OBDD-based Universal Planning for Multiple Synchronized Agents in Non-Deterministic Domains

Rune M. Jensen and **Manuela M. Veloso**
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3891
{runej,mmv}@cs.cmu.edu

Abstract

Model checking representation and search techniques were recently shown to be efficiently applicable to planning, in particular to non-deterministic planning. Ordered Binary Decision Diagrams (OBDDs) encode a planning domain as a non-deterministic finite automaton (NFA) and fast algorithms from model checking search for a solution plan. With proper encodings, OBDDs can effectively scale and can provide universal plans for complex planning domains. We are particularly interested in addressing the complexities arising in non-deterministic, multi-agent domains. In this paper, we present UMOP,[1] a new universal OBDD-based planning framework applicable to non-deterministic and multi-agent domains. We introduce a new planning domain description language, *NADL*,[2] to include the specification of such non-deterministic, multi-agent domains. The language contributes the explicit definition of controllable agents and uncontrollable environment agents. We describe the syntax and semantics of *NADL* and show how to build an efficient OBDD-based representation of an *NADL* description. The UMOP planning systems uses *NADL* and it includes the previously developed strong and strong cyclic planning algorithms (Cimatti et al., 1998a, 1998b). In addition, we introduce a new optimistic planning algorithm, which relaxes optimality guarantees and generates plausible universal plans in some domains where no strong or strong cyclic solution exists. We present empirical results in a previously tested non-deterministic domains. We also introduce three new multi-agent domains with complex environment actions. UMOP is shown to be a rich and efficient planning system.

Introduction

Traditional planning algorithms can be classified according to their search space representation as either state-space, plan-space, or hierarchical task network planners, as surveyed by Weld (1994).

A more recent research trend has been to develop new encodings of planning problems in order to adopt efficient algorithms from other research areas, leading to significant developments in planning algorithms, as surveyed by Weld (1999).

More recently, another new planner MBP (Cimatti et al., 1998b) was introduced that successfully encodes a planning domain as a non-deterministic finite automaton (NFA) represented by an Ordered Binary Decision Diagram (OBDD) (Bryant, 1986). MBP effectively extends to non-deterministic domains producing universal plans. Due to the scalability of the underlying model checking representation and search techniques, MBP can be shown to be an efficient non-deterministic planner (Cimatti et al., 1998a, 1998b).

One of our main research objectives is to develop planning systems suitable for planning in uncertain, and in particular multi-agent environments (Veloso et al., 1998; Stone & Veloso, 1999). The universal planning approach, as originally developed (Schoppers, 1987), is appealing for this type of environments, as a universal plan is a set of state-action rules that aim at covering the possible multiple situations in the non-deterministic environment.

However, the limitations of universal planning have been rightly pointed out (e.g., Ginsberg, 1989), due to the potential exponential growth of the size of the universal plan size with the number of propositions defining a domain state. An important contribution of MBP is thus the use of OBDDs to represent universal plans. In the worst case, this representation may also grow exponentially with the number of domain propositions, but because OBDDs are very compact representations of boolean functions, this is often not the case for domains with a regular structure (Cimatti et al., 1998a). Therefore, we believe that an OBDD-based planning approach combined with appropriate encodings and active learning is a promising approach to a robust integration of planning and real execution in a non-deterministic world.

MBP specifies a planning domain in the action description language $\mathcal{AR}$ (Giunchiglia et al., 1997) and translates it to a corresponding NFA, hence limited to planning problems with finite state spaces. The transition relation of the automaton is encoded as an OBDD, which allows for the use of model checking par-

[1] Universal Multi-agent OBDD-based Planner
[2] Non-deterministic Agent Domain Language.

allel breadth-first search. MBP includes two algorithms for universal planning called *strong* and *strong cyclic* planning.

In this paper we present our OBDD-based planning system, UMOP, standing for Universal Multi-agent OBDD-based Planner, which uses a new OBDD-based encoding, generates universal plans in multi-agent nondeterministic domains, and includes a new *optimistic* planning algorithm.

Our overall approach for designing an OBDD-based planner is similar to the approach developed for MBP. Our main contribution is an efficient encoding of a new front end domain description language, *NADL* (*NADL* stands for Non-deterministic Agent Domain Language.). *NADL* has more resemblance with previous planning languages than the action description language $\mathcal{AR}$ currently used by MBP. It has powerful action descriptions that can perform arithmetic operations on numerical domain variables. Domains comprised of synchronized agents can be modelled by introducing concurrent actions based on a multi-agent decomposition of the domain.

In addition, *NADL* introduces a separate and explicit environment model defined as a set of *uncontrollable* agents, i.e., agents whose actions cannot be a part of the generated plan. *NADL* has been carefully designed to allow for efficient OBDD-encoding. Thus, in contrast to $\mathcal{AR}$ *NADL* allows UMOP to generate a partitioned transition relation representation of the NFA, which is known from model checking to scale up well (Burch et al., 1991). Our empirical experiments suggest that this is also the case for UMOP.

UMOP includes the previously developed algorithms for OBDD-based universal planning. In addition, we introduce a new *optimistic* planning algorithm, which relaxes optimality guarantees and generates plausible universal plans in some domains where no strong or strong cyclic solution exists.

The paper introduces *NADL* and UMOP. It also includes a brief overview of OBDDs which may be skipped by readers already familiar with the subject.

The paper presents empirical experiments that include domains previously tested by MBP showing that our UMOP approach and implementation with its *NADL* extension to multi-agent and environment agents is of comparable effectiveness. Finally, we introduce and show results in a few new multi-agent non-deterministic domains that we hope to contribute to the general multi-agent planning and execution research community.

Introduction to OBDDs

An Ordered Binary Decision Diagram (Bryant, 1986) is a canonical representation of a boolean function with n linear ordered arguments $x_1, x_2, ..., x_n$.

An OBDD is a rooted, directed acyclic graph with one or two terminal nodes of out-degree zero labeled 1 or 0, and a set of variable nodes u of out-degree two. The two outgoing edges are given by the functions $high(u)$ and $low(u)$ (drawn as solid and dotted arrows). Each variable node is associated with a propositional variable in the boolean function the OBDD represents. The graph is ordered in the sense that all paths in the graph respect the ordering of the variables.

An OBDD representing the function $f(x_1, x_2) = x_1 \wedge x_2$ is shown in Figure 1 (left). Given an assignment of the arguments x_1 and x_2, the value of f is determined by a path starting at the root node and iteratively following the high edge, if the associated variable is true, and the low edge, if the associated variable is false. The value of f is *True* if the label of the reached terminal node is 1; otherwise it is *False*.

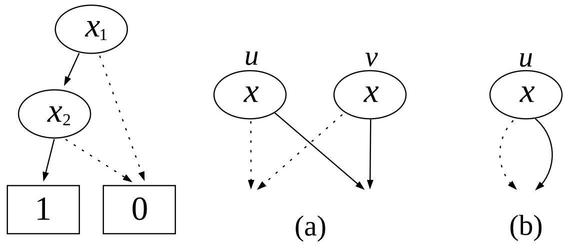

Figure 1: An OBDD representing the function $f(x_1, x_2) = x_1 \wedge x_2$. True and false edges are drawn solid and dotted, respectively. (a) and (b) Reductions of OBDDs.

An OBBD graph is reduced so that no two distinct nodes u and v have the same variable name and low and high successors (Figure 1(a)), and no variable node u has identical low and high successors (Figure 1(b)).

The OBDD representation has two major advantages: First, most commonly encountered functions have a reasonable representation (Bryant, 1986). Second, any operation on two OBDDs, corresponding to a boolean operation on the functions they represent, has a low complexity bounded by the product of their node counts.

In OBDD-based planning OBDDs are used to represent the transition relation semantics of the planning domain. This OBDD representation of finite state transition systems origins from model checking (McMillan, 1993).

NADL

In this section, we first discuss the properties of *NADL* based on an informal definition of the language and a domain encoding example. We then describe the formal syntax and semantics of *NADL*.

An *NADL* domain description consists of: a definition of *state variables*, a description of *system* and *environment agents*, and a specification of an *initial* and *goal conditions*.

The set of state variable assignments defines the state space of the domain. An agent's description is a set of *actions*. The agents change the state of the world by performing actions, which are assumed to be executed synchronously and to have a fixed and equal duration. At each step, all of the agents perform exactly one action, and the resulting action tuple is a *joint action*. The system agents model the behavior of the agents controllable by the planner, while the environment agents model the uncontrollable world. A valid domain description requires that the system and environment agents constrain a disjoint set of variables.

An action has three parts: a set of *state variables*, a *precondition* formula, and an *effect* formula. Intuitively the action takes responsibility of constraining the values of the set of state variables in the next state. It further has exclusive access to these variables during execution.

There are two causes for non-determinism in *NADL* domains: (1) actions not restricting all their constrained variables to a specific value in the next state, and (2) the non-deterministic selection of environment actions.

A simple example of an *NADL* domain description is shown in Figure 2.[3] The domain describes a planning problem for Schoppers' (1987) robot-baby domain. The domain has two state variables: a numerical one, *pos*, with range $\{0,1,2,3\}$ and a propositional one, *robot_works*. The robot is the only system agent and it has two actions *Lift-Block* and *Lower-Block*. The baby is the only environment agent and it has one action *Hit-Robot*. Because each agent must perform exactly one action at each step, there are two joint actions (*Lift-Block*,*Hit-Robot*) and (*Lower-Block*,*Hit-Robot*).

Initially the robot is assumed to hold a block at position 0, and its task is to lift it up to position 3. The *Lift-Block* (and *Lower-Block*) action has a conditional effect described by an if-then-else operator: if *robot_works* is true, *Lift-Block* increases the block position with one, otherwise the block position is unchanged.

Initially *robot_works* is assumed to be true, but it can be made false by the baby. The baby's action *Hit-Robot* is non-deterministic, as it only constrains *robot_works* by the effect expression $\neg robot\_works \Rightarrow \neg robot\_works'$. Thus, when *robot_works* is true in the current state, the effect expression of *Hit-Robot* does not apply, and *robot_works* can either be true or false in the next state. On the other hand, if *robot_works* is false in the current state, *Hit-Robot* keeps it false in the next state.

An NFA representing the domain is shown in Figure 3.

The explicit representation of constrained state variables enables any non-deterministic or deterministic effect of an action to be represented, as the constrained

variables
  nat(4) *pos*
  bool *robot_works*
system
  agt: Robot
    Lift-Block
      con: *pos*
      pre: $pos < 3$
      eff: $robot\_works \rightarrow pos' = pos + 1, pos' = pos$
    Lower-Block
      con: *pos*
      pre: $pos > 0$
      eff: $robot\_works \rightarrow pos' = pos - 1, pos' = pos$
environment
  agt: Baby
    Hit-Robot
      con: *robot_works*
      pre: *true*
      eff: $\neg robot\_works \Rightarrow \neg robot\_works'$
initially
  $pos = 0 \wedge robot\_works$
goal
  $pos = 3$

Figure 2: The robot-baby *NADL* domain: An example.

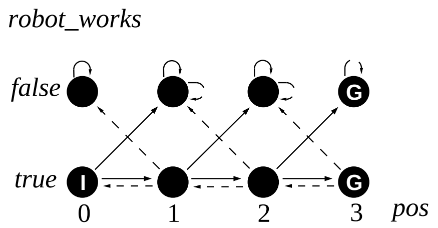

Figure 3: The NFA of the robot-baby domain. The *Lift-Block* and *Lower-Block* actions are drawn with solid and dashed arrows, respectively. States marked with "I" and "G" are initial and goal states.

variables can be assigned to any value in the next state that satisfies the effect formula. It further turns out to have a clear intuitive meaning, as the action takes the "responsibility" of specifying the values of the constrained variables in the next state.

Compared to the action description language $\mathcal{AR}$ that is the only prior language used for non-deterministic OBDD-based planning (Cimatti et al., 1998a, 1998b), *NADL* introduces an explicit environment model, a multi-agent decomposition and numerical state variables. It can further be shown that *NADL* can be used to model any domain that can be modelled with $\mathcal{AR}$ (see Appendix A).

In *NADL* actions cannot be performed concurrently if: 1) they have inconsistent effects, or 2) they constrain an overlapping set of state variables. The first condition is due to the fact that state knowledge is expressed in a monotonic logic which cannot represent inconsistent knowledge. The second rule addresses the

[3] Unquoted, e.g. *pos* and quoted variables, e.g. *pos'* refer to the current and next state, respectively. Another notation like pos_t and pos_{t+1} could have been used. We have chosen the quote notation because it is the common notation in model checking.

problem of sharing resources. Consider for example two agents trying to drink the same glass of water. If only the first rule defined interfering actions both agents, could simultaneously empty the glass, as the effect *glass_empty* of the two actions would be consistent. With the second rule added, these actions are interfering and cannot be performed concurrently.

Syntax

Formally, an *NADL* description is a 7-tuple $D = (SV, S, E, Act, d, I, G)$, where:

- SV finite set of propositional and numerical state variables.
- S is a finite, nonempty set of system agents.
- E is a finite set of environment agents.
- Act is a set of action descriptions (c, p, e) where c is the state variables constrained by the action, p is a precondition state formula in the set *SForm* and e is an effect formula in the set *Form*. The sets *SForm* and *Form* are defined below.
- $d : Agt \rightarrow 2^{Act}$ is a function mapping agents ($Agt = S \cup E$) to their actions.
- $I \in SForm$ is the initial condition.
- $G \in SForm$ is the goal condition.

The set of formulas *Form* are arithmetic and boolean expressions on state variables of the current and next state. $SForm \subset Form$ is a subset of the formulas only referring to current state variables. These formulas are called *state formulas.*

OBDD Representation of *NADL* Descriptions

The formal semantics of a domain description $D = (SV, S, E, Act, d, I, G)$ is given in terms of an NFA M:

Definition 1 (NFA)
A Non-deterministic Finite Automaton is a 3-tuple, $M = (Q, \Sigma, \delta)$, where Q is a set of states, Σ is a set of input values and $\delta : Q \times \Sigma \rightarrow 2^Q$ is a next state function.

The states Q of M equals the set of all possible variable assignments. The input Σ of M is the set of joint actions of system agents. In order to define the next state function δ we express it as a transition relation $T(s, i, s') = (s' \in \delta(s, i))$ and represent it by an OBDD $\tilde{T}$.

To construct $\tilde{T}$ we must define a set of boolean variables to represent the current state s, the joint action input i and the next state s'. Joint action inputs are represented in the following way: assume action a is identified by a number p and can be performed by agent α. a is then defined to be the action of agent α, if the number expressed binary by a set of boolean variables A_α, used to represent the actions of α, is equal to p. Propositional state variables of the current state s and next state s' are represented by a single boolean variable, while numerical state variables are represented binary by a set of boolean variables.

Let A_{e_1} to $A_{e_{|E|}}$ and A_{s_1} to $A_{s_{|S|}}$ denote sets of boolean variables used to represent the joint action of system and environment agents. Further, let $x^k_{v_j}$ and $x'^k_{v_j}$ denote the *k'th* boolean variable used to represent state variable $v_j \in SV$ in the current and next state. An ordering of the boolean variables, known to be efficient from model checking, puts the input variables first followed by an interleaving of the boolean variables of current state and next state variables:

$$
\begin{aligned}
& A_{e_1} \prec \cdots \prec A_{e_{|E|}} \prec A_{s_1} \prec \cdots \prec A_{s_{|S|}} \\
& \quad \prec x^1_{v_1} \prec x'^1_{v_1} \prec \cdots \prec x^{m_1}_{v_1} \prec x'^{m_1}_{v_1} \\
& \quad \cdots \\
& \quad \prec x^1_{v_n} \prec x'^1_{v_n} \prec \cdots \prec x^{m_n}_{v_n} \prec x'^{m_n}_{v_n}
\end{aligned}
$$

where m_i is the number of boolean variables used to represent state variable v_i and n is equal to $|SV|$. An OBDD representing a logical expression is built in the standard way. Arithmetic expressions are represented as lists of OBDDs defining the corresponding binary number. They collapse to single OBDDs when related by arithmetic relations.

$\tilde{T}$ is a conjunction of three relations $\tilde{A}$, $\tilde{F}$ and $\tilde{I}$. We first build a transition relation with the joint actions of both system and environment agents as input and then reduces it to a transition relation with only joint actions of system agents as input.

$\tilde{A}$ defines the constraints on the current state and next state of joint actions. In order to build $\tilde{A}$ we need to refer to the values of the boolean variables representing the actions. Let $i(\alpha)$ be the function that maps an agent α to the value of the boolean variables representing its action and let $b(a)$ be the identifier value of action a. Further let $\tilde{P}(a)$ and $\tilde{E}(a)$ denote OBDD representations of the precondition and effect formula of an action a. $\tilde{A}$ is then given by:

$$
\tilde{A} = \bigwedge_{\substack{\alpha \in Agt \\ a \in d(\alpha)}} \Big(i(\alpha) = b(a) \Rightarrow \tilde{P}(a) \wedge \tilde{E}(a) \Big)
$$

Note that logical operators denote the corresponding OBDD operators in the above formula . $\tilde{A}$ also ensures that actions with inconsistent effects cannot be performed concurrently, as $\tilde{A}$ reduces to false if any pair of actions in a joint action have inconsistent effects. Thus, $\tilde{A}$ also states the first rule for avoiding interference between concurrent actions.

$\tilde{F}$ is a frame relation ensuring that unconstrained variables maintain their value:

$$
\tilde{F} = \bigwedge_{v \in SV} \Big(\big(\bigwedge_{\substack{\alpha \in Agt \\ a \in d(\alpha)}} (i(\alpha) = b(a) \Rightarrow v \notin c(a)) \big) \Rightarrow s'_v = s_v \Big),
$$

where $c(a)$ is the set of constrained variables of action a and $s_v = s'_v$ expresses that all current and next state boolean variables representing v are pairwise equal. The expression $v \notin c(a)$ evaluates to *True* or *False* and is represented by the OBDD for *True* or *False*.

$\tilde{I}$ ensures that concurrent actions constrain a non overlapping set of variables and thus states the second rule for avoiding interference between concurrent actions:

$$\begin{aligned} \tilde{I} \;=\; & \bigwedge_{\substack{(\alpha_1, \alpha_2) \in S^2 \\ (a_1, a_2) \in c(\alpha_1, \alpha_2)}} \Big(i(\alpha_1) = b(a_1) \Rightarrow i(\alpha_2) \neq b(a_2) \Big) \wedge \\ & \bigwedge_{\substack{(\alpha_1, \alpha_2) \in E^2 \\ (a_1, a_2) \in c(\alpha_1, \alpha_2)}} \Big(i(\alpha_1) = b(a_1) \Rightarrow i(\alpha_2) \neq b(a_2) \Big), \end{aligned}$$

where $c(\alpha_1, \alpha_2) = \{(a_1, a_2) \,|\, (a_1, a_2) \in d(\alpha_1) \times d(\alpha_2) \wedge c(a_1) \cap c(a_2) \neq \emptyset\}$.

Finally the OBDD representing the transition relation $\tilde{T}$ is the conjunction of $\tilde{A}$, $\tilde{F}$ and $\tilde{I}$ with action variables of the environment agents existentially quantified:

$$\tilde{T} = \exists A_{e_1}, \cdots, A_{e_{|E|}} \,.\, \tilde{A} \wedge \tilde{F} \wedge \tilde{I}$$

Partitioning the transition relation

The algorithms we use for generating universal plans all consist of a backward search from the states satisfying the goal condition to the states satisfying the initial condition. Empirical studies in model checking have shown that the most complex operation for this kind of algorithms normally is to find the *preimage* of a set of visited states V.

Definition 2 (Preimage) *Given an NFA* $M = (Q, \Sigma, \delta)$ *and a set of states* $V \subseteq Q$*, the preimage of* V *is the set of states* $\{s \,|\, s \in Q \wedge \exists i \in \Sigma, s' \in \delta(s, i) \,.\, s' \in V\}$.

Note that states already belonging to V can also be a part of the preimage of V. Assume that the set of visited states are represented by an OBDD expression $\tilde{V}$ on next state variables and that we for iteration purposes, want to generate the preimage $\tilde{P}$ also expressed in next state variables. An efficient way to calculate the preimage is to use a partitioned representation of the transition relation ($\tilde{T} = \tilde{T}_1 \wedge \cdots \wedge \tilde{T}_n$ combined with early quantification (Burch et al., 1991):

$$\begin{aligned} \tilde{U} &= (\exists \vec{x}'_n \,.\, \tilde{T}_n \wedge \cdots \wedge (\exists \vec{x}'_1 \,.\, \tilde{T}_1 \wedge \tilde{V}) \cdots)[\vec{x}/\vec{x}'] \\ \tilde{P} &= \exists \vec{i} \,.\, \tilde{U} \end{aligned}$$

where $\vec{i}$, $\vec{x}_j$ and $\vec{x}'_j$ denote input, current state and next state variables of partition j, and $[\vec{x}_j/\vec{x}'_j]$ denotes the substitution of current state variables with next state variables of partition j. $\tilde{T}_1$ can refer to all variables, $\tilde{T}_2$ can refer to all variables except $\vec{x}'_1$, $\tilde{T}_3$ can refer to all variables except $\vec{x}'_1$ and $\vec{x}'_2$ and so on.

The set expressed by $\tilde{U}$ consists of state input pairs (s, i), for which the state s belongs to the preimage of V and the input i may cause a transition from s to a state in V.

The input of an NFA representing a planning domain is actions. Thus, for a planning domain the elements in $\tilde{U}$ are state-action pairs. The generated universal plans of the universal planning algorithms presented in the next section are sets of these state-action pairs. We refer to the state-action pairs as *state-action rules*, because they associate states to actions that can be performed in these states.

NADL has been carefully designed to allow a partitioned transition relation representation. The relations A, F and I all consist of a conjunction of subexpressions that normally only refer to a subset of next state variables.

OBDD-based Universal Planning Algorithms

In this section we will describe two prior algorithms for OBDD-based universal planning and discuss which kind of domains they are suitable for. Based on this discussion we present a new algorithm called *optimistic planning* that seems to be suitable for some domains not covered by the prior algorithms.

The three universal planning algorithms discussed are all based on an iteration of preimage calculations. The iteration corresponds to a parallel backward breadth first search starting at the goal states and ending when all initial states are included in the set of visited states (see Figure 4). The main difference between the algorithms is the way the preimage is defined.

Strong and Weak Preimages

Let us introduce two different kinds of preimages namely *strong* and *weak preimages*. A strong preimage is defined by:

Definition 3 (Strong Preimage) *Given an NFA* $M = (Q, \Sigma, \delta)$ *and a set of states* $V \subseteq Q$*, the strong preimage of* V *is the set of states* $\{s \,|\, s \in Q \wedge \exists i \in \Sigma \,.\, \delta(s, i) \subset V\}$.

Thus, for a state s belonging to the strong preimage of a set of states V, there exists at least one action i where all the transitions from s associated with i leads into V. Consider the example shown in Figure 4. The dots and arrows in this figure denote states and transitions for an NFA with a single action. For the set of goal states shown in the figure the three states having a transition into the goal set is a strong preimage (indicated by a solid ellipse), as all transitions from these states lead to a goal state.

A weak preimage is equal to an ordinary preimage defined in Definition 2. Thus, in Figure 4 all the strong

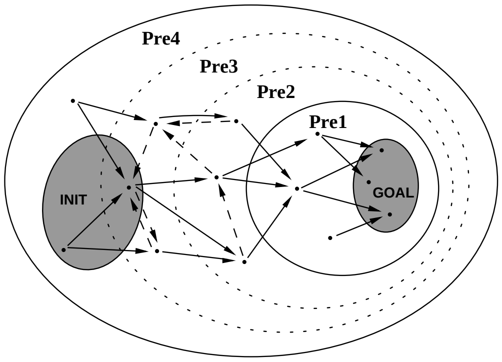

Figure 4: Preimage calculations: Solid and dashed ellipses denote preimages that are both strong and weak, and only weak, respectively. The domain has only one action. Dashed transitions cause a state to belong to a weak preimage rather than to a strong preimage.

preimages are also weak preimages, but the preimages shown by dashed ellipses are only weak preimages, as the dashed transitions do not satisfy the strong preimage definition.

Strong and Strong Cyclic Planning

A strong or strong cyclic plan is the union of the state-action rules U found when calculating the preimages necessary for covering the set of initial states.

Strong planning only considers strong preimages. If a sequence of strong preimages starting at the set of goal states can be calculated, such that the set of initial states is covered, strong planning succeeds and returns the universal plan consisting of the union of all the state-action rules of the calculated strong preimages. Otherwise it fails.

Consider the example in Figure 4. As depicted in the figure a strong preimage can be found in the first preimage calculation, but only a weak preimage can be found in the second calculation. Thus, no strong solution exist for this problem.

Strong planning is complete with respect to strong solutions. If a strong plan exists for some planning problem the strong planning algorithm will return it, otherwise, it returns that no solution exists. Strong planning is also optimal due to the breadth first search. Thus, a strong plan with the fewest number of steps in the worst case is returned.

Strong cyclic planning is a relaxed version of strong planning, because it also considers weak preimages. Strong cyclic planning finds a strong plan, if it exists. Otherwise, if the algorithm at some point in the iteration is unable to find a strong preimage it adds a weak preimage. It then tries to prune this preimage by removing all states that have transitions leading out of the preimage and the set of visited states V. If it succeeds, the remaining states in the preimage are added to V and it again tries to add strong preimages. If it fails, it adds a new, weak preimage and repeats the pruning process.

Figure 4 shows a strong cyclic solution that could could have been computed by the strong cyclic planning algorithm. A strong cyclic plan only guarantees progress towards the goal in the strong parts. In the weak parts, cycles can occur.

Strengths and Limitations of Strong and Strong Cyclic Planning

Strong planning and strong cyclic planning algorithms contribute by providing complete OBDD-based algorithms for universal planning.

A limitation of strong and strong cyclic planning is that they can not find a solution in domains where no strong or strong cyclic plan exists. The domains that strong and strong cyclic planning fail in are characterized by having unrecoverable dead-ends that cannot be guaranteed to be avoided.

Unfortunately, real world domains often have these kinds of dead-ends. Consider, for example, Schoppers' robot-baby domain. As depicted in Figure 3, no universal plan represented by a set of state-action rules can guarantee the goal to be reached in a finite or infinite number of steps, as all relevant actions may lead to an unrecoverable dead-end.

Another limitation of strong and strong cyclic planning is the inherent pessimism of these algorithms. Consider for example the domain (Domain 1) illustrated in Figure 5. The domain consists of $n+1$ states and two different actions (dashed and solid). The strong cyclic algorithm returns a strong plan

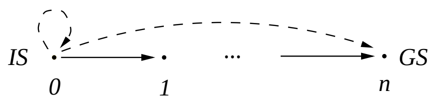

Figure 5: Domain 1.

$\{(0, solid), (1, solid), \cdots, (n-1, solid)\}$. This plan would have a best and worst case length of n. But a strong cyclic plan $\{(0, dashed), (n-1, solid)\}$ also exists and could be preferable because the best case length of 1 of the cyclic solution may have a much higher probability than the infinite worst case length. Strong cyclic planning will always prefer to return a strong plan, if it exists, even though a strong cyclic plan may exist with a shorter, best case plan length.

By adding an unrecoverable dead-end for the dashed action and making solid actions non-deterministic (see Domain 2, Figure 6), strong cyclic planning now returns the strong cyclic plan $\{(0, solid), (1, solid), \cdots, (n-1, solid)\}$. But we might still be interested in the plan $\{(0, dashed), (n-1, solid)\}$ even though the goal is not guaranteed to be achieved.

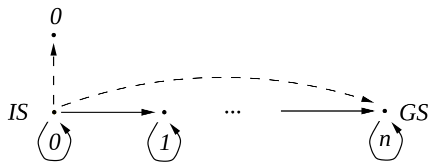

Figure 6: Domain 2.

Optimistic Planning

The analysis in the previous section shows that there are domains and planning problems for which we may want to use a fully relaxed algorithm that always includes the best case plan and returns a solution even, if it includes dead-ends which cannot be guaranteed to be avoided. An algorithm similar to the strong

```
procedure OptimisticPlanning(Init, Goal)
  VisitedStates := Goal
  UniversalPlan := ∅
  while (Init ⊄ VisitedStates)
    StateActions := Preimage(VisitedStates)
    PrunedStateActions := Prune(StateActions, VisitedStates)
    if StateActions ≠ ∅ then
      UniversalPlan := UniversalPlan ∪ PrunedStateActions
      VisitedStates := VisitedStates
        ∪ StatesOf(PrunedStateActions)
    else
      return "No optimistic plan exists"
return UniversalPlan
```

Figure 7: The optimistic planning algorithm.

planning algorithm that adds an ordinary preimage in each iteration has these properties. Because state-action rules leading to unrecoverable dead-ends may be added to the universal plan, we call this algorithm *optimistic planning*. The algorithm is shown in Figure 7. The function Preimage(*VisitedStates*) returns the set of state-action rules U associated with the preimage of the visited states. Prune(*StateActions*, *VisitedStates*) removes the state-action rules, where the state already is included in the set of visited states, and StatesOf(*PrunedStateActions*) returns the set of states of the pruned state-action rules. UMOP includes the optimistic planning algorithm.

The purpose of optimistic planning is not to substitute strong or strong cyclic planning. In domains where strong or strong cyclic plans can be found and goal achievement has the highest priority these algorithms should be used. On the other hand, in domains where goal achievement cannot be guaranteed or the shortest plan should be included in the universal plan, optimistic planning might be the better choice.

Consider again, as an example, the robot-baby domain. For this problem the optimistic solution makes the robot try to lift the block when the position of the block is less than 3 and the robot is working. This seems to be the only reasonable strategy.

Domain	Strong		Strong Cyclic		Optimistic	
	best	worst	best	worst	best	worst
1	n	n	1	∞	1	∞
2	-	-	n	n	1	D

Table 1: The best and worst case plan length of strong, strong cyclic and optimistic planning in Domains 1 and 2 (see Figure 5 and 6). "-" means that no solution exists. "D" means that a solution exists, but may lead to an unrecoverable dead-end.

For domains 1 and 2 shown in Figure 5 and 6, optimistic planning returns a universal plan $\{(0, \mathit{dotted}), (n-1, \mathit{solid})\}$. For both domains this is a universal plan with the shortest best case length. Compared to the strong cyclic solution the price in the first domain is that the plan may have an infinite length, while the price in the second domain is that a dead-end may be reached. The results of strong, strong cyclic and optimistic planning in Domain 1 and 2 are summarized in Table 1.

Empirical Results

The input to UMOP is an *NADL* description[4] and a specification of which planning algorithm to use. This description is then converted to a set of OBDDs representing the partitioned transition relation. The OBDD representation is used by either the Strong, Strong Cyclic or Optimistic planning algorithm to generate a plan. The output of UMOP is an universal plan if it exists.

In the following four subsections we present results obtained with the UMOP planning system.[5] A more detailed description of the experiments can be found in Jensen (1999). *NADL* descriptions of the domains are posted at `http://www.cs.cmu.edu/~runej`.

Domains Tested by MBP

One of the domains solved by MBP is a non-deterministic transportation domain. The domain consists of a set of locations and a set of actions like drive-truck, drive-train and fly to move between the locations. Non-determinism is caused by non-deterministic actions (e.g., a truck may use the last fuel) and environmental changes (e.g., fog at airports). We defined the two domain examples tested by MBP for strong and strong cyclic planning in *NADL* and ran UMOP using strong and strong cyclic planning. Both examples were solved in less than 0.05 seconds. Similar results were obtained with MBP.

The problem in the beam walk domain is for an agent to walk from one end of a beam to the other without falling down. If the agent falls, it has to walk back to the end of the

[4] In fact, the *NADL* description accepted by the current implementation includes only the arithmetic operators + and −, but an implementation of the remaining operators is straightforward and is a part of our current work.

[5] All experiments were carried out on a 450 MHz Pentium PC with 1 GB RAM running Red Hat Linux 4.2.

beam and try again. The finite state machine of the domain is shown in Figure 8. The propositional state variable *up* is true if the agent is on the beam. The numerical state variable *pos* denotes the position of the agent either on the beam or on the ground.

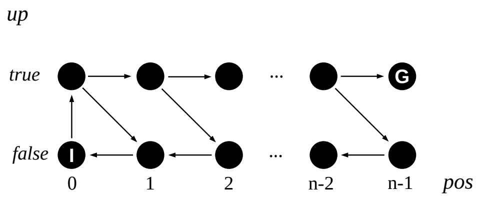

Figure 8: The beam walk domain. The propositional state variable *up* is true if the agent is on the beam.

We implemented a generator program for *NADL* descriptions of beam walk domains and produced domains with 4 to 4096 positions. Because the domain only contains two state variables, UMOP cannot exploit a partitioned transition relation for this domain. As

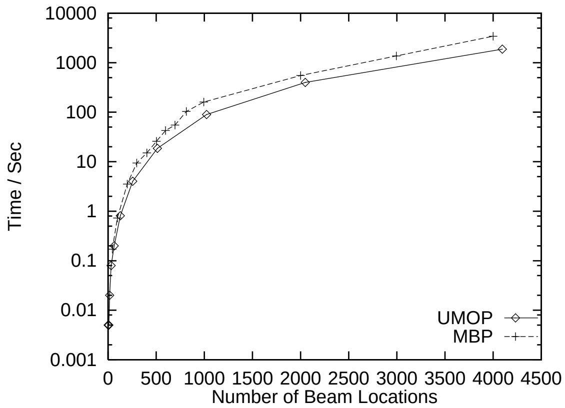

Figure 9: Comparable planning time of UMOP and MBP in the beam walk domain. The MBP data has been extracted with some loss of accuracy from (Cimatti et al., 1998a).

shown in Figure 9 the performance of UMOP is comparable to MBP when using a monolithic transition relation.

The Magnetic Bar Domain

The magnetic bar domain has been constructed to show all the modelling features of *NADL* . It further demonstrates how the different universal planning algorithms can be used to extract information about the domain structure.

The Magnetic Bar Domain is a multi-agent domain consisting of two moving bars and two metal objects. By magnetizing the bars, the objects can be lifted and moved. The goal is to move the objects from random positions on a 8×8 grid to a drop zone in the upper right corner of the grid. A solution to an instance of this problem is shown in Figure 10.

To make the problem non-trivial we assume that the objects and bars can interact in the following way: an object located in front of some moving object blocks its way. An

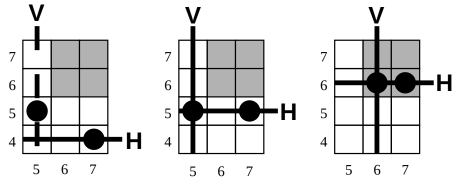

Figure 10: A solution example for the Magnetic Bar Domain. Notice the needed collaboration between the bars in moving the left object into the drop zone.

object attached to a bar blocks the way for the other bar. Thus in the first state in Figure 10 the object under the vertical bar would block the horizontal bar from moving up if the vertical bar was magnetized. Finally, The domain is made non-deterministic by adding an environment agent that at any time can occupy a random grid position. When a grid position is occupied a bar can not pass it.

The *NADL* description of the domain has three agents: Two system agents one for each bar and one environment agent that can occupy grid positions. The size of the state space is 2^{27}. To state a universal planning problem the initial states are defined to be all other states than the goal states.

Using the universal planning algorithms to analyze the domain it turns out that UMOP in less than 0.89 seconds fails to find a strong solution (only one strong preimage can be computed). This is not surprising as the occupation of grid positions can prevent progress towards the goal. A strong cyclic solution is found by UMOP in 602 seconds after 59 preimage. Thus, despite the restrictions there is in fact a cyclic solution covering all the initial states.

The generated plan is large (45 MB) but has a sufficiently low lookup time to be used in a practical implementation (less than 0.001 second). Partitioning of the transition relation (6 partitions in this case) is crucial for the efficiency of UMOP. With a monolithic transition relation UMOP is magnitudes slower and uses more memory after the fifth preimage calculation than is used after the last preimage calculation when using a partitioned transition relation.

The Power Plant Domain

The power plant domain demonstrates a multi-agent domain with an environment model and further exemplifies optimistic planning. It consists of reactors, heat exchangers, turbines and valves. A domain example is shown in Figure 11. In the power plant domain each controllable unit is associated with an agent such that all control actions can be performed simultaneously. The environment consists of a single agent that at any time can fail a number of heat exchanges and turbines and ensure that already failed units remain failed.

The state space of the power plant can be divided into three disjoint sets: good, bad and failed states. In the good states, therefore the goal states, the power plant satisfies its safety and activity requirements. In our example the safety requirements ensure that energy can be transported away from the plant, and that failed units are shut down. The activity requirements state that the energy production equals the demand and that all valves to working turbines are open.

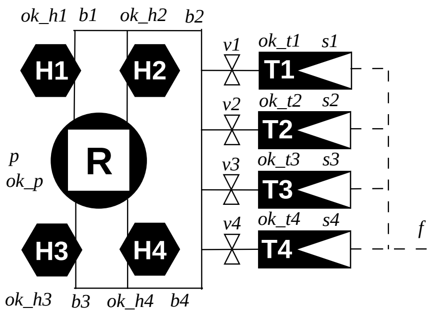

Figure 11: A power plant domain with four heat exchanges (H1-H4) and turbines (T1-T4). The *ok* variables capture the working status of the units.

In a bad state the plant does not satisfy the safety and activity requirements, but on the other hand is not unrecoverably failed. In a failed state all heat exchangers or turbines are failed.

The environment can fail any number of units during execution, thus, for any bad state the resulting joint action may loop back to a bad state or cause the plant to end in a failed state. For this reason no strong or strong cyclic solution exists to the problem.

An optimistic solution simply ignores that joint actions can loop back to a bad state or lead to a failed state and finds a solution to the problem after one preimage calculation.

The size of the state space of the above power plant domain is 2^{24}. An optimistic solution was generated by UMOP in 0.92 seconds and contained 37619 OBDD nodes.

The Soccer Domain

The purpose of the soccer domain is to demonstrate a multi-agent domain with a more elaborate environment model than the power plant domain. It consists of two teams of players that can move in a grid world and pass a ball to each other. The task is to generate a universal plan for one of the teams that can be applied, whenever the team possesses the ball in order to score a goal.

A simple *NADL* description of The team possessing the ball and the opponent team is modeled by a set of system and environment agents, respectively. The goal of the universal plan is to move a player possessing the ball in front of the opponent goal without having any opponents in the goal area.

We implemented an *NADL* generator for soccer domains with different field sizes and numbers of agents. The Multi-Agent graph in Figure 12 shows UMOP's planning time using the strong planning algorithm in soccer domains with 64 locations and one to six players on each team.

The planning time seems to grow exponential with the number of players. This is not surprising as not only the state space but also the number of joint actions grow exponential with the number of agents. To investigate the complexity introduced by joint actions we constructed a version of the soccer domain with only a single system and environment agent and ran UMOP again. The Single-Agent graph in Figure 12 shows the dramatic decrease in computation time. Its is not obvious though, that a parallelization of domain actions increases the computational load, as this

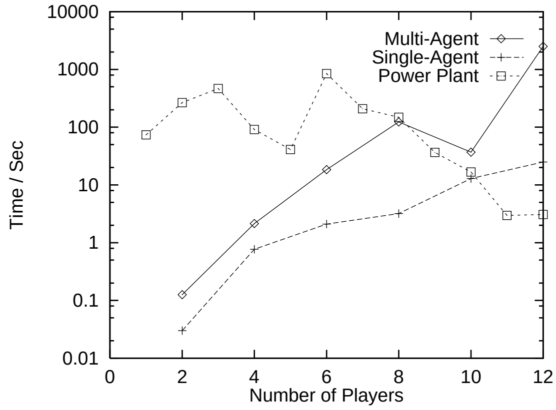

Figure 12: Planning time of UMOP in soccer domains with 1-6 players per team.

normally also reduces the number of preimage calculations, because a larger number of states is reached in each iteration. Indeed, in a deterministic version of the power plant domain we found the planning time to decrease (see the Power Plant graph in Figure 12), when more agents were added (Jensen, 1999).

Previous Work

Universal planning was introduced by Schoppers (1987) who used decision trees to represent plans. Recent approaches include Kabanza et al. (1997) and Cimatti et al. (1998a, 1998b). Kabanza et al. (1997) represents universal plans as a set of Situated Control Rules. Their algorithm incrementally adds SCRs to a final plan. The goal is a formula in temporal logic that must hold on any valid sequence of actions.

Reinforcement Learning (RL) (Sutton & G., 1998) can also be regarded as universal planning. In RL the goal is represented by a reward function in a Markov Decision Process (MDP) model of the domain. In the precursor version of RL, the MDP is assumed to be known and a control policy maximizing the expected reward is found prior to execution. Because RL is a probabilistic approach, its domain representation is more complex than the domain representation used by a non-deterministic planner. Thus, we may expect non-deterministic planners to be able to handle domains with a larger state space than RL.

All previous approaches to universal planning, except Cimatti et al. (1998a, 1998b), use an explicit representation of the universal plan (e.g., SCRs). Thus, in the general case exponential growth of the plan size with the number of propositions defining a domain state must be expected, as argued by Ginsberg (1989).

The compact and implicit representation of universal plans obtained with OBDDs does not necessarily grow exponentially for regular domains as shown by Cimatti et al. (1998a). Further, the OBDD-based representation of the NFA of a non-deterministic domain enables the application of efficient search algorithms from model checking, capable of handling very large state spaces.

Conclusion

In this paper we have presented a new OBDD-based planning system called UMOP for planning in non-deterministic, multi-agent domains. An expressive domain description language called *NADL* has been developed and an efficient OBDD representation of its NFA semantics has been described. We have analyzed previous planning algorithms for OBDD-based planning and deepened the understanding of when these planning algorithms are appropriate. Finally, we have proposed a planning algorithm called optimistic planning for finding sensible solutions in some domains where no strong or strong cyclic solution exists.

Future challenges include extending *NADL* to handle constructive synergetic effects and adding domain knowledge by stating the goal as a formula in temporal logic on the sequence of actions leading to the goal. Further, we are interested in introducing abstraction in OBDD-based planning and defining specialized planning algorithms for multi-agent domains and deterministic domains (Jensen et al., 2000).

Acknowledgments

Special thanks to Paolo Traverso, Marco Roveri and the other members of the IRST group for introducing us to MBP and for many rewarding discussions on OBDD-based planning and model checking. We also wish to thank Randal E. Bryant, Edmund Clarke, Henrik R. Andersen, Jørn Lind-Nielsen and Lars Birkedal for advice on OBDD issues and formal representation.

This research was sponsored in part by Grants Nos. F30602-98-2-0135 and F30602-97-2-0250, and by McKinsey & Company, Selmer & Trane's Fond. The content of this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

Appendix A. *NADL* includes $\mathcal{AR}$

Theorem 1 *If A is a domain description in some $\mathcal{AR}$ language, then there exists a domain description D in NADL with the same semantics as A.*

Proof: let $M_a = (Q, \Sigma, \delta)$ denote the NFA (see Definition 1) equal to the semantics of A as defined by Giunchiglia et al. (1997). An *NADL* domain description D with semantics equal to M_a can obviously be constructed in the following way: let D be a single-agent domain, where all fluents are encoded as numerical state variables and there is an action for each element in the alphabet Σ of M_a. Consider the action a associated to input $i \in \Sigma$. Let the set of constrained state variables of a equal the set of state variables in D. The precondition of a is an expression that defines the set of states having an outgoing transition for input i. The effect condition of a is a conjunction of conditional effects $(P_s \Rightarrow N_s)$. There is one conditional effect for each state that has an outgoing transition for input i. P_s in the conditional effect associated with state s is the characteristic expression for s and N_s is a characteristic expression for the set of next states $\delta(s, i)$. □

References

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *8*, 677–691.

Burch, J., Clarke, E., & Long, D. (1991). Symbolic model checking with partitioned transition relations. In *International Conference on Very Large Scale Integration*, pp. 49–58. North-Holland.

Cimatti, A., Roveri, M., & Traverso, P. (1998a). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of AAAI'98*, pp. 875–881. AAAI Press/The MIT Press.

Cimatti, A., Roveri, M., & Traverso, P. (1998b). Strong planning in non-deterministic domains via model checking. In *Proceedings of AIPS'98*, pp. 36–43. AAAI Press.

Ginsberg, M. L. (1989). Universal planning: An (almost) universal bad idea. *AI Magazine*, *10*(4), 40–44.

Giunchiglia, E., Kartha, G. N., & Lifschitz, Y. (1997). Representing action: Indeterminacy and ramifications. *Artificial Intelligence*, *95*, 409–438.

Jensen, R. M. (1999). OBDD-based universal planning in multi-agent, non-deterministic domains. Master's thesis, Technical University of Denmark, Department of Automation. IAU99F02.

Jensen, R. M., Bryant, R. E., & Veloso, M. M. (2000). Efficient encoding for OBDD-based deterministic planning. In *Proceedings of AIPS'00 workshop on model-theoretic approaches to planning.* Forthcoming.

Kabanza, F., Barbeau, M., & St-Denis, R. (1997). Planning control rules for reactive agents. *Artificial Intelligence*, *95*, 67–113.

McMillan, K. L. (1993). *Symbolic Model Checking.* Kluwer Academic Publ.

Schoppers, M. J. (1987). Universal plans for reactive robots in unpredictable environments. In *Proceedings of IJCAI-87*, pp. 1039–1046. Morgan Kaufmann.

Stone, P., & Veloso, M. (1999). Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, *110*(2), 241–273.

Sutton, R. S., & G., B. A. (1998). *Reinforcement learning: an introduction.* MIT Press.

Veloso, M. M., Pollack, M. E., & Cox, M. T. (1998). Rationale-based monitoring for planning in dynamic environments. In *Proceedings of AIPS'98*, pp. 171–179. AAAI Press.

Weld, D. (1994). An introduction to least-commitment planning. *AI Magazine*, 27–61.

Weld, D. (1999). Recent advances in AI planning. *AI Magazine*, 93–123.