Deterministic State Space Planning with *BDDs*
Technical Report 120 - April 14, 1999

Stefan Edelkamp[1] and Frank Reffel[2]

[1] Institut für Informatik, Albert-Ludwigs-Universität, Am Flughafen 17, D-79110 Freiburg; eMail: edelkamp@informatik.uni-freiburg.de
[2] Institut für Logik, Komplexität und Deduktionssysteme, Universität Karlsruhe, Am Fasanengarten 5, D-76128 Karlsruhe; eMail: reffel@ira.uka.de

Abstract. The paper proposes a planner that uses *BDDs* to compactly represent blocks of propositionally represented states. Using this representation, accurate reachability analysis and backward chaining can apparently carried out without necessarily encountering exponential representation explosion. The main objectives are the interest in optimal solutions, the generality and the conciseness of the approach.
The algorithms are tested against the AIPS98 planning competition problems and lead to substantial improvements to existing solutions.

1 Introduction

Research is challenge. In 1998 five contestants (Blackbox by Kautz and Selman, HSP by Bonet and Geffner, IPP by Koehler, SGP by Anderson and Weld and STAN by Long and Fox) participated in the first AI planning systems competition (AIPS98). The organizers write: "The competition seemed to succeed in its goals of generating excitement and establishing how well top-quality planning algorithms actually perform [...] We encourage others to run their systems on the problems and compare the results[...]."[1]

IPP, Blackbox, SGP and STAN are based on *Graphplan* [2], whereas HSP has been build on a learning variant of real-time A*, LRTA* for short [18]. Graphplan constructs a layered planning graph containing two types of nodes, action nodes and proposition nodes. In a given layer the preconditions of all operators are matched, such that Graphplan consideres instanciated actions at specific points in time. On the other hand it generates partially ordered plans to exhibit concurrent actions. Graphplan alternates between two phases: *graph extension* to extend the search depth and *solution extraction* to terminate the planning process. Graphplan finds optimal parallel plans and, as almost all AI planning algorithms, it bypasses the state space representation of a planning domain. A plan is essential a flow within the planning network graph.

LRTA* performs heuristic real-time state space search within the planning domain. When it expands a state the following steps are executed until a goal state has been found: The heuristic estimate is set to the best value of all adjacent

[1] `http://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html`.

states and the algorithm executes the move to the node that has this value. The heuristic is of domain-independent design and improves over time. Therefore, the main focus of HSP is reinforcement learning and memorization. Note, that LRTA* has also been extended to solve non-deterministic planning domains [16].

This paper offers new insights to be gleant from the third perspective of a symbolic exploration of the state space search tree based on binary decisision diagrams (*BDDs*). It is devised to overcome the limitations based on its typical exponential growths. The approach is guided by *reachability analysis* in model checking which in this context we prefer to denote by *symbolic breadth first search* [22]. It applies to both deterministic and non-deterministic domains, but we have evaluated the technique only to the former ones. The generated plans are optimal in the number of sequential execution steps.

Since Bryant's invention of *BDDs* [4] they have attracted both theoreticans and practioners. *BDDs* have the expressional power to efficiently encode and manipulate large sets of bitstrings. Therefore, *BDDs* are capable of symbolically representing both the states within the planning space and the operators applied.

Cimatti, Roven and Traverso present a *BDD*-based planning approach capable to deal with non-deterministic domains specified in AR [5–7,11] and recent work is devised to extend the approach from AR to the formalism C [12,13]. In contrast we applied our approach to problems given in PDDL to evaluate our algorithm in challenging determinstic planning domains. Even we have not yet completed a domain independent parser which would be necessary for a fair comparison with other planning systems we believe that similar results will be achieved. We did not not hand-optimize each single domain but followed an informal strategy. In addition simple breadth first search was extended to bidirectional and heuristic search, a straightforward adaption to BDD-based planning.

The following section introduces prelimiaries concerning BDDs and methods for reachability analysis. In Section 3 the BDD planning algorithm is explained, while in Section 4 some comments to the implementation and the results of the experiments with the planning contest problems are presented. Finally, we end with a conclusion and a perspective for future work.

2 Reachability Analysis

In briefly reviewing model-based reachability analysis we use the notion of search problems given by (V, O, s, G), where V is the set of states, O the set of operators, s the start state, and G the set of goal states.

2.1 Binary Decision Diagrams

Reduced ordered binary decision diagrams, *ROBDD* for short, with respect to a fixed variable ordering are a canonical representation for boolean functions [4]. Throughout this paper we write *BDDs*, although we alway mean *ROBDDs*.

Definition 1. A *BDD* of the function f with respect to the variable ordering π is an acyclic graph with one source and two sinks labelled with 0 and 1. All other (internal) nodes are labelled with a boolean variable x_i of f and have two outgoing edges labelled 0 and 1.

The evaluation of $f(x_1, \ldots, x_n)$ according to the input $a = (a_1, \ldots, a_n)$ is found as follows. At each x_j labelled node we choose the a_j labelled edge until we reach a sink. Every binary boolean operation (such as $\wedge$, $\Leftrightarrow$, $\oplus$, $\Rightarrow$ and $\vee$) can be applied to merge *BDD* G_1 and G_2 of the same variable ordering in $O(|G_1||G_2|)$ worst-case time. The key operation in model checking is the following.

Definition 2. The *relational product* of a vector of variables v and two boolean functions f and g ist defined as $\exists v(f \wedge g)$.

Since existential quantification of one boolean variable x_i in the boolean function f is equal to disjunction $f|_{x_i=0} \vee f|_{x_i=1}$, the quantification of v results in a series of subproblem disjunctions[2]. Although computing the relational product is NP-hard in general, in many practical applications its determination is quite manageable: For most domains memory limitations are more crucial than time.

2.2 Encoding Sets of States

Let $C : V \rightarrow \{0,1\}^k$ be a suitable binary encoding of V. For our purposes it suffices to know that k is a small constant.

Definition 3. Let $S \subset V$ be a set of states. The *characteristic function* ϕ_S evaluates an input a to 1 if a is the binary encoding of one state s in S.

The BDD of ϕ_S represents S symbolically. By rearranging the variables in ϕ we may assume that π is the trivial permutation. For example let V be the set of states $\{0,1,2,3\}$ encoded by their binary value, that is $C(0) = 00, C(1) = 01, C(2) = 10, C(3) = 11$. The characteristic funcion of a single state is the minterm of the binary encoding, i.g. $\phi_{\{0\}}(x) = \overline{x_1} \wedge \overline{x_2}$. The resulting BDD has two inner nodes. The cruical observation is that the BDD representation of S increases by far slower as $|S|$. For example the BDD for $\phi_{\{0,1\}} = \overline{x_1}$ consists of one internal node and the BDD for $\phi_{\{0,1,2,3\}}$ is given by the 1-sink only.

2.3 The Transition Relation

An operator can also been seen as an encoding of a set. In contrast to the previous section a member of the transition relation corresponds to a pair of states (s', s) if s' is a predecessor of s. Since we already have an encoding C for states we simply take $(C(s'), C(s))$ as a bitstring of length $2k$.

Definition 4. The *transition relation* T is the disjunction of the characteristic functions of all pairs (s', s) with s' is a predecessor to s.

---

2 The number of subproblem disjunctions has proven to be a good platform independent measure of the work to be performed [23].

Subsequently, the transition relation T evaluates to 1 if and only if s' is a predecessor of s. Enumerating the cross product of the entire state space is by far too expensive. Fortunately, we can set up T symbolically by defining which variables change due to an operator and which variables do not.

Definition 5. Let $o \in O$ be an operator in a search problem (V, O, s, G) and I_o be the set of indices i in the binary encoding $x'_1, \ldots, x'_k$ of states in V that are potentially manipulated by o to obtain successor states encoded by $(x_1, \ldots, x_k)$. The *frame* F_o is defined as the boolean function $\bigwedge_{i \in \{1,\ldots,k\} - I_o} (x'_i \Leftrightarrow x_i)$.

In the general case the description of an operator is generic, that is it does not only depend on the state description but on a set of parameters. Later on, we will see how to unfold these parameters automatically, such that we can restrict our considerations to instanciated operators.

Corollary 6. *Let T be the transition relation of a search problem (V, O, s, G), O be the set of instanciated operators and I_o for $o \in O$ be defined as above. Let A_o be the expression, which defines the change of variables with index I_o in the binary encoding due to the application of o. Then $T(s, s') = \bigvee_{o \in O} (A_o(s, s') \wedge F_o(s, s'))$.*

2.4 Unidirectional Search

Let S_i be the set of states reachable from the start state in i steps, initialized by $S_0 = \{s\}$. The follwing equation determines ϕ_{S_i} given $\phi_{S_{i-1}}$ and the transition relation: $\Phi_{S_i}(s) = \exists s' \ \Phi_{S_{i-1}}(s') \wedge T(s', s)$. In other words we perform breadth first search with BDDs. The formula calculating the BDD for Φ_{S_i} is a relational product. A state s belongs to S_i if it has a predecessor in the set S_{i-1} and there exists an operator which transforms s' into s. It should be noted that Φ depends once on s and once on s'. Therefore, it is necessary to calculate the BDD for Φ_{S_i} where s is substituted with s'. Fortunately, this substitution corresponds to a simple renaming of variables. In order to terminate the search we test, whether a state is represented in the intersection of the symbolic representation of the set S_i and G. This can also be decided by applying relational product: *goalReached* $= \exists x \ \phi_{S_i} \wedge \phi_G$. Since we have processed $S_0, \ldots, S_{i-1}$ in case *goalReached* evaluates to true then i is known to be the optimal solution length.

2.5 Bidirectional Search

Backward search is the opposite to forward search. We start with the goal set $B_0 = G$ and iterate as long as we encounter the start state. In backward search we take advantage of the fact that T has been defined as a relation. Therefore we iterate according to the formula $\Phi_{B_{i-1}} = \exists x \ \Phi_{B_i} \wedge T$. In bidirectional breadth-first search forward and backward search are carried out concurrently. On the one hand we have the forward search frontier F_f with $F_0 = S$ and the backward search frontier B_b with $B_0 = G$. The search direction is chosen to favour the smaller BDD. When the two search frontiers meet we have found an optimal solution of length $f + b$, i.e. *goalReached* $= \exists s \ \phi_{F_f} \wedge \phi_{B_b}$ evalutates to true.

2.6 Heuristic Search

The success in solving a lot of difficult problems with heuristic search [14,19,20] gives rise to the question if breadth first search with *BDDs* can be extended by means of a heuristic estimation. Fortunately, the answer is yes. The algorithm *BDDA** combining heuristic search with *BDD* exploration has been effectively applied to the $(n^2 - 1)$-puzzle and to Sokoban[3] to challenge both traditional heuristic search algorithms and uninformed reachability analysis [9]. For planning domains, Geffner and McDermott have illustrated how a problem independent heuristic can be devised [3,21].

3 The *BDD* Planning Algorithm

Instead of inventing a new specification language to describe a planning problem we translate a given specification such that BDDs can be applied. The running example for our considerations is the following logistic domain. We are given a number of packages at certain locations (such as the airport or the post office) in various American cities. In each city we have a truck connecting the airport and the post office. Furthermore, there is a certain number of airplanes connecting the cities. All airplanes and trucks can be loaded and unloaded with packages. In a given city we can drive the truck from one location to another. The goal is to deliver the packages to their goal locations.

3.1 Strips

The Strips [10] specification of the logistic domain includes predicates and actions, e.g. the *load-truck* action:

```
(define (domain logistics-strips)
  (:predicates (OBJ ?obj) (TRUCK ?truck) (LOCATION ?loc)
               (AIRPLANE ?airplane) (CITY ?city) (AIRPORT ?airport)
               (at ?obj ?loc)  (in ?obj ?obj) (in-city ?obj ?city))
(:action LOAD-TRUCK
  :parameters (?obj ?truck ?loc)
  :precondition (and (OBJ ?obj) (TRUCK ?truck) (LOCATION ?loc)
                     (at ?truck ?loc) (at ?obj ?loc))
  :effect (and (not (at ?obj ?loc)) (in ?obj ?truck))) ...)
```

In Strips simple predicates and the start state are enumerated facts[4]:

```
(OBJ package1) (OBJ package2) ...
(AIRPLANE airplane1) (AIRPLANE airplane2) ...
(at package1 pgh-po) (at package2 pgh-po) ...
```

[3] http://xsokoban.lcs.mit.edu/xsokoban.html

[4] Dots (...) are printed if source fragments are left out.

3.2 Encoding of Planning States

The difference between state space search and planning is subject of several hot debates [17]. We will concentrate on pragmatics. If it is possible to easily encode the states in the benchmark planning domains, we have no hesitation in doing so. However, since all competing planners do not integrate problem specific knowledge and since we currently have not completed the domain independent parser, we have to be careful in the presentation to convince the reader of the generality, conciseness and domain-independence of the given approach.

It is not difficult to generate a valid binary encoding of states in the example problem. The binary predicate `at` combines packages, trucks and airplanes with locations. By examining the effect lists of the operators we see that it changes over time, such that it has to be included in the state description. By enumeration we can count the number of instanciations of a parameter in an operator. If l is the number of all locations and p is the number of all objects in our example for the `at` predicate we add $p\lceil \log l \rceil$ binary variables to the state description each of the p blocks encoding the interval $[0..l-1]$. Let t and a be the number of trucks and airplanes respectively. Then the predicate `in` leads to $p\lceil \log(t+a) \rceil$ extra bits in the state description length.

In the considered benchmark set predicates are $[n:l]$ relations between objects. Usually we have either $n=1$ or $l=1$, i.g. `at` is $[1:l]$ and assigns locations to packages, trucks and airplanes. Furthermore, by analysing the strips specification we exibit that no object can be placed at different locations. The state space is devised as follows: For each $[1:l]$ predicate $(type_1, type_2)$ with an assingnment of p objects of $type_1$ to l objects of $type_2$ we take p bitvectors of length $\lceil \log l \rceil$, Processing every predicate we finally get a a complete description of the state space, that can be dealt with *BDDs*. In case of $n>1$ and $l>1$ or other multivalued predicates the bitvector representation can be found by projecting the predicate to its arguments. This approach increases the length of the bitvector, however, multivalued predicates reveal bad modelling and difficulties to all planning systems due to the resulting large branching factor.

3.3 Building the Transition Function

Let O be the set of all (Strips) operators. Then the transition relation T for the planning problem is given by the disjunction $T(x',x) = \bigvee_{o \in O} o(x',x)$. To build the sub-relation $o(x',x)$ we use enumeration of all possible values of the parameters. For example to find the binary representation $Lt(x',x)$ of the *load-truck* operator we loop on every package, on every truck, and on every location: $Lt(x',x) = \bigvee_{i,j,k} \textit{load-truck}(p_i, t_j, l_k, x', x)$.

Let I_{Lt} be the set of indices in the state description that are affected by the instanciated operator Lt. We have seen that the frame F_{Lt} is given by $F_{Lt}(x',x) = \bigwedge_{i \notin I_{Lt}} (x'_i \Leftrightarrow x_i)$. Hence, we get the following boolean specification of the operator *load-truck*: $At(x', p_i, l_k) \wedge \neg In(x', p_i, t_j) \wedge At(x', t_j, l_k) \wedge \neg At(x, p_i, l_k) \wedge In(x, p_i, t_j) \wedge At(x, t_j, l_k) \wedge F_{Lt}(x', x)$

The BDD for the predicate At extracts the variable block according to the parameter p_i or t_j in x and x' and generates a characteristic *BDD* w.r.t. the value l_k. Furthermore, it marks the used block in the global list L of indices to determine I_{Lt}. After the calculation of the *BDD* F_{Lt} we reset L.

3.4 *ADL*

ADL has been designed to fill the gap between Strips and situation calculus. The difference of *ADL* compared to Strips is that *ADL* allows context-dependent action effects. The need of quantification is apparent: if we move a truck from one location to another everything inside should be moved as well.

```
(drive-truck
  :parameter (TRUCK truck) (LOCATION ?from) (LOCATION ?to)
  :precondition (at ?truck ?from)
  :effect (and (at ?truck ?to) (not (at ?truck ?from))
               (forall OBJ ?x (when (in ?x ?truck)
                  (and (at ?x ?to) (not (at ?x ?from)))))))
```

As we have seen *BDDs* have the expressional power to deal with existential quantification. The $\forall$ quantification is easily obtained through negation. *BDDs* can also represent if-then-else: $ite(g, f, h) = (g \wedge f) \vee (\overline{g} \wedge h)$, which replaces the conditional effect labelled by *when*.

4 Implementation

Although the problem definitions are given in Lisp all successful planning systems are written in C or C++ and our system is no exception. The software package *Mips* abbreviates *intelligent model checking and planning system*. The project encapsulates algorithms and data structures for model checking and planning.

First of all we have newly implemented a very efficient and concise index based depth first search *BDD* package *StaticBdd*. The performance study of different existing *BDD* systems has shown that this approach challenges even existing professional packages [23]. In about 1K lines all necessary features of a complete *BDD* system (such as a unique table, a cache, a free list, a compressed state description, a dirty bit garbage collector, the reduction integrated synthesis and relational product algorithm) are provided. Since we aim to handle millions of *BDD* nodes a very compressed representation of a *BDD* node is neccessary. Hence, we use a bit field to encode both successor indices, the variable index and the two traversal bits. Our *BDD*-planning system *Mips* provides a trivial domain-independent lower bound on the solution length: It counts the number of unachieved predicates w.r.t the goal state.

Each planning problem partitions in a domain specific and in an instance specific part. The domain is given by its operators, whereas relevent facts, initial and goal conditions are specified in instances to the domain. Transferring the instance specific definition of an example problem is unproblematic. We used a (domain-dependent) parser that automatically translates the lisp like definition in C++-Code. Its application supports the reliability of the encodings within the set of problem instances.

In the following we present the source code to unfold the part of the transition function depending on the *load-truck* operator. Note, that for the application of binary operators we utilized the overloading technique of C++.

```
BDD transition() {
  BDD lt=BDD.zero;
  for (int o=0;o<objects;o++) { if (OBJ(o)) {
      for (int t=0;t<objects;t++) { if (TRUCK(t)) {
          for (int l=0;l<objects;l++) { if (LOCATION(l)) {
              for (int c=0;c<objects;c++) { if (CITY(c)) {
                  if (in_city(l,c) && in_city(t,c)) {
                    lt = lt | load_truck(o,t,l); }}}}}}}}}}
```

For simple predicates we preferred booleans values instead single-node *BDDs*, although both approaches are valid. We finish the implementation issues for the example problem with the source of the *load-truck* operator.

```
BDD At(var* v, int o, int l) { v->used(o); return v->is(o) == l;}
BDD Frame() {
  BDD f = equal(pre,eff,F); for (int i=0;i<state->size;i++) F[i] = true;
  return f; }
BDD load_truck(int o, int t, int l) {
  return At(pre,o,l) & ~In(pre,o,t) & At(pre,t,l) &
         ~At(eff,o,l) & In(eff,o,t) & At(eff,t,l) & Frame(); }
```

A variable `var` is an instanciation to a given (sub)domain. It merely consists of an array that addresses the boolean variable indices in the *BDD*. The function `used` sets the variable pairs of the frame `F` that are used in an operation to false and `is` extracts the subvariable according to the given index.

Given the state representation and the Strips source it is aparent that C++ source for the logistic domain can be easily inferred. Figure 1 shows a typical profile of *symbolic bidirectional breadth first search* in the example problem. The curves show the BDD sizes of the two search frontiers. Note, that in iteration 45 we have reached a settlement. This reflects the major advantage of *BDDs*: Even though the number of represented states in the search grows exponentially the *BDD* sizes do not necessarily do the same.

4.1 Solving the Planning Contest Problems

Round 1 of the AIPS competition consisted of two tracks, labeled *ADL* and Strips and Round 2 of Strips problems only. All data was produced on a Sun Ultra Sparc Station. Table 1 depictes our experimental results to the logistic problem suite (Velosa) – which was the only one where we applied bidirectional search– and the following five Strips benchmarks:

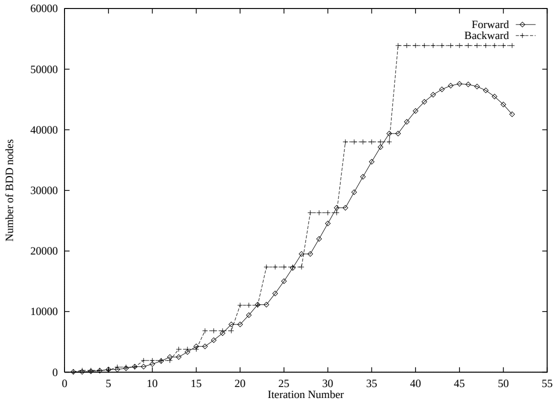

Fig. 1. Profile of solving one example problem to the logistic suite.

Movie (Anderson): In this domain, the goal is to have lots of snacks in order to watch a movie. This problem is not a problem at all since every planner can solve every problem optimally.

Gripper (Koehler): There is a robot with two grippers. It can carry a ball in both of them. The goal is to take balls from one room to another. Like *HSP* we solved every problem instance in this domain, but its solution lengths are at about 20 % higher than our optimal ones.

Mystery (McDermott): There is a planar graph of nodes. At each node are vehicles, cargo items, and some amount of fuel. Objects can be loaded onto vehicles up to their capacity, and the vehicles can move between nodes; but a vehicle can leave a node only if there is a nonzero amount of fuel there, and the amount decreases by one unit. The goal is to get cargo items from various nodes to various new nodes. The crucial fact in this domain is that our approach can prove the unsolvability of given problem instances, since in level 4 we performed a complete state space exploration.

Mprime (McDermott): This is the mystery domain with one extra action, the ability to squirt a unit of fuel from any node to a neighboring node, provided the originating node has at least two units. Given the extra operator, shorter solutions may appear as in the case of the unsolvable problem 4 which in this domain becomes solvable.

Grid (Koehler): There is a square grid of locations. A robot can move one grid square at a time horizontally or vertically. If a square is locked, the robot can move to it only by unlocking it, which requires having a key of the same shape as the lock. The goal is to get keys from various locations to various new locations. In this example we used the model checker μcke [1]. It also uses BDDs but offers a more comfortable input language, so it took less time to perform the experiments. The first four levels were solved, but level 3 was intractable with the breadth-first search approach. To overcome memory limitations we applied

BDDA* with a simple heuristic summarizing the distances of the keys to their respective goal fields. Integrating problem knowledge of this kind is not allowed in general proposed planning, but on the other hand the heuristic can be generated automatically and solves the problem.

The **Assembly** domain (McDermott) is given in *ADL*: The goal is to assemble a complex object made out of subassemblies. The sequence of steps must obey a given partial order. In addition, through poor engineering design, many subassemblies must be installed temporarily in one assembly, then removed and given a permanent home in another. In the assembly domain we also applied the model checker μcke. Exemplarily, we implemented the first two levels and found optimal solutions for the problems `assem-x-1` and `assem-x-2` in 20 s and 30 s, with solution lengths of 28 and 27, respectively. Note, that no other planning system solved any problem instance.

Sokoban is a challenging planning problem not in the AIPS competition that can be specified similarily to the logistics domain above. The bidirectional *BDD* based exploration method is able to solve 6 instances of the Sokoban problem suite move-optimally [8]. We have implemented this domain to measure the performance of our underlying *BDD* package. The first level can be solved in 45 seconds. Note, that no other planning system can solve any of the 90 benchmark problems. It is possible to integrate a BDD for a set of dead patterns in the search, which has proven to be the most effective idea in solving the puzzle [14], but for a small growth of the *BDD* sizes there is a trade-off between the number of states and the number of constraints.

5 Conclusion and Discussion

Modelling and solving planning problems have always been topics of AI research. We have demonstrated how the state description can be inferred from Strips or *ADL* and how to utilize *BDDs* to profit from its compact state representation. The generality of our approach is proven by the fact that we found solutions to every domain of the AIPS98 competition.

The approach in this paper may be seen as an elaboration on the idea of satisfiability planning in which the planning domain is encoded in conjunctive normal form [15]. However, the given algorithm origins in BDD based model checking and does not refer to SAT-plan.

Small changes in the transition relation provides execution of independent actions at the same time and to enhance practical applicability since in several applications operators can be carried out simultaneously. Furthermore, nondeterminism is achieved by specifying different outcomes to a given operator.

One major question in applying BDD techniques to given problem areas is the limitations of the approach. The experimental results suggest that in deterministic planning reachability analysis and its extension are well-placed and worthwhile of further explorations. However, in some challenging planning tasks the requirements of both time and space in the symbolic exploration are considerably high. Therefore, to tackle these difficulties we are currently working on small but expressive heuristics for the planning problems, the integration of macro-operators and a bidirectional heuristic search version of the *BDD* based exploration. The crucial task is to complete the domain-independent parser to automatically map planning problems into *BDD* related notation. Up to now we have a lex/yacc parser transferring the specification into data structures. Furthermore, we symbolically determine *constant* predicates and *semi-constant* predicates that do not change or change in only one direction, respectively. Finally, we automatically merge predicates such as `at` and `in` in Logistics, `craves` and `fears` in Mystery/MPrime, `at` and `holding` in Grid, `at` and `carry` in Gripper. We still have to restrict the ranges of predicates and to apply heuristics for a good variable ordering in the transition function to face other systems in a planning competition. As we said at the beginning: Research is challenge.

References

1. A. Biere. μcke - efficient μ-calculus model checking. In *Computer Aided Verification*, volume 1254 of *LNCS*, pages 468–471, 1997.
2. A. Blum and M. Furst. Fast planning through planning graph analysis. In *IJCAI'95*, pages 1636–1642, 1995.
3. B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *AAAI-97*, pages 714–719, 1997.
4. R. E. Bryant. Symbolic manipulation of boolean functions using a graphical representation. In *ACM/IEEE Design Automation*, pages 688–694, 1985.
5. A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: A decision procedure for AR. In ECP'97, Toulouse, France.
6. A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *AAAI*, pages 875–881, 1995.
7. A. Cimatti, M. Roveri, and P. Traverso. Strong planning in non-deterministic domains via model checking. Technical Report 9801-07, IRST, 1998.
8. J. Doppelhamer and J. K. Lehnert. Optimal solution for sokoban using OBDDs. University of Trier, available from the authors, 1998.
9. S. Edelkamp and F. Reffel. OBDDs in heuristic search. In *KI-98*, volume 1504 of *LNAI*, pages 81–92, Berlin, 1998. Springer.
10. R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving.
11. E. Giunchiglia, G. N. Kartha, and V. Lifschitz. Representing action: Indeterminacy and ramifications. *Artificial Intelligence*, 95(2):409–438, 1997.
12. E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: Preliminary report. In *AAAI*, pages 623–630, 1998.
13. A. Goel. OBDD-based planning with real-valued variables in non-deterministic environments. In *AAAI*, page To appear, 1999.
14. A. Junghanns and J. Schaeffer. Single agent search in the presence of deadlocks. In *AAAI-98*, pages 228–233, 1998.
15. H. Kautz and B. Selman. Pushing the envelope: Planning propositional logic, and stochastic search. In *AAAI*, pages 1194–1201, 1996.
16. S. Koenig and R. Simmons. Real-time search in non-deterministic domains. In *IJCAI*, pages 1660–1667, 1995.
17. R. E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

18. R. E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.
19. R. E. Korf. Finding optimal solutions to Rubik's Cube using pattern databa ses. In *AAAI*, pages 700–705. AAAI Press, 1997.
20. R. E. Korf and L. A. Taylor. Finding optimal solutions to the twenty-four puzzle. In *AAAI*, pages 1202–1207. AAAI Press, 1996.
21. D. McDermott. A heuristic estimator for means ends analysis in planning. *AIPS-96*, pages 142–149, 1996.
22. K. McMillan. *Symbolic Model Checking*. Kluwer Academic Press, 1993.
23. 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 G. Gopalakrishnan, editor, *Formal methods in Computer-Aided Design*, LNCS.

Domain	Problem	*BDD* size		Time (min:sec)					Solution Length				
	number	States	Trans	**Mips**	HSP	IPP	Stan	BB	**Mips**	HSP	IPP	Stan	BB
Movie	1-1	34	55	**0:01**	0:02	0:00	0:00	0:00	**7**	7	7	7	7
	...	34	55	**0:01**	0:02	0:00	0:00	0:00	**7**	7	7	7	7
	1-30	34	55	**0:01**	0:03	0:00	0:00	0:00	**7**	7	7	7	7
Gripper	1-1	73	245	**0:02**	0:02	0:00	0:00	0:00	**11**	13	11	11	11
	1-2	170	421	**0:02**	0:02	0:00	0:01	0:07	**17**	21	17	17	17
	1-3	391	719	**0:02**	0:02	0:10	0:54	-	**23**	31	23	23	-
	1-4	703	1003	**0:03**	0:03	0:29	6:03	-	**29**	37	29	29	-
	1-5	1335	1343	**0:03**	0:03	3:45	-	-	**35**	47	47	-	-
	1-6	2230	1723	**0:04**	0:03	-	-	-	**41**	53	-	-	-
	1-7	3594	2281	**0:04**	0:04	-	-	-	**47**	63	-	-	-
	1-8	5430	2769	**0:06**	0:05	-	-	-	**53**	71	-	-	-
	1-9	7922	3313	**0:08**	0:06	-	-	-	**59**	79	-	-	-
	1-10	11201	3897	**0:10**	0:07	-	-	-	**65**	85	-	-	-
	1-11	15457	4545	**0:15**	0:09	-	-	-	**71**	93	-	-	-
	1-12	20801	5225	**0:23**	0:10	-	-	-	**77**	101	-	-	-
	1-13	27449	5961	**0:38**	0:12	-	-	-	**83**	109	-	-	-
	1-14	35588	6737	**0:52**	0:15	-	-	-	**89**	119	-	-	-
	1-15	45573	7795	**1:26**	0:16	-	-	-	**95**	125	-	-	-
	1-16	57359	8679	**2:29**	0:20	-	-	-	**101**	135	-	-	-
	1-17	71321	9619	**4:10**	0:23	-	-	-	**107**	143	-	-	-
	1-18	87702	10599	**6:01**	0:26	-	-	-	**113**	151	-	-	-
	1-19	106820	11643	**8:56**	0:29	-	-	-	**119**	157	-	-	-
	1-20	128866	12719	**13:30**	0:33	-	-	-	**125**	165	-	-	-
Mystery	1-1	219	682	**0:15**	0:02	0:00	0:00	0:00	**5**	5	5	5	5
	1-2	-	-	-	0:67	0:12	0:00	0:06	-	22	9	9	9
	1-3	3272	12800	**1:04**	0:03	0:01	0:00	0:00	**4**	4	4	4	4
	1-4	310166	2702	**9:49**	-	-	-	-	∞	-	-	-	-
	1-9	50463	11950	**3:26**	0:07	0:01	0:00	0:01	**8**	8	8	8	8
	1-10	-	-	-	-	-	0:09	-	-	-	-	8	-
	1-11	2354	2330	**0:18**	0:04	0:00	0:00	0:00	**7**	12	7	7	7
	1-13	-	-	-	-	-	0:01	-	-	**16**	-	-	-
	1-17	-	-	-	0:86	-	0:01	0:03	-	**10**	4	4	4
	1-19	-	-	-	0:02	0:13	0:00	0:07	-	**9**	8	8	-
	1-20	-	-	-	-	0:56	-	-	-	**10**	-	-	-
	1-25	195	592	**0:15**	0:02	0:00	0:00	0:00	**4**	4	4	6	4
	1-26	18467	13187	**1:26**	0:11	0:01	0:00	0:01	**6**	13	6	10	8
	1-27	2739	4771	**0:34**	0:04	0:00	0:00	0:00	**5**	5	7	7	7
	1-28	1032	1137	**0:15**	0:02	0:00	0:00	0:00	**7**	18	7	7	7
	1-29	1238	5904	**0:44**	0:03	0:00	0:00	0:00	**4**	5	4	4	4
	1-30	-	-	-	1:40	0:09	0:07	0:05	-	**18**	13	13	12
Logistics	1-1	143718	2843	**3:28**	1:19	0:00	0:00	0:02	$\mathbf{26}^b$	43	26	27	27
	1-2	-	-	-	1:37	-	0:00	0:00	-	44	-	32	32
	1-5	30393	2839	**2:32**	0:14	15:47	0:00	-	$\mathbf{22}^b$	26	24	29	-
	1-7	-	-	-	13:08	-	-	-	$\mathbf{22}^b$	112	-	-	-
	1-11	-	-	-	1:26	-	-	-	$\mathbf{22}^b$	-	-	-	-
	2-1	1113	1328	**0:08**	0:04	0:00	0:00	0:00	$\mathbf{13}^b$	13	13	13	13
	2-2	1678	1286	**2:00**	0:05	0:00	0:00	0:00	$\mathbf{20}^b$	27	20	20	20
	2-3	115658	2511	**4:18**	0:10	0:00	0:00	0:00	$\mathbf{27}^b$	32	27	27	27
	2-4	-	-	-	2:50	-	-	-	-	59	-	-	-
	2-5	-	-	-	-	0:66	-	-	-	-	31	-	-
Mprime	1-1	769	1037	**0:19**	0:03	0:01	0:02	0:00	**5**	5	11	11	9
	1-2	-	-	-	1:50	0:15	0:02	0:06	-	12	9	13	9
	1-3	6244	13860	**2:00**	0:05	0:03	0:10	0:00	**4**	4	5	6	4
	1-4	20353	3574	**1:33**	0:03	0:01	-	0:00	**8**	10	12	6	4
	1-5	-	-	-	-	0:08	0:01	-	-	-	11	15	-
	1-7	-	-	-	0:02	0:01	0:01	-	8	-	5	6	-
	1-8	-	-	-	0:23	0:24	-	0:02	-	7	10	-	7
	1-9	134639	12423	**19:53**	0:07	0:03	0:02	0:01	**8**	8	9	16	8
	1-11	7607	2945	**0:46**	0:07	0:02	0:07	0:01	**7**	19	8	17	8
	1-12	15831	4874	**1:16**	0:51	0:04	-	0:02	**6**	10	9	-	10
	1-16	-	-	-	3:27	0:23	-	0:05	-	13	11	-	11
	1-17	-	-	-	10:08	-	-	0:07	-	4	-	-	4
	1-21	-	-	-	-	-	0:01	-	-	-	-	7	-
	1-25	419	775	**0:16**	0:05	0:00	0:00	0:00	**4**	4	4	6	4
	1-26	60612	14489	**8:40**	0:23	0:13	0:08	0:03	**6**	8	7	-	8
	1-27	8192	5459	**2:11**	0:67	0:08	-	0:03	**5**	8	8	-	8
	1-28	3236	1258	**0:22**	0:79	0:01	0:02	0:01	**7**	7	10	14	11
	1-29	2263	6763	**2:01**	0:02	0:05	0:13	0:02	**4**	6	5	7	5
	2-1	10775	41673	**6:07**	0:07	0:04	0:00	0:00	**4**	4	5	5	5
	2-2	20925	8997	**1:22**	0:12	0:02	0:04	0:02	**7**	8	8	9	7
	2-4	8672	26454	**5:45**	0:08	0:11	0:00	0:01	**4**	4	4	5	5
	2-5	2247	3025	**0:24**	0:04	0:02	-	0:00	**5**	6	6	-	6
Grid	2-1	?	?	?	0:09	-	0:02	0:12	$\mathbf{14}^\mu$	24	-	14	14
	2-2	?	?	?	-	-	-	-	$\mathbf{26}^\mu$	-	-	-	-
	2-4	?	?	?	-	-	-	-	$\mathbf{55}^{\mu h}$	-	-	-	-
	2-5	?	?	?	-	-	-	-	$\mathbf{37}^{\mu h}$	-	-	-	-

Table 1. Experimental results of *Mips*: μ checked by μcke, h heuristic, b bidirectional.