Efficient BDD-based Search for Planning

Thesis Proposal

Rune M. Jensen
School of Computer Science
Carnegie Mellon University

April 23, 2001

Committee Members
M. M. Veloso, Carnegie Mellon University (Co-chair)
R. E. Bryant, Carnegie Mellon University (Co-chair)
R. Simmons, Carnegie Mellon University
P. Traverso, Istituto Trentino Di Cultura, Italy

Abstract

In this thesis we propose to develop specialized search algorithms and domain encodings based on reduced ordered binary decision diagrams (BDDs) for deterministic and non-deterministic planning problems. BDDs are compact representations of Boolean functions that have been successfully applied in model checking to implicitly represent and traverse very large state spaces. Recent research has shown that BDD-based search can be one of the most efficient known approaches for optimal deterministic planning and that BDDs are very suitable for representing universal plans for non-deterministic problems. However, when optimality constraints are relaxed, classic heuristic state-space planners currently outperform BDD approaches.

A BDD-based heuristic search algorithm has been suggested but does not provide a satisfying solution on how to represent heuristic functions. An open question remains as whether there exist efficient BDD-based search algorithms that can trade optimality or even completeness for speed and scalability without severely compromising the quality of the generated plans. The proposed research aims at developing such new algorithms and determine applicability conditions for both deterministic and non-deterministic planning problems. The thesis work will investigate features of BDD-based search applied to effectively scale model-checking problems and apply them to planning problems.

Our proposed approach is to use directed search with efficient heuristic encodings combined with 1) two-level abstraction of the search space derived from GRAPHPLAN's parallel relaxation, 2) transition relation partitioning suitable for deterministic and non-deterministic planning domains, and 3) compact domain encoding of STRIPS and PDDL domains. Furthermore, within non-deterministic planning domains, we propose to investigate the generation of universal plans that are k-fault tolerant, in the sense that some goal can be reached with at most k rare events causing non-determinism. The thesis work will include the algorithmic development and experimental analysis in existing and new domains of relevance to the planning community.

1 Introduction

Planning involves search in large state spaces for the automatic generation of actions to achieve specific goal states. Planning is hard (in general PSPACE-complete [11]) and efficient approaches that extend beyond the most restricted domain languages like STRIPS [28] and PDDL [48] are lacking.

Reduced ordered binary decision diagrams (BDDs) can contribute to both of these problems. A BDD [7] is a compact and robust data structure for Boolean functions that can be used to symbolically represent state spaces. Through their canonical representation BDDs have been effectively applied to the synthesis and verification of hardware circuits [8] and incorporated within the area of *symbolic model checking* [49]. Due to the implicit state space representation BDD-based search often scales to very large state spaces. In addition, the BDD-based symbolic state space representation and search algorithms naturally extend to non-deterministic domains and have furthermore been shown to compactly represent universal plans in these domains [15].

In this thesis we want to address the development of efficient BDD-based search algorithms specialized to deterministic and non-deterministic planning domains. Several questions need to be answered in order to achieve this goal. In deterministic planning recent research [25, 23] (see also Section 4.3) indicates that BDD-based search is one of the best known approaches for optimal deterministic planning.[1] However, it is well known in planning that there is a trade off between solution quality and search speed. As shown by heuristic state-space planners [37, 4] and graph planners [1, 44] solutions to deterministic planning problems can be found very fast when optimality constraints are relaxed. A key question is, if such algorithms can be developed for BDD approaches as well. A BDD-based heuristic search algorithm (BDDA*) has been suggested, but it fails provide an efficient

[1] We define an optimal plan of a deterministic planning problem to be a solution with shortest length.

representation of informative heuristics [21]. An important goal of the thesis is therefore to investigate if other directed BDD-based search algorithms exist that use a less complex representation of the heuristic function. Abstraction is an alternative way to relax a search problem. There is no published work on abstraction for BDD-based planning, but our studies on parallel relaxed transition relations for two-level abstraction in BDD-based search show a potential speed up of several magnitudes without a severe degradation of the solution quality (see Section 4.5). The thesis will continue this research and investigate to what extent the approach can be made generally applicable. In addition, we will study ways of combining directed search methods with abstraction.

Heuristic search and abstraction apply to deterministic planning where only one path to a goal state needs to be found. In non-deterministic domains however, the search problem is different since actions may have multiple possible outcomes. Thus, a plan in a non-deterministic domain has to cover all the states that can be visited and this may prevent an effective use of relaxation. The approach, we will focus on for non-deterministic domains, is universal planning [56]. In BDD-based universal planning [15, 14, 40, 39] a plan is generated incrementally from a backwards breadth-first search algorithm. It is unclear to what extent directed search and abstraction can be applied to this approach without significantly reducing the quality of the generated plan. One way to do this, that we want to investigate, is to consider k-fault tolerant algorithms, where some goal can be reached with at most k rare events causing the non-determinism. In addition, we want to consider how the existing algorithms can be made more efficient for example by partitioning the transition relation representation of the search space [40] and/or reducing the search space by constraining it to only reachable states. Currently BDDs have been shown to be an efficient representation for universal plans [15], but previous research has focused on developing new kinds of universal plan solutions rather than efficient search techniques.

A final question, we plan to address, is how to make efficient Boolean state encodings of STRIPS and PDDL domains where a state is represented by a set of grounded predicates. These encodings are typically very redundant and often exponentially smaller Boolean encodings exist [22]. As discussed in Section 2.2 the complexity of the BDD representation is highly dependent on the size and variable ordering of the Boolean encoding. In the thesis we will continue previous work on extracting structural information from planning domains in order to build compact Boolean encodings. Moreover, we want to study if good variable orderings can be found by static domain analysis.

In summary, the goal of the proposed research is to develop specialized BDD-based search algorithms for deterministic and non-deterministic planning problems that in contrast to current algorithms can trade optimality

or even completeness for speed and scalability without severely compromising the quality of the generated plans. The proposed research aims at developing such algorithms using directed search methods combined with abstraction, partitioning and compact domain encoding.

The remainder of this proposal is organized as follows. First we give a brief introduction to BDD-based planning in Section 2. It may be skipped by readers already familiar with the topic. Then in Section 3 we describe related work on deterministic and non-deterministic planning. In Section 4 we give a detailed description of current issues in BDD-based planning including some of our unpublished work. Finally, in Section 5, we give a summary of the proposed thesis work.

2 Background

2.1 Planning

Planning is one of the oldest and most formalized subfields of AI. A planning domain is a discrete representation of some target world, where actions are modelled as transitions between states. The states and actions of a domain are defined by a *domain theory* given in some planning language.

The STRIPS planning language has been widely used to represent deterministic planning problems. In STRIPS instantiated predicates are *facts*. A *state* in a planning problem is a set of facts. An *operator* is a triple (P, A, D) of *preconditions*, *add-effects* and *del-effects*, with P, A and D being sets of facts. An operator is applicable if all preconditions are fulfilled. The state is altered by adding the corresponding add-effects and removing the del-effects. Operators or actions are defined by *schemas* that contain free variables, to be instantiated by the facts given in a state.

A disadvantage of STRIPS is that it can not represent non-deterministic domains. For this reason, we will assume that domains are described in the Non-deterministic Agent Domain Language (*NADL*) that extends to both non-deterministic and multi-agent domains [40].

An *NADL* domain description consists of: a definition of *state variables*, a description of *system* and *environment agents*, and a specification of *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*. At each step, all of the agents perform exactly one action, and the resulting action tuple is a *joint action*.[2] The system agents model the behavior of the

[2]In the remainder of this presentation we will simply refer to joint actions as actions.

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 of 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. As an example consider the *NADL* domain shown in Figure 1. This domain has two agents. One system agent, *Lift* , with two actions *Up* and *Down* and one environment agent, *Env*, with one action *Brake*. There is a single numerical state variable *pos* and Boolean state variable *lift_works*. *Up* and *Down* constrain *Pos* in the next state but are conditioned by *lift_works*. *Brake* constrains *lift_works* and is a non-deterministic action. If *lift_works* is true in the current state *lift_works* can either be true or false in the next state. The semantics of an *NADL* description is a non-deterministic finite transition system with initial and goal states. We will call this a *domain*. A domain is a 5-tuple $D = (S, A, T, I, G)$ where:

1. S is the finite set of states given by the *NADL* state variables.
2. A is the finite set of joint actions of system agents .
3. $T \subseteq S \times A \times S$ is a transition relation, such that $T(s, a, s')$ iff a is applicable in s and causes a transition to s'.
4. $I \subseteq S$ is the set of initial states.
5. $G \subseteq S$ is the set of goal states.

The domain of the *NADL* example in Figure 1 is illustrated in Figure 2.

As shown in Figure 6 the transition relation $T(s, a, s')$ can be represented by a BDD. For a detailed discussion on how to represent *NADL* domains with BDDs, we refer the reader to Jensen and Veloso (2000).

An applicable action in a deterministic domain has exactly one possible outcome. Thus, if $T(s, a, s')$ holds for a state s, an action a and a next state $s' \in S'$ then S' is a singleton set. In non-deterministic domains, actions can have several possible outcomes. Thus, S' may be an arbitrary nonempty subset of S. A *path* from s_0 to s_n in a domain D is a sequence of states $s_0 s_1 \cdots s_n$ such that $\exists a \in A \,.\, T(a, s_j, s_{j+1})$ forall $0 \leq j < n$. The *length* of a path $s_0 s_1 \cdots s_n$ is the number of transitions, n. A sequential plan is a sequence of actions $a_1 a_2 \cdots a_n$. We say that a sequential plan $a_1 a_2 \cdots a_n$ starting at s_0

variables
  nat(4) *pos* **bool** *lift_works*
system
  agt: Lift
    Up
      con: *pos*
      pre: $pos < 3$
      eff: $lift\_works \rightarrow pos' = pos + 1, pos' = pos$
    Down
      con: *pos*
      pre: $pos > 0$
      eff: $lift\_works \rightarrow pos' = pos - 1, pos' = pos$
environment
  agt: Env
    Brake
      con: $lift\_works$
      pre: $true$
      eff: $\neg lift\_works \Rightarrow \neg lift\_works'$
initially
  $pos = 0 \wedge lift\_works$
goal
  $pos = 3$

Figure 1: An *NADL* domain example with two agents. One system agent, *Lift* , with two actions *Up* and *Down* and one environment agent, *Env*, with one action *Brake*.

can generate a path $s_0 s_1 \cdots s_n$ if $T(a_{j+1}, s_j, s_{j+1})$ holds for all $0 \leq j < n$. The length of a plan is simply the number of actions in the plan. Given these basic definitions we can now define a **deterministic planning problem**:

Input: A domain $D = (S, A, T, I, G)$.

Output: A sequential plan that can generate a path from an initial state $i \in I$ to a goal state $g \in G$.

We define an *optimal deterministic plan* to be a sequential plan solution with shortest

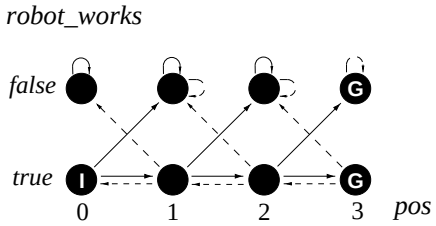

Figure 2: The domain representation of the *NADL* example. Solid and dashed arrows denote the action *Up* and *Down* respectively. States marked with "I" and "G" are initial and goal states.

length.

In general solutions to non-deterministic planning problems have to be more expressive than sequential plans since actions may lead to several next states. In this introduction we assume that solutions are given as *universal plans* [56]. A universal plan U for a domain D is a partial mapping from states to sets of actions $U : S \to 2^A$. A universal plan is executed by iteratively observing the current state and executing an action in the plan associated to the state. Thus, a universal plan U starting at s_0 can generate a path $s_0 s_1 \cdots s_n$ if $\exists a \in U(s_j) \,.\, T(a, s_j, s_{j+1})$ for all $0 \leq j < n$. The optimality of a universal plan depends on the definition of the non-deterministic planning problem. To illustrate this, we define the **optimistic non-deterministic planning problem**:

Input: A domain $D = (S, A, T, I, G)$.

Output: A universal plan $U : S \to 2^A$ such that the initial states are covered by U, $I \subseteq \mathrm{dom}(U)$, and for all states in the plan, $s \in \mathrm{dom}(U)$, U can generate a path from s to a goal state $s_g \in G$.

An *optimal optimistic solution* is an optimistic universal plan mapping each state s to the largest set of actions such that each action has a possible next state being the second state on a shortest length path from s to a goal state.

2.2 BDD-based Planning

An ordered reduced binary decision diagram (BDD) is a canonical representation of a Boolean function with n linear ordered arguments $x_1, x_2, ..., x_n$. A BDD 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 $low(u)$. Each variable node is associated with a Boolean variable in the function the BDD represents. The graph is ordered in the sense that all paths in the graph respect the ordering of the variables.

A BDD representing the function $f(x_1, x_2) = x_1 \wedge x_2$ is shown in Figure 3. 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* or *False* if the label of the reached terminal node is 1 or 0, respectively. A BBD graph is reduced so that

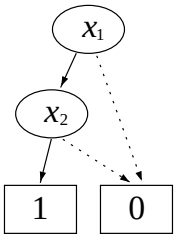

Figure 3: A BDD representing the function $f(x_1, x_2) = x_1 \wedge x_2$. High and low edges are drawn as solid and dotted lines, respectively.

no two distinct nodes u and v have the same variable name and low and high successors (Figure 4a), and no variable node u has identical low and high successors (Figure 4b). The BDD representation has two major advantages: First, due to the reductions, it is a

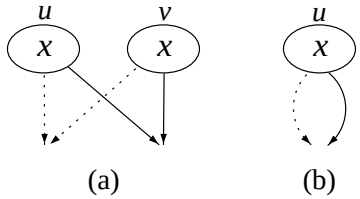

Figure 4: Reductions of BDDs. (a) nodes associated to the same variable with equal low and high successors will be converted to a single node. (b) nodes causing redundant tests on a variable are eliminated.

canonical representation of Boolean functions that can be exponentially more compact than their corresponding truth table representation [8]. Second, any operation on two BDDs, corresponding to a Boolean operation on the functions they represent, has a low complexity bounded by the product of their node size [8]. A disadvantage of BDDs is that

the size of a BDD representing some function is highly dependent on the ordering of the variables. To find an optimal variable ordering is a co-NP-complete problem in itself [8], but as illustrated in Figure 5 a good heuristic for choosing an ordering is to locate related variables near each other [16].

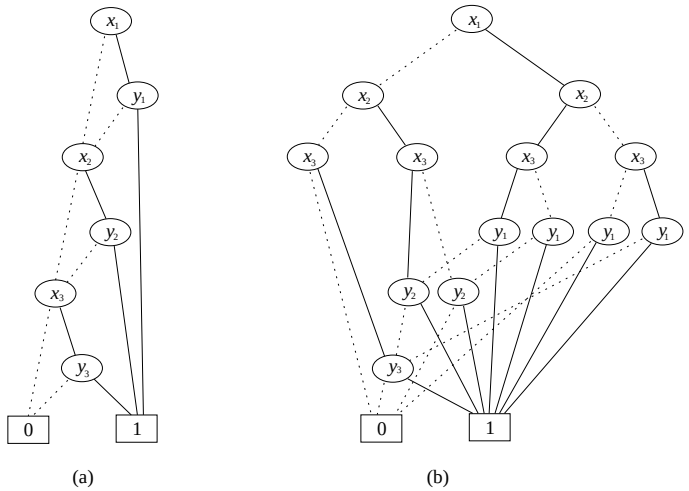

Figure 5: This Figure shows the effect of variable ordering for the expression $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee (x_3 \wedge y_3)$. The BDD in (a) only grows linearly with the number of variables in the expression, while the BDD in (b) grows exponentially. The example illustrates that placing related variables near to each other in the ordering often is a good heuristic.

BDDs can represent planning domains implicitly. As an example, consider the non-deterministic planning domain shown in Figure 6a. In this domain there are four states given by the four possible value assignments of the two Boolean state variables x_1 and x_2. The actions in the domain are defined by the Boolean variable a. The BDD representing the transition relation $T(a, x_1, x_1', x_2, x_2')$ of the domain is shown in Figure 6b. The definition of T is straightforward: for some assignment of its arguments, T is true iff action a causes a transition from the current state given by the value of x_1 and x_2 to the next state given by the value of x_1' and x_2'. Note that the BDD representing T for the example turns out not to depend on x_2'.

Assume that the state 01 is a goal state G. A key operation for solving planning problems for reaching G is to find all the state action pairs (s, a) such that G can be reached from s by executing a. This set is labeled P_1 in Figure 6c. To find P_1 from T we constrain x_1' to *False* and x_2' to *True* in T. This reduces T to the BDD shown in Figure 6d. The resulting BDD represents P_1 with the states described in the current state variables

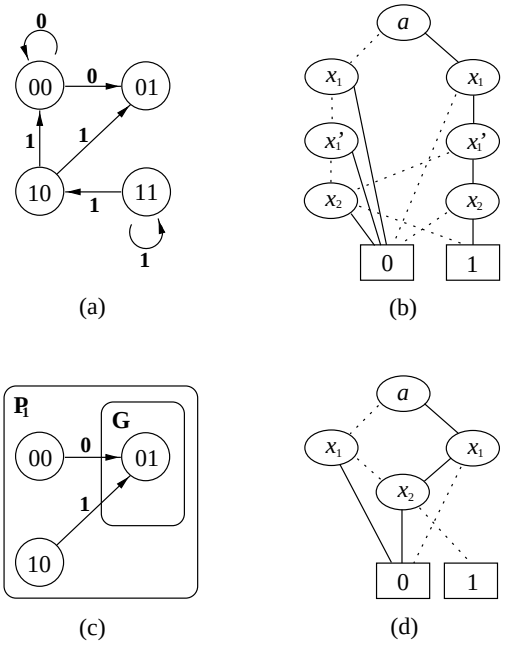

Figure 6: A non-deterministic planning domain is shown in (a). States are defined by Boolean state variables x_1 and x_2, and the two actions in the domain represented by the Boolean variable a. The symbolic representation of the transition relation is shown in (b). In (c), P_1 is the set of state action pairs for which, execution of the action can lead to the goal. The symbolic representation of P_1 is shown in (d). It is obtained from the transition relation by restricting the next state to 01.

x_1 and x_2. Logically we performed the computation $\exists x_1', x_2' \,.\, \neg x_1' \wedge x_2' \wedge T$ to obtain the BDD representing P_1.

We end this section by showing a simple backward BDD-based search algorithm for solving non-deterministic planning problems (see Figure 7). Let the *preimage* of a set of states V of a domain D be a partial mapping, $PreImage(T, V) : S \rightarrow 2^A$, where $a \in PreImage(T, V)(s)$ iff $T(a, s, s')$ and $s' \in V$. That is, a can cause a transition from s into a state s' in V. In a similar way, we can define the *image* of a set of states that is used for forward search algorithms. The algorithm generates an optimal optimistic universal plan U for reaching a set of goal states G from a set of initial states I by performing a backward breadth-first search from the goal states to the initial states. In each step the preimage is computed of the visited states, V. All states previously visited are removed from the preimage by the function *FrontierSet* and the result is stored in the optimistic universal plan component U_c. If U_c is empty a fixed point of V has been

```
function Plan(T, I, G)
    U := ∅; V := G
    while I ⊄ V
        P := PreImage(T, V)
        U_c := FrontierSet(P, V)
        if U_c = ∅ then return failure
        else
            U := U ∪ U_c
            V := V ∪ states(U_c)
    return U
```

Figure 7: Optimistic BDD-based planning algorithm.

reached that does not cover the initial states. Since this means that no universal plan can be generated that covers the initial states, the algorithm returns *failure*. Otherwise, U_c is added to the universal plan and the states in U_c are added to the set of visited states. All variables in the algorithm are sets. In particular, the universal plan and the preimage are represented by a set of state action pairs (s, a). These sets are implicitly manipulated by BDDs operating on their characteristic function. For the *NADL* example domain the generated universal plan maps every state to the action *Lift*.

If the domain is deterministic a sequential plan can be extracted from the universal plan by a forward traversal of the universal plan from an initial state to a goal state by adding a single action from the universal plan to the sequential plan for each state visited.

3 Related Work

Major related work is shown in Figure 8 and is divided into four groups according to its use of BDDs and the type of target domain.

Previous work on deterministic planning, not using BDDs, include state space planners (e.g. PRODIGY [51]), plan space planners (e.g. UCPOP [58]) and hierarchical planners (e.g. SIPE [59]). In PRODIGY a plan is generated from a single-state bidirectional search guided by control rules. UCPOP, on the other hand, carries out a search in the space of possible plans guided by the least commitment principle where orderings between actions only are introduced if the actions are causally linked or interfere. SIPE uses hierarchical task networks (HTNs) to apply abstraction in the search. First, a solution is found at an abstract level which then is refined to a concrete plan. A problem for the above ap-

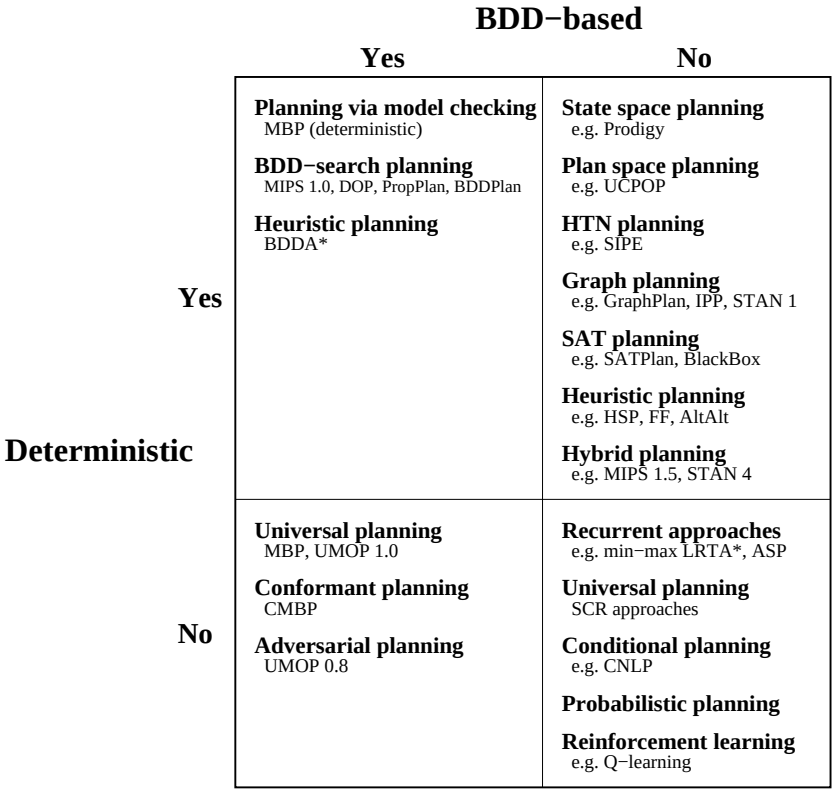

Figure 8: Related work on deterministic or non-deterministic planning.

proaches is that their explicit state representation makes it hard for them to scale to large domains. GraphPlan [1] avoids the state explosion problem by representing reachable states implicitly. Graph planners [1, 47, 44] use a two step approach. In the first step a planning graph is generated. The planning graph consists of alternating action and state layers and keep track of the interferences between actions and states resulting in a compact representation of the reachable states. In the second step a plan is extracted from the planning graph by a backwards search. Graph planners relax optimality constraints by only finding parallel optimal plans, i.e., plans with shortest length assuming that actions can be applied concurrently in each step. More recently this relaxation has been used by SAT planners and heuristic planners. SAT planners, like SATPlan [42], encode a planning problem as a satisfiability problem of a Boolean expression stating goal achievement within a certain number of steps. Using binary search this approach can be optimal but better results have been obtained using GraphPlan's parallel relaxation by

encoding goal achievement of the planning graph as a SAT problem (BLACKBOX,[43]). The problem with planning as satisfiability is that even for small problems the number of Boolean variables in the expression is high. Heuristic planners like HSP, FF and ALTALT [4, 61, 37] have returned to state-space planning and avoid to traverse a large number of states by heuristic directed search. The current success of the approach relies on efficient but non-admissible heuristics for deterministic planning problems that are closely related to the number of steps in the planning graph. However, as discussed in Section 4.3, heuristic single-state search is currently not a suitable approach for optimal deterministic planning. Finally, a new trend in planning is to combine several approaches in one hybrid system. The hybrid planners include MIPS 1.5 [23] (combining heuristic search and BDD-based search) and STAN 4 [31] (combining graph planning and heuristic search).

The first application of BDDs for deterministic planning (deterministic MBP, [12]) was based on symbolic model checking where the plan solution corresponded to a counter example of the verified property. The approach was shown to be competitive with GRAPHPLAN and SATPLAN in several classical domains. More recent approaches are MIPS 1.0, DOP, BDDPLAN and PROPPLAN [25, 38, 29]. All of these planners rely on blind breadth-first search from model checking. MIPS 1.0 uses a specialized preprocessing of domains to find compact Boolean state encodings [22] whereas DOP (see Section 4.5) uses an analysis derived from the TIM preprocessing module [30]. Both planners apply bidirectional search from the initial and goal states. BDDPLAN and PROPPLAN are more simple BDD-based planners without domain preprocessing and therefore worse performance than MIPS 1.0 and DOP. As discussed in Section 4.3 blind BDD-based search is currently one of the most efficient approaches for optimal deterministic planning. However, when optimality constraints are relaxed, heuristic state-space planners like HSP and FF, outperform BDD-based blind search. A heuristic BDD-based search algorithm called BDDA* has been suggested [24]. It represents the priority queue of A* [36] as a BDD. The approach relies on a BDD representation of the heuristic function for each state. Since this representation becomes complex for the heuristics used by HSP and FF, the approach does not scale as well as the heuristic state-space planners [21].

Related work on non-deterministic planning, not relying on BDDs, include conditional [27, 52, 3], probabilistic [20, 18, 2] and universal planning [56, 41]. For example, the CNLP partial order conditional planner handles non-determinism by constructing a conditional plan that accounts for each possible situation or contingency that could arise [52]. Probabilistic planners try to maximize the probability of goal satisfaction, given conditional actions with probabilistic effects. Drummond and Bresina (1990) represent plans as a set of Situated Control Rules (SCRs) [19] mapping situations to actions. The planning algorithm begins by adding SCRs corresponding to the most probable execution path that achieves the goal. It then continues adding SCRs for less probable paths and may end

with a complete plan taking all possible paths into account. Universal planning was introduced by Schoppers (1987) who used decision trees to represent plans. A recent approach also represents plans as a set of Situated Control Rules [41]. This algorithm incrementally adds SCRs to a final plan in a way similar to Drummond and Bresina (1990). Reinforcement Learning (RL) [57] can be regarded as kind of universal planning. In RL the goal is represented by a reward function in a Markov Decision Process (MDP) model of the domain. Value iteration is applied to generate control policies maximizing the expected reward. 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 larger state spaces than RL. An alternative to RL and non-deterministic planning is to interleave planning with execution. This approach has been widely used in non-deterministic robotic domains e.g. [33, 32, 60, 35]. A group of planners suitable for this purpose is action selectors based on heuristic search [45, 5].

Several BDD-based planning approaches has been developed for non-deterministic domains. They can be divided into two groups according to the form of the synthesized plan. The first group keeps the plan on sequential form and may not guarantee goal achievement due to non-determinism [12] or only allow sequential plans that succeed despite of non-determinism (conformant planning, [13]). The second group generates universal plans that are partial mappings from domain states to relevant actions. Several algorithms for synthesizing universal plans with different properties have been developed. The strength of these plans range from *strong solutions*, that guarantee progress towards the goal in each step [15], to *strong cyclic* and *optimistic* (or *weak*) solutions that either may cycle forever or reach dead ends [14, 40]. The latter two solutions have further been extended to reason explicitly about the actions of an adversarial environment [39].

4 Issues in BDD-based Planning

In this section we describe the state of the art of BDD-based planning. The purpose of the presented techniques is to reduce the complexity of one or more of the four major steps involved in BDD-based planning:

1. Make a Boolean encoding of the domain states.
2. Build a symbolic BDD representation of the transition relation of this encoding.
3. Perform a state space search.
4. Extract a plan.

4.1 BDD package optimization

A BDD package is an implementation of efficient data structures and algorithms for computing basic operations on BDDs. Modern BDD packages typically share the following common implementation features based on [6, 54]: 1) a single shared BDD with several roots representing a set of BDDs, 2) a set of dynamic programming algorithms for carrying out operations on the BDDs that due to a large number of distinct subproblems use a cache instead of a memoization table, and 3) data structures that facilitate dynamic variable reordering and garbage collection of unreferenced BDD nodes that is invoked when the percentage of unreferenced BDD nodes exceeds a preset threshold.

The three major parameters are: the initial number of nodes allocated to the shared BDD, the cache size, and the type of dynamic variable reordering if any. Experiments indicate that these parameters should be adjusted differently for BDD-based model checking compared to circuit verification [62]. The experiments show that model checking computations have a large number of repeated subproblems across the top level operations. Thus, a large cache size is more important for model checking than for circuit verification. Furthermore, model checking computations can have a very high death and rebirth rate (unreferenced nodes being referenced again) compared to circuit computations. Thus, garbage collection should occur less frequently, which for example can be accomplished by initially allocating a large number of nodes for the shared BDD. Finally, dynamic reordering of variables is efficient given an initial bad variable ordering, but given a good initial variable ordering the time spend on reordering does not pay off.

Since the reachability analysis that forms the core of symbolic model checking resembles the state space search of BDD-based planning, we would expect these results to apply to deterministic and non-deterministic BDD-based planning as well. No systematic experiments have been carried out to confirm this, but our experiences with deterministic planning problems fit well with the hypothesis: 1) a planning problem initiated with a good variable order seems always to perform better without dynamic variable reordering, 2) each garbage collection seems to impair performance by deleting nodes that later must be recomputed, and 3) a too little cache can cause a performance degradation of several factors (a cache size that works well in practice is about 10 percent of the number of allocated nodes). To illustrate the latter Figure 9 shows the complexity of a subset of the Logistics problems in AIPS-00 using a cache with 10000 and 400000 nodes, respectively. In this experiment bidirectional search with frontier set simplification was used based on a monolithic transition relation representation with no action representation. Garbage collection was suppressed by allocating "enough" initial BDD nodes.[3]

[3]The experiments reported in this proposal are based on the UMOP planning framework and the BuDDy BDD package [46]. The experiments were carried out on a 500 MHz Pentium III PC with 512

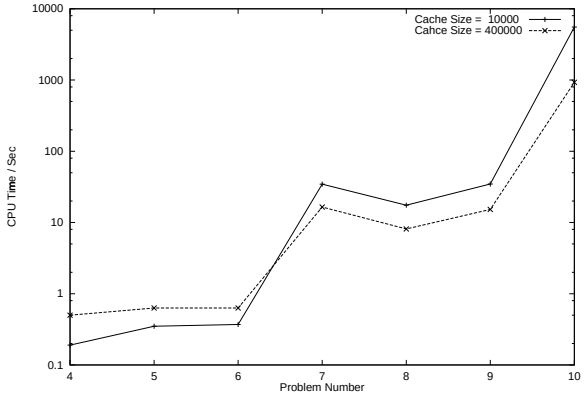

Figure 9: Impact of the cache size on AIPS-00 Logistics planning problems.

A detailed study of the reuse of BDD computations in planning problems compared to symbolic model checking and circuit verification may be crucial for developing efficient search algorithms tuned to the BDD packages. As discussed in Section 4.3 the typical exponential blow up in the size of a blind search fringe for planning domains may indicate a significant difference between planning and model checking domains.

4.2 Efficient Domain Encodings

Finding an efficient domain encoding consists of two completely different tasks. The first is to analyze the domain description in order to find domain structures that may lead to more efficient Boolean state representations than a direct translation. The second is to find a good ordering of the variables of the transition relation for this state representation.

For *NADL* domains the first task is trivial since the state variables are assumed to compactly represent the state space. However, for STRIPS and PDDL domains this task is difficult since the fact based state representation often hides important structure of the domain. Boolean state encoding for BDD-based PDDL planning has been studied by Edelkamp and Helmert (1999). Their results rely on previous work by [30, 34]. Since the size of the BDD representation of the transition relation and the symbolic state exploration are highly dependent on the number of Boolean state variables, the goal of their approach is to find state encodings with minimal Boolean description length. A naive encoding using a single Boolean state variable for each grounded predicate of the domain is only feasible for very small problem instances. Instead sets of balanced predicates used

MB RAM running Linux 5.2.

to encode state variables are identified. The approach typically leads to a logarithmic reduction in the number Boolean states variables since most planning problems consider moving objects between physical locations. Figure 10 shows the performance improvement of this approach for a subset of the AIPS-00 Blocks problems. In this experiment bidirectional search with frontier set simplification was used based on a disjunctive partitioned transition relation with no action representation. Again garbage collection was suppressed by allocating "enough" initial BDD nodes. One direction for future research

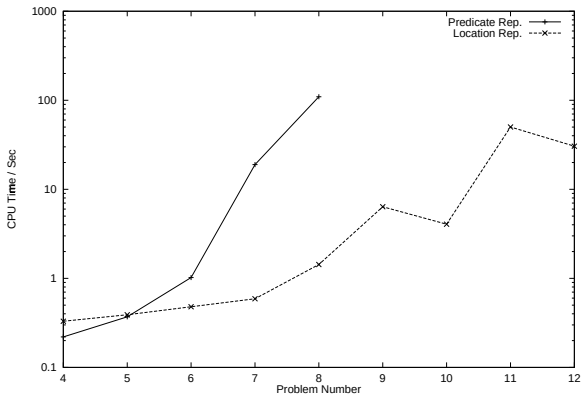

Figure 10: Results of a naive one-to-one encoding of grounded predicates of a subset of the AIPS-00 Blocks problems compared with an encoding of the location of each block.

is to extend this approach with knowledge substitution. For instance in the AIPS-98 Gripper domain a substitution of the $free(gripper)$ predicates with expressions on the ball locations improves performance substantially.

Given a minimal Boolean state representation, the second task is to find a good order of the variables in the transition relation. The BDD variables fall into 3 groups: variables to represent the current state, the action and the next state. Since the same ordering is shared by all BDDs finding a good ordering not only influence the size of the transition relation representation but also the size of BDDs used to represent sets of states expressed either in current state variables or next state variables. As discussed in Section 2.2 related variables should be near each other in the ordering. In practice it is a good heuristic to place the action variables first followed by the current and next state variables interleaved. However, no previous work has addressed how the current and next state variables should be ordered internally. Heuristics like the *fan in* and *weight* heuristic for constructing good variable orders of combinational circuits [50] still need to be developed for both deterministic and non-deterministic BDD-based planning.

4.3 Search algorithms

As discussed in next section the complexity of the first phase of BDD-based planning, where the transition relation is being constructed, can be lowered by partitioning. Unfortunately no equally efficient technique for reducing the complexity exists in the second phase, where the search is carried out.

In deterministic planning the first application of BDDs [12] was based on symbolic model checking where the plan solution corresponded to a counter example of the verified property. The approach was shown to be competitive with GRAPHPLAN and SATPLAN in several classical domains. Subsequent applications of blind search methods from model checking by MIPS 1.0, BDDPLAN, UMOP and PROPPLAN have lead to good results compared to other planners in the Gripper domain (AIPS-98) , Miconic 10 domain (AIPS-00) and to some extent the Logistics domain (AIPS-98). Furthermore, the optimal breadth-first search of these methods make them among the most efficient known today for optimal deterministic planning. In particular this approach seems to scale better than optimal directed single-state search as indicated by an experiment, we carried out, comparing UMOP and MIPS 1.0 with ALTALT and HSP 2.0 using admissible heuristics (all other parameters adjusted for best performance). The results are shown in Figure 11.

A disadvantage of BDD-based blind search is that it can not exploit that deterministic planning problems often have many solutions. The AIPS planning competitions have shown that one of these solutions typically can be found very fast using directed but possibly suboptimal single-state search methods. Blind search even seems to be particularly hard for planning domains. In the following experiment we measured the CPU time as a function of the size of Boolean state encoding for a range of planning problems (see Figure 12). A blind bidirectional search method with frontier set simplification was used and the number of initial BDD nodes, cache nodes and variable order was varied manually for best performance. We then made a profile of the size of BDD nodes used to represent the forward and backward search frontier for problems with 32 Boolean state variables (see Figure 13). The experiment shows two crucial problems with blind search for deterministic planning: 1) the fringe representation of most planning domains grows exponentially or worse, and 2) the growth order of the fringe is the main source of complexity since it predicts the general complexity of problems of a domain.

The size of the search fringe can be reduced by directed search approaches. A directed BDD-based search algorithm has been developed for deterministic planning [24, 21] and has been shown to improve performance considerably in the Logistics domain. The algorithm is called BDDA* and corresponds to A* search with the priority queue represented by a BDD. The approach relies on a BDD-based encoding of the heuristic function that may be fairly complex. It therefore seems likely that other informed search algorithms simplifying

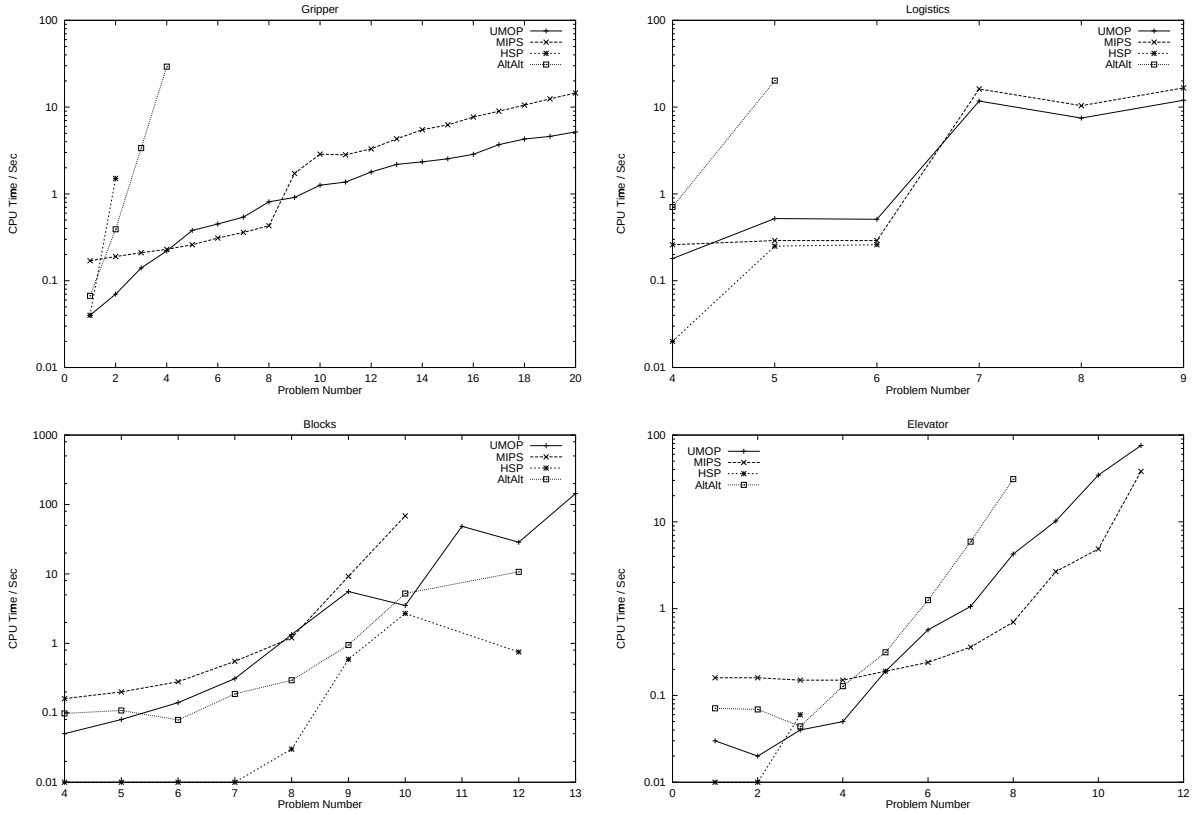

Figure 11: CPU time for UMOP and MIPS compared to ALTALT and HSP 2.0 in the Blocks, Elevator and Logistics domains of AIPS-00 and the Gripper domain of AIPS-98.

this problem may have better performance.

Frontier set simplification is another way to reduce the search fringe. The technique has origins in model checking [17]. The idea is to include previously visited states in the search fringe only if it reduces the size of the BDD representation. Unfortunately in deterministic and non-deterministic BDD-based planning the full power of frontier set simplification is hard to exploit since the plan extraction requires that no previous visited states are included in the search fringe. As shown by the above experiment pruning of previous visited states does not prevent a high fringe growth rate for most planning domains.

The growth rate of the search fringe usually depends highly on the search direction. Figure 14 shows the growth of the search fringe for forward and backward search in the

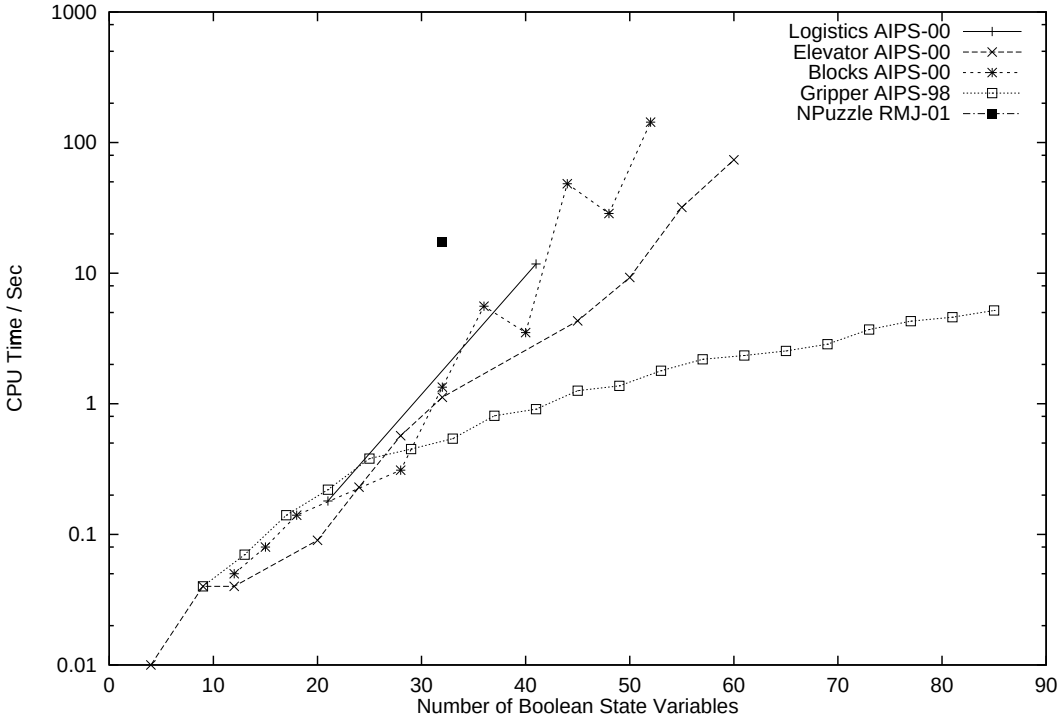

Figure 12: CPU time as a function of Boolean state encoding length for a range of planning domains.

AIPS-00 Logistics problem 6-0.[4] As for most other deterministic planning problems the backward search fringe grows faster than the forward. But for most problems the first couple of backward fringes are cheap to generate. Thus, a bidirectional search approach often pays off in deterministic domains. A good heuristic for choosing which direction to expand the search is simply the to measure the CPU time used for the last expansion in each direction and choose the direction with smallest previous CPU time [23]. An interesting direction for future research is to combine directed and bidirectional search for deterministic planning.

Unfortunately these techniques only apply to deterministic BDD-based planning. BDD-based universal planning for non-deterministic domains relies on a blind backward search to build a plan. No previous work has considered if more efficient algorithms exist. There

[4]It is common that the fringe size decreases towards the end of the search. The effect is due to the fact that BDDs representing either small or large fractions of a state space normally are smaller than BDDs representing fractions of the state space between these extremes.

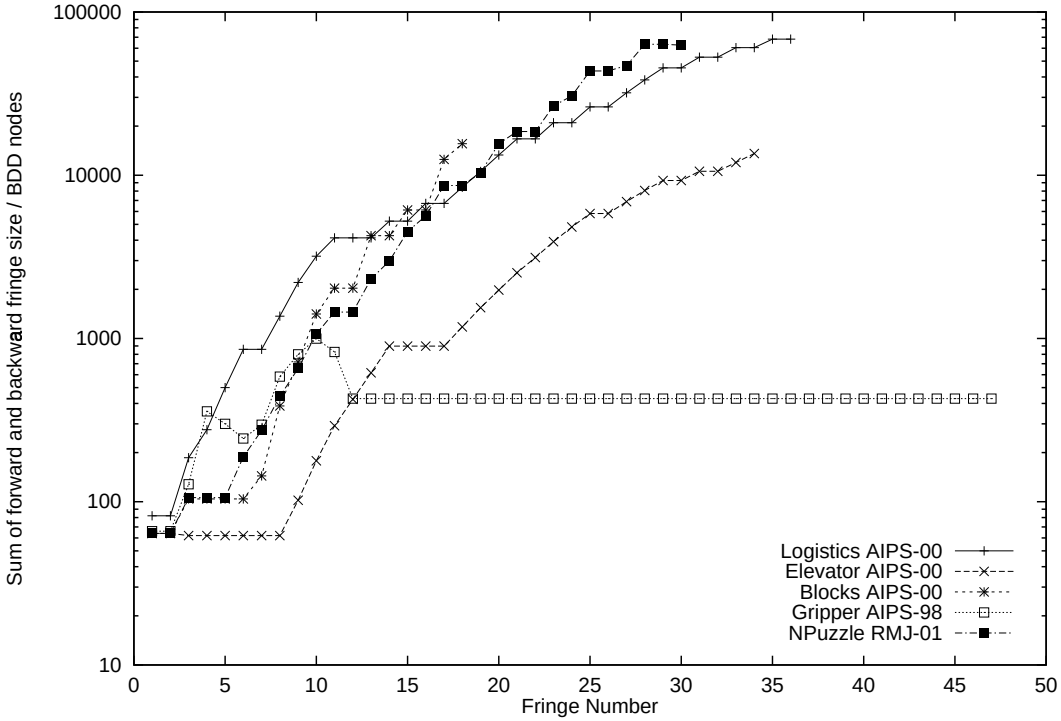

Figure 13: Fringe growth rate of different planning domains.

are good reasons to believe this since most planning domains are more efficiently traversed by forward search. A new algorithm could for instance start with a forward search and restrict the backward search only to reachable states. Furthermore, there might exist very efficient algorithms for domains with sparse non-determinism and algorithms trading optimality or even completeness for speed and scalability without severely compromising on the quality of the generated solutions. As an example consider domains with sparse non-determinism caused by infrequently occurring errors. Since errors are rare, solutions that can handle up to k of them (k-fault tolerant solutions) may be quite strong. An interesting question is whether such solutions can be generated by a directed forward search resembling the search used for deterministic domains.

4.4 Partitioning

Partitioning is another well known technique from model checking [10, 53]. The basic idea is to split the transition relation expression into a conjunction or disjunction of smaller

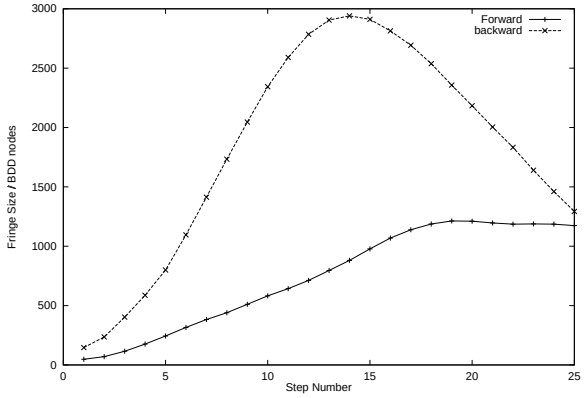

Figure 14: Fringe size profile for forward and backward search in the AIPS-00 Logistics problem 6-0.

partitions. The gain of doing this is twofold. First, it often reduces the complexity of computing the transition relation in the first phase of BDD-based planning, and second it may reduce the complexity of the fringe calculations in the second phase.

4.4.1 Conjunctive Partitioning

Assume that the transition relation $T(s, a, s')$ is represented by a Boolean function $T(s_v, a_v, s'_v)$ where s_v, a_v and s'_v are sets of Boolean variables used to represent the current state, the action and the next state. A conjunctive partitioning with n partitions is then n Boolean expressions $P_i(s_v, a_v, s'_v)$ where $0 < i \leq n$ such that:

$$T(s_v, a_v, s'_v) = \bigwedge_{i=1}^{n} P_i(s_v, a_v, s'_v)$$

Since we save $n - 1$ conjunctions for computing this representation the total size of the partitioned representation may be much smaller than the size of a monolithic representation and therefore less complex to compute. Another important reason for partitioning the transition relation is that it can reduce the complexity of computing the preimage (or image for that matter). The characteristic function of the preimage $PreImage(s_v, a_v)$ of a set of states $A(s'_v)$ is:

$$PreImage(s_v, a_v) = \exists s'_v \,.\, T(s_v, a_v, s'_v) \wedge A(s'_v)$$

Unfortunately the complexity of this computation is exponential in the number of quantified variables s'_v. To reduce the complexity we want to quantify variables as soon

as possible in the calculation when the intermediate BDDs still may be small. This can be accomplished with a conjunctive partitioning because a variable can be quantified early if no later partitions depend on it:

$$PreImage(s_v, a_v) = \exists s'_{v_n} \,.\, P_n \wedge \cdots \exists s'_{v_2} \,.\, P_2 \wedge \exists s'_{v_1} \,.\, P_1 \wedge A(s'_v)$$

where $s_{v_i} \notin \bigcup_{j=n}^{i+1} s'_{v_j}$ for $0 < i < n$. For a non-deterministic planning problem, described in *NADL*, it is straight forward to define conjunctive partitioning that works well in practice [40]. This is not the case for deterministic planning. The reason is that we want to discard the actions in the search phase and only consider them when extracting a sequential plan. We therefore need to find a partitioning without action representation that can be early quantified. Unfortunately this does not seem to be possible without adding extra information about which action executes. We have developed *bit partitions* where an extra Boolean variable per partition provides this information. In a bit partitioning each partition constrains a set of next state variables disjoint from the set of next state variables constrained by any other partition. Assume that an *NADL* domain has m actions only constraining a subset of the next state variables $V \subset S_v$. The expression of the bit partitioning for V is then:

$$b \wedge (A_1 \wedge \mathrm{IV}(v_1) \vee A_2 \wedge \mathrm{IV}(v_2) \vee \cdots \vee A_m \wedge \mathrm{IV}(v_m)) \quad \vee \quad \neg b \wedge \mathrm{IV}(V)$$

A_i is the subexpression for action i in the partition, and $IV(w)$ expresses that the state variables w are unchanged between the current and next state. The extra bit of information given by b denotes if an action in the partition executes or not. Bit partitions for *NADL* can be defined both for forward and backward search. However, for all deterministic planning domains, we have studied, a monolithic transition relation outperforms a bit partitioned. Figure 15 shows the CPU time for solving a single problem from a range of domains varying the threshold for recombining partitions. At the smallest threshold level as many partitions as possible are used. At the largest threshold level only one partition (a monolithic transition relation) is used. As depicted, the best CPU time for these domains is obtained with a monolithic transition relation. Apparently the cost of adding an extra variable to each partition is higher than the savings in the fringe calculations. This is not always the case. In model checking conjunctive partitions with extra information have been shown to improve overall performance for some problems.[5] The bit partitioning results are disappointing. But it is important to stress that the complexity of the fringe calculations at least is linear with the size of the fringe. If the fringe grows exponential so will the fringe calculations no matter how sophisticated the partitioning

[5]Communication with Pankajkumar Chauhan.

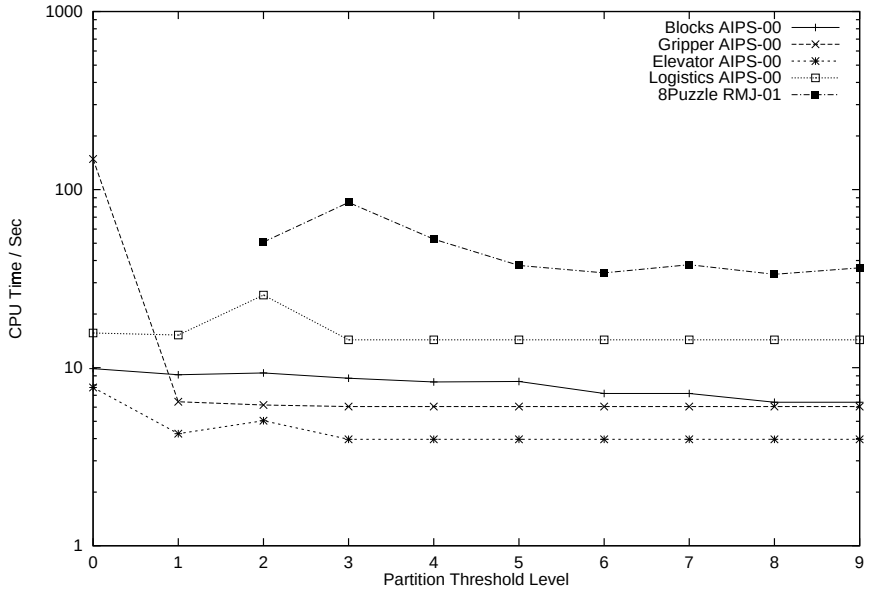

Figure 15: CPU time as a function of bit partition threshold from a range of planning problems.

is. As discussed in Section 4.3 an exponential growth of the search fringe seems to be a particular bottleneck for BDD-based planning based on blind search. When using directed search methods the efficiency of partitioning may become more important.

4.4.2 Disjunctive Partitioning

Disjunctive partitioning techniques have been developed in model checking for asynchronous circuits [16]. Since actions in a planning domain correspond to units in an asynchronous circuit this approach can also be applied to deterministic planning without adding any extra information to the partitions. A disjunctive partitioning with n partitions is n Boolean expressions, $P_i(s_v, a_v, s'_v)$ where $0 < i \leq n$, such that:

$$T(s_v, a_v, s'_v) = \bigvee_{i=1}^{n} P_i(s_v, a_v, s'_v)$$

Similar to a conjunctive partitioning, a disjunctive partitioning may be less complex to compute than a monolithic representation, and again, the preimage $PreImage(s_v, a_v)$ of

a set of states $A(s_v)$ can be efficiently computed:

$$P(s_v, a_v) = \bigvee_{i=1}^{n} \exists s'_{v_i} \,.\, P_i(s_v, a_v, s'_v) \wedge A(s_v - s_{v_i}, s'_{v_i})$$

The representation of the set of states A is a little tricky. The arguments $s_v - s_{v_i}$ are in current state variables since they are not changed by P_i, but the arguments s'_{v_i} are in next state variables because they are changed by P_i and need to be quantified. A less efficient approach quantifying all next state variables has been implemented in MIPS version 1.1. It is simple to define disjunctive partitions of *NADL* domains for both forward and backward search algorithms. However, disjunctive partitioning only improves performance if the peak fringe size is small compared to the size of the transition relation. We carried out an experiment for disjunctive partitioning similar to the experiment for conjunctive partitioning. As depicted in Figure 16, the CPU time was approximately cut in half for problems where the size of the peak fringe was small relative to the size of the monolithic transition relation (the peak fringe fraction of the monolithic transition relation is shown to the right of the domain name in the figure). Compared to conjunctive partitioning,

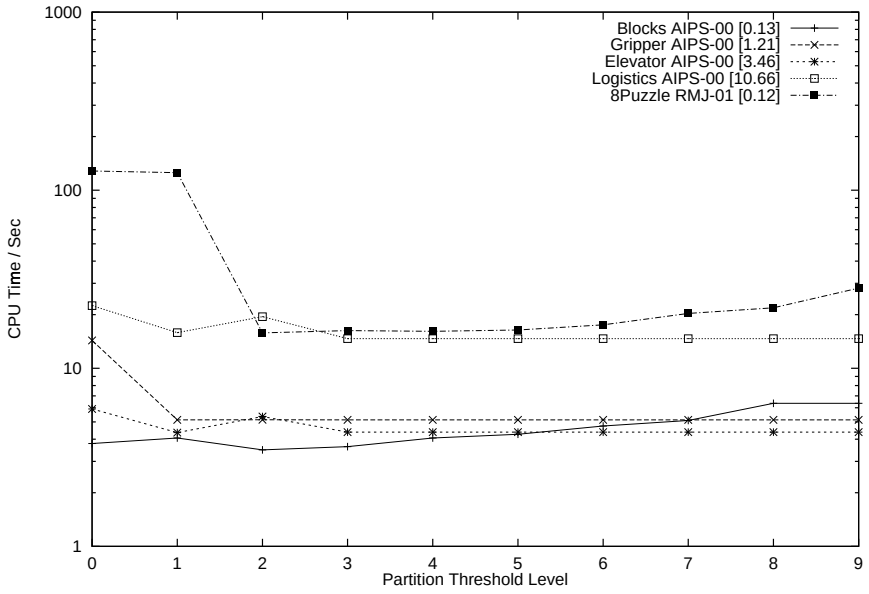

Figure 16: CPU time as a function of disjunctive partition threshold for a range of planning problems.

disjunctive partitioning suffers from introducing the A expression in each sub-computation

of the preimage. In conjunctive partitioning, A is only introduced once. For this reason it is still worthwhile to investigate if more powerful conjunctive partitionings exist for deterministic planning.

4.5 Abstraction

Abstract and hierarchical search are a classical approaches in deterministic planning [55, 59, 26, 58]. The main idea is to reduce the search complexity by first solving the planning problem in an abstract search space and then refining each abstract search step to a sequential plan.

We have been working on defining abstract search spaces for deterministic BDD-based planning where each transition corresponds to the effect of a set of actions instead of just a single action. Recall that the set of actions of each partition in a bit partitioning constrains a disjoint set of next state variables. If we further can prove that the effect of actions in each partition do not interfere with the preconditions of actions in the other partitions then actions from different partitions can be executed concurrently. In this case a partitioned abstract transition relation can be computed from a bit partitioning simply by removing the partitioning bit from each partition by existential quantification.

Another approach is to reason explicitly about which applicable actions that can be executed concurrently. Consider the following definition of the abstract transition relation T_{ABS}:

$$
\begin{aligned}
T_{ABS}(s,s') &= \exists I \,.\, \Big(\bigwedge_{a \in I} A(s,a,s') \Big) \wedge F(s,I,s') \wedge C(I) \\
F(s,I,s') &= \bigwedge_{p \notin Chg_I} (p' = p) \\
C(I) &= \forall v, w \in I \,.\, (v \neq w) \Rightarrow \neg Md(v,w)
\end{aligned}
$$

I is a set of concurrent actions. $A(s,a,s')$ expresses the precondition and effect of action a. $F(s,I,s')$ is a frame expression ensuring variables not constrained by the actions in I keep their value in the next state. Finally, $C(I)$ is a consistency expression that ensures that no distinct actions in I are mutual dependent. Two actions are mutual dependent if, for each action, the effect of the action interferes with the precondition of the other action. The consistency expression does not guarantee that the set of actions can be executed concurrently. But for some domains, including the Logistics domain, the condition is sufficient.

We implemented a planning system called DOP using this abstraction transition relation combined with domain preprocessing based on TIM [30] for generating compact

Boolean state encodings. An experiment measuring the CPU time of DOP in 6 Logistics problems derived from the AIPS-98 Logistics problem 1-1 shows several magnitudes of speed up compared to ordinary bidirectional search (see Table 1). The abstract search

	Abstract		Ordinary	
Problem	Time / sec	Length	Time / sec	Length
1	5.08	8	1.07	7
2	5.63	12	2.37	10
3	7.48	18	11.32	16
4	10.73	28	145.61	24
5	15.90	31	1908.19	26
6	22.78	34	-	-

Table 1: Abstract search compared to ordinary search for 6 Logistics problems derived from the Logistics problem 1-1 of the AIPS-98 planning competition.

algorithm of DOP first finds an abstract solution based on bidirectional search in the abstract search space and subsequently refines each abstract step using ordinary bidirectional search. The approach is sketched in Figure 17. We will refer to this as two-level abstraction. Notice that the plans generated by abstract search are suboptimal. In our

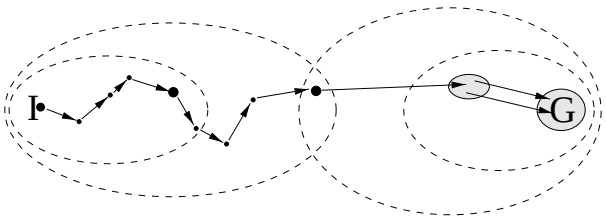

Figure 17: Abstract bidirectional search planning.

experiment they are a few steps longer than the optimal plans generated by the ordinary approach. The reason for this is that the concurrent action sequences considered by the abstract search only constitute a subset of all possible action sequences. Thus, this abstraction is a relaxation of the problem. Optimal abstraction techniques can be obtained with *iterative squaring* [9]. However, this approach often turns out to be too complex in practice.

No abstraction techniques have been developed for non-deterministic BDD-based planning and the approaches studied for deterministic planning still need further investigation.

Continued work in this direction is promising since: 1) abstraction alters the search fringe and thus may avoid exponential growth, and 2) it can be combined with directed search methods.

5 Summary of Proposed Research

As already mentioned, deterministic and non-deterministic search techniques differ because of dissimilar demands on the search and the solution formats. For this reason we choose to describe our approach to each problem separately.

5.1 BDD-based Deterministic Planning

Due to the exponential blow up of the fringe in blind search, we focus on directed search methods possibly combined with one or more additional approaches for reducing the search fringe size. Since our previous research already has shown that BDD-based planning using blind search is one of the fastest optimal planning approaches known today, we further choose to concentrate on fast but possibly suboptimal or even incomplete search algorithms not severely compromising with the solution quality.

Initially we plan to investigate directed search methods based on simpler heuristics than used by BDDA*. The heuristic function could for example be the number of BDD variables different from a possible goal value. This simple heuristic would enable an efficient partitioning of the search fringe into three sets: states with smaller (S), unchanged (U) and larger (L) heuristic value. A complete heuristic search would be easy to implement as a depth first search expanding the partitioned fringe in the order S,U,L. Further, an incomplete but possibly faster version of this search algorithm could simply skip the L set in each expansion. The example is easy to extend to bidirectional search. However, for more elaborate versions this might not be the case. To summarize, we propose to:

1. **Investigate simple directed BDD-based search algorithms that can trade optimality or even completeness for speed and scalability without significantly reducing the solution quality.**
2. **Experiment with different partial fringe expansion paradigms.**
3. **Generalize the approaches to bidirectional search if possible.**

An interesting extension of this research is to combine it with abstraction techniques and partitioning for blind search. We therefore propose to:

1. **Define two-level abstraction approaches for directed and undirected BDD-based search based on:**
   (a) **Concurrent actions.**
   (b) **Other abstractions (e.g. similar to ABSTRIPS [55]).**
2. **Continue current research on disjunctive and conjunctive partitioning approaches suitable for deterministic domains.**

For comparison experiments we plan to develop a fully automatic BDD-based PDDL planner. In order to do this more research is needed on generating minimal length Boolean state encodings of PDDL planning domains. We plan to:

1. **Continue our research on balanced predicate analysis.**
2. **Consider approaches for predicate substitution.**

5.2 BDD-based Non-Deterministic Planning

Initially we want to concentrate on finding more efficient BDD-based search approaches for the known strong, cyclic and optimistic BDD-based universal planning algorithms. The current approach is to perform a blind backward search and incrementally synthesize a universal plan from the preimages. As described in the previous section one obvious way to reduce this search is to constrain it only to states reachable from the initial states. It might also be possible to constrain the search further without violating the properties of the solution. Such search algorithms could be directed and cutting a subset of alternative paths to the goal. Another interesting direction is to consider specific algorithms for domains with particularly sparse or dense non-determinism. In short, we propose to:

1. **Investigate approaches for reducing the search space of BDD-based universal planning algorithms by:**
   (a) **Reachability analysis.**
   (b) **Cutting off valid solutions paths from the produced plan.**
2. **Study to what extent approaches reducing the number of solutions paths can be made directed as in deterministic planning.**
3. **Develop specialized algorithms for domains with sparse or dense non-determinism.**

As for deterministic planning, we also want to continue the work on partitioning for non-deterministic domains. As shown by our previous research, partitioning can speed up the search by at least a factor of two in some non-deterministic domains. So, partitioning may be more important for non-deterministic planning than deterministic planning. Also it may be possible to define abstraction for non-deterministic planning even though the plan is incrementally build. In summary, we propose to:

1. **Continue our work on partitioning techniques for non-deterministic domains**

2. **Investigate possible abstraction methods for non-deterministic planning that still allows an efficient plan synthesis.**

References

[1] A. Blum and M. L. Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1636–1642, 1995.

[2] J. Blythe. *Planning under Uncertainty in Dynamic Domains*. PhD thesis, Computer Science Department, Carnegie Mellon University, 1998. CMU-CS-98-147.

[3] J. Blythe and M. M. Veloso. Analogical replay for efficient conditional planning. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97)*, pages 668–673. AAAI Press, 1997.

[4] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proceedings of the European Conference on Planning (ECP-99)*, 1999.

[5] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97)*, pages 714–719. AAAI Press, 1997.

[6] K. Brace, R. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 40–45, 1990.

[7] R. E. Bryant. Symbolic manipulation of boolean functions using a graphical representation. In *DAC*, pages 688–694, 1985.

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

[9] J. R. Burch, E. M. Clarke, and K. McMillan. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science*, pages 428–439, 1990.

[10] J.R. Burch, E.M. Clarke, and D.E. Long. Symbolic model checking with partitioned transition relations. In *International Conference on Very Large Scale Integration*, pages 49–58. North-Holland, 1991.

[11] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

[12] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: A decision procedure for $\mathcal{AR}$. In *Proceedings of the 4th European Conference on Planning (ECP'97)*, Lecture Notes in Artificial Intelligence, pages 130–142. Springer-Verlag, 1997.

[13] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[14] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 875–881. AAAI Press, 1998.

[15] A. Cimatti, M. Roveri, and P. Traverso. Strong planning in non-deterministic domains via model checking. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning System (AIPS'98)*, pages 36–43. AAAI Press, 1998.

[16] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[17] O. Coudert, C. Berthet, and J. Madre. Verification of sequential machines using symbolic execution. *Automatic Verification Methods for Finite State Machines*, pages 365–373, 1989.

[18] T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76:35–74, 1995.

[19] M. Drummond. Situated control rules. In *Proceedings of the 1'st International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 103–113. Morgan Kaufmann, 1989.

[20] M. Drummond and J. Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the 8th Conference on Artificial Intelligence*, pages 138–144. AAAI Press, 1990.

[21] S. Edelkamp. Directed symbolic exploration in AI-planning. In *AAAI Spring Symposium*, 2001.

[22] S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *Proceedings of the 6th European Conference on Planning (ECP'99)*, pages 135–147, 1999.

[23] S. Edelkamp and M. Helmert. On the implementation of MIPS. In *Proceedings of AIPS-2000 Workshop on Decision-Theoretic Planning*, pages 18–25, 2000.

[24] S. Edelkamp and F. Reffel. OBDDs in heuristic search. In *KI*, pages 81–92, 1998.

[25] S. Edelkamp and F. Reffel. Deterministic state space planning with BDDs. In *Proceedings of the 5th European Conference on Planning (ECP-99)*, pages 381–382, 1999.

[26] K. Erol, J. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 1123–1128, 1994.

[27] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An approach for planning with incomplete information. In *Proceedings of the 3'rd International Conference on Principles of Knowledge Representation and Reasoning*, 1992.

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

[29] M. P. Fourman. Propositional planning. In *AIPS-2000 Workshop on Model-Theoretic Approaches to Planning*, pages 10–17, 2000.

[30] M. Fox and D. Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, 1998.

[31] M. Fox and D. Long. Hybrid stan: Identifying and managing combinatorial subproblems in planning. In *UK Planning and Scheduling SIG Workshop*, 2000.

[32] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92)*, pages 809–815. AAAI Press, 1992.

[33] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI'87)*, pages 677–682. Morgan Kaufmann, 1987.

[34] A. Gerevini and L. Schubert. Inferring state constraints for domain-independent planning. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 905–912, 1998.

[35] K. Z. Haigh and M. M. Veloso. Planning, execution and learning in a robotic agent. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, pages 120–127. AAAI Press, 1998.

[36] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for heuristic determination of minimum path cost. *IEEE Transactions on SSC*, 100(4), 1968.

[37] J. Hoffman and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Submitted, Journal of Artificial Intelligence Research*, 2001.

[38] S. Holldouble and H.-P. Stör. Solving the entailment problem in the fluent calculus using binary decision diagrams. In *AIPS-2000 Workshop on Model-Theoretic Approaches to Planning*, pages 32–39, 2000.

[39] R. M. Jensen, M. M. Veloso, and M. Bowling. Optimistic and strong cyclic adversarial planning. 2001. To be submitted to ECP'01.

[40] R.M. Jensen and M. M. Veloso. OBDD-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research*, 13:189–226, 2000.

[41] F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence*, 95:67–113, 1997.

[42] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI'96)*, volume 2, pages 1194–1201. AAAI Press, 1996.

[43] H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, volume 1, pages 318–325. Morgan Kaufmann, 1999.

[44] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *Proceedings of the 4th European Conference on Planning (ECP'97)*, Lecture Notes in Artificial Intelligence, pages 273–285. Springer-Verlag, 1997.

[45] S. Koenig and R. G. Simmons. Real-time search in non-deterministic domains. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1660–1667. Morgan Kaufmann, 1995.

[46] J. Lind-Nielsen. BuDDy - A Binary Decision Diagram Package. Technical Report IT-TR: 1999-028, Institute of Information Technology, Technical University of Denmark, 1999. `http://cs.it.dtu.dk/buddy`.

[47] D. Long and M. Fox. Type analysis of planning domain descriptions. In *Proceedings of 17th Workshop of UK Planning and Scheduling SIG*, 1998.

[48] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. Technical report, Yale Center fro Computational Vision and Control, 1998.

[49] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publ., 1993.

[50] C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design.* Springer, 1998.

[51] J. S. Penberthy and D. S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the 3'rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114. Morgan Kaufmann, 1992.

[52] M. Peot and D. Smith. Conditional nonlinear planning. In *Proceedings of the 1'st International Conference on Artificial Intelligence Planning Systems (AIPS'92)*, pages 189–197. Morgan Kaufmann, 1992.

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

[54] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the International Conference on Computer-Aided Design*, pages 139–144, 1993.

[55] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[56] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 1039–1046. Morgan Kaufmann, 1987.

[57] R. S. Sutton and A. G. Barto. *Reinforcement Learning: an Introduction*. MIT Press, 1998.

[58] M. Veloso, J. Carbonell, A. Pérez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

[59] D. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufman, 1988.

[60] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:197–227, 1994.

[61] N. XuanLong and S. Kambhampati. Extracting effective and admissible state space heuristics from the planning graph. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00)*, pages 798–805, 2000.

[62] B. Yang, R. E. Bryant, D. R. O'Hallaron, A. Biere, O. Coudert, G. Janssen, R. K. Ranjan, and F. Somenzi. A performance study of BDD-based model checking. In *Formal Methods in Computer-Aided Design FMCAD'98*, pages 255–289, 1998.