal{I}$. The following is an ordered list of solution concepts increasing in strength.*

1. *π is a weak solution for agent i if and only if for any state in $\mathcal{I}$ some state in $\mathcal{G}_i$ is reachable.*

2. *π is a strong cyclic solution for agent i if and only if from any state in Q some state in $\mathcal{G}_i$ is reachable.*

3. *π is a strong solution for agent i if and only if* all *execution paths, including infinite length paths, from a state in Q contain a state in $\mathcal{G}_i$.*

4. *π is a perfect solution for agent i if and only if for* all *execution paths $s_0, s_1, s_2 \ldots$ from a state in Q there exists some $n \geq 0$ such that $\forall i \geq n$, $s_i \in \mathcal{G}_i$.*

A state-action table's strength STRENGTH$(\mathcal{D}, \mathcal{P}, i, \pi)$ *is the largest number whose condition above applies for agent i. If no conditions apply then* STRENGTH$(\mathcal{D}, \mathcal{P}, i, \pi) = 0$.

For our doorway domain, the joint state-action table is a strong cyclic solution for both agents but not strong (i.e., it has a strength of 2 for both agents). This means that there is a path to the goal from any reachable state. But there are also paths that do not include either agents' goal states, and so it is not a strong solution for either agent.

The plans from the soccer domain can also be described under this solution framework. The plan that handles the nondeterministic agent B is a strong cyclic solution since a goal state is always reachable but there are infinite execution paths where agent A does not reach the goal (e.g., if agent B holds indefinitely). For the teammate case, the plan is a perfect solution since it is guaranteed to reach the goal in three steps and remain there. The same is true for the situation where agent B's goal is to have the ball scored in the southern square. In the adversarial case, the plan is only weak since some execution paths result in losing the ball and failing.

Notice that the adversarial planning algorithms also give similar guarantees. For example, a strong cyclic adversarial plan along with *any* plan by the other agent has a STRENGTH of 2. Likewise, an optimistic adversarial plan along with *any* plan by the other agent has a STRENGTH of 1.

These solutions define what it means for one agent to be successful given a joint state-action table. The goal of planning from one agent's perspective is to find a plan that has the highest strength given the plans of the other agents. But the other agents' selection of a plan is equally contingent upon the first agent's plan. This recursive dependency leads to the main contribution of this section: multiagent planning equilibria.

Definition 10 (Multiagent Planning Equilibria)
Let $\mathcal{D}$ be a multiagent planning domain and $\mathcal{P} = \langle \mathcal{D}, \mathcal{I}, \mathcal{G}_{i=1\ldots n} \rangle$ be a multiagent planning problem. Let π be a complete *joint state-action table for $\mathcal{D}$. Let $K = \langle Q, T \rangle$ be the execution structure induced by π from $\mathcal{I}$. π is an equilibrium solution to $\mathcal{P}$ if and only if for all agents i and for any complete joint state-action table π'*

such that $\pi'_{j \neq i} = \pi_j$,

$$\text{STRENGTH}(\mathcal{D}, \mathcal{P}, i, \pi) \geq \text{STRENGTH}(\mathcal{D}, \mathcal{P}, i, \pi').$$

I.e., each agent's state-action table attains the strongest solution concept possible given the state-action tables of the other agents.

Note that our example joint state-action table for the doorway domain is *not* an equilibrium. Both agents A and B currently have strength 2, but B can achieve a strength of 4 by choosing a different state-action table. Specifically, B should select the wait (W) action from the initial state and the go (G) action in state 1.

6.2 Examples

To make the concept of planning equilibria clearer, we will examine it in a number of illustrative domains. We first examine the doorway domain along with a couple of variants. We then consider a domain representation of the children's game Rock-Paper-Scissors, and finally we reexamine the various plans in the soccer domain.

6.2.1 Doorway Domain

We gave above an example joint state-action table that is not a multiagent planning equilibria for this domain. An equilibria is the following state-action tables:

$$\begin{aligned} \pi_A &= \{\langle 0, G\rangle, \langle 1, W\rangle, \langle 2, G\rangle, \langle 3, W\rangle\}, \\ \pi_B &= \{\langle 0, W\rangle, \langle 1, G\rangle, \langle 2, W\rangle, \langle 3, W\rangle\}. \end{aligned}$$

In this case agent A goes through the door while agent B waits and then follows through the door. This is a perfect plan for both agents and so obviously no agent can achieve a higher strength with a different state-action table. Similarly, the symmetric tables where agent B goes through the door while agent A waits is also an equilibrium. There is an additional equilibrium,

$$\begin{aligned} \pi_A &= \{\langle 0, G\rangle, \langle 0, W\rangle, \langle 1, W\rangle, \langle 2, G\rangle, \langle 3, W\rangle\}, \\ \pi_B &= \{\langle 0, G\rangle, \langle 0, W\rangle, \langle 1, G\rangle, \langle 2, W\rangle, \langle 3, W\rangle\}. \end{aligned}$$

Here both agents nondeterministically decide between going through the door and waiting. This results in a strong cyclic solution for both agents, but given this state-action table for the other agent no strong or perfect plan exists for either agent. So this is also an equilibrium although obviously inferior to the other equilibria where both agents have higher strength plans. In game theory, such a joint strategy is called Pareto dominated.

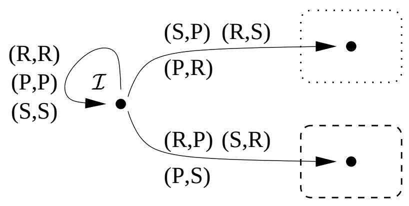

Figure 10: Rock-Paper-Scissors as multiagent planning.

Collision variation. Consider a variation on this domain where collisions (when both agents choose G) result in the robots becoming damaged and unable to move. In this case, the first two state-action tables above remain equilibria, but the third inferior table no longer is an equilibrium. This joint plan is now only a weak solution for both agents since there is a possibility of never achieving the goal. Each agent can also change to a different plan where it waits for the other agent to get through the door thus achieving a strong cyclic plan and a higher strength.

Door closing variation. Finally, consider that one agent entering the room sometimes causes the door to close behind it. Once the door is closed it cannot be opened and the doorway cannot be used. In this case, the same two joint plans are again an equilibrium but now they have different strengths for the different agents. The first joint state-action table is a strong plan for agent A, but only a weak plan for agent B, though it can do no better. The second is just a symmetry of this.

6.2.2 Rock-Paper-Scissors

Consider a planning domain representation of the children's game Rock-Paper-Scissors. Each agent simultaneously chooses one of rock (R), paper (P), or scissors (S). The winner is determined by a cyclic rule: rock loses to paper, paper loses to scissors, scissors loses to rock. Figure 10 gives the enumeration of states, transitions, and goals for this planning problem. In this case, there is a unique planning equilibrium where each agent's state-action table contains every action. This joint plan is a weak solution (strength 1) for both agents and neither agent can switch to a different plan and get a higher strength. This plan is analogous to the game's game theoretic equilibrium which randomizes evenly between all three actions [6].

6.2.3 Soccer Domain

Let us reconsider the soccer-like domain. We presented three distinct planning problems where agent A's goals remained constant, but agent B's goals varied from having identical goals to A, opposing goals to A, and a subset of A's goals. The example plans described for these situations, if we add in the implied plan for agent

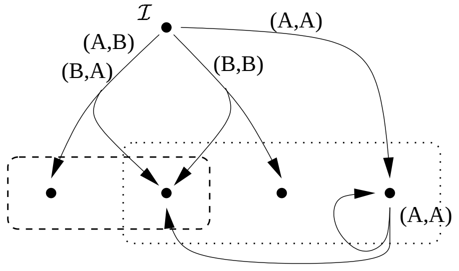

Figure 11: Domain without an equilibrium.

B, are all equilibria to their respective multiagent planning problems. In the teammate case and the overlapping goal case, the equilibrium is a perfect solution for both agents. So, obviously, no agent can switch plans to improve on this solution. In the adversarial case, it is a weak solution for both agents, and neither agent can improve on this strength. This formalization of the planning equilibrium matches well with our intuitive notions of "good" plans in multiagent domains.

6.3 Discussion

Multiagent planning equilibria is a powerful concept both to understand the multiagent planning problem and as a viable solution that accounts for the other agents' goals. It also opens up many avenues for further research and understanding. We consider a couple important questions this work raises.

The first issue is the number of planning equilibria. The doorway domain shows that multiple equilibria may exist. Although some equilibria are obviously inferior to others, the equilibria framework need not define a single solution plan. For example, the two symmetric equilibria in the doorway domain are not equivalent, nor is one Pareto dominant. This calls for coordination or communication mechanisms to decide between competing equilibria. In addition, some problems may have no equilibria. Figure 11 gives an example planning problem with no equilibrium. Each agent has three possible complete state-action tables, and a simple examination of the nine possible pairs will demonstrate that none are equilibria. Still, large classes of domains can be proven to have equilibria (e.g., team domains and adversarial domains.) Other interesting questions are whether equilibria exist in most useful domains, or what are reasonable plans when they do not exist.

Second, this work presents a compelling framework and solution concept for multiagent planning, and gives the challenge of devising methods to find planning equilibria. The equilibrium definition involves universal quantification over an agent's possible plans, which is exponential in the number of states, which in turn is exponential in the number of state variables. This is intractable for anything but domains with a handful of state variables. This opens up a new realm of interesting issues relating to efficiently finding equilibria under different planning frameworks, languages, or classes of domains. Planning for agents with identical goals is essentially a single-agent planning problem. In the case of adversarial settings, the algorithm in Section 3 can find half of an equilibrium plan. These special case algorithms are evidence that multiagent planning equilibria can be both a theoretically and practically powerful concept.

One possible general technique for finding equilibria comes from game theory's alternating-move Cournot process [6]. The basic idea would be to start each agent with a complete state-action table and then each agent alternates finding a new complete state-action table that achieves the highest strength given the other agents' tables. This process is stopped if none of the agent's can improve on their current table given the others' and so the joint state-action table is an equilibrium. This technique may hold some promise. If the initial state-action tables include all available actions to the players, this process could actually find equilibria in every example presented in this chapter. Details of the technique, such as how to select among equally strong state-action tables, would be critical to an actual implementation and analysis. This does, though, give insight into how planning equilibrium may be found in practice.

7 Conclusion

In this chapter, we explored the importance of accounting for all of the agents' goals when planning in a multiagent environment. We both examined this fact in planning for an adversary, as well as defining a more general solution concept that explicitly depends on all of the agents' goals.

We contributed a new OBDD-based universal planning algorithm, strong cyclic adversarial planning. This algorithm naturally extend the previous strong cyclic algorithm to adversarial environments. We have proven and shown empirically that, in contrast to strong cyclic plans, a strong cyclic adversarial plan always eventually reach the goal.

We also presented a formalization of multiagent planning where all agents have individually specified goals and introduced the concept of a multiagent planning equilibrium. This is the first known solution concept that explicitly accounts for the goals of all the agents. This provides a unifying framework for considering planning in multiagent domains with identical, competing, or overlapping goals. It also opens up many exciting questions related to practical algorithms for finding equilibria, the existence of equilibria, and the coordination of equilibria selection.

Acknowledgments

This research is sponsored in part by the Danish Research Agency and the United States Air Force under Grants Nos F30602-00-2-0549 and F30602-98-2-0135. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force, or the US Government.

References

[1] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 8:677–691, 1986.

[2] J. G. Carbonell. Counterplanning: A strategy-based model of adversary planning in real-world situations. *Artificial Intelligence*, 16(3):257–294, 1981.

[3] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence*, 147(1-2), 2003.

[4] L. De Alfaro, T. A. Henzinger, and O. Kupferman. Concurrent reachability games. In *IEEE Symposium on Foundations of Computer Science*, pages 564–575, 1998.

[5] E. H. Durfee. *Coordination of Distributed Problem Solvers*. Kluwer Academic Press, 1988.

[6] Drew Fudenberg and David K. Levine. *The Theory of Learning in Games*. The MIT Press, 1999.

[7] M. P. Georgeff. Communication and interaction in multiagent planning. In *Proceedings of the 3rd National Conference on Artificial Intelligence (AAAI'83)*, pages 125–129, 1983.

[8] P. Haddawy and M. Suwandi. Decision-theoretic refinement planning using inheritance abstraction. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-92)*, 1994.

[9] N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.

[10] R. M. Jensen. The BDD-based InFoRmed planning and cOntroller Synthesis Tool (BIFROST) version 0.7. `http://www.cs.cmu.edu/~runej`, 2003.

[11] T. Kreifelts and F. Martial. A negotiation framework for autonomous agents. In *Proceedings of the 2nd European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds*, pages 169–182, 1990.

[12] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995.

[13] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufman, 1994.

[14] Allen Newell and Herbert A. Simon. GPS, a program that simulates human thought. In Edward A. Feigenbaum and Julian Feldman, editors, *Computers and Thought*, pages 279–296. McGraw-Hill, New York, 1963.

[15] M. J. Osborne and A. Rubinstein. *A course in game theory*. MIT Press, 1994.

[16] Guillermo Owen. *Game Theory*. Academic Press, 1995.

[17] J. Pearl. *Heuristics : Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[18] M. Schoppers. Universal planning for reactive robots in unpredictable environments. In *Proceedings of IJCAI-87*, pages 1039–1046, 1987.

[19] G. Zlotkin and J. S. Rosenschein. Incomplete information and deception in multi-agent negotiation. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'85)*, pages 225–231, 1985.