

Heuristic Search + Symbolic Model Checking = Efficient Conformant Planning

Piergiorgio Bertoli¹, Alessandro Cimatti¹, Marco Roveri^{1,2}

¹ ITC-IRST, Via Sommarive 18, 38055 Povo, Trento, Italy
{bertoli,cimatti,roveri}@irst.itc.it

² DSI, University of Milano, Via Comelico 39, 20135 Milano, Italy

Abstract

Planning in nondeterministic domains has gained more and more importance. Conformant planning is the problem of finding a sequential plan that guarantees the achievement of a goal regardless of the initial uncertainty and of nondeterministic action effects. In this paper, we present a new and efficient approach to conformant planning. The search paradigm, called heuristic-symbolic search, relies on a tight integration of symbolic techniques, based on the use of Binary Decision Diagrams, and heuristic search, driven by selection functions taking into account the degree of uncertainty. An extensive experimental evaluation of our planner HSCP against the state of the art conformant planners shows that our approach is extremely effective. In terms of search time, HSCP gains up to three orders of magnitude over the breadth-first, symbolic approach of CMBP, and up to five orders of magnitude over the heuristic search of GPT, requiring, at the same time, a much lower amount of memory.

1 Introduction

Planning in nondeterministic domains is being recognized as increasingly important, and much harder than classical planning. In order to find plans that guarantee the achievement of the goal, it is necessary to deal with uncertainty in the initial condition, and with nondeterministic action effects. Several approaches to different planning problems in nondeterministic domains have been recently proposed. In [Pryor and Collins, 1996; Kabanza *et al.*, 1997; Weld *et al.*, 1998; Cimatti *et al.*, 1998; Rintanen, 1999] the problem of producing a conditional plan under the hypothesis of (total or partial) run-time observability has been considered. Conformant planning [Goldman and Boddy, 1996] is the problem of finding a sequential plan that will achieve the goal assuming that no information will be available at run-time. Conformant planning has also a close relation with the problem of finding synchronization sequences in hardware circuits [Kohavi, 1978].

Conformant planning has been recently tackled in different ways. The first efficient approach, based on GRAPH-PLAN, was presented in [Smith and Weld, 1998]. Bonet and Geffner [2000] formulate conformant planning as a problem

of search in the space of belief states. The approach, implemented within the GPT system, relies on the use of heuristics to drive an A^* -style search algorithm. The state of the art in conformant planning is [Cimatti and Roveri, 2000]. This approach extends symbolic model checking techniques, to compactly represent and efficiently explore the search space. Although the approach inherits from symbolic model checking a blind, breadth-first search style, the CMBP planner shows surprising efficiency, the reported results being superior to the heuristic search of GPT, sometimes by more than two orders of magnitude.

In this paper, we present a new, *heuristic-symbolic* search paradigm, based on the combination of heuristic search with ideas taken from symbolic model checking. We modify and extend the data structures defined in [Cimatti and Roveri, 2000], implementing the new planner HSCP (Heuristic-Symbolic Conformant Planner). Heuristic-symbolic search overcomes the bottleneck of the breadth-first approach while retaining the advantages of symbolic techniques: by means of a simple, domain-independent heuristic, HSCP outperforms CMBP, gaining up to three orders of magnitude in search time, with a much lower memory consumption.

This paper is structured as follows. We first define the problem of conformant planning. Then, we present the combined use of symbolic and heuristic mechanisms for the representation and exploration of the search space. We describe the planning algorithm, and present an experimental analysis comparing HSCP with CMBP and GPT. Finally, we discuss some other related work and draw the conclusions.

2 Conformant Planning

We consider nondeterministic planning domains, where actions can have preconditions, conditional and uncertain effects, and the initial state can be only partly specified.

Definition 1 (Planning Domain) A *Planning Domain* is a 4-tuple $\mathcal{D} = (\mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R})$, where $\mathcal{P}$ is the (finite) set of atomic propositions, $\mathcal{S} \subseteq \text{Pow}(\mathcal{P})$ is the set of states, $\mathcal{A}$ is the (finite) set of actions, and $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the transition relation.

Intuitively, a proposition is in a state if and only if it holds in that state. In the following we assume that a planning domain $\mathcal{D}$ is given. We use s , s' and s'' to denote states of $\mathcal{D}$, and α to denote actions. $\mathcal{R}(s, \alpha, s')$ holds iff when executing the action α in the state s the state s' is a possible outcome. We say that an action α is applicable in s iff there is at least one state s' such that $\mathcal{R}(s, \alpha, s')$ holds. We say that an action

α is deterministic in s iff there is a unique state s' such that $\mathcal{R}(s, \alpha, s')$ holds. An action α has an uncertain outcome in s if there are at least two distinct states s' and s'' such that $\mathcal{R}(s, \alpha, s')$ and $\mathcal{R}(s, \alpha, s'')$ hold.

Plans are sequences of actions, i.e. elements of $\mathcal{A}^*$. We use ϵ for the 0-length plan, π and ρ to denote plans, and $\pi; \rho$ for concatenation. Conformant planning is the problem of finding a plan that, if executed in any initial state in $\mathcal{I} \subseteq \mathcal{S}$, takes the domain into a set of states $\mathcal{G} \subseteq \mathcal{S}$, regardless of non-deterministic action effects. Following [Bonet and Geffner, 2000], we model this problem as search in the space of *belief states*. A belief state is a set of states, intuitively expressing a condition of uncertainty, by collecting together all the states that are equally possible. Conformant planning amounts to searching $\text{Pow}(\mathcal{S})$, i.e. the powerset of the set of states of the domain. In order to tackle this problem, we need the ability to define the applicability and effect of actions in a belief state, under a condition of uncertainty.

Definition 2 An action α is applicable in a belief state $\emptyset \neq Bs \subseteq \mathcal{S}$ iff α is applicable in every state in Bs . If α is applicable in Bs , its execution $\text{Exec}[\alpha](Bs)$ is the set $\{s' \mid \mathcal{R}(s, \alpha, s'), \text{ and } s \in Bs\}$. The execution of a plan π in a belief state, written $\text{Exec}[\pi](Bs)$, is defined as follows.

$$\begin{aligned} \text{Exec}[\epsilon](Bs) &\doteq Bs \\ \text{Exec}[\pi](\emptyset) &\doteq \emptyset \\ \text{Exec}[\alpha; \pi](Bs) &\doteq \emptyset, \text{ if } \alpha \text{ is not applicable in } Bs \\ \text{Exec}[\alpha; \pi](Bs) &\doteq \text{Exec}[\pi](\text{Exec}[\alpha](Bs)), \text{ otherwise} \end{aligned}$$

We say that a plan is applicable in a belief state when its execution is not empty. For a conformant planning problem, solutions are applicable plans, for which all the final states must be goal states.

Definition 3 Let $\emptyset \neq \mathcal{I} \subseteq \mathcal{S}$ and $\emptyset \neq \mathcal{G} \subseteq \mathcal{S}$. The plan π is a conformant solution to the problem $\langle \mathcal{I} : \mathcal{G} \rangle$ iff $\emptyset \neq \text{Exec}[\pi](\mathcal{I}) \subseteq \mathcal{G}$.

3 Heuristic-Symbolic Representation

3.1 Representation of Planning Domains

We represent planning domains symbolically, by means of BDDs (Binary Decision Diagrams) [Bryant, 1992], borrowing from the formal verification community the standard machinery used in symbolic model checking [McMillan, 1993]. BDDs are an efficient mechanism for the representation of propositional formulae. A BDD is a binary directed acyclic graph. The terminal nodes are either *True* or *False*. Each non-terminal node is associated with a boolean variable, and two BDDs, the left and right branches, corresponding to the assignment of the true and false values to the node variable. BDDs enjoy a canonical form, resulting from the imposition of a total order over the variables associated to nonterminal nodes. This allows for equivalence checking in constant time, and for the efficient implementation of boolean transformation (e.g. conjunction, disjunction, negation) and QBF transformations (i.e. universal and existential quantification of boolean variables). Efficient software libraries for the manipulation of BDDs, called BDD packages, are available. For lack of space the reader is referred to [Bryant, 1992] for a thorough description of BDDs.

The symbolic representation of the automaton for a given domain can be efficiently built starting from the domain description language (see for instance [Cimatti *et al.*, 1997]). For a given planning domain, we use two vectors of BDD variables, called current and next state variables, written $\mathbf{x}$ and $\mathbf{x}'$ respectively, to represent atomic propositions of the domain. A state s can be seen as an assignment to such variables. A set of states $Bs \subseteq \mathcal{S}$ is associated with a unique corresponding BDD, the models of which are the assignments corresponding to the states in Bs . We represent actions by means of a vector of action variables α . We write $\alpha = \alpha$ for the BDD in the α variables representing the action α . We assume a mapping between the set-theoretic view of the planning domain and the corresponding BDD-based representation. When clear from the context, we confuse the set-theoretic and symbolic representations. For instance, we use equivalently the *False* BDD and $\emptyset$. We write $\mathcal{R}(\mathbf{x}, \alpha, \mathbf{x}')$ for the BDD representing the transition relation, to stress the dependency on BDD variables. The notions defined in previous section at the set-theoretic level have a direct counterpart in terms of BDD transformations. For instance, the applicability relation $\text{APPL}(\mathbf{x}, \alpha)$ is computed symbolically by the projection operation $\exists \mathbf{x}'. \mathcal{R}(\mathbf{x}, \alpha, \mathbf{x}')$, the result being a BDD in the current state variables and action variables, whose assignments are the state-action pairs $\langle s \cdot \alpha \rangle$ such that α is applicable in s . In order to check whether an action α is applicable in a belief state $\emptyset \neq Bs \subseteq \mathcal{S}$, it is possible to check whether $\exists \alpha. \forall \mathbf{x}. ((Bs(\mathbf{x}) \wedge \alpha = \alpha) \rightarrow \text{APPL}(\mathbf{x}, \alpha))$ is not the *False* BDD. The execution of α in $Bs(\mathbf{x})$ is computed as $\exists \mathbf{x}. (Bs(\mathbf{x}) \wedge \exists \alpha. (\alpha = \alpha \wedge \mathcal{R}(\mathbf{x}, \alpha, \mathbf{x}')))[\mathbf{x}'/\mathbf{x}]$, where $[\mathbf{x}'/\mathbf{x}]$ represents the parallel substitution of the next state variables with the current state variables.

3.2 Representation of the Search Space

The search space for conformant planning is $\text{Pow}(\mathcal{S})$ (outlined in Figure 1, with belief states represented as circles). The search space can be constructed forward, starting from the initial set of states, or backward, from the goal. In the first case, the arrows outgoing from belief state 1 represent the fact that the labeling action A [B, C, respectively] is applicable to all the states in 1, and can result in any of the states in belief state 2 [3, 4, resp.]. In the backward case, the situation is dual: for instance, belief state -3 is the maximal belief state where the labeling action B is applicable, and the result is contained in the belief state -1. For both search directions, a belief state Bs is directly represented by the corresponding BDD $Bs(\mathbf{x})$. Given the canonical form of BDDs, a belief state is simply the pointer to the unique corresponding BDD. In Figure 2, the lower box depicts a possible status of the BDD package. The column on the left shows the variables in the BDD package. The horizontal dashed line separates state variables (below the line) from action variable. A belief state is a subgraph in the state variables. Below the dashed line, a possible configuration for the forward search space of Figure 1 is given. For each belief state, there is a corresponding BDD in the state variables. Solid [dashed, respectively] arcs in a BDD represent the positive [negative, resp.] assignment to the variable in the originating node. For instance, the leftmost BDD $Bs1$ encodes the formula $(x1 \leftrightarrow \neg x2) \wedge x3 \wedge x4$.

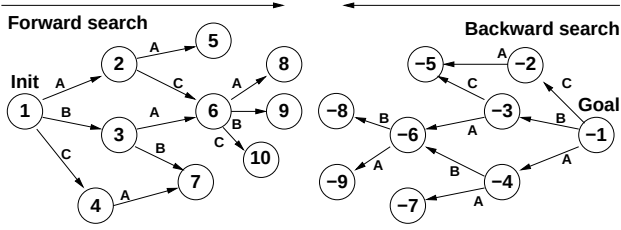


Figure 1: The search space

For the sake of simplicity, only the paths leading to *True* are shown. The advantage over an enumerative representation of belief states (e.g. the list of the state vectors associated to each state contained in the Bs) is twofold. In terms of memory, there is no direct connection between the cardinality of the belief state and the size of the corresponding BDD. (Consider for instance that, with 10 boolean variables, the set of all the states, of cardinality 1024, is represented by the *True* BDD, requiring one node.) Furthermore, BDD packages are designed to maximize the sharing between different BDDs, and minimize memory occupation. In terms of efficiency, the set-theoretic operations for the transformation and combination of belief states (e.g. projection, equivalence, inclusion) can be efficiently performed as BDD transformations, with the primitives provided by the BDD package, which make aggressive use of memoizing of previously computed subproblems.

One of the novelties of our approach is based on the integration of the symbolic representation described above with the standard data structures used in heuristic search. In order to allow for heuristic search, a hash table is used to store and retrieve the belief states visited during the search (upper box in Figure 2). Each entry directly points to the BDD representing the belief state, and is annotated with the suitable information (e.g. the plan, the cost factors). For instance, for belief state 6 the stored plan is the sequence B;A, i.e. one of the sequences of actions needed to reach belief state 6 from the initial belief state 1. The hash table is accessed using as key the pointer to the BDDs representing the belief state. The approach relies on the canonical form of BDDs, thanks to which only one BDD representative for equivalent boolean functions is required. The hash table is accessed with a Bs, and if a corresponding entry is present, then it is returned, otherwise a new entry is created. Notice that the memory occupation of the belief state for each entry in the hash table is fixed to a single pointer: the BDD package is responsible to compress the information, as shown in Figure 2. Notice also that this approach allows to reuse a standard BDD package, without interfering with its internal mechanisms (e.g. hashing, memoizing, garbage collection).

The construction of the search space is performed by expanding an individual belief state, resulting in a set of the form $\{\langle Bs_1 \cdot \alpha_1 \rangle, \dots, \langle Bs_n \cdot \alpha_n \rangle\}$. For instance, when expanding in forward the initial belief state 1 in Figure 1, the resulting expansion is the set $\{\langle 2 \cdot A \rangle, \langle 3 \cdot B \rangle, \langle 4 \cdot C \rangle\}$, meaning that *A*, *B* and *C* are applicable in 1, and the corresponding execution results are the belief state 2, 3 and 4. Dually, when expanding -6 backward, the result is $\{\langle -8 \cdot B \rangle, \langle -9 \cdot A \rangle\}$, meaning that *B* can be applied in -8 and results in -6. The ability to expand belief states sym-

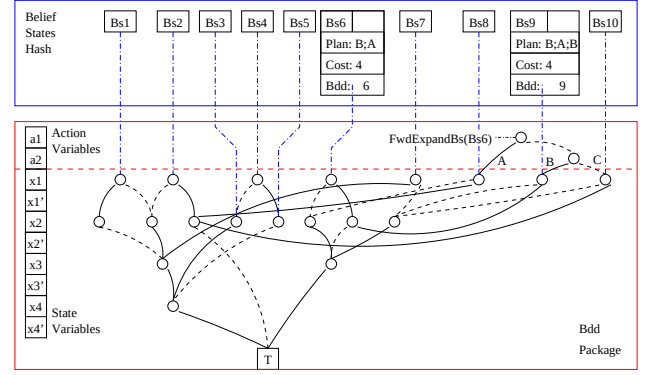


Figure 2: The combined use of the hash and the BDD

bolically is one of the keys to the efficient search. The backward step of expansion $BWEXPANDBS$, applied to the BDD $Bs(\mathbf{x})$, constructs the BDD $\forall \mathbf{x}' . (\mathcal{R}(\mathbf{x}, \alpha, \mathbf{x}') \rightarrow Bs(\mathbf{x})[\mathbf{x}/\mathbf{x}']) \wedge \text{APPL}(\mathbf{x}, \alpha)$. The result is a BDD in the current state variables $\mathbf{x}$ and action variables α , representing an association from actions to states $\{\langle s_1^1 \cdot \alpha_1 \rangle, \dots, \langle s_{n_1}^1 \cdot \alpha_1 \rangle, \dots, \langle s_1^k \cdot \alpha_k \rangle, \dots, \langle s_{n_k}^k \cdot \alpha_k \rangle\}$, that can be interpreted as an association between actions and belief states, i.e. $\{\langle \{s_1^1, \dots, s_{n_1}^1\} \cdot \alpha_1 \rangle, \dots, \langle \{s_1^k, \dots, s_{n_k}^k\} \cdot \alpha_k \rangle\}$. The dual forward step $FWEXPANDBS$ expands $Bs(\mathbf{x})$ by computing the executions in Bs of every applicable action, i.e. $(\exists \mathbf{x} . (Bs(\mathbf{x}) \wedge (\forall \mathbf{x} . (Bs(\mathbf{x}) \rightarrow \text{APPL}(\mathbf{x}, \alpha)) \wedge \mathcal{R}(\mathbf{x}, \alpha, \mathbf{x}'))))[\mathbf{x}'/\mathbf{x}]$. The data structure resulting from the expansion is again a BDD in the $\mathbf{x}$ and α variables. In Figure 2 the BDD resulting from the forward expansion of belief state 6 is shown: the paths at the levels of action variables represent the different actions *A*, *B* and *C*, and the attached subgraphs represent the corresponding belief states 8, 9 and 10. The link between the symbolic representation of the expansion and the hash-based representation of the search space is the special purpose primitive $PRUNEBSEXPANSION$. For either search directions, every time a belief state is expanded, the resulting belief states have to be compared with the previously visited belief states. If not present, they must be inserted in the hash of the visited belief states, otherwise eliminated. This analysis is performed by $PRUNEBSEXPANSION$, that operates directly on the BDD resulting from the expansion of the Bs, and allows to combine the symbolic expansion of belief states with the hashing mechanism. $PRUNEBSEXPANSION$ recursively descends the BDD in the action and state variables, and interprets as a belief state every BDD node having a state variable at its top. Nodes corresponding to previously visited belief states are pruned, while the new ones are inserted in the hash. $PRUNEBSEXPANSION$ assumes that in the BDD package action variables precede state variables, as shown in Figure 2. This does not appear to be a significant limitation in many practical cases. This approach takes care of a very significant source of redundancy: often a belief state can be associated with a large number of equivalent plans (in Figure 1, *A;C* and *B;A* are two equivalent action plans associated with 1). The pruning mechanism makes sure that, for each Bs, only one plan is left.

4 The Planning Algorithm

In this section we describe the planning algorithm, based on the data structures and primitives described in previous section. Figure 3 depicts the backward search algorithm, that takes in input the problem description in form of the BDDs $\mathcal{I}$ and $\mathcal{G}$, while the transition relation $\mathcal{R}$ is assumed to be globally available. *OpenBsPool* contains the (annotated) belief states which have been reached but still have to be explored, and is initialized to the first belief state of the search, i.e. $\mathcal{G}$, annotated with the empty plan ϵ . *BSMARKVISITED* inserts $\mathcal{G}$ into the hash table of visited belief states, also updating the cost information. The algorithm loops (lines 3-11) until a solution is found or all the search space has been exhausted. First, an annotated belief state $\langle Bs . \pi \rangle$ is extracted from the open pool (line 4) by *EXTRACT*. The belief state is expanded by *BWEXPANDBS*, computing the corresponding BDD in the $\mathbf{x}$ and α variables. The resulting Bs expansion, stored in *BsExp*, is traversed by *PRUNEBSEXPANSION* as explained in previous section: for each belief state in the expansion, the hash table of the visited belief states is accessed, and only the belief states that are not present are accumulated in the returned list.

Each of the resulting belief states is compared with the initial set of states $\mathcal{I}$. If $\mathcal{I} \subseteq Bs_i$, then the associated plan $\alpha_i; \pi$ is a solution to the problem, the loop is exited and the plan is returned. Otherwise, the annotated belief state $\langle Bs_i . \alpha_i; \pi \rangle$ is inserted in *OpenBsPool* and the loop is resumed. If *OpenBsPool* becomes empty and a solution has not been found, then a fix point has been reached, i.e. all the reachable space of belief states has been covered, and the algorithm terminates with failure. The dual algorithm for forward search, not reported here for lack of space, shares the structure of the backward planning algorithm described in Figure 3, but applies the forward expansion steps defined in previous section. The termination of the algorithms is guaranteed by the fact that the set of explored belief sets (stored in the visited hash table) is monotonically increasing. Since the set of accumulated belief states is contained in $\text{Pow}(\mathcal{S})$, which is finite, a fix point is eventually reached. Furthermore, a solution plan is returned if and only if the problem admits a solution, otherwise a failure is returned.

The length of the constructed plans depends on the effectiveness of the heuristic with respect to the problem being tackled, in particular on the primitives *EXTRACT* and *INSERT*. We use a very simple heuristic function. When proceeding backward, it selects the belief state with the highest cardinality. We consider as more promising a belief state with a low degree of knowledge: intuitively, the associated plan requires a low knowledge to lead to the goal. Dually, in forward, we select the belief states with lowest cardinality, that is associated with a plan that yields a guarantee of high knowledge. Our approach departs significantly from the heuristic function used in [Bonet and Geffner, 2000], that are a simple combination of heuristic functions over the states in the belief state. The behavior of GPT on the SQUARE and CUBE variations shows that this approach can lead to dramatic failures, as discussed in [Cimatti and Roveri, 2000]. Although our selection strategy is in general not admissible (in the sense of A^*), it seems to be extremely effective, especially in the backward

```

procedure HeurSymBwdConformant( $\mathcal{I}, \mathcal{G}$ )
0  begin
1     $\text{OpenBsPool} := \{\langle \mathcal{G} . \epsilon \rangle\}$ ; BSMARKVISITED( $\mathcal{G}$ );
2     $\text{Solved} := \mathcal{I} \subseteq \mathcal{G}$ ;  $\text{Solution} := \epsilon$ ;
3    while ( $\text{OpenBsPool} \neq \emptyset \wedge \neg \text{Solved}$ ) do
4       $\langle Bs . \pi \rangle := \text{EXTRACT}(\text{OpenBsPool})$ ;
5       $\text{BsExp} := \text{BWEXPANDBS}(Bs)$ ;
6       $\text{BsPList} := \text{PRUNEBSEXPANSION}(\text{BsExp})$ ;
7      for  $\langle Bs_i . \alpha_i \rangle$  in  $\text{BsPList}$  do
8        if  $\mathcal{I} \subseteq Bs_i$  then
9           $\text{Solved} := \text{True}$ ;  $\text{Solution} := \alpha_i; \pi$ ; break;
10         else INSERT( $\langle Bs_i . \alpha_i; \pi \rangle, \text{OpenBsPool}$ ) endif;
11      end while
12      if  $\text{Solved}$  then return  $\text{Solution}$ ;
13      else return Fail;
14  end

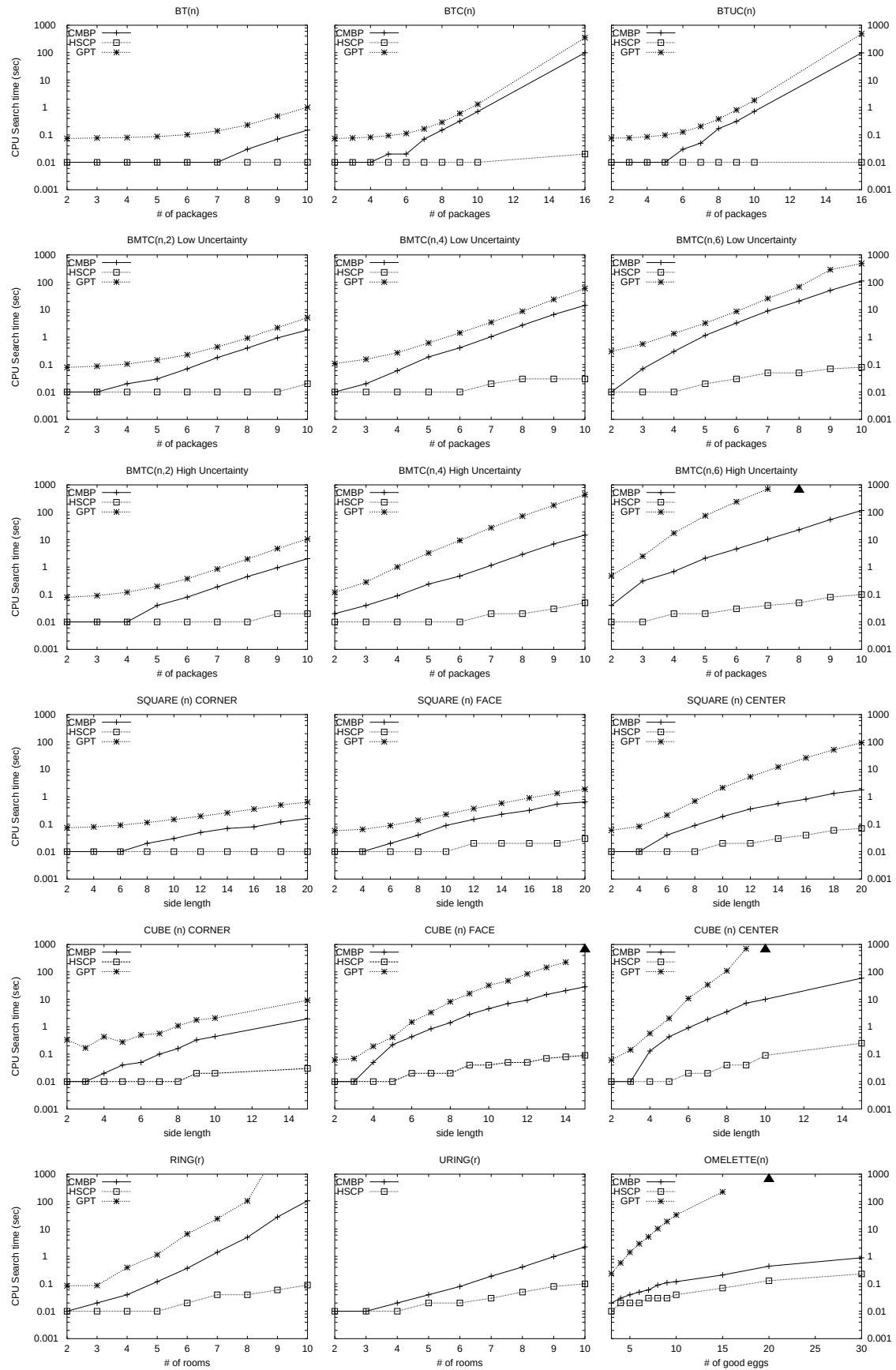
```

Figure 3: The backward conformant planning algorithm case.

5 Experimental Evaluation

We implemented the data structures and the algorithms described in this paper, in the following called HSCP (Heuristic-Symbolic Conformant Planner), on top of CMBP [Cimatti and Roveri, 2000] (see <http://sra.itc.it/tools/mbp/>). We carried out an extensive experimental evaluation of our approach, covering all the test cases presented so far in the literature to evaluate conformant planners. (For lack of space, we refer to [Cimatti and Roveri, 2000] for the description of the problems.) The most relevant systems for the comparison are GPT, for its heuristic search style, and CMBP, for its use of model checking techniques. We do not consider other conformant planners, such as CGP [Smith and Weld, 1998] and QBFPLAN [Rintanen, 1999], that are outperformed by GPT and CMBP, as shown in [Cimatti and Roveri, 2000]. For CMBP and HSCP, the backward search algorithms were run. All the experiments were run on an Pentium II 300MHz with 512Mb of memory running Linux, fixing a memory limit of 128Mb and a CPU time limit of 2 hours. The results of the comparison are depicted in the following. Each plot refers to a problem class. For each problem instance, we report (in seconds on a logarithmic scale) the search time, i.e. the CPU time required to find a solution for the given planning problem, or to discover that the problem admits no solution. (We do not consider the preprocessing time. In the case of GPT, this includes the generation of a C++ file, that is compiled and then linked to the solver program. In the case of CMBP and HSCP, for all the reported experiments, the automaton construction, optimized with respect to the results reported in [Cimatti and Roveri, 2000], requires at most 0.44 secs, in the OMELETTE(30) problem.)

The experimental evaluation shows that the heuristic-symbolic search of HSCP outperforms CMBP (that, in turn, outperforms GPT [Cimatti and Roveri, 2000]). On average, the improvement is of about two orders of magnitude, with a minimum of one order of magnitude in the case of SQUARE and OMELETTE, and a maximum of about three orders of magnitude in the case of BTUC and of BMTC(n,6) with high uncertainty. In the RING(9) problem, GPT reaches the fixed



time limit without finding a solution. The URING problems are not representable in GPT. It is interesting to notice that, while GPT reaches the memory limit in 4 sets of experiments (solid triangles in the plots), CMBP and HSCP complete all the runs within the given limit. This appears to be due to the use of the BDD based representation. In terms of memory, on average, CMBP requires about twice more memory than HSCP: the main reason for this seems to be that CMBP represents plans symbolically, and this requires the introduction of a new set of action variables for each layer of the search. Although CMBP is guaranteed to return plans of minimal length, HSCP returns plans that are surprisingly short. For all the experiments, HSCP returns minimal length plans, with the exception of some of the BMTC problems. Finally, the OMELETTE problem, differently from all the others, admits no conformant solution, and therefore it requires the complete exploration of the state space. We expected CMBP to outperform HSCP, because of its ability to expand a large number of states in one single operation. However, HSCP outperforms CMBP, having to deal with more BDDs, but of smaller size.

6 Related Work and Conclusions

In this paper, we have presented a new, heuristic-symbolic search paradigm, resulting in an extremely effective approach to conformant planning. BDD-based, symbolic techniques are combined with search algorithms that exploit heuristic selection functions taking into account the degree of uncertainty. Our approach gains up to three order of magnitude in search time over the breadth-first symbolic approach of CMBP, and up to five over the heuristic search of GPT, with lower memory requirements.

In the following we describe additional related works. In hardware design, synchronization sequences are used for testing and equivalence checking. The problem of finding a synchronization sequence can be seen as a particular form of conformant planning: a synchronization sequence is a sequence of inputs that takes a circuit from an unknown state into a completely defined one. In [Cimatti *et al.*, 2001], we tackle this problem within the heuristic-symbolic paradigm. A comparison over the standard benchmarks shows that our approach is more than competitive with specialized BDD-based algorithms [Pixley *et al.*, 1992; Rho *et al.*, 1993]. The work in [Finzi *et al.*, 2000] is limited to deterministic actions, and relies on user-defined heuristics to achieve an efficiency comparable with CMBP. [Ferraris and Giunchiglia, 2000] tackles conformant planning in nondeterministic domains described in *C* language, that allows for parallel actions. Although the problems tackled by CMBP and HSCP are expressed in the *AR* language, that is limited to sequential actions, the planners are not subject to this restriction. For instance, in [Cimatti *et al.*, 2001] HSCP is used to analyze circuits with for parallel inputs (actions) expressed in the SMV language [McMillan, 1993]. The planner of [Ferraris and Giunchiglia, 2000] can not discover when a problem admits no conformant plan, and the reported results are comparable with CGP [Smith and Weld, 1998] (see also [Cimatti and Roveri, 2000] for a detailed discussion). In [Bertoli *et al.*, 2001], the approach presented in this paper is used as a basis for planning under partial observability. As a future work, we will investigate the

use of more accurate heuristic functions, that combine a measure of the degree of uncertainty with domain knowledge in the style of GPT. Furthermore, we will extend the approach presented in this paper with the techniques of [Cimatti and Roveri, 2000], that allow to expand symbolically several belief states at a time. We will also generalize our work to deal with extended goals expressed in different forms of temporal logic specifications.

References

- [Bertoli *et al.*, 2001] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. In *Proc. IJCAI-01*. AAAI Press, 2001.
- [Bonet and Geffner, 2000] B. Bonet and H. Geffner. Planning with Incomplete Information as Heuristic Search in Belief Space. In *Proc. AIPS-00*, pages 52–61. AAAI Press, April 2000.
- [Bryant, 1992] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.
- [Cimatti and Roveri, 2000] A. Cimatti and M. Roveri. Conformant Planning via Symbolic Model Checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.
- [Cimatti *et al.*, 1997] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via Model Checking: A Decision Procedure for *AR*. In *Proc. ECP-97*, LNCS 1348, pages 130–142, Toulouse, France, September 1997. Springer-Verlag.
- [Cimatti *et al.*, 1998] A. Cimatti, M. Roveri, and P. Traverso. Strong Planning in Non-Deterministic Domains via Model Checking. In *Proc. AIPS-98*, Pittsburgh, USA, June 1998. AAAI Press.
- [Cimatti *et al.*, 2001] A. Cimatti, M. Roveri, and P. Bertoli. Searching Powerset Automata by Combining Explicit-State and Symbolic Model Checking. In *Proc. TACAS 2001*, pages 313–327. Springer, April 2001.
- [Ferraris and Giunchiglia, 2000] P. Ferraris and E. Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proc. AAAI'00*, Austin, Texas, July-August 2000. AAAI Press.
- [Finzi *et al.*, 2000] A. Finzi, F. Pirri, and R. Reiter. Open world planning in the situation calculus. In *Proc. AAAI'00*, Austin, Texas, July-August 2000. AAAI Press.
- [Goldman and Boddy, 1996] R.P. Goldman and M.S. Boddy. Expressive Planning and Explicit Knowledge. In *Proc. AIPS-96*, pages 110–117. AAAI Press, 1996.
- [Kabanza *et al.*, 1997] F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence*, 95(1):67–113, 1997.
- [Kohavi, 1978] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill Book Company, New York, 1978. ISBN 0-07-035310-7.
- [McMillan, 1993] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publ., 1993.
- [Pixley *et al.*, 1992] C. Pixley, S.-W. Jeong, and G. D. Hachtel. Exact calculation of synchronization sequences based on binary decision diagrams. In *Proc. of the 29th Design Automation Conference*, IEEE CS Press, 1992.
- [Pryor and Collins, 1996] L. Pryor and G. Collins. Planning for Contingency: a Decision Based Approach. *J. of Artificial Intelligence Research*, 4:81–120, 1996.
- [Rho *et al.*, 1993] J.-K. Rho, F. Somenzi, and C. Pixley. Minimum length synchronizing sequences of finite state machine. In *Proc. of the 30th ACM/IEEE Design Automation Conference*, pages 463–468, Dallas, TX, June 1993. ACM Press.
- [Rintanen, 1999] J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.
- [Smith and Weld, 1998] D. E. Smith and D. S. Weld. Conformant graphplan. In *Proc. AAAI-98*, pages 889–896, Menlo Park, July 26–30 1998. AAAI Press.
- [Weld *et al.*, 1998] D. S. Weld, C. R. Anderson, and D. E. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proc. AAAI-98*, pages 897–904, Menlo Park, July 26–30 1998. AAAI Press.