Directed Symbolic Exploration in AI-Planning

Stefan Edelkamp
Institut für Informatik
Albert-Ludwigs-Universität
Georges-Köhler-Allee
D-79110 Freiburg
edelkamp@informatik.uni-freiburg.de

Abstract

In this paper we study traditional and enhanced BDD-based exploration procedures capable of handling large planning problems. On the one hand, reachability analysis and model checking have eventually approached AI-Planning. Unfortunately, they typically rely on uninformed *blind* search. On the other hand, heuristic search and especially lower bound techniques have matured in effectively directing the exploration even for large problem spaces. Therefore, with heuristic symbolic search we address the unexplored middle ground between single state and symbolic planning engines to establish algorithms that can gain from both sides. To this end we implement and evaluate heuristics found in state-of-the-art heuristic single-state search planners.

Introduction

One currently very successful trend in deterministic fully-automated planning is heuristic search. The search space incorporates states as lists of instantiated predicates (also called atoms or fluents). The success of the heuristic search correlates with the quality of the estimate; the more informed the heuristic the better the achieved results. Heuristic search planners have outperformed other approaches on a sizable collection of deterministic domains. In the fully automated track of the AIPS-2000 planning competition (`http://www.cs.toronto.edu/aips2000`) chaired by Fahim Baccus the System FF (by Hoffmann) was awarded for outstanding performance while HSP2 (by Geffner and Bonet), STAN (by Fox and Long), and MIPS (by Edelkamp and Helmert) were placed shared second.

Historically, the first heuristic search planner was HSP (Bonet & Geffner 1999), which also competed in AIPS-1998. HSP computes the heuristic values of a state by adding (or maximizing) depth values for each fluent for an overestimating (or admissible) estimate. These values are retrieved from the fix point of a relaxed exploration. Since the technique is similar to the first phase of building the layered graph structure in GRAPHPLAN (Blum & Furst 1995), HSPr (Bonet, Loerincs, & Geffner 1997) extends the approach by regression/backward search and excludes *mutuals* similar to the original planning graph algorithm. In the competition version HSP2 of the planner the *max-pair heuristic* computes a distance value to the goal for each pair of atoms. The underlying search algorithm is a weighted version of A* (Pearl 1985) implementing a higher influence of the heuristic by the cost of non-optimal solutions. Due to the observed overhead at run-time, high-order heuristics have not been applied yet.

HSP has inspired the planners GRT (Refanidis & Vlahavas 1999) and FF (Hoffmann 2000) and influenced the development of the planners STAN and MIPS. In AIPS-2000 the heuristic of GRT was too weak to compete with the improvements applied in HSP2 and in FF (for fast-forward planning). FF solves a relaxed planning problem for *every* encountered state in a combined forward *and* backward traversal. Therefore, the *FF-Heuristic* is an elaboration to the *HSP-Heuristic*, since the latter only considers the first phase. The efforts in computing a very accurate heuristic estimate correlates with data in solving Sokoban (Junghanns 1999), which applies a $O(n^3)$ estimate, and suggests that even involved work for improving the heuristic pays off. With *enforced hill climbing* FF further employs another search strategy and reduces the explored portion of search space. It makes use of the fact that phenomena like big plateaus or local minima do not occur very often in benchmark planning problems. STAN is a hybrid of two strategies: The GRAPHPLAN-based algorithm and a forward planner using a heuristic function based on the length of the relaxed plan (as in HSP and FF). STAN performs a domain analysis techniques to select between these strategies. *Generic Types* automatically choose an appropriate algorithm for problem instance at hand (Long & Fox 2000).

An orthogonal approach in tackling huge search spaces is a symbolic representation of sets of states. The SATPLAN approach by Kautz and Selman (Kautz & Selman 1996) has shown that representional issues can be resolved by parsing the planning domain into a collection of Boolean formulae (one for each depth level). The system BLACKBOX (Kautz & Selman 1999), a hybrid planner based on merging SATPLAN with GRAPHPLAN, performed well on AIPS-1998, but failed to solve as many problems as the heuristic search planners on the domains in AIPS-2000. However, it should be denoted that the results of SATPLAN (GRAPHPLAN) are optimal in the number of sequential (parallel) steps, while heuristic search planners tend to overestimate in order to cope with state space sizes of 10^{20} and beyond.

Although efficient satisfiability solvers have been developed in the last decade, the blow-up in the size of the formulae even for simple planning domains calls for a concise representation. This leads to reduced ordered binary decision diagrams (BDDs) (Bryant 1985), an efficient data structure for Boolean functions. Through their unique representation BDDs are effectively applied to the synthesis and verification of hardware circuits (Bryant 1986) and incorporated within the area of *model checking* (Burch *et al.* 1992). Nowadays BDDs are a fundamental tool in various research areas of computer science and very recently BDDs are encountering AI-research topics like *heuristic search* (Edelkamp & Reffel 1998) and *planning* (Giunchiglia & Traverso 1999). The diverse research aspects of *traditional STRIPS planning* (Edelkamp & Reffel 1999), *non-deterministic planning* (Cimatti *et al.* 1997), *universal planning* (Cimatti, Roveri, & Traverso 1998), and *conformant planning* (Cimatti & Roveri 1999) indicate the wide range of BDD-related planning.

The planner MIPS (Edelkamp & Helmert 2000) uses BDDs to compactly store and maintain sets of propositionally represented states. The concise state representation is inferred in an analysis prior to the search and, by utilizing this representation, accurate reachability analysis and backward chaining are carried out without necessarily encountering exponential representation explosion. MIPS was originally designed to prove that BDD-based exploration methods are an efficient means for implementing a domain-independent planning system with some nice features, especially guaranteed optimality of the plans generated. If problems become harder and information on the solution length is available, MIPS invokes its incorporated heuristic single state search engine (similar to FF), thus featuring two entirely different planning algorithms, aimed to assist each other on the same state representation. The other two BDD planners in AIPS-2000, BDDPLAN (Holldobler & Stör 2000) and PROPPLAN (Fourman 2000), lack the precompiling phase of MIPS. Therefore, these approaches were too slow for traditional STRIPS problems. Moreover, a single state extension to their planners has not being provided. In the generalized ADL settings, however, PROPPLAN has proven to be effective compared with the FF approach, which solves more problems in less time, but fails to find optimal solutions.

This paper extends the idea of BDD representations and exploration in the context of heuristic search. The heuristic estimate is based on subpositions (called patterns) calculated prior to the search. Therefore, the heuristic is a form of a pattern database with planning patterns corresponding to (one or a collection of) fluents. This heuristic will be integrated into a BDD-based version of the A* algorithm, called BDDA*. Moreover, we alter the concept of BDDA* to *pure heuristic search* which seems to be more suited at least to some planning problems. Thereby, we allow non-optimistic heuristics and sacrifice optimality but succeed in searching larger problem spaces. The paper is structured as follows. First of all, we give a simple planning example and briefly introduce BDDs basics. Thereafter, we turn to the exploration algorithms, starting with blind search then turning to the directed approach BDDA*, its adaption to planning, and its refinement for *pure heuristic search*. We end with some experimental data and draw conclusions.

BDD Representation

Let us consider an example of a planning problem. A truck has to deliver a package from Los Angeles to San Francisco. In STRIPS notation the start state is given by `(PACKAGE package)`, `(TRUCK truck)`, `(LOCATION los-angeles)`, `(LOCATION san-francisco)`, `(AT package los-angeles)`, and `(AT truck los-angeles)` while the goal state is specified by `(AT package san-francisco)`. We have three operator schemas in the domain, namely `LOAD` (for loading a truck with a certain package at a certain location), `UNLOAD` (the inverse operation), and `DRIVE` (a certain truck from one city to another). The operator schemas are expressed in form of preconditions and effects.

The precompiler to infer a small state encoding consists of three phases (Edelkamp & Helmert 1999). In a first *constant predicate* phase it observes that the predicates `PACKAGE`, `TRUCK` and `LOCATION` remain unchanged by the operators. In the next *merging* phase the precompiler determines that `at` and `in` should be encoded together, since a `PACKAGE` can exclusively be at a `LOCATION` or in a `TRUCK`. By *fact space exploration* (a simplified but complete exploration of the planning space) the following fluents are generated: `(AT package los-angeles)`, `(AT package san-francisco)`, `(AT truck los-angeles)`, `(AT truck san-francisco)`, and `(IN package truck)`. This leads to a total encoding length of three bits. Using two bits x_0 and x_1 the fluents `(AT package los-angeles)`, `(AT package san-francisco)`, and `(IN package truck)` are encoded with 00, 01, and 10, respectively, while the variable x_2 represents the fluents `(AT truck los-angeles)` and `(AT truck san-francisco)`.

Therefore, a Boolean representation of the start state is given by $\overline{x_0} \wedge \overline{x_1} \wedge \overline{x_2}$ while the set of goal states is simply formalized with the expression $\overline{x_0} \wedge x_1$. More generally, for a set of states S the *characteristic function* $\phi_S(a)$ evaluates to *true* if a is the binary encoding of one state x in S. As the formula for the set of goal states indicate, the symbolic representation for a large set of states is typically smaller than the cardinality of the represented set.

Since the satisfiability problem for Boolean formulae is NP hard, binary decision diagrams are used to for their efficient and unique graph representation. The nodes in the directed acyclic graph structure are labeled with the variables to be tested. Two outgoing edges labeled *true* and *false* direct the evaluation process with the result found at one of the two sinks. We assume a fixed variable ordering on every path from the root node to the sink and that each variable is tested at most once. The BDD size can be exponential in the number of variables but, fortunately, this effect rarely appears in practice. The satisfiability test is trivial and given two BDDs G_f and G_g and a Boolean operator $\otimes$, the BDD $G_{f \otimes g}$ can be computed efficiently. The most important operation for exploration is the *relational product* of a set of variables v and two Boolean functions f and g. It is defined as $\exists v\ (f \wedge g)$. Since existential quantification of one variable x_i in a Boolean function f is equal to disjunction $f_{\overline{x_i}} \vee f_{x_i}$, the quantification of v results in a sequence of subproblem disjunctions. Although computing the relational product is NP-hard, specialized algorithms have been developed leading good results for many practical applications.

An operator can also be seen as an encoding of a set. The *transition relation* T is defined as the disjunction of the characteristic functions of all pairs (x', x) with x' being the predecessor of x. For the example problem, `(LOAD package truck los-angeles)` corresponds to the pair $(00|0, 10|0)$ and `(LOAD package truck san-francisco)` to $(01|1, 10|1)$. Subsequently, the `UNLOAD` operator is given by $(10|0, 00|0)$ and $(10|1, 10|1)$. The `DRIVE` action for the truck is represented by the strings $(00|*, 00|*)$ $(01|*, 01|*)$, and $(10*, 10|*)$ with $* \in \{0, 1\}$. For a concise BDD representation (cf. Figure 1) the variable ordering is chosen that the set of variable in x' and x are *interleaved*, i.e. given in alternating order.

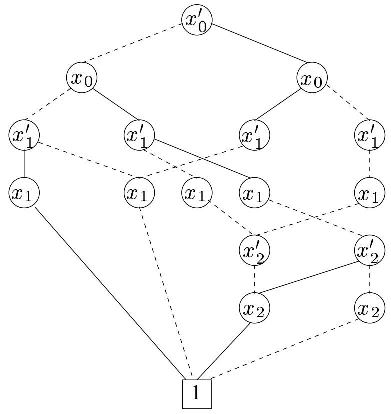

Figure 1: The transition relation for the example problem. For the sake of clarity, the *false* sink has been omitted. Dashed lines and solid lines indicate edges labeled *false* and *true*, respectively.

BDD-Based Blind Search

Let S_i be the set of states reachable from the initial state s in i steps, initialized by $S_0 = \{s\}$. The following equation determines ϕ_{S_i} given both $\phi_{S_{i-1}}$ and the transition relation:

$$\phi_{S_i}(x) = \exists x'\ (\phi_{S_{i-1}}(x') \wedge T(x', x)).$$

The formula calculating the successor function is a relational product. A state x belongs to S_i if it has a predecessor x' in the set S_{i-1} and there exists an operator which transforms x' into x. Note that on the right hand side of the equation ϕ depends on x' compared to x on the left hand side. Thus, it is necessary to substitute x with x' in ϕ_{S_i} beforehand, which can be achieved by a simple textual replacement of the node labels in the diagram structure. In order to terminate the search, we successively test, whether a state is represented in the intersection of the set S_i and the set of goal states G by testing the identity of $\phi_{S_i} \wedge \phi_G$ with the trivial zero function. Since we enumerated $S_0, \ldots, S_{i-1}$ the iteration index i is known to be the optimal solution length.

Let *Open* be the representation of the search horizon and *Succ* the BDD for the set of successors. Then the algorithm can be realized as the pseudo-code Figure 2 suggests.

procedure *Breadth-First Search*
  Open $\leftarrow \phi_{\{s\}}$
  do
    Succ $\leftarrow \exists x'\ (Open(x') \wedge T(x', x))$
    Open $\leftarrow$ *Succ*
  while (*Open* $\wedge\ \phi_G \equiv 0$)

Figure 2: Breadth-first search implemented with BDDs.

This simulates a breadth-first exploration and leads to three iterations for the example problem. We start with the initial state represented by a BDD of three inner nodes for the function $\overline{x_0} \wedge \overline{x_1} \wedge \overline{x_2}$. After the first iteration we get a BDD size of four representing three states and the function $(\overline{x_0} \wedge \overline{x_1}) \vee (x_0 \wedge \overline{x_1} \wedge \overline{x_2})$. The next iteration leads to four states in a BDD of one internal node for $\overline{x_1}$, while the last iteration results in a BDD containing a goal state.

Bidirectional Search

In backward search we start with the goal set B_0 and iterate until we encounter the start state. We take advantage of the fact that T has been defined as a relation. Therefore, we iterate according to the formula $\phi_{B_i}(x') = \exists x\ (\phi_{B_{i-1}}(x) \wedge T(x', x))$. 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 on the other hand the backward search frontier B_b with $B_0 = G$. When the two search frontiers meet ($\phi_{F_f} \wedge \phi_{B_b} \not\equiv 0$) we have found an optimal solution of length $f + b$. With the two horizons *fOpen* and *bOpen* the algorithm can be implemented as shown in Figure 3.

procedure *Bidirectional Breadth-First Search*
  fOpen $\leftarrow \phi_{\{s\}}$; *bOpen* $\leftarrow \phi_G$
  do
    if (*forward*)
      Succ $\leftarrow \exists x'\ (fOpen(x') \wedge T(x', x))$
      fOpen $\leftarrow$ *Succ*
    else
      Succ $\leftarrow \exists x\ (bOpen(x) \wedge T(x', x))$
      bOpen $\leftarrow$ *Succ*
  while (*fOpen* $\wedge$ *bOpen* $\equiv 0$)

Figure 3: Bidirectional BFS implemented with BDDs.

Forward Set Simplification

The introduction of a list *Closed* containing all states ever expanded is an apparent very common approach in single state exploration to avoid duplicates in the search. The memory structure is realized as a transposition table. For symbolic search this technique is called *forward set simplification* (cf. Figure 4).

procedure *Forward Set Simplification*
    Closed $\leftarrow$ *Open* $\leftarrow \phi_{\{s\}}$
    do
        Succ $\leftarrow \exists x' \; (\mathit{Open}(x') \wedge T(x', x))$
        Open $\leftarrow$ *Succ* $\wedge \neg$ *Closed*
        Closed $\leftarrow$ *Closed* $\vee$ *Succ*
    while (*Open* $\wedge \phi_G \equiv 0$)

Figure 4: Symbolic BFS with *forward set simplification*.

The effect in the given example is that after the first iteration the number of states shrinks from three to two while the new BDD for $(\overline{x_0} \wedge \overline{x_1} \wedge x_2) \vee (x_0 \wedge \overline{x_1} \wedge \overline{x_2})$ has five inner nodes. For the second iteration only one newly encountered state is left with three inner BDD nodes representing $x_0 \wedge \overline{x_1} \wedge \overline{x_2}$. Forward set simplification terminates the search in case of a complete planning space exploration. Note that any set in between the successor set *Succ* and the simplified successor set *Succ* $-$ *Closed* will be a valid choice for the horizon *Open* in the next iteration. Therefore, one may choose a set R that minimizes the BDD representation instead of minimizing the set of represented states. Without going into details we denote that such image size optimizing operators are available in several BDD packages (Coudert, Berthet, & Madre 1989).

BDD-Based Directed Search

Before turning to the BDD-based algorithms for directed search we take a brief look at Dijkstra's single-source shortest path algorithm, *Dijkstra* for short, which finds a solution path with minimal length within a weighted problem graph (Dijkstra 1959). *Dijkstra* differs from breadth-first search in ranking the states next to be expanded. A priority queue is used, in which the states are ordered with respect to an increasing f-value. Initially, the queue contains only the initial state s. In each step the state with the minimum merit f is dequeued and expanded. Then the successor states are inserted into the queue according to their newly determined f-value. The algorithm terminates when the dequeued element is a goal state and returns the minimal solution path.

As said, BDDs allow sets of states to be represented very efficiently. Therefore, the priority queue *Open* can be represented by a BDD based on tuples of the form (*value*, *state*). The variables should be ordered in a way which allows the most significant variables to be tested at the top. The variables for the encoding of *value* should have smaller indices than the variables encoding *state*, since this leads to small BDDs and allows an intuitive understanding of the BDD and its association with the priority queue.

Let the *weighted transition relation* $T(w, x', x)$ evaluates to 1 if and only if the step from x' to x has costs w (encoded in binary). The symbolic version of Dijkstra (cf. Figure 5)

procedure Symbolic-Version-of-Dijkstra
    $\mathit{Open}(f, x) \leftarrow \; (f = 0) \wedge \phi_{S^0}(x)$
    do
        $f_{\min} = \min\{f \mid f \wedge \mathit{Open} \neq \emptyset\}$
        $\mathit{Min}(x) \leftarrow \exists f \; (\mathit{Open} \wedge f = f_{\min})$
        $\mathit{Rest}(f, x) \leftarrow \mathit{Open} \wedge \neg \mathit{Min}$
        $\mathit{Succ}(f, x) \leftarrow \exists x', w \; (\mathit{Min}(x') \wedge$
            $T(w, x', x) \wedge \mathit{add}(f_{\min}, w, f))$
        Open $\leftarrow$ *Rest* $\vee$ *Succ*
    while (*Open* $\wedge \phi_G \equiv 0$)

Figure 5: Dijkstra's single-source shortest-path algorithm implemented with BDDs.

now reads as follows. The BDD *Open* is set to the representation of the start state with value zero. Until we find a goal state in each iteration we extract *all* states with minimal f-value $f_{\min}$, determine the successor set and update the priority queue. Successively, we compute the minimal f-value $f_{\min}$, the BDD *Min* of all states in the priority queue with value $f_{\min}$, and the BDD of the remaining set of states. If no goal state is found, the variables in *Min* are substituted as above before the (weighted) transition relation $T(w, x', x)$ is applied to determine the BDD for the set of successor states. To attach new f-values to this set we have to retain the old f-value $f_{\min}$ and in order to calculate $f = f_{\min} + w$. Finally, the BDD *Open* for the next iteration is obtained by the disjunction of the successor set with the remaining queue.

It remains to show how to perform the arithmetics using BDDs. Since the f-values are restricted to a finite domain, the Boolean function *add* with parameters a, b and c can be built being *true* if c is equal to the sum of a and b. A recursive calculation of $\mathit{add}(a, b, c)$ should be prefered:

$$\mathit{add}(a, b, c) = ((b = 0) \wedge \; (a = c)) \vee$$
$$\exists \, b', c' \; (\mathit{inc}(b', b) \; \wedge \; \mathit{inc}(c', c) \; \wedge \; \mathit{add}(a, b', c')),$$

with *inc* representing all pairs of the form $(i, i+1)$. Therefore, symbolic breadth-first search can be applied to determine the fixpoint of *add*.

Heuristic Pattern Databases

For symbolically constructing the heuristic function a simplification T' to the transition relation T that regains tractability of the state space is desirable. However, obvious simplification rules might not be available. Therefore, in heuristic search we often consider relaxations of the problem that result in subpositions. More formally, a state v is a *subposition* of another state u if and only if the characteristic function of u logically implies the characteristic function of v, e.g., $\phi_{\{u\}} = \overline{x_1} \wedge x_2 \wedge x_3 \wedge \overline{x_4} \wedge x_5$ and $\phi_{\{v\}} = x_2 \wedge x_3$ results in $\phi_{\{u\}} \Rightarrow \phi_{\{v\}}$. As a simple example take the Manhattan distance in sliding tile solitaire games like the famous Fifteen-Puzzle. It is the sum of solutions of single tile problems that occur in the overall puzzle.

More generally, a *heuristic pattern data base* is a collection of pairs of the form (*value*, *pattern*) found by optimally solving problem relaxations that respect the subposition property (Culberson & Schaeffer 1996). The solution lengths of the patterns are then combined to an overall heuristic by taking the maximum (leading to an admissible heuristic) or the sum of the individual values (in which case we overestimate).

Heuristic pattern data bases have been effectively applied in the domains of Sokoban (Junghanns 1999), to the Fifteen-Puzzle (Culberson & Schaeffer 1996), and to Rubik's Cube (Korf 1997). In single-state search heuristic pattern databases are implemented by hash table, but in symbolic search we have to construct the estimator symollically, only using logical combinators and Boolean quantification.

Since heuristic search itself can be considered as the matter of introducing lower bound relaxations into the search process, in the following we will maximize the relaxed solution path values. The maximizing relation $max(a, b, c)$, evaluates to 1 if c is the maximum of a and b and is based on the relation *greater*, since

$$max(a, b, c) = (greater(a, b) \wedge\ (a = c)) \vee (\neg greater(a, b) \wedge\ (b = c)).$$

The relation $greater(a, b)$ itself might be implemeted by existential quantifying the add relation:

$$greater(a, b) = \exists t\ add(b, t, a)$$

Next we will find a way to automatically infer the heuristic estimate. To combine n fluent pattern $p_1, \ldots, p_n$ with estimated distances $d_1, \ldots, d_n$ to the goal we use $n + 1$ additional slack variables $t_0, \ldots, t_n$ which are existenially quantified later on. We define subfunctions H_i of the form

$$H_i(t_i, t_{i+1}, state) = (\neg\, p_i\ \wedge\ (t_i = t_{i+1})) \vee (p_i\ \wedge\ max(d_i, t_i, t_{i+1})),$$

with $H_i(t_i, t_{i+1}, state)$ denoting the following relation: If the accumulated heuristic value up to fluent i is t_i, then the accumulated value including fluent i is t_{i+1}. Therefore, we can combine the subfunctions to the overall heuristic estimate as follows.

$$H(value, state) = \exists\, t_1, \ldots, t_n$$
$$(t_0 = 0) \wedge\ H(t_n, value, state)\ \wedge\ \bigwedge_{i=0}^{n-1} H_i(t_i, t_{i+1}, state).$$

In some problem graphs subpositions or patterns might constitute a feature in which every position containing it is unsolvable. These *deadlocks* are frequent in directed search problems like Sokoban and can be learned domain or problem specifically. Deadlocks are heuristic patterns with an infinite heuristic estimate. Therefore, a deadlock table DT is the disjunction of the characteristic functions according to subpositions that are unsolvable. The integration of *deadlock tables* in the search algorithm is quite simple. For the BDD for DT we assign the new horizon *Open* as

$$Open \wedge \neg(Open \Rightarrow DT) = Open \wedge \neg DT.$$

The simplest patterns in planning are fluents. The estimated distance of each single fluent p to the goal is a heuristic value associated with p. We examine two heuristics.

HSP-Heuristic: In HSP the values are recursively calculated by the formula $h(p) = min\{h(p), 1 + h(C)\}$ where $h(C)$ is the cost of achieving the conjunct C, which in case of HSPr is the list of preconditions. For determining the heuristic the planning space has been simplified by omitting the delete effects. The algorithms in HSP and HSPr are variants of pure heuristic search incorporated with restarts, plateau moves, and overestimation.

The exploration phase to minimize the state description length in our planner has been extended to output an estimate $h(p)$ for each fluent p. Since we avoid duplicate fluents in the breadth-first *fact-space-exploration*, with each encountered fluent we associate a depth by adding the value 1 to its predecessor. The quality of the achieved distance values are not as good as in HSPr since we are not concerned about mutual exclusions in any form. Giving the list of value/fluent pairs a symbolic representation of the sub-relations and the overall heuristic is computed.

In the example `(AT ball los-angeles)` and `(AT truck los-angeles)` have distance 0 from the initial state `(AT truck san-francisco)` `(IN ball truck)` have a depth of one and `(AT ball san-francisco)` has depth two. Figure 6 depicts the BDD representation of the overall heuristic function.

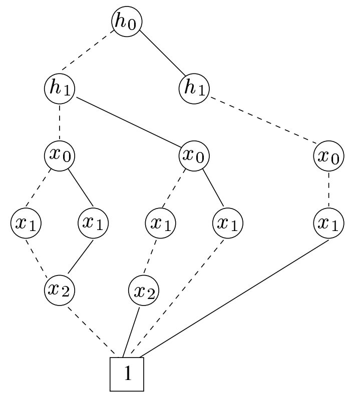

Figure 6: The BDD representation for the heuristic function in the example problem. In this case the individual pattern values have been maximized.

FF-Heuristic: FF solves the relaxed planning problem (delete-facts omitted) with GRAPHPLAN on-line for each state. For estimation FF builds the plan graph *and* extracts a simplified solution by counting the number of instantiated operators that at least have to fire Since the branching factor is large (one state has up to hundreds of successors) by determining *helpful actions*, only a *relevant* part of all successors is considered. The overall search phase is entitled *enforced hill-climbing*. Until the next smaller heuristic value is found a breadth-first search is invoked. Then the search process iterates with one state evaluating to this value.

In our planner we have (re-)implemented the FF-approach both to have an efficient heuristic single-state search engine at hand and to build an improved estimate for symbolic search. Since the FF approach is based on states and not on fluents, we cannot directly infer a symbolic version of the heuristic. We have to weaken the state-dependent character of the heuristic down to fluents. Moreover, simplifying the start state to a fluent may give no heuristic value at all, since the goal will not necessarily be reached by the relaxed exploration. Therefore, the estimate for each fluent is calculated by partitioning the goal state instead. Since we get improved distance estimates with respect to the initial state, we obtain a heuristic for backward search. However this is no limitation, since the concept of STRIPS operators can be inverted, yielding a heuristic in the usual direction.

BDDA*

In *informed search* with every state in the search space we associate a lower bound estimate h. By reweighting the edges the algorithm of Dijkstra can be transformed into A*. The new weight $\hat{w}$ is set to the old one w minus the h-value of the source node x', plus the value of the target node x resulting in the equation $\hat{w}(x', x) = w(x', x) - h(x') + h(x)$. The length of the shortest paths will be preserved and no new negative weighted cycle is introduced (Cormen, Leiserson, & Rivest 1990). More formally, if we denote $\delta(s, g)$ for the length of the shortest path from s to a goal state g in the original graph, and $\hat{\delta}(s, g)$ the shortest path in the reweigthed graph then $w(p) = \delta(s, g)$ if and only if $\hat{w}(p) = \hat{\delta}(s, g)$.

The rank of a node is the combined value $f = g + h$ of the generating path length g and the estimate h. The information h allows us to search in the direction of the goal and its quality mainly influences the number of nodes to be expanded until the goal is reached.

In the symbolic version of A*, called BDDA*, the relational product algorithm determines all successor states in one evaluation step. It remains to determine their values. For the dequeued state x' we have $f(x') = g(x') + h(x')$. Since we can access f, but usually not g, the new value $f(x)$ of a successor x has to be calculated in the following way

$$f(x) = g(x) + h(x) = g(x') + w(x', x) + h(x) =$$

$$f(x') + w(x', x) - h(x') + h(x).$$

The estimator H can be seen as a relation of tuples (*value*, *state*) which is *true* if and only if h(*state*)=*value*. We assume that H can be represented as a BDD for the entire problem space. The cost values of the successor set are calculated according to the equation mentioned above. The arithmetics for *formula*(h', h, w, f', f) based on the old and new heuristic value (h' and h, respectively), and the old and new merit (f' and f, respectively) are given as follows.

$$\mathit{formula}(h', h, w, f', f) = \exists\, t_1, t_2\ \mathit{add}(t_1, h', f') \wedge$$

$$\mathit{add}(t_1, w, t_2)\ \wedge\ \mathit{add}(h, t_2, f).$$

The implementation of BDDA* is depicted in Figure 7. Since all successor states are reinserted in the queue we expand the search tree in best-first manner. Optimality and completeness is inherited by the fact that given an optimistic heuristic A* will find an optimal solution.

Given a uniform weighted problem graph and a consistent heuristic the worst-case number of iterations in BDDA* is $O(f^{*2})$, with f^* being the optimal solution length (Edelkamp & Reffel 1998). In (a moderately difficult instance to) the Fifteen-Puzzle, the 4×4 version of the well-known sliding-tile (n^2-1)-Puzzles, a minimal solution was found by BDDA* within 176 iterations. With a breadth-first search approach it was impossible to find any solutions because of memory limitations. Already after 19 iterations more than 1 million BDD-nodes were needed to represent more than 1.4 million states.

procedure BDDA*
$\mathit{Open}(f, x) \leftarrow H(f, x) \wedge \phi_{S^0}(x)$
do
  $f_{\min} = \min\{f \mid f \wedge \mathit{Open} \neq \emptyset\}$
  $\mathit{Min}(x) \leftarrow \exists f\ (\mathit{Open} \wedge f = f_{\min})$
  $\mathit{Rest}(f, x) \leftarrow \mathit{Open} \wedge \neg\, \mathit{Min}$
  $\mathit{Succ}(f, x) \leftarrow \exists w, x'\ (\mathit{Min}(x') \wedge T(w, x', x) \wedge$
    $\exists h'\ (H(h', x') \wedge \exists h\ (H(h, x) \wedge$
    $\mathit{formula}(h', h, w, f_{\min}, f))))$
  $\mathit{Open} \leftarrow \mathit{Rest} \vee \mathit{Succ}$
while $(\mathit{Open} \wedge \phi_G \equiv 0)$

Figure 7: A* implemented with BDDs.

To find the minimal solution in the first problem to Sokoban (6 balls) the BDDA* algorithm was invoked with a very poor heuristic, counting the number of balls not on a goal position. Breadth-first search finds the optimal solution with a peak BDD of 250,000 nodes representing 61,000,000 states in the optimal number of 230 iterations. BDDA* with the heuristic leads to 419 iterations and to a peak BDD of 68,000 nodes representing 4,300,00 states. Note that even with such a poor heuristic, the number of nodes expanded by BDDA* is significantly smaller than in a breadth-first-search approach and their representation is more memory efficient. The number of represented states is up to 250 times larger than the number of BDD nodes.

Best-First-Search

A variant of BDDA*, called *Symbolic Best-First-Search*, can be obtained by ordering the priority queue only according to the h values. In this case the calculation of the successor relation simplifies to $\exists x'\ (\mathit{Min}(x') \wedge T(x', x) \wedge H(f, x))$ as shown in Figure 8. The old f-value are replaced.

procedure Best-First-Search
$\mathit{Open} \leftarrow H(f, x) \wedge \phi_{S^0}$
do
  $f_{\min} = \min\{f \mid f \wedge \mathit{Open} \neq \emptyset\}$
  $\mathit{Min}(x) \leftarrow \exists f\ \mathit{Open} \wedge f = f_{\min}$
  $\mathit{Rest}(f, x) \leftarrow \mathit{Open} \wedge \neg\, \mathit{Min}$
  $\mathit{Succ} \leftarrow \exists x'\ (\mathit{Min}(x') \wedge T(x', x) \wedge H(f, x))$
  $\mathit{Open} \leftarrow \mathit{Rest} \vee \mathit{Succ}$
while $(\mathit{Open} \wedge \phi_G \equiv 0)$

Figure 8: Best-first search implemented with BDDs.

Unfortunately, even for an optimistic heuristic the algorithm is not admissible and, therefore, will not necessarily find an optimal solution. The hope is that in huge problem spaces the estimate is good enough to lead the solver into a promising goal direction. Therefore, especially heuristics with overestimizations can support this aim.

On solution paths the heuristic values eventually decrease. Hence, Best-first search profits from the fact that the most promising states are in the front of the priority queue, have a smaller BDD representation, and are explored first. This compares to BDDA* in which the combined merit on the solution paths eventually increases. A good trade-off between exploitation and exploration has to be found. In FF breadth-first search for the next heuristic estimate consolidates pure heuristic search for a complete search strategy.

Figure 9 depicts the different dequeued BDDs *Min* together with the encoded heuristic in the exploration phase of *Pure* BDDA* for the example problem.

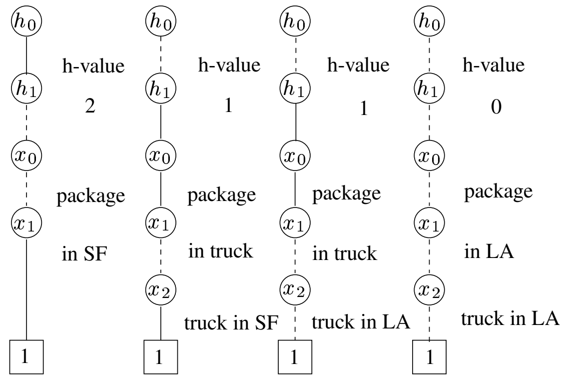

Figure 9: Backward exploration of the example problem in *Pure* BDDA*. In each iteration step the BDD *Min* with associated h-value is shown. Note that when using forward set simplification these BDDs additionally correspond to a snapshot of the priority queue *Open*.

Experiments

From given results on the different heuristic search planners (Hoffmann 2000) it can be obtained that heuristics pay off best in the *Gripper* and the *Logistics* domain.

Gripper

The effect of forward set simplification and optimization can best be studied in the scalable *Gripper* domain depicted in Table 1[1] When the problem instances get larger the additional computations pay off. In *Gripper* bidirectional search leads to no advantage since due to the symmetry of the problem the climax of the BDD sizes is achieved in the middle of the exploration. This is an important advantage to BDD-based exploration: Although the number of states grows continuously, the BDD representation might settle and become smaller. The data further suggests that optimizing the BDD structure with the proposed optimization is helpful only in large problems. *Gripper* is not a problem to *BDD*-based search, whereas it is suprisingly hard for GRAPHPLAN.

	Solution Length	BFS	+B	+BF	+BFO
1-1	11	0.00	0.01	0.01	0.01
1-2	17	0.01	0.01	0.02	0.02
1-3	23	0.02	0.03	0.02	0.02
1-4	29	0.03	0.03	0.04	0.04
1-5	35	0.04	0.04	0.07	0.07
1-6	41	0.06	0.06	0.08	0.08
1-7	47	0.08	0.08	0.11	0.14
1-8	53	0.12	0.13	0.19	0.20
1-9	59	0.35	0.36	1.33	1.58
1-10	65	0.72	1.93	2.06	2.15
1-11	71	1.27	2.33	2.36	2.43
1-12	77	1.95	3.21	3.05	3.13
1-13	83	2.80	3.91	3.48	3.49
1-14	89	3.80	5.04	4.28	4.36
1-15	95	4.93	6.26	5.29	5.43
1-16	101	6.32	7.21	6.41	6.07
1-17	107	7.72	8.94	7.26	7.52
1-18	113	9.82	10.91	8.65	8.61
1-19	119	24.73	26.11	15.28	15.35
1-20	125	34.59	36.73	20.41	20.08

Table 1: Solution lengths and computation times in solving Gripper with breadth-first bidirectional search, forward set simplification and optimization; B abbreviates *bidirectional search*, O denotes BDD image *optimization*, and F is *forward set simplification*.

Logistics

Due to the first round results in AIPS-2000 it can be deduced that FF's, STAN's and MIPS's heuristic single search engine are state-of-the-art in this domain, but Logistics problems turn out to be suprisingly hard for BDD exploration and therefore a good benchmark domain for BDD inventions. For example Jensen's BDD-based planning system, called UMOP, fails to solve any of the AIPS-1998 (first-round) problems (Jensen & Veloso 2000) and breadth-first search in *MIPS* yields only two domains to be solved optimally. This is due to high parallelism in the plans, since optimal parallel (Graphplan-based) planners, like IPP (by Köhler), Blackbox (by Kautz and Selman), Graphplan (by Blum and Furst), STAN (by Fox and Long) perform well on Logistics. Note, that heuristic search planners, such as (parallel) HSP2 with an IDA* like search engine loose their perfomance gains when optimality has to be preserved.

With best-first-search and the FF-Heuristic, however, we can solve 11 of the 30 problem instances. The dauting problem is that – due to the large minimized encoding size of the problems – the transition function becomes too large to be build. Therefore, the Logistics benchmark suite in the *Blackbox* distribution and in AIPS-2000 scale better. In AIPS-2000 we can solve the entire first set of problems with heuristic sybolic search and Table 2 visulizes the effect of

[1]The CPU-times in the experiments are given in seconds on a Linux-PC (Pentium III/450 MHz/128 MByte).

best-first search for the *Logistics* suite of the *Blackbox* distribution, in which all 30 problems have encodings of less than 100 bits. We measured the time, and the length of the found solution. H^{HSP}_{add} and H^{HSP}_{max} abbreviate best-first search according to the *add* and the *max* relation in the HSP-heuristic, respectively. H^{FF}_{add} and H^{FF}_{max} are defined analogously. The depicted times are not containing the efforts for determining the heuristic functions, which takes about a few seconds for each problem. Obviously, searching with the *max*-Heuristic achieves a better solution quality, but on the other hand it takes by far more time. The data indicates that on average the *FF-Heuristic* leads to shorter solutions and to smaller execution times. This was expected, since the average heuristic value per fluent in H^{FF} is larger than in H^{HSP}, e.g. in the first problem it increases from 2.96 to 4.43 and on the whole set we measured an average increase of 41.25 % for the estimate.

	BFS		H^{HSP}_{add}		H^{HSP}_{max}		H^{FF}_{add}		H^{FF}_{max}	
1	25	0.66	30	0.06	25	1.05	30	0.92	25	0.49
2	24	121	27	5.33	24	129	31	1.27	26	3.52
3	-	-	29	3.30	26	35.98	28	1.18	26	30.22
4	-	-	59	6.53	52	37.10	59	3.49	52	22.74
5	-	-	52	5.64	42	4.56	51	3.11	43	3.41
6	42	72	63	7.22	51	67.18	64	2.45	52	11.37
7	-	-	83	14.89	-	-	80	11.87	-	-
8	-	-	84	19.14	-	-	80	15.05	-	-
9	-	-	84	13.07	-	-	80	8.94	-	-
10	-	-	47	13.93	40	484	45	8.15	40	421
11	-	-	54	10.10	-	-	52	7.30	-	-
12	-	-	37	1.19	-	-	36	3.90	-	-
13	-	-	77	15.18	-	-	78	9.89	-	-
14	-	-	74	18.58	-	-	83	13.36	-	-
15	-	-	64	17.16	-	-	68	10.08	-	-
16	39	580	49	7.19	41	4.64	46	2.78	40	1.73
17	43	277	51	9.97	43	3.91	50	2.60	43	3.38
18	-	-	56	21.53	-	-	54	15.76	-	-
19	-	-	53	12.85	-	-	57	8.01	-	-
20	-	-	101	20.42	-	-	95	13.58	-	-
21	-	-	73	16.16	-	-	69	10.47	-	-
22	-	-	94	18.45	-	-	87	14.54	-	-
23	-	-	72	13.95	-	-	71	10.81	-	-
24	-	-	79	14.18	-	-	75	9.50	-	-
25	-	-	73	14.81	-	-	66	9.03	-	-
26	-	-	60	14.23	-	-	61	9.35	-	-
27	-	-	81	15.31	-	-	80	12.72	-	-
28	-	-	87	27.15	-	-	89	23.74	-	-
29	-	-	51	21.58	-	-	52	16.70	-	-
30	-	-	59	13.41	-	-	59	9.61	-	-

Table 2: Solution lengths and computation times in solving Logistics with best-first search.

The backward search component - here applied in the regression space (thus corresponding to forward search in progression space) is used as a breadth-first *target enlargement*. With higher search tree depths this approach definitely profits from the symbolic representation of states.

In best-first-search forward simplification is used to avoid recurrences in the set of expanded states. However, if the set of reachable states from the first bucket in the priority queue returns with failure, we are not done, since the set of goal states according to the minimum may not be reachable.

Planning as Model Checking

The model checking problem determines whether a formula is true in a concrete model and is based on the following issues (Giunchiglia & Traverso 1999):

1. A domain of interest (e.g, a computer program or a reactive system) is described by a formal model.
2. A desired property of finite domain (e.g. a specification of a program, a safety requirement for a reactive system) is described by a formula typically using temporal logic.
3. The fact that a domain satisfies a desired property (e. g. that a reactive system never ends up in a dangerous state) is determined by checking whether or not the formula is true in the initial state of the model.

The crucial observation is that exploring (deterministic or non-deterministic) planning problem spaces is in fact a model checking problem. In model checking the assumed structure is described as a Kripke structure (W, W_0, T, L), where W is the set of states, W_0 the set of initial states, T the transition relation and L a labeling function that assigns to each state the set of atomic propositions which evaluate to *true* in this state. The properties are usually stated in a temporal formalism like linear time logic LTL (used in SPIN) or branching time logic CTL eventually enhanced with fairness constraints. In practice, however, the characteristics people mainly try to verify are simple safety properties expressible in all of the logics mentioned above. They can be checked through a simple calculation of all reachable states. An iterative calculation of Boolean expressions has to be performed to verify the formula *EF Goal* in the temporal logic CTL which is dual to the verification of *AG* $\neg$*Goal*. The computation of a (minimal) witness delivers a solution. BDD-based planning approaches are capable of dealing with non-deterministic domains (Cimatti, Roveri, & Traverso 1998; Ferraris & Guinchiglia 2000). Due to the non-determinism the authors refer to plans as complete state action tables. Therefore, actions are included in the transition relation, resulting in a representation of the form $T(\alpha, x', x)$. The concept of *strong cyclic plans* expresses that from each state on a path a goal state is eventually reachable.

Conclusion and Outlook

Symbolic breadth-first search and BDDA* have been applied to the areas *search* (Edelkamp & Reffel 1998) and *model checking* (Reffel & Edelkamp 1999). The experiments in *(heuristic) search* indicate the potential power of the symbolic exploration technique (in Sokoban) and the lower bound information (in the Fifteen Puzzle). In *model checking* we encounter a real-world problem of finding errors in hardware devices. BDD sizes of 25 million nodes reveal that even with symbolic representations we operate at the limit of main memory. However, this study of domain independent *planning* proves the generality of BDDA*.

The BDD representation of the space allows to reduce the planning problem to model checking: reachability analysis verifies the formula **EF** *Goal* in the temporal logic CTL.

The directed BDD-based search techniques bridge the gap between heuristic search planners and symbolic methods. Especially, best-first search and the FF-heuristic seem very promising to be studied in more detail and to be evaluated in other application areas. Due to off-line computation and restriction to one atom in the planning patterns, the symbolic HSP- and FF-heuristics are not as informative as their respective single-state correspondants in FF and HSP2, but, nevertheless, lead to good results.

The extension of the approach to planning patterns with more than one encoded atom is challenging. One possibility is a regressive breadth-first exploration through an abstraction of the state-space to build a pattern-estimate data-base. In (Edelkamp 2001) we show how this approach leads to a very good admissible estimate in explicite search. We have experimented with an extension to the *max-pair heuristic* with a weighted bipartite minimum-matching, but explicite pattern data bases lead to better results. Together with the wide range of applicability, we expect that with the same heuristic information a symbolic planner is competitive with a explicite one if at least moderate-sized sets of states have to be explored. In future we will also consider other symbolic heuristic search algorithms such as *Symbolic Hill-Climbing* and *Sybolic Weighted A**.

Acknowledgments We thank F. Reffel and M. Helmert for their cooperation concerning this research. The work is supported by DFG in a project entitled *Heuristic Search and Its Application in Protocol Verification*.

References

Blum, A., and Furst, M. L. 1995. Fast planning through planning graph analysis. In *IJCAI*, 1636–1642.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *ECP*, 359–371.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *AAAI*, 714–719.

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

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers* 35:677–691.

Burch, J. R.; M.Clarke, E.; McMillian, K. L.; and Hwang, J. 1992. Symbolic model checking: 10^{20} states and beyond. *Information and Computation* 98(2):142–170.

Cimatti, A., and Roveri, M. 1999. Conformant planning via model checking. In *ECP*, 21–33.

Cimatti, A.; Giunchiglia, E.; Giunchiglia, F.; and Traverso, P. 1997. Planning via model checking: A decision procedure for AR. In *ECP*.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *AAAI*, 875–881.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms*. The MIT Press.

Coudert, O.; Berthet, C.; and Madre, J. 1989. Verification of synchronous sequential machines using symbolic execution. In *Automatic Verification Methods for Finite State Machines*, 365–373.

Culberson, J. C., and Schaeffer, J. 1996. Searching with pattern databases. In *CSCSI*, 402–416.

Dijkstra, E. W. 1959. A note on two problems in connection with graphs. *Numerische Mathematik* 1:269–271.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *ECP*, 135–147.

Edelkamp, S., and Helmert, M. 2000. On the implementation of Mips. In *AIPS-Workshop on Model-Theoretic Approaches to Planning*, 18–25.

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

Edelkamp, S., and Reffel, F. 1999. Deterministic state space planning with BDDs. In *ECP*, 381–382.

Edelkamp, S. 2001. Planning with pattern databases. Submitted to IJCAI-01.

Ferraris, P., and Guinchiglia, E. 2000. Planning as satisfiability in simple nondeterministc domains. In *AIPS-Workshop on Model-Theoretic Approaches to Planning*, 10–17.

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

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In *ECP*, 1–19.

Hoffmann, J. 2000. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Symposium on Methodologies for Intelligent Systems*.

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

Jensen, R. M., and Veloso, M. 2000. OBDD-based deterministic planning using the UMOP planning framework. In *AIPS-Workshop on Model-Theoretic Approaches to Planning*, 26–31.

Junghanns, A. 1999. *Pushing the Limits: New Developments in Single-Agent Search*. Ph.D. Dissertation, University of Alberta.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning propositional logic, and stochastic search. In *AAAI*, 1194–1201.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and Graph-based planning. In *IJCAI*, 318–325.

Korf, R. E. 1997. Finding optimal solutions to Rubik's cube using pattern databases. In *AAAI*, 700–705.

Long, D., and Fox, M. 2000. Automatic synthesis and use of generic types in planning. In *AIPS*, 196–205.

Pearl, J. 1985. *Heuristics*. Addison-Wesley.

Refanidis, I., and Vlahavas, I. 1999. A domain independent heuristic for STRIPS worlds based on greedy regression tables. In *ECP*, 346–358.

Reffel, F., and Edelkamp, S. 1999. Error detection with directed symbolic model checking. In *FM*, 195–211.