OBDD-based Deterministic Planning using the UMOP Planning Framework

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. Ordered Binary Decision Diagrams (OBDDs) encode a planning domain as a finite transition system and fast algorithms from model checking search for a solution plan. With proper encodings, OBDDs can effectively scale and can provide plans for complex planning domains. In this paper, we present results obtained in classical deterministic domains using UMOP,[1] a new universal OBDD-based planning framework applicable to non-deterministic and multi-agent domains (Jensen & Veloso, 1999). A key difference between UMOP and previous OBDD-based planning systems is that the OBDD encoding of planning problems is partitioned. This representation is known from model checking to scale up the problem size that can be handled (Ranjan et al., 1995). Experimental results from the STRIPS track of the AIPS'98 planning competition show that this is also the case for OBDD-based planning. The results further indicate that UMOP is an efficient deterministic 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 new 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 (Weld, 1999). This class of planning algorithms includes GRAPHPLAN (Blum & Furst, 1995), which uses a flow-graph encoding to constrain the search and SATPLAN (Kautz & Selman, 1996), which encodes the planning problem as a satisfiability problem and uses fast model satisfaction algorithms to find a solution.

Ordered Binary Decision Diagrams (OBDDs, Bryant, 1986) have been used to encode the transition relation of planning domains. The encoding originates in model checking (McMillan, 1993), where efficient techniques have been developed for search in OBDD-based transition systems (Burch et al., 1991). Fast planning algorithms based on these techniques have been developed for domains ranging from single-agent and deterministic with no environment model to multi-agent and non-deterministic with an explicit environment model (Cimatti et al., 1998; Jensen & Veloso, 1999).

OBDD-based planning is a promising approach for classical deterministic planning. In this paper we present results obtained with our OBDD-based planning framework, UMOP (UMOP stands for Universal Multi-agent OBDD-based Planner), in deterministic domains.

Compared to a previous OBDD-based planning systems for deterministic domains (Di Manzo et al., 1998; Edelkamp & Reffel, 1999) the logical expression of the frame relation can be split into conjunctive partitions. This allows UMOP to use a partitioned transition relation encoding of planning domains, which is known from model checking to scale up the problem size that can be handled (Ranjan et al., 1995). Using a simple backward chaining algorithm we demonstrate that this is also the case for OBDD-based planning and show that UMOP is an efficient deterministic planning system.

In order to explain the OBDD encoding used by UMOP, we briefly describe its front end language *NADL*.[2] *NADL* is a description language for multi-agent planning domains allowing non-deterministic actions. In this paper, though, only classical deterministic single-agent domains are modeled.

The paper is organized as follows. First, we give a brief overview of OBDDs which may be skipped by readers already familiar with the subject. Second, the non-deterministic planning language, *NADL* , and its OBDD encoding is described. Third, we introduce the backward chaining algorithm used by UMOP for deterministic domains, and finally we present empirical results from the AIPS'98 planning competition domains and draw conclusion.

[1] UMOP stands for Universal Multi-agent OBDD-based Planner.

[2] *NADL* stands for Non-deterministic Agent Domain Language.

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 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*.

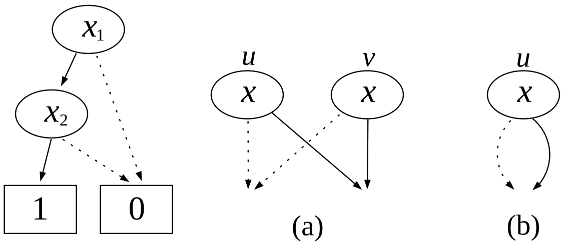

Figure 1: An OBDD representing the function $f(x_1, x_2) = x_1 \wedge x_2$. True and false edges are drawn solid and dotted, respectively. (a) and (b) Reductions of OBDDs.

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. We then describe its formal syntax and semantics.

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.

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 in 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 in 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 environment and system 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. The ordering of the boolean variables, we use, puts the input variables first followed by an interleaving of the boolean variables of current state and next state variables:

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

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 where each OBDD express the value of a digit in 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 algorithm we use for generating plans 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 (Ranjan et al., 1995).

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 deterministic planning algorithm presented in the next section generates a plan from these state-action pairs.

NADL has been carefully designed to allow a partitioned transition relation representation. Thus, the relations A, F and I all consist of a conjunction of subexpressions that for most planning problems only refer to a small subset of next state variables.

The Deterministic Planning Algorithm

To translate a classical deterministic STRIPS domain (Fikes & Nilsson, 1971) to an *NADL* description all the actions in the STRIPS domain are defined to be the actions of a single system agent. Since only one system agent is defined no concurrent actions exist. Further, since no environment agents exist and the actions of the system agent are deterministic the transition system describing the planning domain is deterministic.

The deterministic planning algorithm is derived from an OBDD-based universal planning algorithm developed by Cimatti et al. (1998). The algorithm first generates a set of state-action pairs relevant for reaching a goal state. This set is produced by an iteration of preimage calculations starting at the set of goal states and ending when the initial state is covered. This backward search is illustrated in Figure 2. Next, a sequential plan is generated from the set of state-action pairs by starting at the initial state and iteratively adding an action from the state-action set until a goal state is reached. The plan steps of the plan extracted for the example illustrated in Figure 2 is indicated by thick arrows.

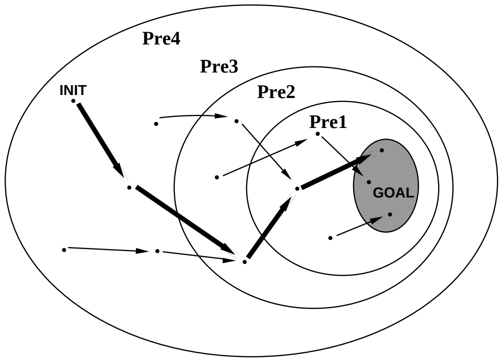

Figure 2: Generation of state-action pairs by a backward iteration of preimage calculations starting at the set of goal states. Arrows denote state-action pairs. Thick arrows are plan steps in the final plan. Solid ellipses denote preimages.

All operations in the planning algorithm are carried out on OBDDs. Thus large sets of states and state-action pairs can be represented and efficiently manipulated. The planning algorithm is shown below:

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

Figure 3: The deterministic planning algorithm. The *Prune* function removes states from the preimage that are already visited. The *ExtractPlan* function finds the sequence of plan steps in the final plan by starting in the initial state and adding state-action pairs until a goal state is reached.

Empirical Results

In the following three subsections we present results obtained in three of the AIPS'98 planning competition domains.[3] *NADL* descriptions of the domains are posted at `http://www.cs.cmu.edu/~runej`.

Five planning systems BLACKBOX, IPP, STAN, HSP and SGP participated in the competition. Only the first four of these planners competed in the three examined domains. BLACKBOX is based on SATPLAN, while IPP and STAN are

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

graphplan-based planners. HSP uses a heuristic search approach based on a preprocessing of the domain. The AIPS'98 planners were run on 233/400 MHz[4] Pentium PCs with 128 MB RAM equipped with Linux.

The Gripper Domain

The gripper domain consists of two rooms A and B, a robot with a left and right gripper and a number of balls that can be moved by the robot. The task is to move all the balls from room A to room B, with the robot initially in room A. The state variables of the *NADL* encoding of the domain are the position of the robot and the position of the balls. The position of the robot is either 0 (room A) or 1 (room B), while the position of a ball can be 0 (room A), 1 (room B), 2 (in left gripper) or 3 (in right gripper). For the AIPS'98 gripper problems the number of plan steps in an optimal plan grows linearly with the problem number. Problem 1 contains 4 balls, and the number of balls grows by two for each problem. The result of the experiment is shown in Figure 4 together with the results of the planners in the AIPS'98 competition. UMOP generates minimum-

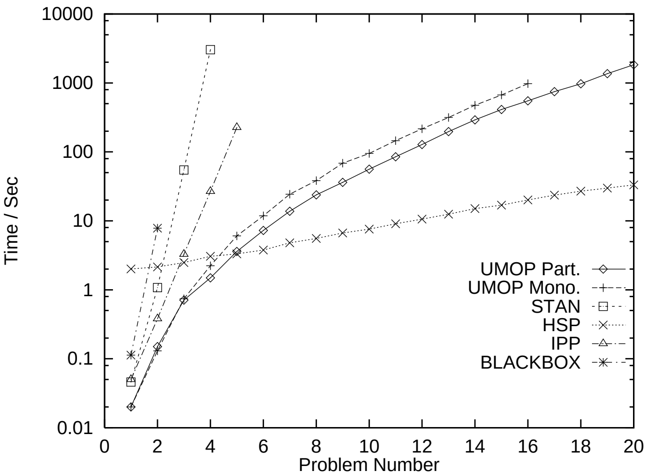

Figure 4: Execution time for UMOP and the AIPS'98 competition planners for the gripper domain problems. UMOP Part. and UMOP Mono. show the execution time for UMOP using a partitioned and a monolithic transition relation respectively. 1 to 5 partitions were used.

length plans due to its parallel breadth first search algorithm. As depicted in Figure 4, it avoids the exponential growth of the execution time that characterizes all of the competition planners except HSP. When using a partitioned transition relation UMOP is the only planner capable of generating minimum-length plans for all the problems. For this domain the transition relation of an *NADL* description can be divided into $n+1$ basic partitions, where n is the number of balls. The optimal number of partitions is not necessarily the largest number of partitions. For the results in Figure 4 each partition equaled a conjunction of 10 basic partitions. Compared to the monolithic transition relation representation the results obtained with the partitioned transition relation was significantly better on the larger problems. The memory usage for problem 20 with a partitioned transition relation was 87 MB, while it, for the monolithic transition relation, exceeded the limit of 128 MB at problem 17.

[4] Unfortunately no exact record has been kept on the machines and there is some disagreement about their clock frequency. According to Drew McDermott, who chaired the competition, they were 233 MHz Pentiums, but Derek Long (STAN) believes, they were at least 400 MHz Pentiums, as STAN performed worse on a 300 MHz Pentium than in the competition.

The Movie Domain

In the movie domain the task is to get chips, dip, pop, cheese and crackers, rewind a movie and set the counter to zero. The only interference between the subgoals is that the movie must be rewound, before the counter can be set to zero. The problems in the movie domain only differs by the number of objects of each type of food. The number of objects increases linearly from 5 for problem 1 to 34 for problem 30.

Problem	umop	stan	hsp	ipp	blackbox
1	14	19	2121	10	11
2	12	18	2104	10	12
3	14	19	2144	10	14
4	4	20	2188	10	16
5	14	21	2208	10	18
6	16	22	2617	10	20
7	14	22	2316	20	22
8	12	23	2315	20	24
9	14	25	2357	-	26
10	14	26	2511	10	29
11	14	27	2427	30	30
12	4	28	2456	30	32
13	16	29	3070	20	36
14	14	31	2573	30	35
15	16	32	2577	30	38
16	14	34	2699	10	39
17	16	35	2645	30	41
18	14	37	2686	10	43
19	16	39	2727	30	45
20	12	40	2787	20	47
21	16	42	2834	20	49
22	14	45	2834	20	51
23	16	48	2866	20	53
24	14	50	3341	20	55
25	16	52	2997	30	57
26	16	54	3013	40	58
27	16	57	3253	50	60
28	4	62	3049	40	63
29	18	64	3384	50	64
30	16	67	3127	40	66

Table 1: Movie domain results. For each planner the run time in milliseconds is shown. All planners found a plan with minimal length. "-" indicates the planner failed. UMOP used far less than 128 MB for any problem in this domain.

The *NADL* description of the movie domain represents each type of food as a numerical state variable with a range equal to the number of objects of that type of food. Table 1 shows the execution time for UMOP and the competition planners for the movie domain problems. In this experiment UMOP used its default partitioning of the transition relation. For every problem all the planners find the optimal plan. Like the competition planners UMOP has a low computation time, but it is the only planner not showing any increase in computation time even though, the size of the state space of its encoding increases from 2^{24} to 2^{39}.

The Logistics Domain

The logistics domain consists of cities, trucks, airplanes and packages. The task is to move packages to specific locations. Problems differ by the number of packages, cities, airplanes and trucks. The logistics domain is hard and only problem 1,2,5,7 and 11 of the 30 problems were solved by any planner in the AIPS'98 competition (see Table 2). The *NADL* description of the logistics domain uses numerical state variables to represent locations of packages, where trucks and airplanes are treated as special locations. Even though, the state space for the small problems is moderate, UMOP fails to solve any of the problems in the domain. It succeeds to generate the transition relation but fails to finish the preimage calculations. The reason for this might be a bad representation or variable ordering. It might also be that no compact OBDD representation exists for this domain in the same way, that no compact OBDD representation exists for the integer multiplier (Bryant, 1986). More research is needed to decide this.

Prob.	stan		hsp		ipp		blackbox	
1	767	27	79682	43	900	26	2062	27
2	4319	32	97114	44	-	-	6436	32
5	364932	29	144413	26	2400	24	-	-
7	-	-	788914	112	-	-	-	-
11	12806	34	86195	30	6940	33	6544	32

Table 2: Logistics domain results. For each planner column one and two show the run time in milliseconds and the plan length. (- -) means the planner was unable to find a solution.

Conclusion

In this paper we have presented results obtained with UMOP in classical deterministic domains. The results show that UMOP is an efficient deterministic planning system and that the partitioned transition relation used by UMOP also can be exploited in deterministic domains.

We believe a more efficient OBDD-based deterministic planning system may be designed by using a compact, partitioned encoding of the domain based on type and domain knowledge analysis (Long & Fox, 1998; Etzioni, 1990) and using the GRAPHPLAN relaxation when searching for a solution. We are currently developing such a system.

Acknowledgments

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.

References

Blum, A., & Furst, M. L. (1995). Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, pp. 1636–1642. Morgan Kaufmann.

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. (1998). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of AAAI'98*, pp. 875–881. AAAI Press/The MIT Press.

Di Manzo, M., Giunchiglia, E., & Ruffino, S. (1998). Planning via model checking in deterministic domains: Preliminary report. In *Proceedings of the 8'th International Conference on Artificial Intelligence: Methodology, Systems and Applications (AIMSA'98)*, pp. 221–229. Springer-Verlag.

Edelkamp, S., & Reffel, F. (1999). Deterministic state space planning with BDDs. In *Proceedings of ECP-99*, pp. 381–382. Springer.

Etzioni, O. (1990). *A structural theory of explanation-based learning*. Ph.D. thesis, Computer Science Department, Carnegie Mellon University.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Jensen, R. M., & Veloso, M. M. (1999). OBDD-based universal planning: Specifying and solving planning problems for synchronized agents in non-determinitic domains. In Wooldrige, M. J., & Veloso, M. (Eds.), *Artificial Intelligence Today, Recent Trends and Developments*. Springer-Verlag.

Kautz, H., & Selman, B. (1996). Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the 13'th National Conference on Artificial Intelligence (AAAI'96)*, Vol. 2, pp. 1194–1201. AAAI Press/MIT Press.

Long, D., & Fox, M. (1998). The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, *9*, 367–421.

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

Ranjan, R. K., Aziz, A., Brayton, R. K., Plessier, B., & Pixley, C. (1995). Efficient BDD algorithms for FSM synthesis and verification. In *IEEE/ACM Proceedings International Workshop on Logic Synthesis*.

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.