Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Artificial Intelligence 159 (2004) 127–206

Artificial Intelligence

www.elsevier.com/locate/artint

Conformant planning via symbolic model checking and heuristic search

A. Cimatti [*], M. Roveri, P. Bertoli

ITC – IRST, Via Sommarive 18, 38055 Povo, Trento, Italy

Received 8 August 2003; accepted 4 May 2004

Abstract

In this paper we tackle the problem of Conformant Planning: find a sequence of actions that guarantees goal achievement regardless of an uncertain initial condition and of nondeterministic action effects. Our approach, set in the framework of search in the belief space, is based on two main intuitions. First, symbolic model checking techniques, Binary Decision Diagrams in particular, can be used to represent and expand the search space, and to provide an efficient computational platform. Second, driving the search solely with reachability information is not effective, and the notion of knowledge has to be taken explicitly into account.

We make the following contributions. First, we thoroughly analyze reachability heuristics: we formally prove their admissibility, we show how they can be implemented by means of symbolic techniques, and experimentally evaluate them. Second, we analyze the limitations of reachability heuristics, and propose more informed, yet admissible, heuristics, based on a formal notion of knowledge. Third, we present a practical conformant planning algorithm, where the search is explicitly based on the notion of target knowledge, i.e., the amount of information that has to be available in order to reach the goal. The search alternates between the "Acquire Knowledge" mode, where actions have the precise purpose of gathering necessary information, and the "Reach Goal" mode, where actions are directed towards the goal. Finally, we provide a thorough experimental comparison between several conformant planners. Our approach is sometimes able to outperform the competitor systems by orders of magnitude.
© 2004 Elsevier B.V. All rights reserved.

[*] Corresponding author.
E-mail addresses: cimatti@irst.itc.it (A. Cimatti), roveri@irst.itc.it (M. Roveri), bertoli@irst.itc.it (P. Bertoli).

0004-3702/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2004.05.003

Keywords: Planning in nondeterministic domains; Conformant planning; Heuristic search in belief space; Symbolic model checking; Binary decision diagrams

1. Introduction

Planning in nondeterministic domains relaxes a number of assumptions underlying classical planning, that are often unrealistic when modeling real-world domains. For instance, the initial condition may be incompletely specified, actions may have nondeterministic effects, and the status may be only partial observable. Conformant Planning is the problem of finding a plan that guarantees goal achievement in a nondeterministic domain, under the hypothesis that no information can be acquired at run time. The Blind Robot problem [38], describing a sensor-less robot that has to reach a certain objective, despite its unability to sense the environment, is an early example of conformant planning. Conformant planning can also find a suitable course of action under the hypothesis that all the information that could be acquired and/or inferred is not enough to rule out uncertainty: this is the case of autonomous control, when on-board diagnosis may leave a certain degree of uncertainty regarding the existence and precise location of faults.

Conformant planning may seem similar to classical planning, since a solution is a sequence of actions. However, when dealing with nondeterministic domains, a sequence of actions is associated with a set of runs (rather than to a single run, as in classical planning). Different runs may start from different possible initial states, or may result from the branching associated with nondeterministic action effects. Therefore, conformant planning turns out to be remarkably harder than classical planning [34,43].

A number of different approaches and techniques have been proposed to tackle conformant planning in the recent years [4,6,8,14,20,29,35,43,44]. Some of these (e.g., [14, 29,43]) can be classified as a generate-and-test approach: a candidate plan is constructed by analyzing some of the associated traces, and then it is tested for conformance on all possible traces. A potential problem of this approach is the number of invalid candidate plans, that solve the problem for some runs but not for all. An alternative formulation, initially proposed in [6], tackles conformant planning as search in the *belief space*: the information characterizing the execution of a plan is represented as a belief state, i.e., a set of the possible, undistinguishable states. Although plans are conformant by construction, the search space is potentially exponential in the state space of the planning domain.

In this paper, we tackle conformant planning in the framework of search in the belief space, and improve it in two main directions. First, we provide an efficient platform to implement *symbolic search in the belief space*. We use and extend techniques from symbolic model checking, with belief states compactly represented as Binary Decision Diagrams (BDDs), while the search expansion primitives are directly implemented by means of logical transformations. The major advantage of a symbolic approach is in that sets of large cardinality can be often compactly represented and efficiently manipulated by exploiting structural regularities. Second, we tackle the key problem of *directing the search in belief space*. The approach implemented in [6] is based on the idea of a reachability heuristic, where the distance of a belief state to the goal can be estimated as the distance of the farthest state in the belief state. This approach, though providing admissible heuristics, results sometimes in very coarse estimates. We tackle this problem by introducing the notion of knowledge associated with a belief state. By exploiting the idea of necessary knowledge (i.e., a certain amount of information has to be available to reach the goal), we define more informed yet admissible heuristics. Then, we propose a conformant planning algorithm explicitly based on knowledge acquisition, where the search alternates between two modes: in the "Acquire Knowledge" mode, actions have the precise purpose of gathering necessary information; in the "Reach Goal" mode, actions are directed towards the goal. A thorough experimental comparison shows that our approach, implemented in the MBPplanner, is in fact very competitive, and is able to outperform the competitor systems, sometimes by orders of magnitude. The paper is structured as follows. In Section 2 we provide a formal definition of conformant planning, and in Section 3 we recast the problem as heuristic search in the belief space. In Section 4 we describe the use of symbolic techniques for an efficient implementation of the heuristic search framework. In Section 5 we define a general class of reachability heuristics and their symbolic characterization, prove their admissibility, and experimentally analyze their effectiveness. In Section6, we characterize the limitations of reachability heuristics, introduce the notion of knowledge associated with a belief state, and define more informed heuristics based on the idea of necessary knowledge. In Section 7, we present the conformant planner based on knowledge acquisition. In Section 8 we discuss the other approaches to conformant planning. In Section 9 we compare our approach with the most efficient conformant planners available. In Section 10 we present the lines for future research and conclude. The proofs of the theorems are reported in Appendices A–D. The complete set of the experiments, as well as the MBP system, are available from http://sra.itc.it/tools/mbp/AIJ04.

2. Conformant planning

The aim of this section is to provide a precise notion of conformant planning. We first define nondeterministic planning domains (Section 2.1); then, we present the plan structure and the associated notion of execution (Section 2.2); finally, we define the notion of conformant planning problem (Section 2.3). Our presentation is at the semantic level, independently of the specific language used for the description of the planning domain.

2.1. Nondeterministic planning domains

A (*nondeterministic*) *planning domain* can be described in terms of *state variables* (also referred to as *fluents*), which may assume different values in different *states*, of *actions*, and of a *transition relation* describing how (the execution of) an action leads from one state to possibly many different states.

Definition 1 (*Nondeterministic planning domain*). A nondeterministic planning domain $\mathcal{D}$ is a tuple $\langle \mathcal{X}, \mathcal{V}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$, where

- $\mathcal{X}$ is a finite set of state variables;

- $\mathcal{V}$ is a range function over $\mathcal{X}$, such that $\mathcal{V}(x)$ is a finite, non-empty set for every variable $x \in \mathcal{X}$;
- $\mathcal{S}$ is a set of states, a state being a function over $\mathcal{X}$ that associates to each variable $x \in \mathcal{X}$ an element in $\mathcal{V}(x)$;
- $\mathcal{A}$ is a finite set of actions;
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the transition relation.

Notice that we provide an explicit treatment of non-boolean state variables, even though they could be encoded using boolean variables when the set of possible values is finite. This allows us to represent the original structure of the domain, which would be lost with a boolean encoding. As we will see, this is crucial for effectively extracting heuristic information to efficiently drive the search.

The transition relation describes the effects of action execution. An action α is *applicable* in a state $s \in \mathcal{S}$ iff there exists a state s' such that $(s, \alpha, s') \in \mathcal{R}$. An action α is *deterministic* [*nondeterministic*, respectively] in a state s iff there exists exactly one [more than one] state s' such that $(s, \alpha, s') \in \mathcal{R}$. Given a domain $\mathcal{D}$, we denote with $\textsc{ActionsOf}(s)$ the set of actions that are applicable in state s. We denote with $\textsc{Appl}$ the applicability relation, i.e., the set of state-action pairs such that the action is applicable in the state.

$$\textsc{Appl} = \{(s, \alpha)\colon\ \alpha \in \textsc{ActionsOf}(s)\}.$$

We write $\textsc{Appl}(s, \alpha)$ for $(s, \alpha) \in \textsc{Appl}$, and $\mathcal{R}(s, \alpha, s')$ for $(s, \alpha, s') \in \mathcal{R}$. We call *execution* of α in s (denoted by $\textsc{Exec}[\alpha](s)$) the set of the states that can be reached from s after performing the action α:

$$\textsc{Exec}[\alpha](s) = \{s'\colon\ (s, \alpha, s') \in \mathcal{R}\}.$$

Example 1. For explanatory purposes, we consider a simple navigation domain depicted in Fig. 1. A robot can move in the four directions in a 4×4 square. The positions in the square can be represented by means of two state variables x and y, both ranging from 0 to 3. Thus, $\mathcal{X} = \{x, y\}$ and $\mathcal{V}(x) = \mathcal{V}(y) = \{0, 1, 2, 3\}$. The state corresponding to the robot being in the upper-left corner is the tuple $(\langle \mathtt{x}\ .\ \mathtt{0}\rangle, \langle \mathtt{y}\ .\ \mathtt{0}\rangle)$. In the following we simply refer to $(\langle \mathtt{x}\ .\ \mathtt{v_x}\rangle, \langle \mathtt{y}\ .\ \mathtt{v_y}\rangle)$ with $(\mathtt{v_x}, \mathtt{v_y})$. Thus, for this domain the set of states is $\{(\mathtt{v_x}, \mathtt{v_y})\colon\ \mathtt{v_x}, \mathtt{v_y} \in \{0, 1, 2, 3\}\}$. The set of actions $\mathcal{A}$ that model the movement directions of the robot is $\{\mathtt{GoWest}, \mathtt{GoEast}, \mathtt{GoNorth}, \mathtt{GoSouth}\}$. The black location in $(\mathtt{1}, \mathtt{1})$ is a hole that, if entered, will cause the robot to crash. This means, for instance, that the $\mathtt{GoEast}$ action is not applicable in $(\mathtt{0}, \mathtt{1})$. When moving towards a wall, the robot does not move. For instance, if the robot performs a $\mathtt{GoEast}$ action in location $(\mathtt{3}, \mathtt{3})$, it will remain in $(\mathtt{0}, \mathtt{0})$. The location in $(\mathtt{2}, \mathtt{2})$ is a slippery spot, that will make the robot move

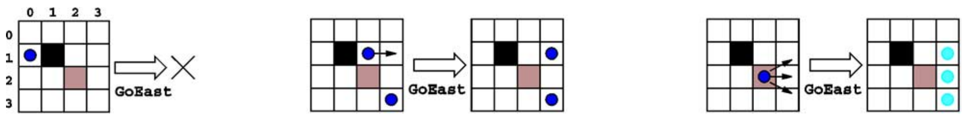


Fig. 1. A simple nondeterministic robot navigation domain.

unpredictably sideways when performing an action in it. This introduces nondeterministic action effects. For instance, the effect of performing a `GoEast` action starting from state $(\mathtt{2}, \mathtt{2})$ may results in any of the states in $\{(\mathtt{3}, \mathtt{1}), (\mathtt{3}, \mathtt{2}), (\mathtt{3}, \mathtt{3})\}$.

In the following, unless otherwise specified, we assume a planning domain $\mathcal{D} = \langle \mathcal{X}, \mathcal{V}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ as given. We use $s, s_0, s_1, \ldots$ to denote states, $a, a_0, a_1, \ldots$ to denote actions, $x, y, z, x_0, x_1, \ldots$ to denote state variables, $v, v_0, v_1, \ldots$ to denote values.

We define the set $\mathcal{P}$ of atomic propositions as the set of equalities between the state variables and their possible values.

$$\mathcal{P} \doteq \bigl\{x = v\colon\ x \in \mathcal{X} \text{ and } v \in \mathcal{V}(x)\bigr\}.$$

Intuitively, a state is a collection of the atomic propositions holding in it. Thus, in the example of Fig. 1 the propositions are all the equalities between the variables and their possible values (e.g., $x = 0$, $y = 3$). We say that an atomic proposition $(x = v)$ holds in a state s, denoted with $s \models (x = v)$, iff $s(x) = v$.

Definition 2 (*Boolean formula*: *Syntax, denotation*). Let $\mathcal{P}$ be a set of atomic propositions, and let $p \in \mathcal{P}$. The set of boolean formulae *bwff* over $\mathcal{P}$ is defined as follows:

- $\mathit{bwff} := \mathcal{P} \mid \neg \mathit{bwff} \mid \mathit{bwff} \wedge \mathit{bwff} \mid \mathit{bwff} \vee \mathit{bwff}$.

The denotation of a boolean formula is defined as:

- $[\![p]\!] \doteq \{s \in \mathcal{S}\colon\ s \models p\}$, if p is an atomic proposition;
- $[\![\phi_1 \wedge \phi_2]\!] \doteq [\![\phi_1]\!] \cap [\![\phi_2]\!]$;
- $[\![\phi_1 \vee \phi_2]\!] \doteq [\![\phi_1]\!] \cup [\![\phi_2]\!]$;
- $[\![\neg\phi]\!] \doteq \mathcal{S} \setminus [\![\phi]\!]$.

Intuitively, the denotation of a formula is the set of states where the formula holds. We say that s *satisfies* a boolean formula ϕ, written $s \models \phi$, iff $s \in [\![\phi]\!]$. We write $x \in \{v_1, \ldots, v_n\}$, where $v_i \in \mathcal{V}(x)$, for $(x = v_1 \vee \cdots \vee x = v_n)$, and $x = y$ for

$$\bigvee_{v \in (\mathcal{V}(x) \cap \mathcal{V}(y))} (x = v) \wedge (y = v).$$

We also assume that $x = v$ stands for $\perp$ whenever $v \notin \mathcal{V}(x)$.

2.2. Plans and plan execution

In this paper, we consider plans that are sequences of actions.

Definition 3 (*Plan, plan length*). A plan π for a planning domain $\mathcal{D}$ is an element of $\mathcal{A}^*$, i.e., a finite, possibly empty sequence $a_1, \ldots, a_n$ such that for all $a_i \in \mathcal{A}$ with $1 \leqslant i \leqslant n$. n is the length of the plan.

We use ε to denote the 0-length plan, π and ρ to denote plans, $\pi; \rho$ to denote plan concatenation, and $|\pi|$ to denote the length of π.

Definition 4 (*Runs of a plan*). Let $\pi = a_1, \ldots, a_n$ be a plan of length n. The sequence $\sigma = s_0, s_1, \ldots, s_n$ is a run for π from s_0 iff, for $0 \leqslant i < n$, $(s_i, a_{i+1}, s_{i+1}) \in \mathcal{R}$. s_n is the final state of the run. The set of runs of π from $B \subset \mathcal{S}$ is the set of all runs of π from any $s \in B$.

We notice that, whenever S contains more than one state, the set of runs of π from S also contains multiple runs. Furthermore, a plan can be associated with many runs, even if starting from the same state, due to the uncertain effects of actions. Our definition of plan applicability guarantees that no action may be attempted in a state where it is not applicable.

Definition 5 (*Plan applicability*). The empty plan ε is applicable in every state $s \in \mathcal{S}$. The plan $\alpha; \pi$ is applicable in $s \in \mathcal{S}$ iff α is applicable in s and π is applicable in all s' such that $(s, \alpha, s') \in \mathcal{R}$. Let S be a non-empty set of states. A plan π is applicable in S iff it is applicable in all $s \in S$.

Example 2. The plan $\pi_1 =$ `GoNorth`; `GoEast`; `GoEast`; `GoEast`; `GoSouth` is applicable in state $(0, 1)$. The only associated run is:

$$\sigma_0 = (0,1), (0,0), (1,0), (2,0), (3,0), (3,1).$$

Plan π_1 is not applicable in $(0, 2)$, since the second action would be attempted in $(0, 1)$, which violates the applicability condition. Plan $\pi_2 =$ `GoEast`; `GoEast`; `GoEast`; `GoSouth` is applicable in state $(0, 2)$ and its three runs are:

$$\sigma_1 = (0,2), (1,2), (2,2), (3,1), (3,2),$$
$$\sigma_2 = (0,2), (1,2), (2,2), (3,2), (3,3),$$
$$\sigma_3 = (0,2), (1,2), (2,2), (3,3), (3,3).$$

Plan $\pi_3 =$ `GoSouth`; `GoEast`; `GoEastGoEast`; `GoSouth`; `GoNorth` is applicable in $\{(0,1), (0,2)\}$, and the corresponding runs, depicted in Fig. 2, are:

$$\sigma_4 = (0,1), (0,2), (1,2), (2,2), (3,1), (3,2), (3,1),$$
$$\sigma_5 = (0,1), (0,2), (1,2), (2,2), (3,2), (3,3), (3,2),$$
$$\sigma_6 = (0,1), (0,2), (1,2), (2,2), (3,3), (3,3), (3,2),$$
$$\sigma_7 = (0,2), (0,3), (1,3), (2,3), (3,3), (3,3), (3,2).$$

2.3. Conformant planning problem

Intuitively, a conformant planning problem for a given domain is characterized by a set of possible initial states, and a set of goal states. A plan is a solution to a conformant planning problem if it is applicable in all the initial states, and all the associated runs end in a goal state.

Definition 6 (*Conformant planning problem, solution*). A conformant planning problem is a triple $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ where $\mathcal{D}$ is a planning domain, $\mathcal{I}$ and $\mathcal{G}$ are non-empty sets of states,

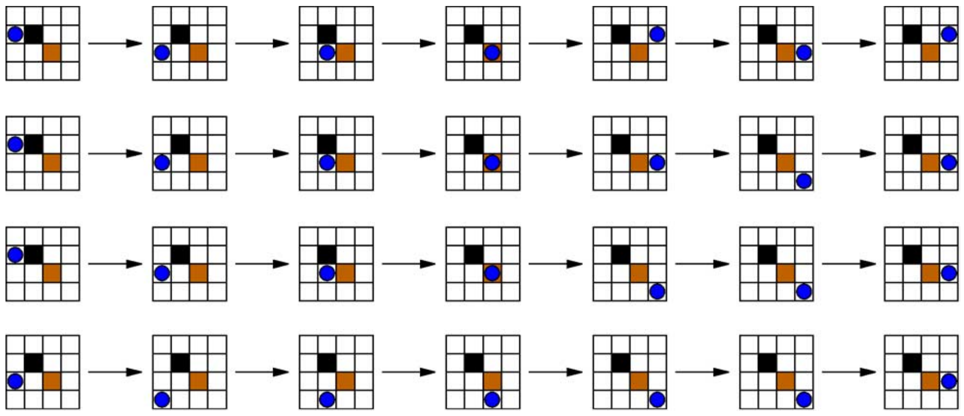


Fig. 2. The runs associated with the plan `GoSouth`; `GoEast`; `GoEast`; `GoEast`; `GoSouth`; `GoNorth`.

called the set of initial states and the set of goal states, respectively. A plan π is a solution to a conformant planning problem $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ iff

- it is applicable in $\mathcal{I}$, and
- every run of π from $\mathcal{I}$ has its final state in $\mathcal{G}$.

At this point, it should be clear that the problem we are tackling is much harder than the classical planning problem. Suppose we are given a possible conformant plan, having a run from one initial state to the goal; we still have to check that it is a valid conformant plan, i.e., it is applicable in each state in $\mathcal{I}$, and that the final state of each run is in $\mathcal{G}$. In fact, conformant planning reduces to classical planning if the set of initial states is a singleton and the domain is deterministic.

Example 3. In the robot navigation domain of Fig. 1, let us consider the conformant planning problem where the initial set of states is $\{(\mathtt{0},\mathtt{1}),(\mathtt{0},\mathtt{2})\}$ and the set of goal states is $\{(\mathtt{3},\mathtt{1}),(\mathtt{3},\mathtt{2})\}$. A possible solution to this problem is the plan $\pi_3 \doteq$ `GoSouth`; `GoEast`; `GoEastGoEast`; `GoSouth`; `GoNorth` described in Example 2. The plan is a conformant solution: for all the possible associated runs (depicted in Fig. 2) the plan never violates the applicability conditions of the attempted actions, and the final states are all in the goal. Notice that the plan is not particularly “smart”, if the runs are considered individually. For instance, the first two runs reach the goal after four actions. However, other actions are needed to make sure that the problem is solved for all runs.

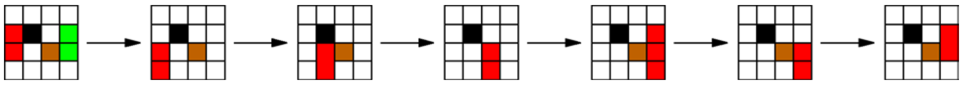
3. Conformant planning as search in the belief space

3.1. The belief space

In this section we provide a way to analyze the condition of uncertainty associated with a certain course of action. We refer to a set of indistinguishable states as *belief state*.

A belief state is a non-empty set of states, intuitively expressing a condition of uncertainty, by collecting together all the states which are indistinguishable.

Example 4. Let us consider again the problem of Example 3. The robot can initially be either in $(\mathtt{0},\mathtt{1})$ or $(\mathtt{0},\mathtt{2})$. Since we are under the hypothesis that no information can be gathered at run-time, the robot is not able to determine its position. Therefore, it believes that its position could be either $(\mathtt{0},\mathtt{1})$ or $(\mathtt{0},\mathtt{2})$, i.e., it is in the belief state $\{(\mathtt{0},\mathtt{1}),(\mathtt{0},\mathtt{2})\}$. The execution of `GoSouth` in the initial uncertainty state would result either in state $(\mathtt{0},\mathtt{2})$ or $(\mathtt{0},\mathtt{3})$ if the initial position of the robot is $(\mathtt{0},\mathtt{1})$ or $(\mathtt{0},\mathtt{2})$ respectively, but again it is not possible to distinguish them. The associated belief state is therefore $\{(\mathtt{0},\mathtt{2}),(\mathtt{0},\mathtt{3})\}$. In general, we can construct the belief states associated with each of the prefixes of length i of the conformant plan `GoSouth`; `GoEast`; `GoEast`; `GoEast`; `GoSouth`; `GoNorth` by collecting all the ith states of the associated runs:



The plan is a conformant solution since the final belief state is contained in $\mathcal{G}$.

Belief states are a convenient way to represent the uncertainty in the conformant planning process. Intuitively, instead of analyzing set of traces associated with plans, we collect the uncertainty into belief states; conformant planning reduces to deterministic search in the space of belief states, called the *belief space*. The belief space for a given domain is basically the power-set of the set of states of the domain. For technical reasons, we explicitly restrict our reasoning to non-empty belief states, and define the belief space as $\mathit{Pow}^+(\mathcal{S}) \doteq \mathit{Pow}(\mathcal{S}) \setminus \emptyset$.

The execution of actions is lifted from states to belief states by the following definition.

Definition 7 (*Action applicability, execution*). An action α is applicable in a belief state Bs iff α is applicable in every state in Bs. If α is applicable in a belief state Bs, its execution in Bs, written $\textsc{Exec}[\alpha](\mathit{Bs})$, is defined as follows:

$$\textsc{Exec}[\alpha](\mathit{Bs}) \doteq \bigl\{s'\colon\ s \in \mathit{Bs} \text{ and } (s,\alpha,s') \in \mathcal{R}\bigr\}.$$

Definition 8 (*Plan applicability, execution*). The execution of plan π in a belief state Bs, written $\textsc{Exec}[\pi](\mathit{Bs})$, is defined as follows:

$$\begin{aligned}
&\textsc{Exec}[\varepsilon](\mathit{Bs}) \doteq \mathit{Bs},\\
&\textsc{Exec}[\pi](\bot) \doteq \bot,\\
&\textsc{Exec}[\alpha;\pi](\mathit{Bs}) \doteq \bot, \quad \text{if } \alpha \text{ is not applicable in } \mathit{Bs},\\
&\textsc{Exec}[\alpha;\pi](\mathit{Bs}) \doteq \textsc{Exec}[\pi]\bigl(\textsc{Exec}[\alpha](\mathit{Bs})\bigr), \quad \text{otherwise.}
\end{aligned}$$

$\bot$ is a distinguished symbol representing violation of action applicability. Plan π is applicable in a belief state Bs iff $\textsc{Exec}[\pi](\mathit{Bs}) \neq \bot$.

For a conformant planning problem, solutions are applicable plans, for which all the final states must be goal states (see Definition 6).

Recasting search from the space of plans to the belief space is straightforward: a plan π for $\mathcal{D}$ is a conformant solution to the planning problem $\langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ iff π is applicable in $\mathcal{I}$ and $\text{E}\textsc{xec}[\pi](\mathcal{I}) \subseteq \mathcal{G}$. This allows analyzing all the runs associated with a plan in one shot.

3.2. *Forward search in the belief space*

We now address the problem of searching the belief space in order to solve a conformant planning problem.

Example 5. Fig. 3 outlines (a subset of) the search space for the problem described in Example 4, constructed forward, starting from the initial set of states toward the goal. An expansion step consists in considering the belief states resulting from the execution of all the applicable actions. Notice that different plans can result in the same belief state, and that cycles are possible. Furthermore, even for this simple example, three different solutions are possible (since the three leftmost belief states are all goal states). Their characteristics are quite different: for instance, after the plan associated with the upper row (`GoNorth` twice; `GoEast` three times; `GoSouth`) the uncertainty in the robot location is eliminated.

Finally, notice the potential complexity of the problem. Since a belief state is a subset of $\mathcal{S}$, conformant planning amounts to searching all possible paths in the belief space. In the simple example depicted in Fig. 1, 15 states (assuming the hole is not an admissible condition) induce 2^{15} belief states (although not all of them are reachable).

The forward progression algorithm depicted in Fig. 4 searches the belief space, proceeding forward from the set of initial states $\mathcal{I}$ towards the goal $\mathcal{G}$. The algorithm can be seen as a standard best-first algorithm, where search nodes are (uniquely indexed by) belief states. *Open* contains a list of open nodes to be expanded, and *Closed* contains a

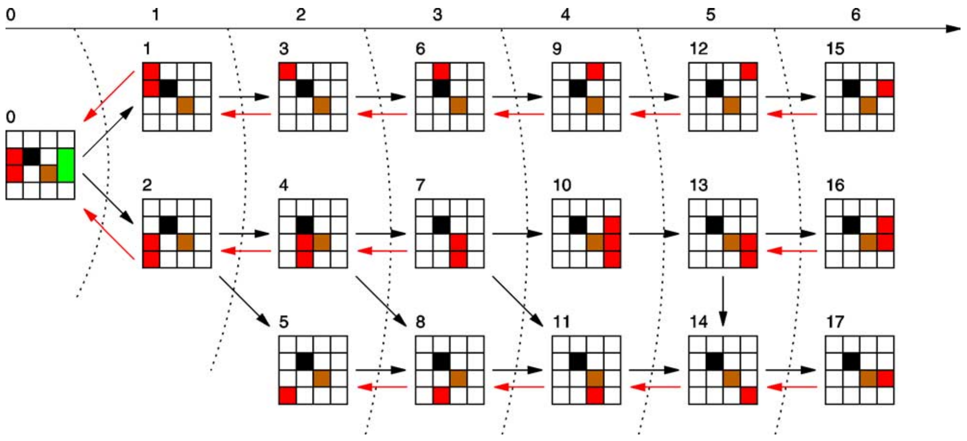


Fig. 3. A fragment of the forward search space for the robot navigation domain.

```
 1 function HeurConformantFwd($\mathcal{I}, \mathcal{G}$)
 2   $Open := \{\langle \mathcal{I} . \varepsilon \rangle\}$;
 3   $Closed := \emptyset$;
 4   $Solved := False$;
 5   while ($Open \neq \emptyset \wedge \neg Solved$) do
 6     $\langle Bs . \pi \rangle$ := ExtractBest($Open$);
 7     Insert($\langle Bs.\pi \rangle$, $Closed$);
 8     if $Bs \subseteq \mathcal{G}$ then
 9       $Solved := True$; $Solution := \pi$;
10     else
11       $BsExp$ := FwdExpandBs($Bs$);
12       $BsPList$ := PruneBsExpansion($BsExp$, $Closed$);
13       for $\langle Bs_i . \alpha_i \rangle$ in $BsPList$ do
14         Insert($\langle Bs_i . \pi ; \alpha_i \rangle$, $Open$)
15       done
16     endif
17   done
18   if $Solved$ then
19     return $Solution$;
20   else
21     return $Fail$;
22   endif
23 end
```

Fig. 4. The forward conformant planning algorithm.

list of closed nodes that have already been expanded. After the initialization phase, *Open* contains (the node indexed by) $\mathcal{I}$, while *Closed* is empty. The algorithm then enters a loop, where it extracts a node from the open list, stores it into the closed list, and checks if it is a success node (line 8) (i.e., it a subset of $\mathcal{G}$); if so, a solution has been found and the iteration is exited. Otherwise, the successor nodes are generated, and the ones that have already been expanded are pruned. The remaining nodes are stored in *Open*, and the iteration restarts. Each belief state *Bs* is associated with a plan π, that is applicable in $\mathcal{I}$, and that results exactly in *Bs*, i.e., $\textsc{Exec}[\pi](\mathcal{I}) = Bs$.

The algorithm loops (lines 5–17) until either a solution has been found ($Solved = True$) or all the search space has been exhausted ($Open = \emptyset$). A belief state *Bs* is extracted from the open pool (line 6), and it is inserted in closed pool (line 7). The belief states *Bs* is expanded (line 11) by means of the FwdExpandBs primitive. PruneBsExpansion (line 12) removes from the result of the expansion of *Bs* all the belief state that are in the *Closed*, and returns the pruned list of belief states. If *Open* becomes empty and no solution has been found, the algorithm returns with *Fail* to indicate that the planning problem admits no conformant solution. The expansion primitive FwdExpandBs takes as input a belief state *Bs*, and builds a set of pairs $\langle Bs_i . \alpha_i \rangle$ such that α_i is executable in *Bs* and the execution of α_i in *Bs* is contained in Bs_i. Notice that α_i is a conformant solution for the planning problem of reaching Bs_i from any non-empty subset of *Bs*.

$$\textsc{FwdExpandBs}(Bs) \doteq \big\{\langle Bs_i . \alpha_i \rangle \colon\ Bs_i = \textsc{Exec}[\alpha_i](Bs) \neq \bot\big\}.$$

Function PRUNEBSEXPANSION takes as input a result of an expansion of a belief state and *Closed*, and returns the subset of the expansion containing the pairs where each belief state has not been expanded. The PRUNEBSEXPANSION function can be defined as:

$$\text{PRUNEBSEXPANSION}(BsP, \mathit{Closed}) \doteq \big\{\langle Bs_i \,.\, \alpha_i\rangle\colon\ \langle Bs_i \,.\, \alpha_i\rangle \in BsP,\ \text{ and } \langle Bs_i \,.\, \pi\rangle \in \mathit{Closed} \text{ for no plan } \pi\big\}.$$

When an annotated belief state $\langle Bs \,.\, \pi\rangle$ is inserted in *Open*, INSERT checks if another annotated belief state $\langle Bs \,.\, \pi'\rangle$ exists; the length of π and π' are compared, and only the pair with the shortest plan is retained.

The algorithm described above can implement several search strategies, e.g., depth-first or breadth-first, depending on the implementation of the functions EXTRACTBEST (line 6) and INSERT (line 14). In the following sections, we will interpret the search algorithm as a standard A* algorithm, implementing a best-first heuristic search.

Example 6. A portion of the search space traversed by the algorithm while attempting to solve the problem of reaching $\{(\mathtt{0},\mathtt{1}),(\mathtt{0},\mathtt{2})\}$ from the $\{(\mathtt{3},\mathtt{1}),(\mathtt{3},\mathtt{2})\}$ is depicted in Fig. 3. Numbers on the upper left corner name belief states. *Open* is initialized with belief state 0 annotated with the empty plan. When expanded, it results in belief state 1 and 2 (`GoNorth` and `GoSouth` actions, respectively). 0 is also re-generated (action `GoEast`), but it is pruned away since already closed. Action `GoWest` is not applicable. Since neither 1 nor 2 are subsets of $\mathcal{G}$, they are added to *Open*. Say 1 is extracted; from its expansion we re-obtain belief states 0 (action `GoSouth`), pruned since closed, and 3 (action `GoEast`), new. Belief state 3 is not a subset of the goal, and it is thus added to *Open* for further expansion. The search proceeds by considering belief state 2 from *Open*. The expansion of belief state 2 leads to belief states 0, 4, 5. Belief state 0 has been visited before and is discarded, while belief states 4 and 5 that are not subset of the goal are added to *Open*.

The algorithm enjoys the following properties. First, it always terminates. Second, it returns a solution if the problem is solvable, otherwise it returns failure.

Theorem 1. *Let $P = \langle\mathcal{D},\mathcal{I},\mathcal{G}\rangle$ be a planning problem.* HEURCONFORMANTFWD$(\mathcal{I},\mathcal{G})$ *always terminates.*

Termination is proved by noticing that at each step, at least one belief state is extracted from *Open* and inserted into *Closed*, while a possibly empty set of new ⟨belief state, plan⟩ pairs, such that the belief state has not yet been visited, is inserted in *Open*. Given that the number of belief states is finite and we do not add to *Open* already visited belief states, the algorithm is guaranteed to terminate.

Theorem 2. *Let $P = \langle\mathcal{D},\mathcal{I},\mathcal{G}\rangle$ be a planning problem and let π be the value returned by the function* HEURCONFORMANTFWD$(\mathcal{I},\mathcal{G})$.

- *If $\pi \neq Fail$, then π is a conformant solution for the planning problem P.*
- *If $\pi = Fail$, instead, then there is no conformant solution for the planning problem P.*

```
 1 function HEURCONFORMANTBWD($\mathcal{I}, \mathcal{G}$)
 2   $Open := \{\langle \mathcal{G} . \varepsilon \rangle\}$;
 3   $Closed := \emptyset$;
 4   $Solved := False$;
 5   while ($Open \neq \emptyset \wedge \neg Solved$) do
 6     $\langle Bs.\pi \rangle :=$ EXTRACTBEST($Open$);
 7     INSERT($\langle Bs . \pi \rangle, Closed$);
 8     if $\mathcal{I} \subseteq Bs$ then
 9       $Solved := True$; $Solution := \pi$;
10     else
11       $BsExp :=$ BWDEXPANDBS($Bs$);
12       $BsPList :=$ PRUNEBSEXPANSION($BsExp, Closed$);
13       for $\langle Bs_i . \alpha_i \rangle$ in $BsPList$ do
14         INSERT($\langle Bs_i . \alpha; \pi_i \rangle, Open$)
15       done
16     endif
17   done
18   if $Solved$ then
19     return $Solution$;
20   else
21     return $Fail$;
22   endif
23 end
```

Fig. 5. The backward conformant planning algorithm.

To prove this theorem we notice that the invariant preserved by the algorithm is that each belief state plan pair $\langle Bs . \pi \rangle$ in the *Open* is such that π is applicable in $\mathcal{I}$ and $\text{EXEC}[\pi](\mathcal{I}) = Bs$. In other words, the plan is applicable in the initial belief state, and it results in the belief state *Bs*.

3.3. Backward search in the belief space

The dual for the algorithm described in Fig. 4 where the belief space is traversed backward, from the goal towards the initial belief state, is depicted in Fig. 5. The algorithms share the very same control structure, but different computations are performed. In particular, *Open* is initialized with the goal belief state annotated with the empty plan (line 2); the success test (at line 8) checks if the extracted belief state *Bs* contains $\mathcal{I}$ (rather than checking for containment in $\mathcal{G}$); the expansion of *Bs* (at line 11) is carried out by the BWDEXPANDBS primitive; finally, the annotated belief state inserted at line 14 is associated with a plan where the action α_i is prepended to π (rather than being appended as in the forward case).

BWDEXPANDBS, similarly to FWDEXPANDBS, takes as input a belief state *Bs*, and builds a set whose elements $\langle Bs_i . \alpha_i \rangle$ are such that α_i is executable in Bs_i and the execution of α_i in Bs_i is guaranteed to end up in states contained in *Bs*. Thus, α_i is a conformant solution for the planning problem of reaching *Bs* from any non-empty subset of Bs_i.

$$\text{BWDEXPANDBS}(Bs) \doteq \left\{\langle Bs_i . \alpha_i \rangle\colon\ Bs_i = \text{EXEC}^{-1}[\alpha_i](Bs) \neq \emptyset\right\},$$

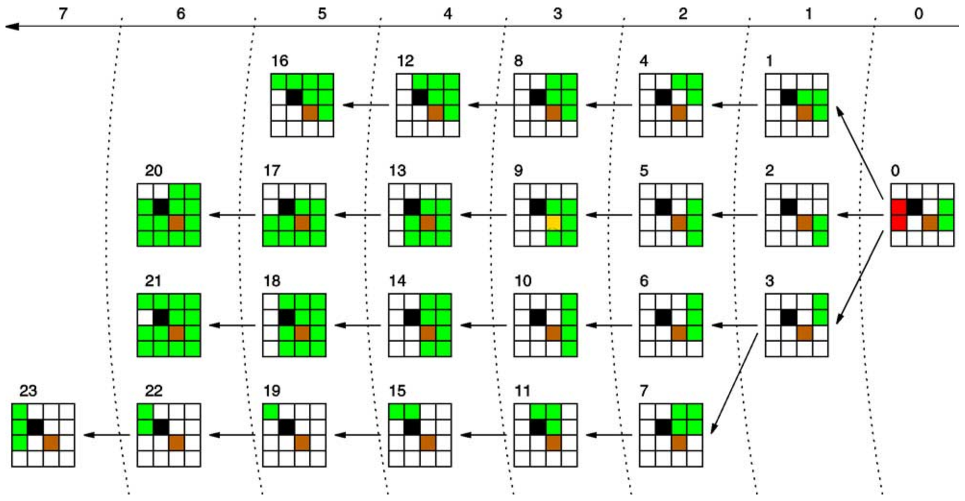


Fig. 6. A fragment of the backward search space for the robot navigation domain.

where

$$\text{E}{\scriptstyle\text{XEC}}^{-1}[\alpha](Bs)\bigl\{s\colon\ \text{A}{\scriptstyle\text{PPL}}(\alpha, s) \text{ and, for all } s' \in Bs, \mathcal{R}(s, \alpha, s')\bigr\}.$$

A fragment of the search space traversed by the algorithm of Fig. 5 is depicted in Fig. 6. Compare the expansion primitives for the backward and forward search algorithms: while proceeding backwards, the expansion step guarantees the applicability of the extended plans, while in the forward case we first need to compute the applicable actions, and then to project their effects. In the forward case, a plan is associated with the "minimal" belief state resulting from its execution in the initial condition; in the backward case, a plan is associated with the "maximal" set for which it guarantees reaching the goal. These differences may result in very different search spaces (compare also Figs. 3 and 6).

This algorithm enjoys the same properties of the one of Fig. 4. Termination can be proved similarly to the forward case. Correctness is proved by considering that the algorithm preserves the following invariants: each belief state plan pair $\langle Bs \,.\, \pi \rangle$ in *Open* is such that π is applicable in *Bs*, and $\text{E}{\scriptstyle\text{XEC}}[\pi](Bs) \subseteq \mathcal{G}$. In other words, the plan is applicable in the belief state it is associated with, and it ends up in the goal.

For both search directions, there are two critical factors that can affect the performance of the algorithms. The first is the ability to efficiently implement the basic steps, i.e., to store the visited belief states, and to compute the successors of a belief state. The second is the ability to drive the search in the "right" direction, i.e., to limit the number of expanded belief states. The former is addressed in the next section, by means of symbolic techniques; the latter is addressed in the following sections by means of heuristic search.

4. Conformant planning via symbolic model checking

In this section, we show how Symbolic Model Checking techniques can be extended to provide an efficient implementation platform for the search paradigm presented in previous section. Model Checking is a formal verification technique based on the exploration of finite state automata [25]. *Symbolic* model checking [37] is a particular form of model checking using Binary Decision Diagrams (BDDs) [11] to compactly represent and efficiently analyze finite state automata. The introduction of symbolic techniques into model checking led to a breakthrough in the size of models which could be analyzed [13], and made it possible for model checking to be routinely applied in industry, especially in the design of semiconductors (see [24] for a survey).

In the rest of this section, we will give an introduction to BDDs (Section 4.1), then we will show how BDDs are used to represent planning domains (Section 4.2); we will discuss the extension which allows to symbolically represent belief states and their transformations, thus allowing for an efficient implementation of the algorithm described in previous section (Section 4.3).

4.1. Binary decision diagrams

A Reduced Ordered Binary Decision Diagram [9,11] (improperly called BDD) is a directed acyclic graph (DAG). The terminal nodes are either *True* or *False*. Each non-terminal node is associated with a boolean variable, and two BDDs, called left and right branches. Fig. 7(a) depicts a BDD for $(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2) \wedge (a_3 \leftrightarrow b_3)$. At each non-terminal node, the right [left, respectively] branch is depicted as a solid [dashed, respectively] line, and represents the assignment of the value *True* [*False*, respectively] to the corresponding variable. A BDD represents a boolean function. For a given truth

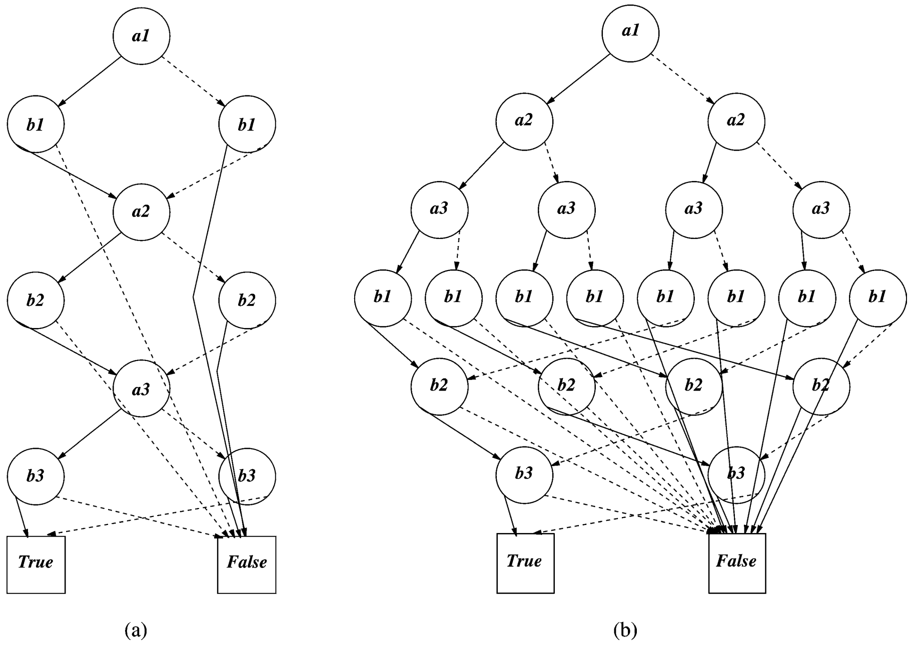


Fig. 7. Two BDD for the formula $(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2) \wedge (a_3 \leftrightarrow b_3)$.

assignment to the variables in the BDD, the value of the function is determined by traversing the graph from the root to the leaves, following each branch indicated by the value assigned to the variables.[1] The reached leaf node is labeled with the resulting truth value. If ϕ is a BDD, its size $|\phi|$ is the number of its nodes. If n is a node, $var(n)$ indicates the variable indexing node n.

BDDs are a canonical representation of boolean functions. The canonicity follows by imposing a total order $<$ over the set of variables used to label nodes, such that for any node n and respective non-terminal child m, their variables must be ordered, i.e., $var(n) < var(m)$, and requiring that the BDD contains no isomorphic subgraphs.

BDDs can be combined with the usual boolean transformations (e.g., negation, conjunction, disjunction). Given two BDDs, for instance, the conjunction operator builds and returns the BDD corresponding to the conjunction of its arguments. Substitution can also be represented as BDD transformations. In the following, if v is a variable, and Φ and ψ are BDDs, we indicate with $\Phi[v/\psi]$ the BDD resulting from the substitution of v with ψ in Φ. If $\boldsymbol{v}$ is a vector of BDD variables, we indicate with $|\boldsymbol{v}|$ the number of elements of the vector. If $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$ are vectors of distinct variables such that $|\boldsymbol{v_1}| = |\boldsymbol{v_2}|$, we indicate with $\Phi[\boldsymbol{v_1}/\boldsymbol{v_2}]$ the parallel substitution in Φ of the variables in vector $\boldsymbol{v_1}$ with the (corresponding) variables in $\boldsymbol{v_2}$.

BDDs also allow for transformations described as quantifications, in the style of Quantified Boolean Formulae (QBF). QBF is a definitional extension to propositional logic, where propositional variables can be universally and existentially quantified. In QBF, quantifiers can be arbitrarily applied and nested. In general, a QBF formula has an equivalent propositional formula, but the conversion is subject to an exponential blow-up. If Φ is a formula, and v_i is one of its variables, the existential quantification of v_i in Φ, written $\exists v_i\ .\ \Phi(v_1, \ldots, v_n)$, is equivalent to $\Phi(v_1, \ldots, v_n)[v_i/\mathit{False}] \vee \Phi(v_1, \ldots, v_n)[v_i/\mathit{True}]$. Analogously, the universal quantification $\forall v_i\ .\ \Phi(v_1, \ldots, v_n)$ is equivalent to $\Phi(v_1, \ldots, v_n)[v_i/\mathit{False}] \wedge \Phi(v_1, \ldots, v_n)[v_i/\mathit{True}]$. In terms of BDD computations, a quantification corresponds to a transformation mapping the BDD of Φ and the variable v_i being quantified into the BDD of the resulting (propositional) formula.

The time complexity of the algorithm for computing a truth-functional boolean transformation $f_1\langle op\rangle f_2$ is $\mathrm{O}(|f_1| \cdot |f_2|)$. As far as quantifications are concerned, the time complexity is quadratic in the size of the BDD being quantified, and linear in the number of variables being quantified, i.e., $\mathrm{O}(|\boldsymbol{v}| \cdot |f|^2)$ [9,11].

BDD *packages* are efficient implementations of such data structures and algorithms (see [7,26,45,46]). Basically, a BDD package deals with a single multi-rooted DAG, where each node represents a boolean function. Memory efficiency is obtained by using a “unique table”, and by sharing common subgraphs between BDDs. The unique table is used to guarantee that at each time there are no isomorphic subgraphs and no redundant nodes in the multi-rooted DAG. Before creating a new node, the unique table is checked to see if the node is already present, and only if this is not the case a new node is created and stored in the unique table. The unique table allows performing the equivalence check between two

[1] A path from the root to a leaf can visit nodes associated with a subset of all the variables of the BDD. See, for instance, the path associated with $a_1, \neg b_1$ in Fig. 7(a).

BDDs in constant time (since two equivalent functions always share the same subgraph) [7, 45]. Time efficiency is obtained by maintaining a “computed table”, which keeps track of the results of recently computed transformations, thus avoiding the re-computation via memoization.

A critical computational factor with BDDs is the order of the variables. (Fig. 7 shows an example of the impact of a change in the variable ordering on the size of a BDD.) For a certain class of boolean functions, the size of the corresponding BDD is exponential in the number of variables for any possible variable ordering [10]. In many practical cases, however, finding a good variable ordering is rather easy. Beside affecting the memory used to represent a boolean function, finding a good variable ordering can have a big impact on computation times, since the complexity of the transformation algorithms depends on the size of the operands. Most BDD packages provide heuristic algorithms for finding good variable orderings, which can be called to try to reduce the overall size of the stored BDDs. The reordering algorithms can also be activated dynamically by the package, during a BDD computation, when the total amount of nodes in the package reaches a predefined threshold (dynamic reordering).

4.2. Symbolic representation of planning domains

A planning domain $\mathcal{D} = \langle \mathcal{X}, \mathcal{V}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ can be represented symbolically using BDDs, as follows. A set of (distinct) BDD variables, called *state* variables, is devoted to the representation of the states $\mathcal{S}$ of the domain. These variables have a direct association with the atomic propositions of the domain. For non-boolean variables, it is common practice in Symbolic Model Checking to use a logarithmic bit encoding. For instance, the robot navigation domain of Fig. 1 is described by means of two variables x and y both having range $\{0, 1, 2, 3\}$. Each range can be encoded using two boolean variables ($x_{1,2}$ and $y_{1,2}$, respectively), with the relation to the values of the two state variables depicted in Fig. 8.

In the following, we write $\boldsymbol{x}$ for the vector of (BDD variables representing the) state variables of the domain. Because the particular order is irrelevant but for performance issues, in the rest of this section we will not distinguish between a proposition and the corresponding BDD representation.

A state is a complete set of assignments to state variables (that corresponds to the propositions which are intended to hold in it). For each state s, there is a corresponding assignment to the state variables $\boldsymbol{x}$, i.e., the assignment where each variable corresponding to a proposition p such that $s \models p$ is assigned *True*, and each other variable is assigned *False*. We represent s with the BDD $\xi(s)$, having such an assignment as its unique satisfying assignment. For instance, the BDD for the state (0,2), written $\xi((\mathtt{0}, \mathtt{2}))$, is $(\neg x_1 \wedge \neg x_2) \wedge (y_1 \wedge \neg y_2)$.

$x = 0 \mapsto \neg x_1 \wedge \neg x_2,$	$y = 0 \mapsto \neg y_1 \wedge \neg y_2,$
$x = 1 \mapsto \neg x_1 \wedge x_2,$	$y = 1 \mapsto \neg y_1 \wedge y_2,$
$x = 2 \mapsto x_1 \wedge \neg x_2,$	$y = 2 \mapsto y_1 \wedge \neg y_2,$
$x = 3 \mapsto x_1 \wedge x_2,$	$y = 3 \mapsto y_1 \wedge y_2.$

Fig. 8. A boolean encoding for variables x and y.

The above representation naturally extends to any *set of states* $Q \subseteq \mathcal{S}$ as follows:

$$\xi(Q) \doteq \bigvee_{s \in Q} \xi(s).$$

In other words, we associate a set of states with the generalized disjunction of the BDDs representing each of the states. Notice that the satisfying assignments of the $\xi(Q)$ are exactly the assignment representations of the states in Q. This representation mechanism is very natural. For instance, the BDD $\xi(\mathcal{I})$ representing the set of initial states of the robot navigation domain $\mathcal{I} \doteq \{(\mathtt{0}, \mathtt{1}), (\mathtt{0}, \mathtt{2})\}$ is:

$$\xi(\mathcal{I}) \doteq \big((\neg x_1 \wedge \neg x_2) \wedge \big((\neg y_1 \wedge y_2) \vee (y_1 \wedge \neg y_2)\big)\big)$$

while for the set of goal states $\mathcal{G} \doteq \{(\mathtt{3}, \mathtt{1}), (\mathtt{3}, \mathtt{2})\}$ the corresponding BDD is

$$\xi(\mathcal{G}) \doteq \big((x_1 \wedge x_2) \wedge \big((\neg y_1 \wedge y_2) \vee (y_1 \wedge \neg y_2)\big)\big).$$

A BDD is also used to represent the set $\mathcal{S}$ of all the states of the planning domain. In the robot navigation example we are considering, $\xi(\mathcal{S}) = \mathit{True}$ because $\mathcal{S} = \mathit{Pow}(\mathcal{P})$.

In a different formulation, where *independent* propositions $x = i$ $(y = i)$, while $x = i$ $(y = i)$ is a BDD variable, are used to represent the assignments to state variables, $\xi(\mathcal{S})$ would be the BDD:

$$\bigvee_{i=0..3} \bigvee_{j=0..3} \left(\left((x = i) \leftrightarrow \bigwedge_{i \neq k=0..3} \neg(x = k)\right) \wedge \left((y = j) \leftrightarrow \bigwedge_{j \neq k=0..3} \neg(y = k)\right)\right).$$

In general, a BDD represents the set of (states which correspond to) its models. As a consequence, set theoretic transformations are naturally represented by propositional operations, as follows:

$$\xi(\mathcal{S} \setminus Q) \doteq \xi(\mathcal{S}) \wedge \neg\xi(Q),$$
$$\xi(Q_1 \cup Q_2) \doteq \xi(Q_1) \vee \xi(Q_2),$$
$$\xi(Q_1 \cap Q_2) \doteq \xi(Q_1) \wedge \xi(Q_2).$$

The main efficiency of this symbolic representation lies in the fact that the cardinality of the represented set is not directly related to the size of the BDD. For instance, $\xi([\![x = 0]\!]) = \xi(\{(\mathtt{0}, \mathtt{0}), (\mathtt{0}, \mathtt{1}), (\mathtt{0}, \mathtt{2}), (\mathtt{0}, \mathtt{3})\}) = \neg x_1 \wedge \neg x_2$ uses two (non-terminal) nodes to represent four states (see Fig. 9), while the set of states where state variable y assume a value in the set $\{1, 2\}$ is represented with three non-terminal nodes. $\xi(\mathcal{I})$ uses five non-terminal nodes to represent two states. As limit cases, for this example $\xi(\mathcal{S})$ and $\xi(\{\})$ are (the leaf BDDs) *True* and *False*, respectively. As a further advantage, symbolic representation is extremely efficient in dealing with irrelevant information. Notice, for instance, that only the boolean variable x_1 occurs in $\xi([\![x \in \{2, 3\}]\!]) = \xi(\{(\mathtt{2}, \{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}\}), (\mathtt{3}, \{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}\})\}) = x_1$. For this reason, a symbolic representation can have dramatic advantages over an explicit, enumerative representation. This is what allows symbolic, BDD-based model checkers to handle finite state automata with a very large number of states (see, for instance, [13]).

Another set of BDD variables, called *action* variables, written $\boldsymbol{\alpha}$, is used to represent actions. We use one action variable for each possible action in $\mathcal{A}$. Intuitively, a BDD action variable is true if and only if the corresponding action is being executed. If we assume

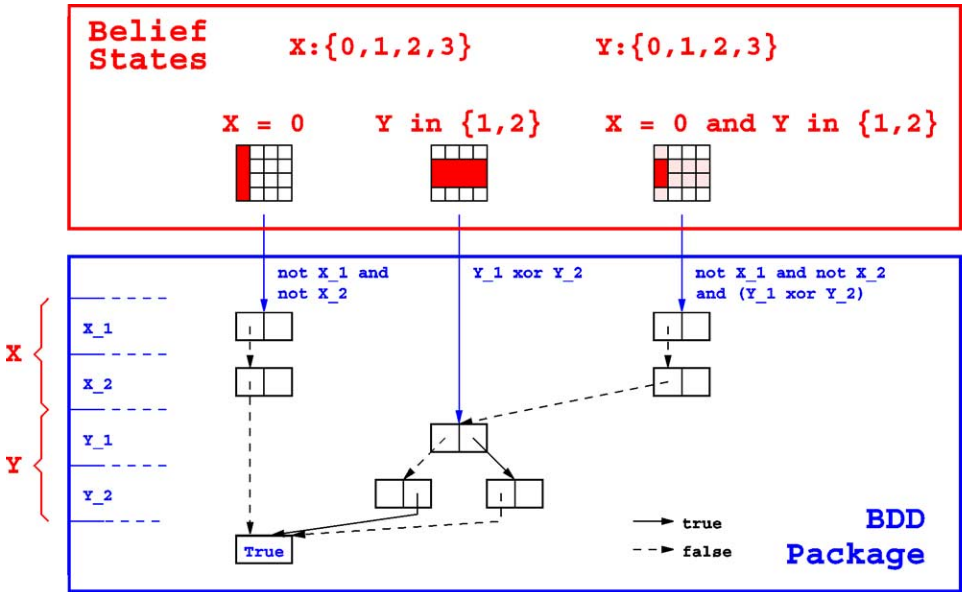


Fig. 9. A symbolic representation of belief states for the robot navigation domain.

that a sequential encoding is used, i.e., no concurrent actions are allowed, we also use a BDD, SEQ($\boldsymbol{\alpha}$), to express that exactly one of the action variables must be true at each time.[2] For the robot navigation problem, where $\mathcal{A}$ contains four actions, we use the four BDD variables `GoNorth`, `GoSouth`, `GoWest` and `GoEast`, while we express the serial encoding constraint with the following BDD:

$$\begin{aligned}\mathrm{SEQ}(\boldsymbol{\alpha}) \doteq {} & (\mathtt{GoNorth} \vee \mathtt{GoSouth} \vee \mathtt{GoWest} \vee \mathtt{GoEast}) \\ & \wedge \neg(\mathtt{GoNorth} \wedge \mathtt{GoSouth}) \wedge \neg(\mathtt{GoNorth} \wedge \mathtt{GoWest}) \\ & \wedge \neg(\mathtt{GoNorth} \wedge \mathtt{GoEast}) \wedge \neg(\mathtt{GoSouth} \wedge \mathtt{GoWest}) \\ & \wedge \neg(\mathtt{GoSouth} \wedge \mathtt{GoEast}) \wedge \neg(\mathtt{GoWest} \wedge \mathtt{GoEast}).\end{aligned}$$

Similarly to the case of state variables, we are referring to BDD action variables with symbolic names for the sake of simplicity. The position of the BDD action variables in the ordering of the BDD package is totally irrelevant in logical terms but is relevant to performance issues.

A BDD in the variables $\boldsymbol{x}$ and $\boldsymbol{\alpha}$ represents a set of state-action pairs, i.e., a relation between states and actions. For instance, the applicability relation that says that action

[2] In the specific case of sequential encoding, an alternative approach using only $\lceil \log |\mathcal{A}| \rceil$ is possible: an assignment to the action variables denotes a specific action to be executed. Since two assignments are mutually exclusive, the constraint SEQ($\boldsymbol{\alpha}$) needs not to be represented. When the cardinality of the set of actions is not a power of two, the standard solution is to associate more than one assignment to certain values. This optimized solution, which is actually used in the implementation, is not described here for the sake of simplicity.

`GoEast` is applicable in any state but $(\mathtt{0},\mathtt{1})$ is represented by the BDD $\mathtt{GoEast} \wedge \xi(\mathcal{S} \setminus \{(\mathtt{0},\mathtt{1})\}) = \mathtt{GoEast} \wedge \neg x_1 \wedge \neg x_2 \wedge \neg y_1 \wedge y_2$. Notice that it represents a set of 15 state-action pairs, each associating a state with the `GoEast` action.

A transition is a 3-tuple composed of a state (the initial state of the transition), an action (the action being executed), and a state (the resulting state of the transition). To represent transitions, another vector $\boldsymbol{x}'$ of BDD variables, called *next state* variables, is allocated in the BDD package. We write $\xi'(s)$ for the representation of the state s in the next state variables. We use $\xi'(Q)$ to denote the construction of the BDD corresponding to the set of states Q, using each variable in the next state vector $\boldsymbol{x}'$ in place of each current state variables in $\boldsymbol{x}$. We require that $|\boldsymbol{x}| = |\boldsymbol{x}'|$, and assume that the variables in similar positions in $\boldsymbol{x}$ and in $\boldsymbol{x}'$ correspond. We define the representation of a set of states in the next variables as $\xi'(s) \doteq \xi(s)[\boldsymbol{x}/\boldsymbol{x}']$. We call the operation $\Phi[\boldsymbol{x}/\boldsymbol{x}']$ “forward shifting”, because it transforms the representation of a set of “current” states into the representation of an analogous set of “next” states, where each state is renamed to its corresponding “next state”. The dual operation $\Phi[\boldsymbol{x}'/\boldsymbol{x}]$ is called backward shifting. In the following, we call $\boldsymbol{x}$ *current* state variables to distinguish them from next state variables $\boldsymbol{x}'$. A transition is represented as an assignment to $\boldsymbol{x}$, $\boldsymbol{\alpha}$ and $\boldsymbol{x}'$. For the robot navigation domain, the transition corresponding to the application of action `GoEast` in state $(\mathtt{0},\mathtt{0})$ resulting in state $(\mathtt{1},\mathtt{0})$ is represented by the following BDD

$$\xi\big(\big\langle(\mathtt{0},\mathtt{0}), \mathtt{GoEast}, (\mathtt{1},\mathtt{0})\big\rangle\big) \doteq \xi\big((\mathtt{0},\mathtt{0})\big) \wedge \mathtt{GoEast} \wedge \xi'\big((\mathtt{1},\mathtt{0})\big).$$

The transition relation $\mathcal{R}$ of the automaton corresponding to a planning domain is simply a set of transitions, and is thus represented by a BDD in the BDD variables $\boldsymbol{x}$, $\boldsymbol{\alpha}$ and $\boldsymbol{x}'$, where each satisfying assignment represents a possible transition.

$$\xi(\mathcal{R}) \doteq \mathrm{SEQ}(\boldsymbol{\alpha}) \wedge \bigvee_{t\in\mathcal{R}} \xi(t).$$

In the rest of this paper, we assume that the BDD representation of a planning domain is given. In particular, we assume as given the vectors of variables $\boldsymbol{x}$, $\boldsymbol{x}'$, $\boldsymbol{\alpha}$, the encoding functions ξ and ξ', and we simply call $\mathcal{S}$, $\mathcal{R}$, $\mathcal{I}$ and $\mathcal{G}$ the BDD representing the states of the domain, the transition relation, the initial states and the goal states, respectively. We write $\Phi(\boldsymbol{v})$ to stress that the BDD Φ depends on the variables in $\boldsymbol{v}$. With this representation, it is possible to reason about plans, simulating symbolically the execution of sets of actions in sets of states, by means of QBF transformations. The $\mathrm{APPL}(\boldsymbol{x},\boldsymbol{\alpha})$ BDD representing the applicability relation can be directly obtained with the following computation:

$$\mathrm{APPL}(\boldsymbol{x},\boldsymbol{\alpha}) \doteq \exists \boldsymbol{x}' \,.\, \mathcal{R}(\boldsymbol{x},\boldsymbol{\alpha},\boldsymbol{x}').$$

The BDD representing the states reachable from Q by executing an action $a \in \mathcal{A}$ is directly obtained by means of the following computation:

$$\mathrm{EXEC}[a]\big(Q(\boldsymbol{x})\big) \doteq \exists \boldsymbol{x} \,.\, \exists \boldsymbol{\alpha} \,.\, \big(\mathcal{R}(\boldsymbol{x},\boldsymbol{\alpha},\boldsymbol{x}') \wedge Q(\boldsymbol{x}) \wedge \xi(a)\big)[\boldsymbol{x}'/\boldsymbol{x}].$$

The BDD representing the states reachable from Q in one step is obtained with the following computation:

$$\exists \boldsymbol{x} \,.\, \exists \boldsymbol{\alpha} \,.\, \big(\mathcal{R}(\boldsymbol{x},\boldsymbol{\alpha},\boldsymbol{x}') \wedge Q(\boldsymbol{x})\big)[\boldsymbol{x}'/\boldsymbol{x}].$$

Notice that, with this single operation, we symbolically simulate the effect of the application of any applicable action in $\mathcal{A}$ to any of the states in Q. Also, in the following, we will refer indifferently to sets of states (or actions), or to the BDDs that represent them.

4.3. Symbolic search in the belief space

In symbolic model checking, BDDs provide a way for compactly representing and efficiently expanding a search frontier for search in the space of states. This machinery is used, with small variations, in [23], where conditional plans are constructed by symbolic breadth-first search in the space of states under the hypothesis of full observability. However, this machinery cannot be applied to tackle conformant planning. Our approach is rather based on the fact that a belief state $Bs \subseteq \mathcal{S}$ is directly represented by the corresponding BDD $Bs(\boldsymbol{x})$. The basic step of the algorithms is the expansion of a belief state that results in a list of newly expanded nodes, together with an update of the costs.

FwdExpandBs given a belief state *Bs* computes the set of belief state action pairs $\langle Bs_i . \alpha \rangle$ such that α is applicable in *Bs* and $\textsc{Exec}[Bs](\alpha) = Bs_i$. This computation can be performed symbolically as follows:

$$\begin{aligned}&\textsc{FwdExpandBs}\big(Bs(\boldsymbol{x})\big)\\&\quad \doteq \Big(\exists \boldsymbol{x} . \Big(Bs(\boldsymbol{x}) \wedge \mathcal{R}(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{x}') \wedge \big(\forall \boldsymbol{x} . \big(Bs(\boldsymbol{x}) \rightarrow \textsc{Appl}(\boldsymbol{x}, \boldsymbol{\alpha})\big)\big)\Big)\Big)[\boldsymbol{x}'/\boldsymbol{x}].\end{aligned}$$

The result is a formula in variables $\boldsymbol{x}$ and $\boldsymbol{\alpha}$ such that the satisfying assignments represent the belief state action pair $\langle Bs_i . \alpha \rangle$ where the action is applicable in *Bs* and whose execution in *Bs* necessarily results is a state in Bs_i. The dual backward computation, BwdExpandBs, is performed symbolically as follows:

$$\textsc{BwdExpandBs}\big(Bs(\boldsymbol{x})\big) \doteq \forall \boldsymbol{x}' . \big(\mathcal{R}(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{x}') \rightarrow Bs(\boldsymbol{x})[\boldsymbol{x}/\boldsymbol{x}']\big) \wedge \textsc{Appl}(\boldsymbol{x}, \boldsymbol{\alpha}).$$

The resulting formula is in variables $\boldsymbol{x}$ and $\boldsymbol{\alpha}$ and the satisfying assignments represents the belief state action pair $\langle Bs_i . \alpha \rangle$ such that the action is applicable in Bs_i and the execution of α in Bs_i necessarily results in a state in *Bs*.

In terms of the implementation, we exploit BDD technology in a fundamental way, to directly link the symbolic machinery with the search framework [4]. Given the canonical form of BDDs, a belief state is simply a pointer to the unique corresponding BDD in the BDD package. Fig. 9 shows how sets of states represented by formulas over fluents (top part of figure) are mapped as BDDs into a BDD package, using the boolean encoding of Fig. 8. For instance, the set of states where the fluent x has value 0 (leftmost, top) corresponds to the BDD for the formula $\neg x_1 \wedge \neg x_2$ (leftmost, bottom). The (BDD pointers representing) belief states generated during the search are stored by means of a hash table, external to the BDD package; this is accessed directly using BDD pointers as keys. In this way, it is very easy to associate additional information to belief states (e.g., the corresponding plan, heuristic information, or marks to avoid the explicit traversal of the *Open* and *Closed* lists). The PruneBsExpansion primitive is also implemented efficiently in terms of BDDs: it takes as input a BDD in the state and action variables, that represents a set of belief state-action pairs. It returns a list of pairs, where the first element is a BDD representing a belief state that has not been expanded, and the second is the corresponding action. Fig. 10 depicts the

```
 1 function PRUNEVISITEDBS($B$, $StateVars$)
 2    if ($B = False$) then
 3       return ($B$);
 4    endif
 5    if (BDD_VAR($B$) $\notin StateVars$) then
 6       $T$ := PRUNEVISITEDBS(BDD_GETTHEN($B$), $StateVars$);
 7       $E$ := PRUNEVISITEDBS(BDD_GETELSE($B$), $StateVars$);
 8       $result$ := BDD_ IFTHENELSE(BDD_VAR($B$), $T$, $E$);
 9    else
10       if (VISITED($B$)) then
11          $result := False$;
12       else
13          $result := B$;
14       endif
15    endif
16    return $result$;
17 end;
```

Fig. 10. The pruning transformation.

high level description of the ad-hoc BDD transformation PRUNEVISITEDBS for pruning previously visited belief states. It takes as input the BDD B in action variables and state variables and the set of state variables *StateVars*. PRUNEVISITEDBS is implemented as a recursive traversal of the input BDD, in the action and state variables. For the sake of clarity, let us assume that action variables precede state variables in the BDD ordering. PRUNEVISITEDBS interprets as a belief state every node associated with a state variable, having a parent node associated with an action variable. Nodes corresponding to previously visited belief states are pruned by substituting *False* for the belief state, while new ones are simply returned. This BDD operation is at the basis of the efficient implementation of the PRUNEBSEXPANSION primitive. If the top variable of BDD B is an action variable (line 5), then the transformation is recursively called on T ("then") and E ("else") part of the BDD (lines 6 and 7, respectively). The two partial results are then combined in the BDD package by BDD_IFTHENELSE (line 8), the primitive for constructing a BDD, which takes into account the issues of minimization and uniqueness [11,45]. If the top variable is a state variable (lines 9), then the BDD represents a belief state. If the belief state has already been visited (i.e., it is marked as "closed"), *False* is returned (line 11), otherwise it is returned (line 13).

5. Heuristic search in belief space

The algorithms presented in Section 3 can implement different search styles, depending on the way nodes are inserted in and extracted from *Open*. For instance, a depth-first search (DFS) can be implemented by selecting the more recently inserted nodes, while breadth-first search (BFS) is obtained by extracting less recently inserted nodes. We assume now that at each step the belief states are selected for extraction from the *Open* based on a score function f, so that the algorithms become instances of A^* algorithms [39]. The score function f for a search node is the sum of g, i.e., the cost to reach the node from the initial node, and of h, i.e., the estimate of the cost to reach a final node. In the algorithms presented in Section 3, search nodes are of the form $\langle Bs\,.\,\pi\rangle$. In the forward case, the selection function $f(\langle Bs\,.\,\pi\rangle)$, used to extract the "best" node from *Open*, is defined as the sum of $g(\langle Bs\,.\,\pi\rangle) = |\pi|$, i.e., the length of plan π for reaching *Bs* from $\mathcal{I}$, plus an estimate $h(\langle Bs\,.\,\pi\rangle)$ of the cost to reach $\mathcal{G}$ from *Bs*. In the backward case, $g(\langle Bs\,.\,\pi\rangle) = |\pi|$, i.e., the length of plan π for reaching $\mathcal{G}$ from *Bs*, plus an estimate $h(\langle Bs\,.\,\pi\rangle)$ of the cost to reach *Bs* from $\mathcal{I}$. The key property of A* algorithms is that if the function h is admissible [39], i.e., it underestimates the actual optimal cost $h^*(\langle Bs\,.\,\pi\rangle)$ to reach the target belief state, then the algorithm is guaranteed to return optimal solutions. As usual, it is trivial to find an admissible but uninformative heuristic. For instance, the function associating 0 to each $\langle Bs\,.\,\pi\rangle$ results in a breadth-first search. The challenge is to find an "informative" heuristic, i.e., a function that is able to provide accurate estimates of the "cost to go" for the open search nodes.

5.1. Symbolic reachability heuristics

We now define a set of heuristic functions for belief states. The function MAXSDIST, reported in Fig. 11, takes as input two belief states, FROM and TO. It proceeds backwards, starting from the TO set, repeatedly applying the STRONGPREIMAGE computation and incrementing the counter i; at each iteration, it constructs a layer of states at increasing distance from TO. The algorithm terminates when the FROM belief state is contained in SOLVED[i], in which case the index i is returned, or when the iteration reaches a fixed point and no new states can be added, in which case it returns ∞. The basic expansion operation is the STRONGPREIMAGE primitive, defined as

$$\begin{aligned}&\textsc{StrongPreImage}(Bs)\\ &\quad\doteq \bigl\{s\colon \exists\alpha\in\mathcal{A}\,.\,\bigl(\textsc{Appl}(\alpha,s)\wedge\forall s'\in\mathcal{S}\,.\,\bigl(R(s,\alpha,s')\rightarrow s'\in Bs\bigr)\bigr)\bigr\}.\end{aligned}$$

This primitive guarantees that, for each $s \in$ STRONGPREIMAGE(*Bs*), there exists an action α that, if applied in s, will necessarily result in states in *Bs*.

```
function MAXSDIST(FROM, TO);
  $i := 0$;
  SOLVED[0] := LAYER[0] := TO;
  while ((FROM $\not\subseteq$ SOLVED[$i$]) $\wedge$ (LAYER[$i$] $\neq \emptyset$)) do
    SOLVED[$i+1$] := SOLVED[$i$] $\cup$ STRONGPREIMAGE(SOLVED[$i$]);
    LAYER[$i+1$] := SOLVED[$i+1$] \ SOLVED[$i$];
    $i := i+1$;
  done;
  if (FROM $\not\subseteq$ SOLVED[$i$]) then
    return $\infty$;
  else
    return $i$;
  endif;
end;
```

Fig. 11. The algorithm for the computation of the Maximal Strong Distance.

The MAXSDIST algorithm is a simplified version of STRONGPLAN, the algorithm for strong planning in nondeterministic domains under full observability [19]. The algorithm returns a state-action table, i.e., a memoryless conditional plan, which guarantees goal achievement in a finite number of steps, regardless of nondeterminism. The set LAYER$[i+1]$ contains all the states for which there exists an action that is guaranteed to enter SOLVED$[i]$, regardless of nondeterministic action effects. Notice the difference with respect to conformant planning: in strong planning under full observability, each state can be associated with the most appropriate action to proceed towards the goal. In conformant planning, on the other hand, all the states in a belief state are not distinguishable, and must all be associated with the *same* action. Therefore, strong planning under full observability can be seen as a relaxation of the problem of conformant planning.

The MAXSDIST algorithm always terminates. This follows from the fact that (a) $\mathcal{S}$ is finite, (b) for any i SOLVED$[i] \subseteq \mathcal{S}$, and (c) the sequence of SOLVED is monotonically increasing, i.e., for any i, either (c.1) SOLVED$[i] \subset$ SOLVED$[i+1]$, or (c.2) SOLVED$[i] =$ SOLVED$[i+1]$, in which case MAXSDIST is exited. The MAXSDIST algorithm can be used to provide admissible heuristics. In the forward case, the cost estimate for the belief state *Bs* is given by MAXSDIST$(Bs, \mathcal{G})$, while in the backward case it is MAXSDIST$(\mathcal{I}, Bs)$. In both search directions, MAXSDIST provides admissible heuristics, and therefore allows construction of optimal plans.

Theorem 3. *Let the optimal cost function for conformant planning h^* be defined as follows*:

$$h^*(Bs, Bs') = \begin{cases} \infty & \textit{if the problem } \langle Bs, Bs'\rangle \textit{ admits no conformant solution}, \\ |\pi| & \textit{where } \pi \textit{ is a conformant solution of optimal length} \\ & \textit{for } \langle Bs, Bs'\rangle. \end{cases}$$

Then, MAXSDIST$(Bs, Bs') \leqslant h^*(Bs, Bs')$.

It is possible to obtain different heuristic functions as variations of the MAXSDIST algorithm. The MAXWDIST is obtained by substituting the STRONGPREIMAGE primitive, at line 5, with the WEAKPREIMAGE primitive:

$$\text{WEAKPREIMAGE}(Bs) \doteq \{s\colon \exists \alpha \in \mathcal{A} \,.\, \exists s' \in \mathcal{S} \,.\, (R(s, \alpha, s') \wedge s' \in Bs)\}$$

which collects all the states that have a possibility (but may not be guaranteed) to reach *Bs* in one step. The main difference between MAXSDIST and MAXWDIST is that the former considers the worst case with respect to nondeterministic action effects, while MAXWDIST is optimistic.

Both algorithms terminate when FROM $\subseteq$ SOLVED$[i]$, i.e., when the "farthest" state is included. This explains the "max" in the name, since MAXSDIST$(Bs, \mathcal{G}) = \max_{s \in Bs}$ MAXSDIST$(\{s\}, \mathcal{G})$. The corresponding "optimistic" versions, MINSDIST and MINWDIST, are obtained by replacing the test (FROM $\not\subseteq$ SOLVED$[i]$) at lines 4 and 9 with ((FROM $\cap$ SOLVED$[i]) \neq \emptyset$). These algorithms stop as soon as a non-empty intersection between FROM and the currently expanded layer is obtained. It is easy to see that MAXSDIST is in general more informed than MAXWDIST, i.e., MAXWDIST$(Bs, Bs') \leqslant$ MAXSDIST(Bs, Bs'). Consider in fact that STRONGPREIMAGE$(Bs) \subseteq$ WEAKPREIMAGE (Bs). Notice also that, when the domain is deterministic, STRONGPREIMAGE$(Bs) =$

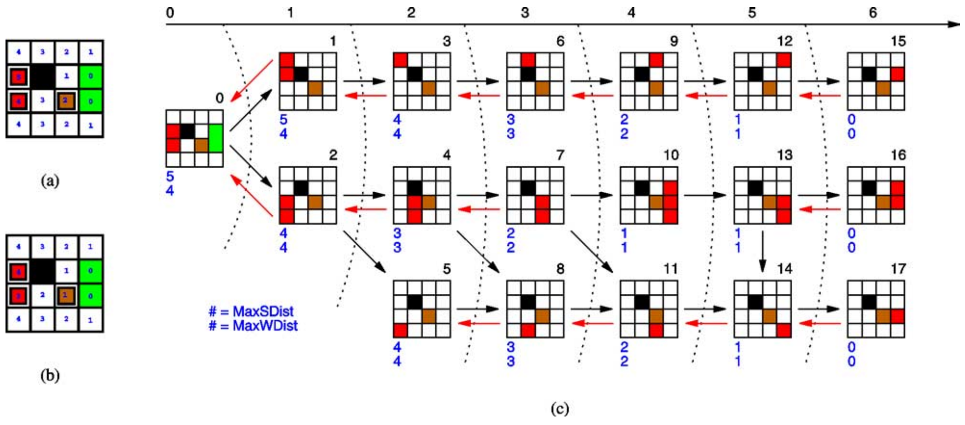


Fig. 12. The values of MAXSDIST and MAXWDIST on the example domain (left), and the heuristic values for the explored belief states (right).

WEAKPREIMAGE(Bs), from which MAXSDIST is as informed as MAXWDIST. Similarly, MAXSDIST [MAXWDIST, respectively] is more informed than MINSDIST [MINWDIST, respectively], from the fact that the layers are the same, but the termination test in the Max case (complete inclusion) is strictly stronger than the Min case (non-empty intersection). The convenience in using a less informed heuristic may be in fact that they can be easier to compute, and that convergence may be reached in a lower number of iterations. In general, however, a more informed heuristics is guaranteed not to expand more nodes than a less informed heuristic.

Example 7. The scores computed by MAXSDIST and MAXWDIST for each state in the case of the example of Fig. 1 are shown in Fig. 12(c). The goal states have score 0. In the case of MAXWDIST the score is basically the manhattan distance. MAXSDIST differs in the score associated to the slippery spot, that is at distance 2: although there is a possibility of reaching the goal in one step there is no way to guarantee that the `GoEast` action will not result in (3.3). The approach is iterated to associate each state with the reported value. In Fig. 12(c), the MAXSDIST and the MAXWDIST scores associated with each of the belief states in the fragment of the forward search space are reported.

Some final remarks. Our approach is very similar in spirit to [6]; in fact, the MAXSDIST computes exactly the same values of the V^*_{dp} heuristic used in GPT (from now on we use V^*_{dp} and MAXSDIST interchangeably). Here we present a class of heuristics, we show that they can be used with both search directions, and with different degrees of informedness. Furthermore, our algorithms are directly implemented by means of symbolic techniques.

When searching in the forward direction, MAXSDIST (or its variations) is called with different values to the FROM parameter, but the TO parameter is always bound to $\mathcal{G}$. It is therefore possible to precompute the distance layers SOLVED, and compute MAXSDIST($Bs, \mathcal{G}$) as a sequence of containment checks $Bs \subseteq$ SOLVED[i], for increasing i; depending on the distance at which a fixed point is reached, it may be convenient to implement the search by means of a bisection procedure, so that the results can be found in at most $\lceil \log(M) \rceil$ subsumption steps, M being the number of iteration of the algorithm to reach the fixpoint; further reductions can be obtained by implementing the construction of layers on demand. Unfortunately, in the backward case the computation is to be re-executed for each belief state to be scored, since it is the first parameter that is fixed. This makes this approach less amenable in practice to backward search.

5.2. *Experimental evaluation*

The reachability heuristics described above are extremely informative. In the case of a classical planning problem, where a single initial state is given and the domain is deterministic, they eliminate the need for search: exactly as many nodes as the number of actions in the returned optimal plan are expanded. In the case of nondeterministic domains, MAXSDIST encodes an optimal course of action under the hypothesis of full observability, i.e., it suggests the best strategy to deal with uncertain action effects. In order to understand the merits of reachability heuristics for conformant planning, we carried out an experimental evaluation.

5.2.1. *Set-up of the evaluation*

The evaluation determines the structure of the (traversed portion of the) search spaces, and disregards efficiency issues, such as memory and run times, related to the actual computation. These are addressed later in the paper. In the evaluation, we take into account all the planning domains reported in the literature to test conformant plannners; these are parameterized in one or more dimensions, to increase difficulty and highlight the asymptotic behavior of the algorithms. For each problem, we consider both the forward and backward search directions. We run the A^* algorithms using MAXSDIST (and also MAXWDIST, in the case of nondeterministic domains) as estimates, where we break ties by expanding the deepest nodes first, and then randomly. We also run the BFS algorithm, which expands the whole search space, to have a measure of the effectiveness of the search.

The gathered information is reported in two ways. First, for each parameterized domain, and for each of the algorithms, we report the number of expanded nodes against the problem size. This helps to understand the width of the search space, also depending on the search direction, and the degree of penetration of the heuristic search. Furthermore, we report the behavior of a specific search algorithm on a specific problem instance by means of so-called *FGH plots*: for each of the nodes expanded during the search we report the corresponding depth g, the estimated cost h, and their sum f. In the case of the example of Fig. 13, these are the results for the MAXSDIST and MAXWDIST, for both search directions.

5.2.2. *The model-based planner* MBP

The experimental evaluation platform is implemented within the Model Based Planner MBP, available from http://sra.itc.it/tools/mbp. MBP is a general system for planning in nondeterministic domains. MBP is based on the symbolic model checking techniques described in Section 4, and is built on top of the symbolic model checker NUSMV [15, 16,18]. MBP can be seen as a two-stage system. The first stage processes a domain description provided as input, and encodes it into a BDD-based representation of the

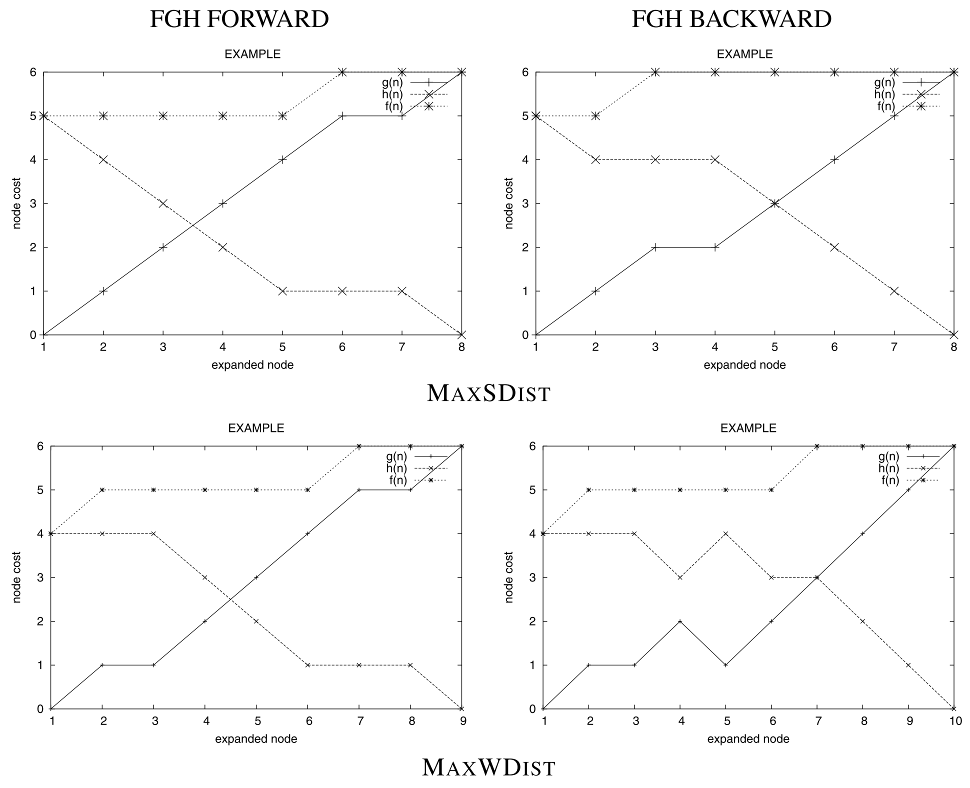


Fig. 13. FGH plots for the running example.

corresponding automaton. Possible languages are the high-level action language $\mathcal{AR}$ [33], a nondeterministic extension of PDDL [2], and the model description language of NUSMV. The problems considered in this paper are written in the language of NUSMV: for each fluent in the domain, the effect of each action is specified. The use of logical assertions allows us to express action preconditions and maintainance of reasonably compact encodings. The use of non-boolean fluents, that are logarithmically encoded into boolean variables during the compilation process, results in a significant compaction of the encodings. Since the NUSMV language only allows for *ground* description of domains (i.e., it is not possible to specify parametric operators, but only ground actions), the different instances are generated by means of scripts.

In the second stage of MBP, different classes of planning algorithms are available, that operate solely on the automaton representation, and are independent of the input language used. The algorithms are able to tackle, beside conformant planning, weak, strong and strong cyclic planning [19,22,23], conditional planning under partial observability [3, 5], and planning for temporally extended goals [27,41,42]. All the algorithms for conformant planning presented in this paper have been implemented in this framework, and instrumented with procedures to mechanically trace their behavior and present the information extracted during the search.

5.2.3. Problems and results

Bomb in the toilet. The first examples are the Bomb in the Toilet (BT) problem [36] and its variations. In the BT(p), there are p packages, one of which contains an initially armed bomb. The goal is to disarm the bomb. There is one toilet. If the bomb is in package i, then dunking the package in the toilet has the effect of disarming the bomb. The solution is to dunk all packages (in any order). In the BT with clogging (BTC(p)) [with uncertain clogging (BTUC(p)), respectively], dunking a package will [may] clog the toilet. A flush action removes the clog. If the toilet is clogged, then a dunk action is not applicable. BMTC(p, t) and BMTUC(p, t) are the multiple-toilets versions of BTC and BTUC, respectively.

The results are shown in Fig. 14. For the BT, the distance is uniformly one. For the BTC and the BTUC it is one for the states where the toilet is not clogged, and two where the toilet is clogged. For the multiple toilet version, the value is one if there is at least one toilet that is not clogged, two otherwise. In all examples, the heuristic turns out to be very poor, and in fact the behavior of the algorithms is not different from the corresponding BFS. Despite its apparent simplicity, the BT problems are far from being trivial, in its original formulation. The plots at the bottom of Fig. 14 refer to the formulation of the BT presented in [40], where each of the packages is associated with an "Armed" predicate, and the goal is expressed as

$$\bigwedge_{1\leqslant i\leqslant n} \neg\mathrm{Armed}_i\,.$$

The behavior of the search clearly shows that this reformulation of the problem makes it trivial to solve.

For the BMT{U}C problems we report only the number of expanded beliefs in Fig. 15, since the behavior of FGH plots is analogous to the one for the BT problems in Fig. 14.

As a final remark, in the BT family of problems the use of reachability based heuristics obtained using full observability to guide the search do not appear to drive the search, thus resulting in almost a breadth-first search.

SQUARE and CUBE navigation. The SQUARE navigation domain [6] is a simplification of the example domain used in this paper, without holes and slippery spots. We consider different problem classes. The initial condition is completely unspecified, i.e., any state is possibly initial. In the SQUARE-CORNER(n) problem, the goal is to reach the $(0,0)$ corner; in the SQUARE-SIDE, the goal is to reach (0, $n/2$); in the SQUARE-CENTER, the goal is to reach ($n/2$, $n/2$). The CUBE domain is the generalization of the square to the three-dimensional case. In CUBE-CORNER(n) the goal is to reach $(0,0,0)$; in the CUBE-EDGE, the goal is (0, 0, $n/2$); in CUBE-SIDE, (0, $n/2$, $n/2$) in SQUARE-CENTER, ($n/2$, $n/2$, $n/2$).

For all cases, for each state the distance is the Manhattan distance to the closest goal state. The results of the search are shown in Figs. 16 and 17 for the SQUARE and CUBE problem instances respectively. The CUBE problems are harder than the corresponding SQUARE ones, but similar in behavior. The difficulty greatly increases from CORNER, to EDGE, to SIDE, to CENTER. Intuitively, this can be justified by the consideration that in

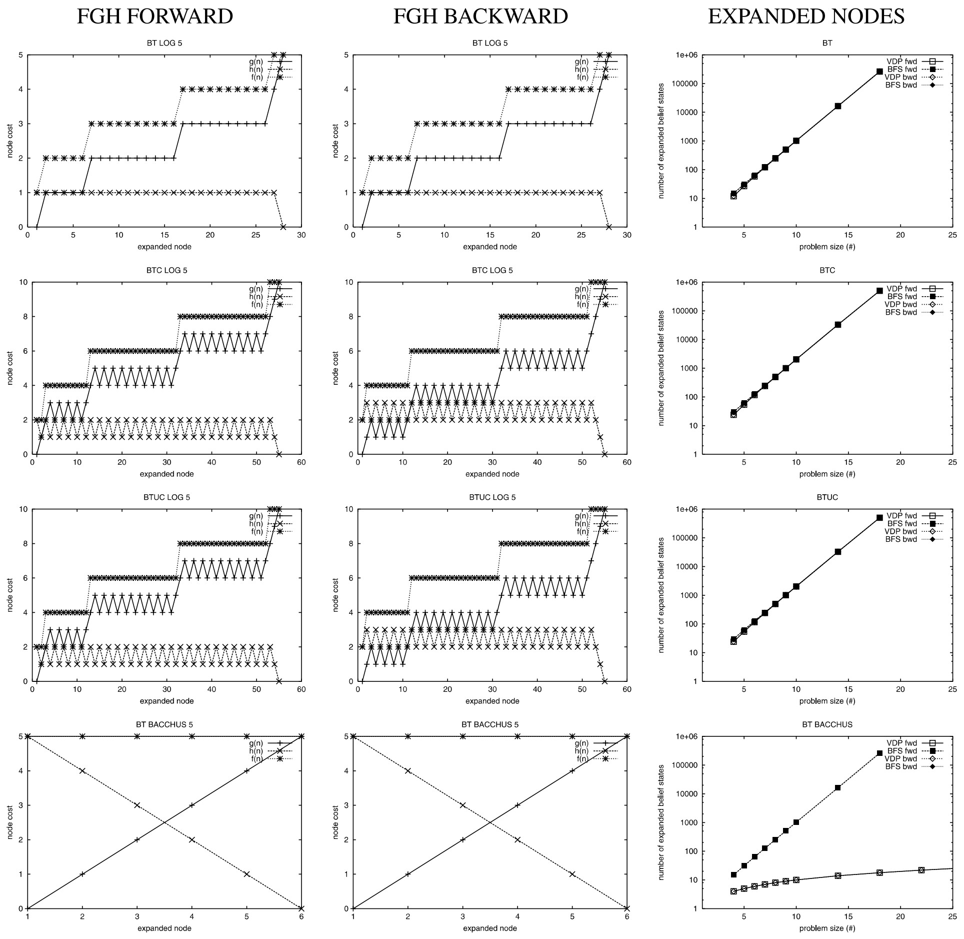


Fig. 14. The results of the search on the (single-toilet) BT domains.

the CORNER case progressing in the direction of the goal also reduces the uncertainty in the position; in the CENTER case, instead, the search is not very different from BFS.

Notice the difference resulting from the search direction: in the backward case, the search is made easier by the fact that the preimage constructs "larger" belief states—in other words, it acquires knowledge whenever possible. In both directions, the impact of heuristic search with respect to BFS tends to reduce for the "harder" instances.

Ring domains. In the RING domain [20], a robot can move in a ring of n rooms. Each room has a window, which can be either open, closed, or locked. The robot can move (either clockwise or counterclockwise), close the window of the room where it is, and lock the windows where it is provided it is closed. The goal is to have all windows locked. In the DETRING(n) problem instance, in the initial condition, both the position of the robot and the status of the windows can be uncertain, while actions are deterministic.

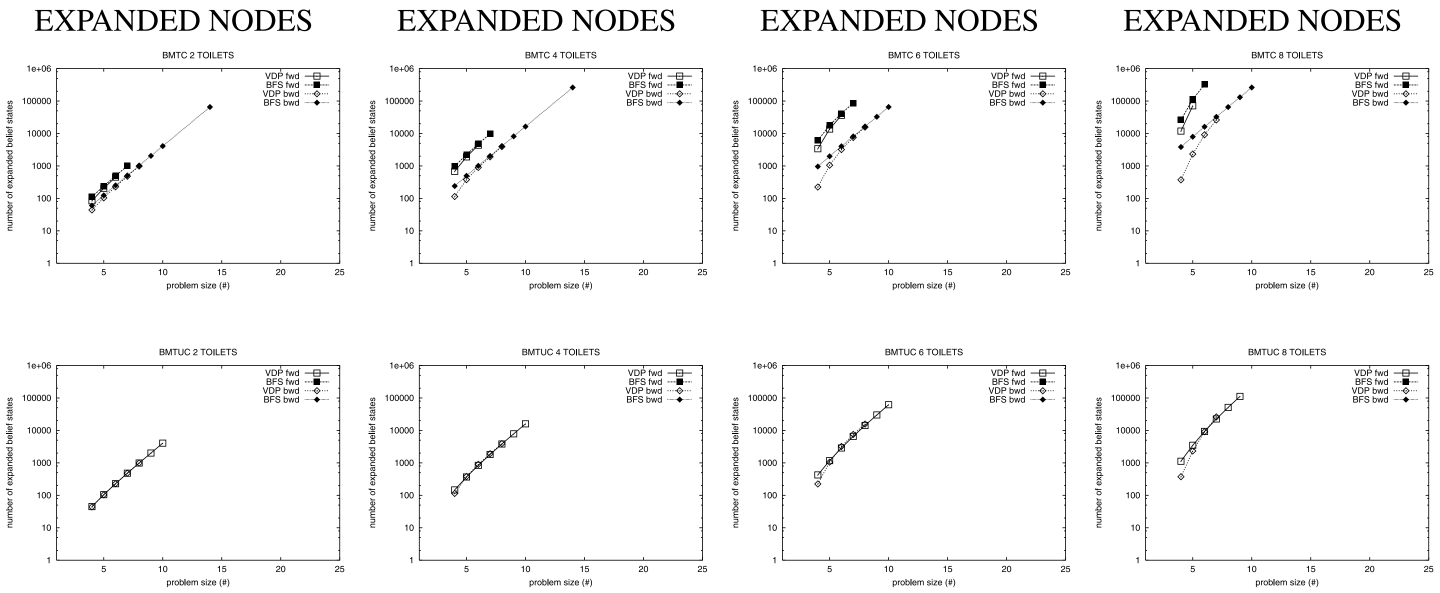


Fig. 15. The results of the search on the BT domains with multiple toilets.

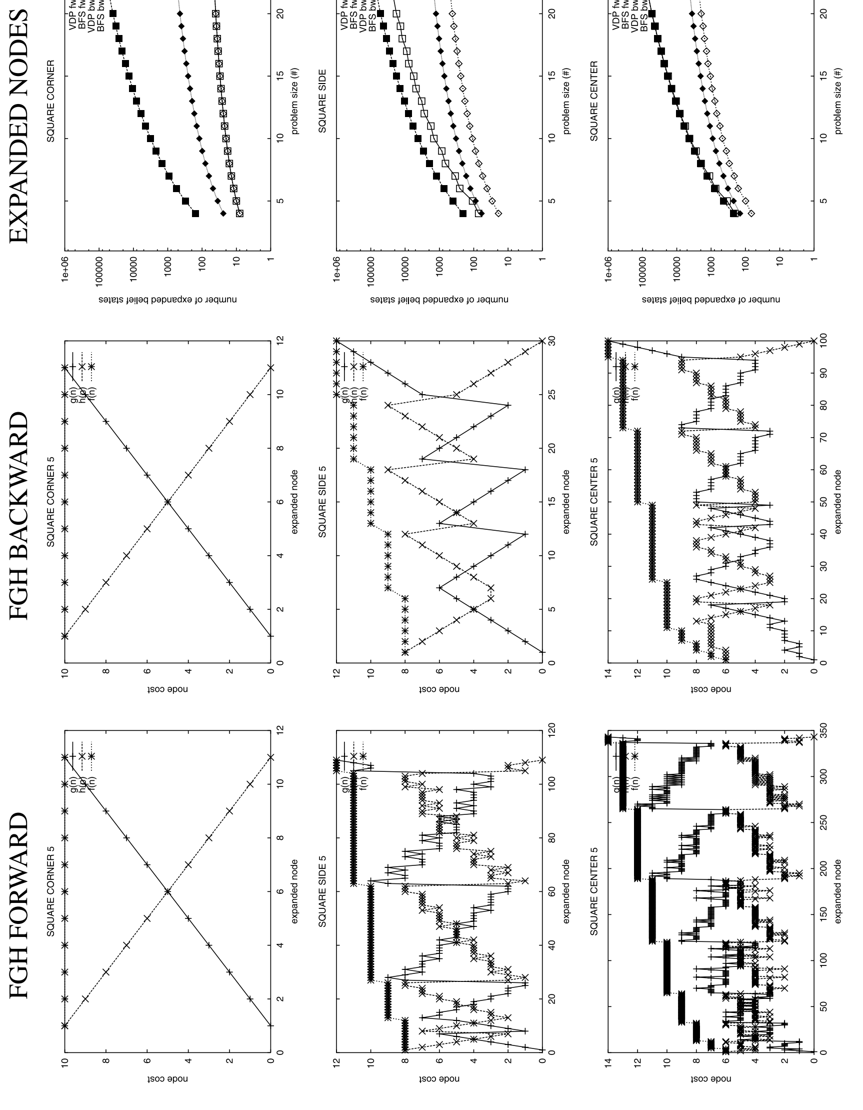


Fig. 16. The results of the search on SQUARE domains.

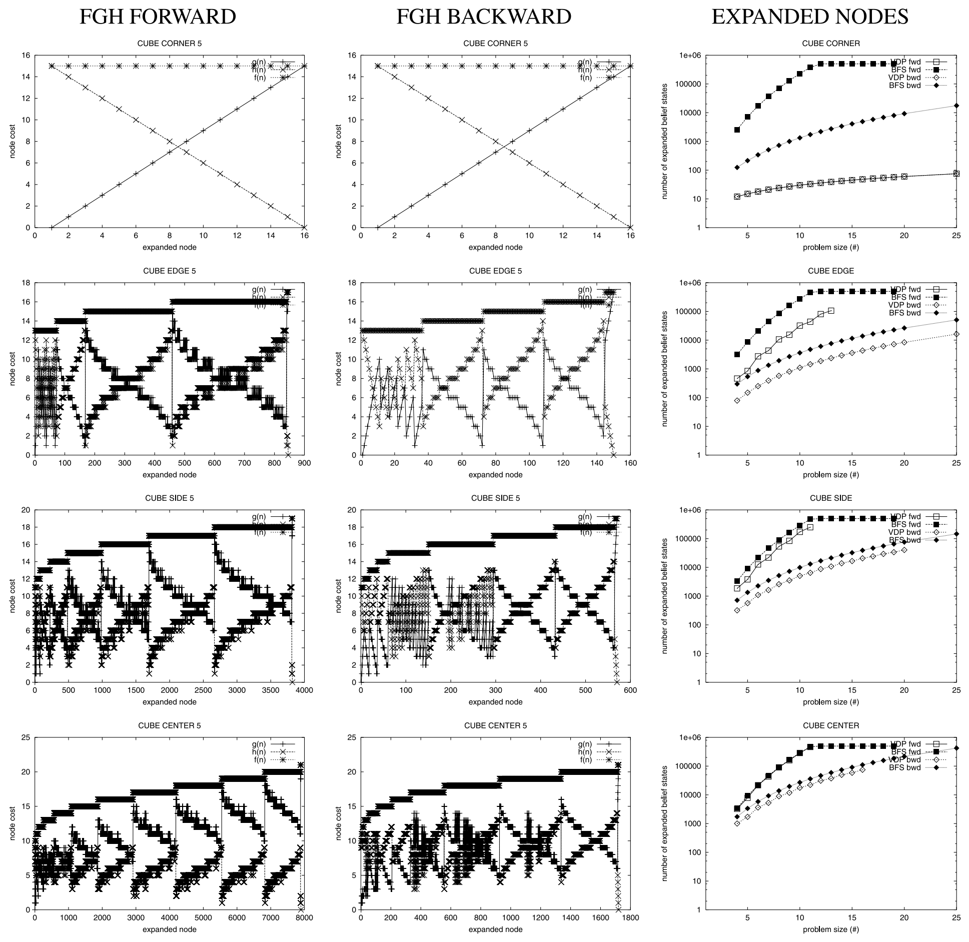


Fig. 17. The results of the search on CUBE domains.

In the NONDETRING(n) problem, if a window is not locked, then it can open or close nondeterministically. For instance, while the robot is moving from room 1 to room 2, the windows in room 3 and 4 could be open or closed if they are not locked. For the DETRING, the distance depends on the number of open windows, and on the distance between the robot position and the farthest open window. For NONDETRING, the distance depends on the window being locked. The results are shown in Fig. 18. In the first row, for the DETRING, the MAXSDIST heuristic turns out to be a perfect oracle, as for the NONDETRING, in the second row. In the third row, we report the impact of MAXWDIST. It is possible to see that there is a significant difference between the estimate provided by the MAXSDIST and by the MAXWDIST.

We also consider the variation where a key is needed to lock and unlock the windows; initially, the position of the key is uncertain. The results are shown in the lower part of Fig. 18. In both search directions, the heuristics are no longer perfect oracles, and a wider

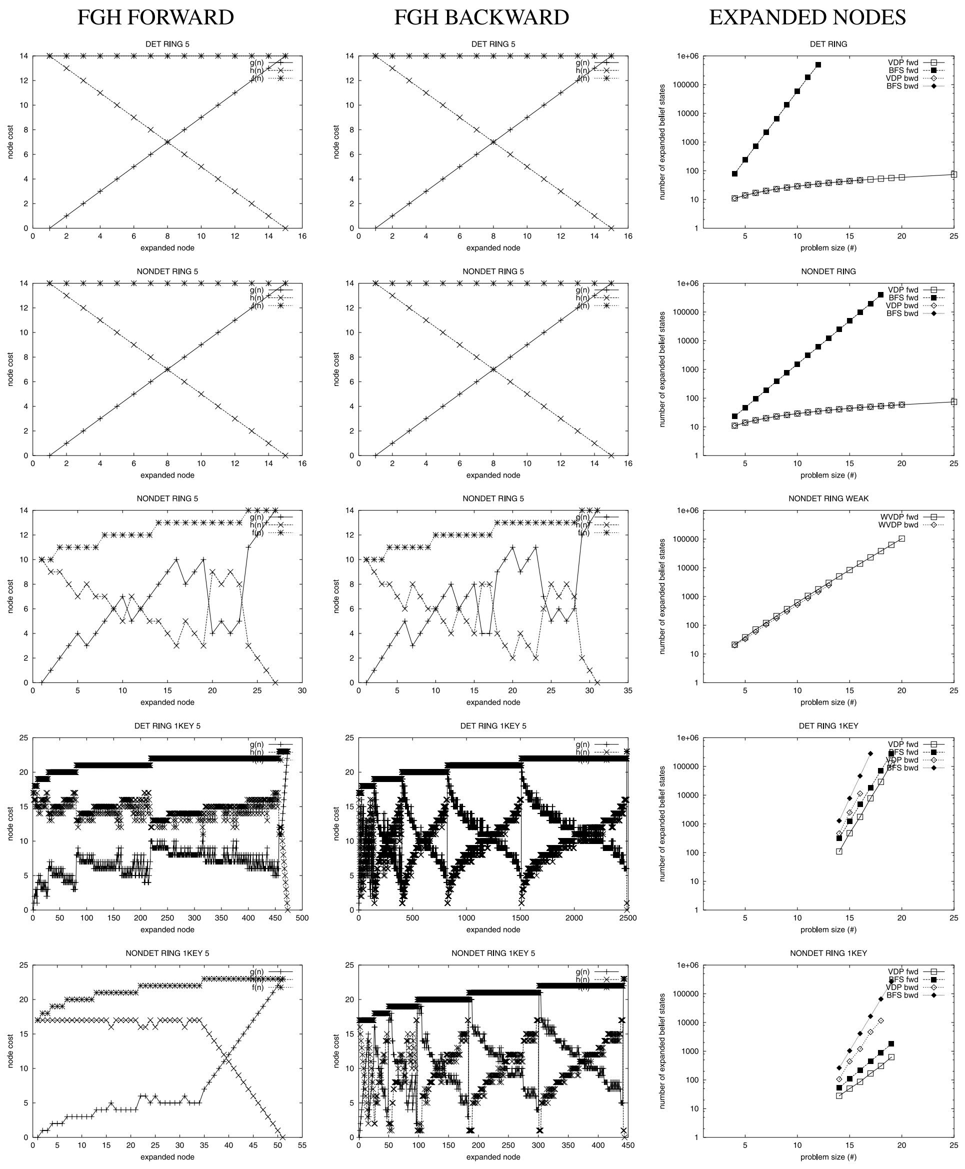


Fig. 18. The results of the search on the RING domains.

segment of the search space needs to be explored. Notice again that in the forward direction MAXSDIST appears to be much more effective than in the backward case. In both cases, when a belief state is reached where the robot is surely holding the key, the problem is reduced to the key-less version, and the heuristic drives the search straight to the solution.

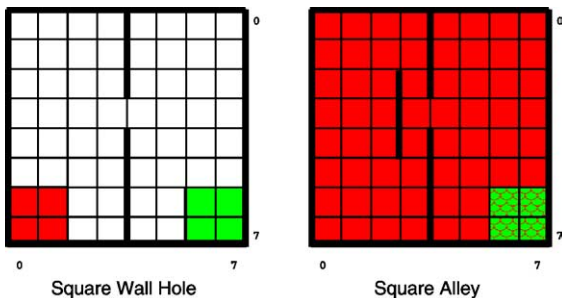


Fig. 19. SQUARE-HOLE and SQUARE-ALLEY domains.

Harder navigation domains. The CUBE-OBSTACLE [1] is a three-dimensional navigation problem, with no side walls, and where each dimension is ring-like (i.e., a GoWest action in $(i, j, n-1)$ leads to $(i, j, 0)$). Location $(1, 1, 1)$ is occupied by a 1-dimensional cube, whose sides behave like walls. The goal is to reach $(n/2, n/2, n/2)$ starting from $\{(n/2, n/2, n/2), (1+n/2, n/2, n/2)\}$. The CUBE-TREASURE problem [1] is similar to the CUBE-OBSTACLE, except that a treasure is located at $(1, 1, 1)$ instead of a wall. A pick-up action can always be attempted, and it succeeds if the position is the same of the treasure. The goal is to get hold of the treasure; the initial position is incompletely specified. We also classify in this family two variants of the empty room navigation domain, SQUARE-HOLE and SQUARE-ALLEY. Their pictorial representation is given in Fig. 19. In the SQUARE-HOLE family, a robot is initially positioned in $x \in \{0, 1\} \wedge y \in \{n-1, n\}$ and the goal is $x \in \{n-1, n\} \wedge y \in \{n-1, n\}$. At $x = n/2$ there is a wall which divide the room in two parts. The two sub-rooms are connected by means of a hole in the mid wall positioned at $y = n/2$. The SQUARE-ALLEY is a variant of the SQUARE-HOLE, where there is an alley to connect the two sub-rooms. Initially the robot can be in any position, and the goal is the same as the one for the SQUARE-HOLE. In all cases, the distance is Manhattan from the goal state. The results shows that the use of reachability heuristics results in an improved search w.r.t. the breadth-first search, but the improvements are not remarkably good in all cases (compare, e.g., the SQUARE-ALLEY with SQUARE-HOLE).

Blocksworld. The Blocksworld conformant domain [8] is a variant with three kinds of operators: put a block x from another block y onto the table; put a block x from a block y onto a different block z; put a block x from the table onto a block y. The uncertainty is that the top blocks on each initial stack are arranged in an unknown order. The results are shown in Fig. 21. They show that the reachability heuristic is quite effective in driving the search in the forward search case, while in the backward case a larger number of belief states is considered w.r.t. the forward case.

5.2.4. *Analysis of results*

From the experimental results, we draw the following conclusions. First, compared to BFS, V^*_{dp} yields significantly more informed search. In several cases, V^*_{dp} turns out to be as informative as a perfect oracle, as highlighted by the “cross-shaped” behavior in the FGH plots. In domains with nondeterministic action effects, the difference between weak and

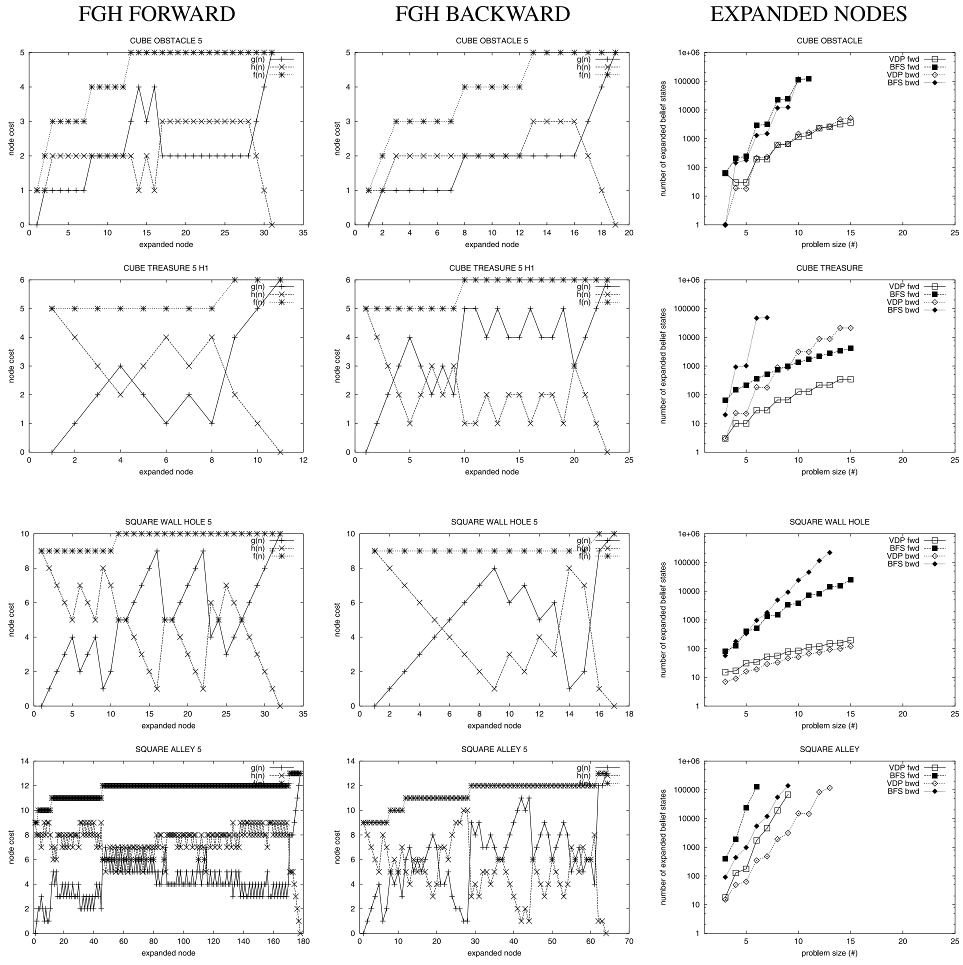
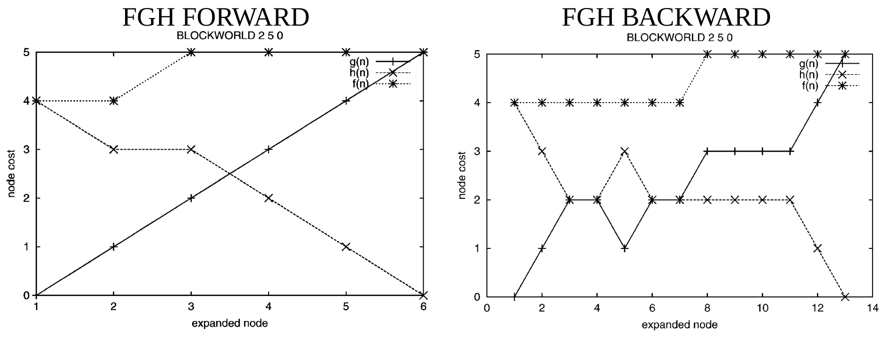
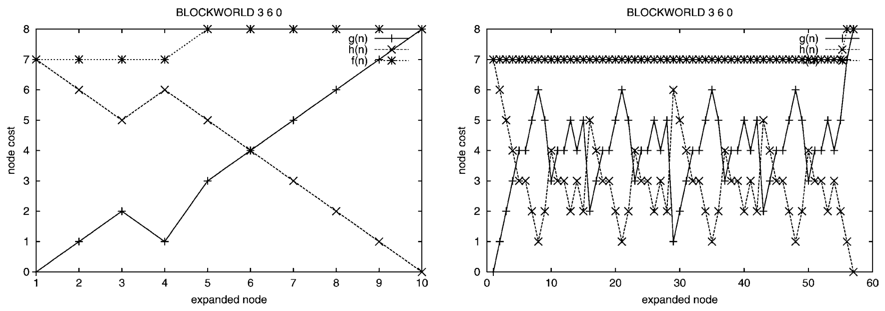


Fig. 20. The results of the search for the hard navigation domains.

strong distance may be relevant (while in the case of deterministic actions strong distance coincides with the weak). By simply looking at the relaxed version it is also possible to detect that a certain problem is unsolvable and avoid the exploration of the belief space. Second, there is a strong dependence on the (forward and backward) search direction. This is clearly highlighted by the number of belief states that are expanded by the BFS algorithms. Third, the experimental analysis presented above is greatly eased by the use of symbolic techniques. Although the discussion of efficiency issues is deferred to later sections, it is interesting to note that for most of the problems the time is dominated by search rather than by the computation of V_{dp}^{*}. The final—and probably most important remark—is that there are several cases in which V_{dp}^{*} turns out not to be very informative, despite the fact that it is very informed under the hypothesis of full observability. Some examples are the SIDE and CORNER instances of the SQUARE and CUBE, where the



5 Blocks with 2 blocks uncertain



6 Blocks with 3 blocks uncertain

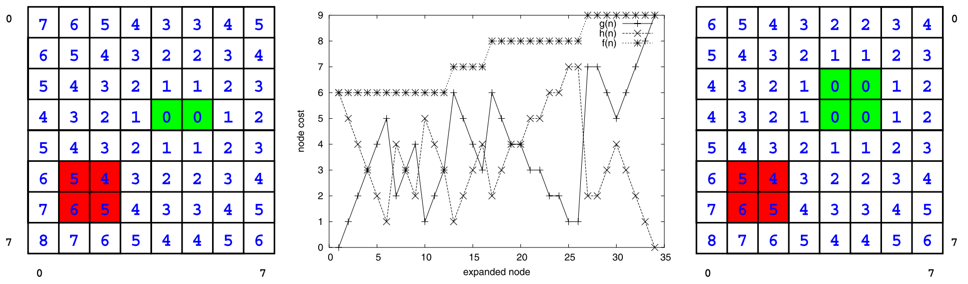
Fig. 21. The results of the search on the Blocksworld domains.

heuristic function grossly underestimates the actual distance. We address this problem in the next section.

6. Improving heuristic search with knowledge

In order to understand the difficulties with V^*_{dp} (which corresponds to MAXSDIST), let us consider the navigation problem described in Fig. 22(a), where the goal is $\mathcal{G} = \{(4,3),(5,3)\}$ and $\mathcal{I} = \{(1,5),(1,6),(2,5),(2,6)\}$. According to V^*_{dp}, the search focuses first on belief states that are "Manhattan-closer" to the goal. However, once the belief state $\{(4,3),(5,3),(4,4),(5,4)\}$ is reached, that has h value 1 and g value 3, V^*_{dp} is in a local minimum, as shown by the FGH plot in Fig. 22(b). The algorithm then repeatedly expands states with decreasing h, getting to a local minimum at value 1, and then backtracks by reconsidering belief states associated with higher values of f. This simple problem clearly shows that moving directly towards the goal, as suggested by V^*_{dp}, is a bad idea.

The reason for these repeated failures is that reaching the goal requires a certain amount of knowledge. In fact, we need to know the value of y without uncertainty, but this information is not available in the initial situation, since two values of y are possible in $\mathcal{I}$. An optimal solution to the problem is to `GoSouth` twice, to limit the uncertainty on the y coordinate, and only after that head towards the goal. Notice that the V^*_{dp} values for the belief states traversed with the `GoSouth` actions are actually increasing with respect



(a) (b) (c)

Fig. 22. An example of the limitations of V^*_{dp}.

to their values in the originating states. Notice also that this behavior depends on the planning problem, and not on the domain. When the goal is $\{(4,2),(5,2),(4,3),(5,3)\}$ (Fig. 22(c)), V^*_{dp} behaves like a perfect oracle. In this case, the acquisition of information is unnecessary.

The weakness of V^*_{dp} that it considers the states in the belief state one by one, as if they were not correlated; in this way, V^*_{dp} does not take into account the fact that, without enough information, it is pointless to "aim at" the goal, also because there is no way to acquire the required knowledge close to the goal.

6.1. *Knowledge in the belief space*

In order to obtain more informed heuristics, we explicitly take into account the idea that a certain degree of knowledge is associated with each belief state.

6.1.1. *The knowledge lattice*

In the following, we no longer look at the belief space $Pow^+(\mathcal{S})$ as a "flat" set; it is a semi-lattice $\mathcal{L} = \langle Pow^+(\mathcal{S}), \subseteq \rangle$, where the bottom element is the set of all states $\mathcal{S}$, and the top (maximal) elements are the singletons. Fig. 23 provides a pictorial representation for the semi-lattice in the case of 2×2 empty room navigation domain. The arrows represent set inclusion. The intuition is that "smaller" states are higher in the lattice, and represent higher knowledge conditions, while in larger belief states, lower in the lattice, the knowledge is lower. In the limit cases of the bottom element, any state of the domain is possible, while a maximal (singleton) belief state corresponds to perfect knowledge on the status of the domain.

The effect of actions in the belief space can be presented as connections on the lattice on $Pow^+(\mathcal{S})$. While proceeding forward, an action α connects Bs_1 to Bs_2 iff α is applicable in Bs_1 and $Bs_2 = \text{EXEC}[\alpha](Bs_1)$. While proceeding backward, α connects two belief states, namely Bs_2 to Bs_1 iff $Bs_1 = \text{EXEC}^{-1}[\alpha](Bs_2)$. A planning problem is solved in the forward case if we have a path from $\mathcal{I}$ to any belief state subsuming (i.e., more informed than) $\mathcal{G}$; in the backward case, we need a path from $\mathcal{G}$ to any belief state subsumed by (i.e., less informed than) the initial belief state. Notice the difference in the expansion primitives in the search spaces: in the forward case, an action returns the most informed condition given

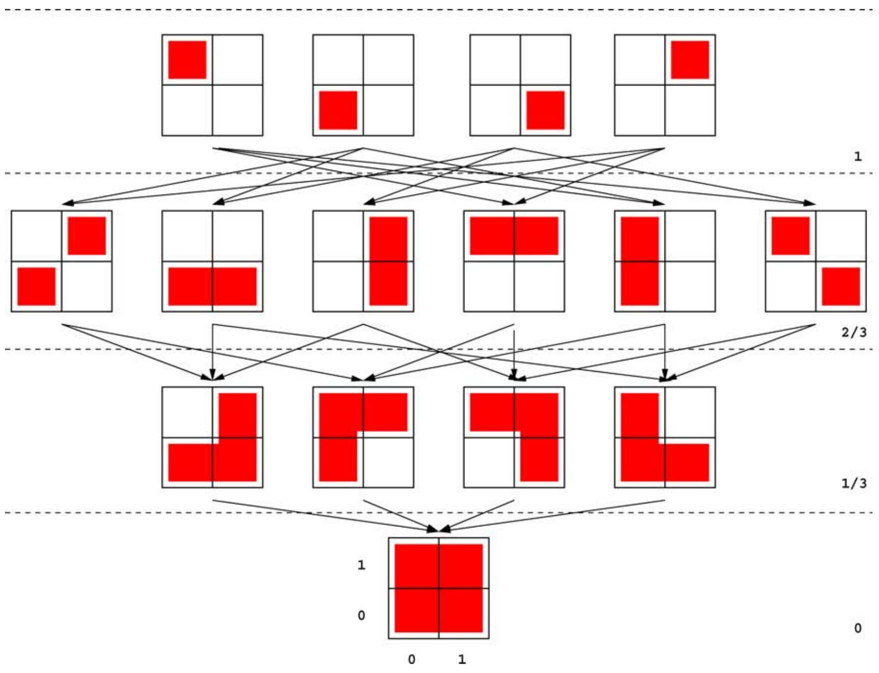


Fig. 23. Semi-lattice for a 2×2 empty-room robot navigation problem.

the starting belief state. In the backward case, the preimage constructs the least informed condition from which we can reach the belief state.

6.1.2. *Knowledge formulae*

The notion of knowledge associated with a belief state is formalized by means of knowledge formulae.

Definition 9 (*Knowledge formulae*). Let *bwff* be a boolean wff. Knowledge formulae (*kwff*) are defined according to the following rules:

$$kwff := \mathcal{K}bwff \mid \neg kwff \mid kwff \wedge kwff \mid kwff \vee kwff.$$

The basic atoms of knowledge formulae are obtained with the application of a knowledge operator to a boolean formula; general knowledge formulae are boolean combinations of basic knowledge atoms. In the following, whenever not specified, we denote with ϕ a boolean wff and with ψ a knowledge formula. We interpret knowledge formulae over $\mathcal{S}$ as follows.

Definition 10 (*Semantics of knowledge formulae*). The denotation of a knowledge formula ψ in $\mathcal{S}$ is defined as follows:

- $[\![\mathcal{K}\phi]\!]_k \doteq \{Bs \in Pow^+(\mathcal{S}){:}\ Bs \subseteq [\![\phi]\!]\}$, if ϕ is a boolean *wff*,

- $[\![\psi_1 \wedge \psi_2]\!]_k \doteq [\![\psi_1]\!]_k \cap [\![\psi_2]\!]_k$,
- $[\![\psi_1 \vee \psi_2]\!]_k \doteq [\![\psi_1]\!]_k \cup [\![\psi_2]\!]_k$,
- $[\![\neg\psi]\!]_k \doteq Pow^+(\mathcal{S}) \setminus [\![\psi]\!]_k$.

We consider that knowledge formulae are basically formulae of modal K, where nesting of knowledge operators is not allowed. Therefore, if ϕ_1 and ϕ_2 are boolean formulae, the following facts hold: $\mathcal{K}(\phi_1 \wedge \phi_2)$ and $\mathcal{K}\phi_1 \wedge \mathcal{K}\phi_2$ are equivalent, i.e., they have the same denotations. $\mathcal{K}(\phi_1 \vee \phi_2)$ is implied both by $\mathcal{K}\phi_1$ and by $\mathcal{K}\phi_2$, but implies neither of them.

The denotation of a knowledge formula is the set of the belief states where this formula is known to hold. Let a belief state *Bs* be given. We say that a knowledge formula ψ holds in *Bs* iff $Bs \in [\![\psi]\!]_k$. Clearly, if $Bs \in [\![\psi]\!]_k$, then $Bs' \in [\![\psi]\!]_k$ for every $Bs' \in Pow^+(\mathcal{S})$ such that $Bs' \subseteq Bs$; this formally states that if a certain fact is known given a certain amount of uncertainty, then it is known whenever the uncertainty is reduced.

If $\mathcal{K}\phi$ holds in *Bs*, we say that ϕ is known to be true in *Bs*. If $\mathcal{K}\neg\phi$ holds in *Bs*, we say that ϕ is known to be false in *Bs*. ϕ is known in *Bs* iff either ϕ or $\neg\phi$ are known to be true in *Bs*. In the case of non-boolean functions, we use the following terminology. Let x be a variable, $v \in \mathcal{V}(x)$, and $V \subseteq \mathcal{V}(x)$. If $\mathcal{K}(x = v)$ holds, we say that x is known to have value v. If $\mathcal{K}(x \in V)$ holds, we say that x is known to range over V. The known range of variable x in *Bs* is the minimal set $V \subseteq \mathcal{V}(x)$ that x is known to range over. Let $\mathcal{K}x \doteq \bigvee_{v \in \mathcal{V}(x)} \mathcal{K}(x = v)$. $\mathcal{K}x$ expresses the fact that the value of x is known: in fact, no belief state in $[\![\mathcal{K}x]\!]_k$ contains two states where x is interpreted differently. The formula $\bigvee_{v \in V} \mathcal{K}(x = v)$ expresses that the value of x is known *and* in V, which is much stronger than $\mathcal{K}(x \in V)$. For instance, in the 2×2 empty room, the belief state $\{(0,0),(1,0)\}$ belongs to $[\![\mathcal{K}\bigvee_{i \in V}(x = i)]\!]_k$ but it does not belong to $[\![\bigvee_{i \in V} \mathcal{K}(x = i)]\!]_k$.

A knowledge formula ψ provides a nice way to separate the belief space into two parts, one composed of the belief states where ψ is known to hold, and one where ψ is not known to hold. For instance, the knowledge formula $\mathcal{K}(x = 0 \vee y = 0)$ cuts the semi-lattice for the 2×2 empty room domain as shown in Fig. 24.

We use this partitioning to identify the notion of *necessary knowledge* for a conformant planning problem. The intuition is that if a knowledge formulae ϕ is necessary, basically there is no way to solve the problem without coming to know ϕ at some point, in the sequence of belief states traversed during the execution of any solution plan. In the case of the problem depicted in Fig. 22 it is clear that it is impossible to reach the goal without coming to know the value of y: this means that there is no sequence of belief states associated with a solution plan that does not "enter" the denotation of $\mathcal{K}y$.

Definition 11 (*Necessary knowledge formula*). Let Bs_1 and Bs_2 be belief states, such that $Bs_1 \not\subseteq Bs_2$, and let the conformant planning problem $\langle Bs_1, Bs_2\rangle$ be solvable. The knowledge formula ψ is necessary for $\langle Bs_1, Bs_2\rangle$ iff for each plan π solving $\langle Bs_1, Bs_2\rangle$ there is a prefix π' such that $\textsc{Exec}[\pi'](Bs_1) \in [\![\psi]\!]_k$.

6.2. Using knowledge in heuristic functions

The notion of necessary knowledge formula is used to improve heuristic functions for searching the belief space. Let us consider a generic knowledge formula ψ, that is intended

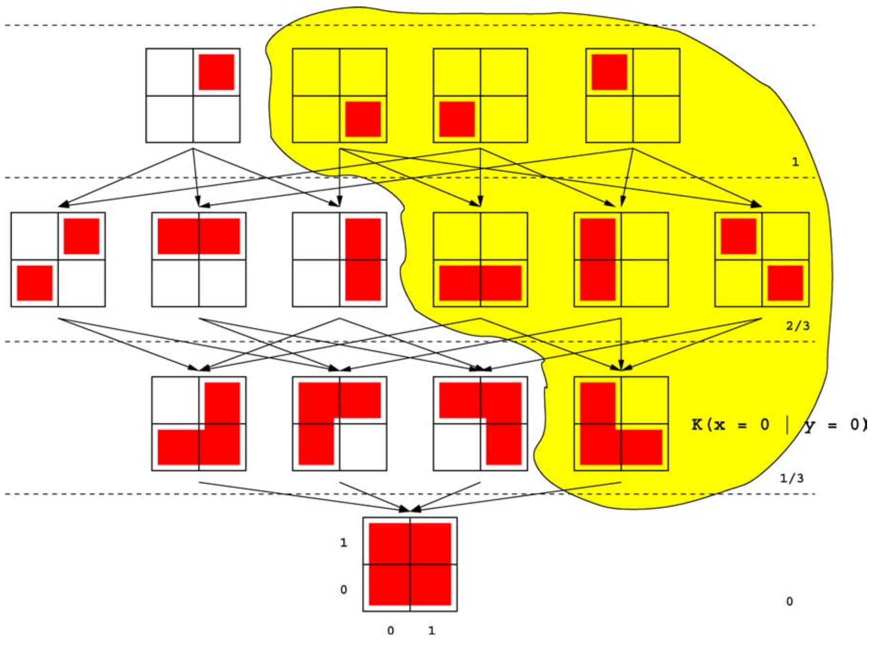


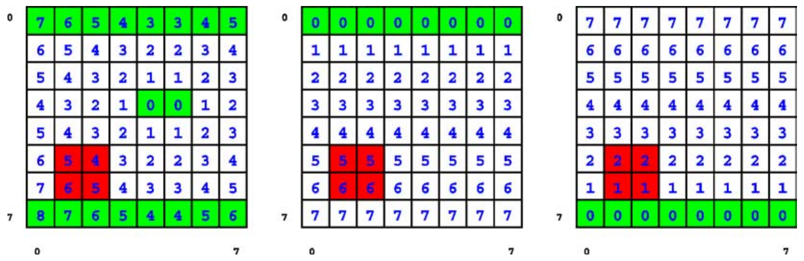
Fig. 24. Semi-lattice for a 2 × 2 empty-room robot navigation problem showing $\mathcal{K}(x = 0 \vee y = 0)$.

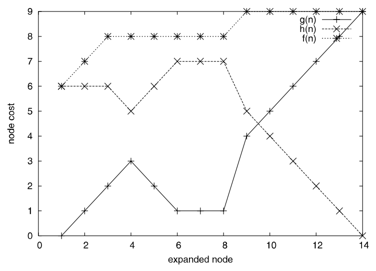
to be necessary for $\langle \mathcal{I}, \mathcal{G} \rangle$. Then, we characterize a knowledge-based heuristic function V_ψ as follows:

$$V_\psi(Bs, \mathcal{G}) \doteq \begin{cases} V^*_{dp}(Bs, \mathcal{G}) \quad \text{if } \psi \text{ holds in } Bs \text{ or if } \psi \text{ is not necessary to reach } \mathcal{G}, \\ \max\big(V^*_{dp}(Bs, \mathcal{G}),\ \min_{Bs' \in [\![\psi]\!]_k} \big(V^*_{dp}(Bs, Bs') + \textsc{MinSDist}(Bs', \mathcal{G})\big)\big) \\ \quad \text{otherwise.} \end{cases}$$

The first clause applies when the belief state being evaluated either satisfies ψ, or ψ has already been satisfied and it is no longer necessary to reach $\mathcal{G}$. In both these cases, there is no need to acquire the knowledge of ψ, and V^*_{dp} is used to proceed towards $\mathcal{G}$. Otherwise, the second clause tries to take into account the cost coming to know ψ, by entering a belief state Bs' in $[\![\psi]\!]_k$, and then proceedings towards $\mathcal{G}$ from Bs'. We have no information on which subset of Bs' will be reached: therefore, in order for the heuristics to be admissible, we evaluate distance between Bs' and $\mathcal{G}$ by means of MinSDist and we choose the min value of the sum of the cost to coming to know ψ and the cost of reaching the goal with ψ known. Finally, it is possible that the computation of the min value given by the second clause is lower than $V^*_{dp}(Bs, \mathcal{G})$; therefore it can be compared with the value provided by V^*_{dp} so as to return their maximum.

Let us consider now the effect of V_ψ on the problem of Fig. 22(a), the necessary knowledge formula being $\psi \doteq \mathcal{K}(y = 0) \vee \mathcal{K}(y = 7)$. In Fig. 25(a), we show the basic elements involved in the computation of the values of V_ψ for this problem. On the left, we have the layers of the distance from the goal; the same layers are used to compute MinSDist from the (maximal) elements of $Bs' \in [\![\psi]\!]_k$, that is $y = 0$ and $y = 7$. Then we have the distance layers from $y = 0$ and $y = 7$. In Fig. 25(b), we show the FGH plot for the


(a)



(b)

Fig. 25. Empty room problem with V_ψ, where $\psi \doteq \mathcal{K}(y=0) \vee \mathcal{K}(y=7)$.

V_ψ heuristic. We can see that the use of V_ψ significantly increases the estimate of goals (compare to the estimates provided by V^*_{dp} in Fig. 22(b)). The use of V_ψ results in a more directed search, where 14 nodes are expanded instead of 34. The expansion towards the south wall to reach a belief state with $y = 7$ is promoted. Once this belief state has been reached, the search is directed towards the goal guided by V^*_{dp}, which does not take into account knowledge.

It is also evident that the new heuristic is far from providing a perfect guidance. As explained above, the MINSDIST cost to reach the goal from $y = 7$ is an underestimate: the distance on the x axis not varied, but this fact is not taken into account.

The formal properties of the V_ψ heuristic function are stated by the following theorem.

Theorem 4. *If ψ is a necessary knowledge formula for the planning problem $\langle \mathcal{I}, \mathcal{G}\rangle$, then V_ψ is admissible, and at least as informed as V^*_{dp}.*

The theorem follows from the admissibility of the min term in the second clause. V_ψ is more informed than V^*_{dp} by construction: for any belief states Bs, $V^*_{dp}(Bs) \leqslant V_\psi(Bs)$. This guarantees that an algorithm using it to guide the search will possibly explore fewer nodes to find an optimal solution [39]. It is also evident that the computation of the new heuristic function is more complex, and there is in general a trade-off to be taken into account.

6.3. Experimental evaluation of knowledge-based heuristics

In order to validate the new heuristic function, we ran some experiments from Section 5 and some new ones, where the necessary knowledge functions are provided manually. We concentrate on the forward search mechanism, and on those problems where the guidance provided by V^*_{dp} is limited.

Empty room. For the empty room domain described in Fig. 22(a), Fig. 26 depicts the behavior of the search driven by V_ψ, when the heuristic function is $\psi \doteq \mathcal{K}(y = 0) \vee \mathcal{K}(y = 7)$ The search proceeds first towards the down wall and then towards the goal, thus resulting in a more directed search.

Empty room with an obstacle. Fig. 28 depicts the behavior for the Empty Room with Obstacle domain (see Fig. 27), where a robot moving in a square room with no side walls, and where opposite sides are connected (i.e., a GoWest action in $(i, n - 1)$ leads to $(i, 0)$). Location $(c_x, c_y) = (1, 1)$ is occupied by an obstacle, whose sides behave like walls. The goal is to reach $x = y = n/2$ starting from $x, y \in \{(n/2) - 1, (n/2)\}$. The knowledge formula provided is

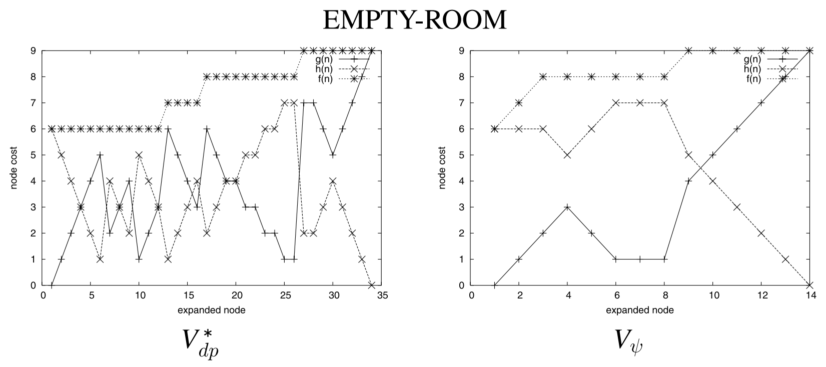


Fig. 26. V^*_{dp} vs. V_ψ.

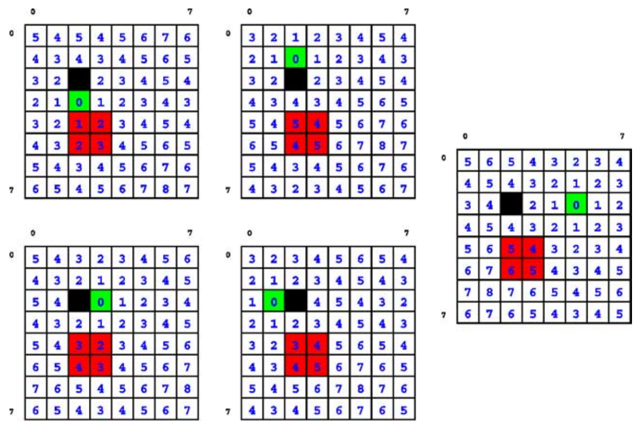


Fig. 27. Empty room with an obstacle.

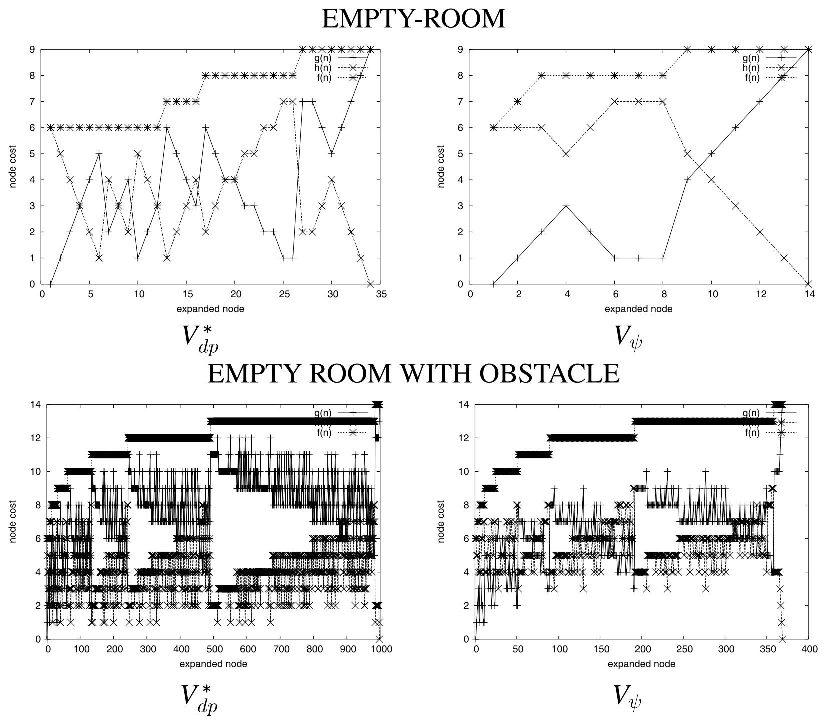


Fig. 28. V_{dp}^* vs. V_ψ.

$$\psi \doteq \mathcal{K}\big((x = c_x - 1) \wedge (y = c_y)\big) \vee \mathcal{K}\big((x = c_x + 1) \wedge (y = c_y)\big)$$
$$\vee\, \mathcal{K}\big((x = c_x) \wedge (y = c_y + 1)\big) \vee \mathcal{K}\big((x = c_x) \wedge (y = c_y - 1)\big).$$

The search first proceeds towards the obstacle to acquire knowledge, and then towards the goal. However, in this example the function is not as effective as in the previous cases, due to the fact that V_{dp}^* is used to drive the search for knowledge, but is not very accurate when distances are close to zero. This causes considerable searching in the proximity of the obstacle, which is carried out until complete knowledge on the value of the domain variables is available. Only once this condition has been achieved, the search proceeds directly towards the goal without reconsidering shallower states.

SQUARE and CUBE domains. In the case of the square domains, necessary formulae are $(\mathcal{K}(x = 0) \vee \mathcal{K}(x = N)$ and $(\mathcal{K}(y = 0) \vee \mathcal{K}(y = N)))$. We provide as ψ the conjunction of these formulae, that characterizes the four corners of the room.[3] The results are depicted in Fig. 29. The number of expanded nodes is dramatically reduced, since the search heads towards the "closest" corner to the goal and then towards the goal. In the case of the cube

[3] Strictly speaking, this is not a necessary formula, since the problem can be solved without entering a corner. However, for this domain this does not change the situation, since the two dimensions are completely independent of each other.

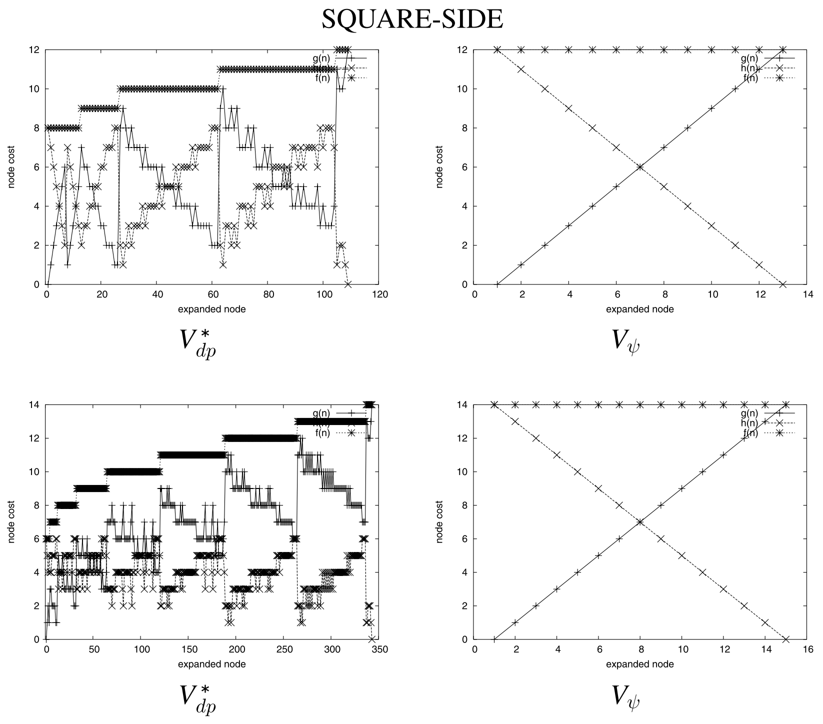


Fig. 29. V^*_{dp} vs. V_ψ.

navigation domains, we generalized the formula used in the square domains to characterize the eight corners of the cube. The results are depicted in Fig. 30. Even in this case the search is dramatically reduced since it heads towards the closest corner to the goal, and from that corner to the goal.

Nondeterministic ring with a key. In the case of the ring with keys, the necessary function is $\mathcal{K}$`HasKey`. However, in this case the use V_ψ function does not improve the search. This is due to the fact that V^*_{dp} is able to discover the need to acquire the key to achieve the goal.

Navigation hard. We tried to identify necessary knowledge formulae for the SQUARE-HOLE and for the CUBE-OBSTACLE domains considered in the previous section. For the SQUARE-HOLE a necessary knowledge formula is $\psi \doteq \mathcal{K}(x = 0) \vee \mathcal{K}(x = N/2)$, while for the CUBE-OBSTACLE we used the generalization of the cube case of the one used for the empty room with an obstacle described above. The results are depicted in Fig. 32. In the case of the SQUARE-HOLE there is an improvement, but it is not as dramatic as the one we have in the case of the CUBE-OBSTACLE.

The results presented from Fig. 26 to Fig. 32 clearly show the advantage of V_ψ over V^*_{dp} in terms of expanded nodes, once a necessary knowledge function is identified. This is of course to be expected, given that V_ψ is strictly more informed than V^*_{dp}. Notice however

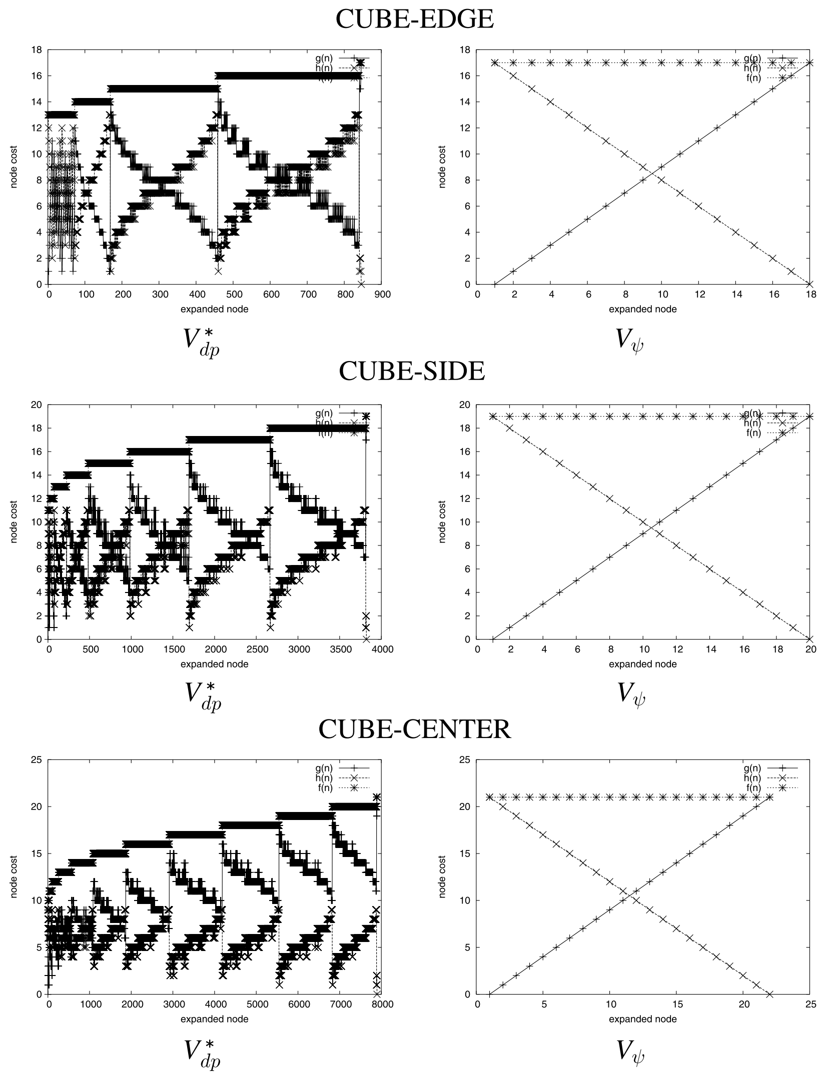


Fig. 30. V^*_{dp} vs. V_ψ.

that V_ψ is not always able to eliminate the search. This can be explained by the fact that V_ψ provides in general a strict under-approximation, since it does not take into account that states in a belief are not distinguishable, and actions may be needed to distinguish them and act on them.

More importantly, even if the experiments clearly show the potential for the idea of knowledge-based heuristic functions in terms of search space, in many cases finding V_ψ may not be practical. In fact, the discovery of non-trivial necessary knowledge formulae for a given problem may not be straightforward. Furthermore, V_ψ requires us to detect at run

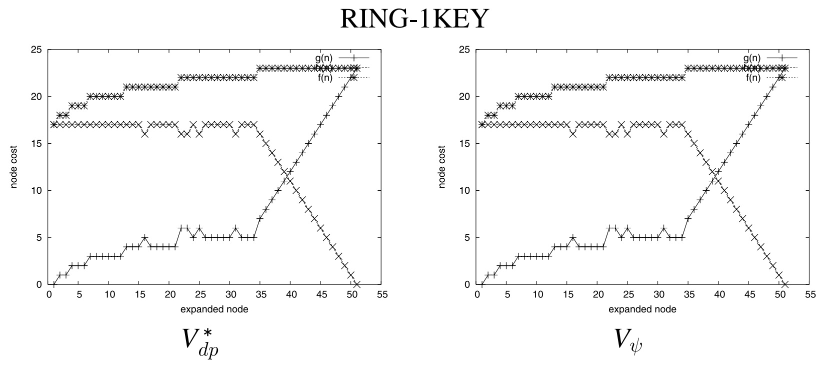


Fig. 31. V^*_{dp} vs. V_ψ.

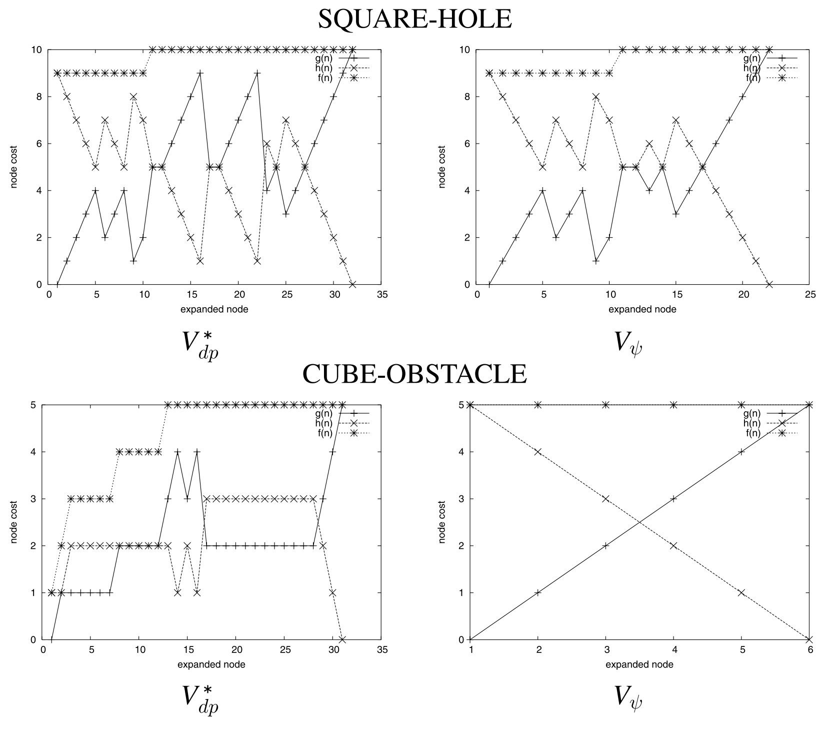


Fig. 32. V^*_{dp} vs. V_ψ.

time whether such knowledge is still necessary for the belief state being evaluated. (In the case of the specific examples defined above, the tests benefit from the fact that it is possible to statically recognize when the given knowledge formula is necessary for a belief state.)

7. Conformant planning with knowledge acquisition

In this section we present an algorithm for conformant planning that tackles the problems highlighted in the previous section. The algorithm gives up the guarantee of optimality, but is able to solve significant problems where reachability heuristics are not effective. The algorithm is based on the following intuitions.

First, the search is based on the notion of *target knowledge*, i.e., the level of knowledge that must be available before trying to reach the goal. If the target knowledge is available, search is directed towards the goal. Otherwise, the algorithm tries to perform actions to acquire the target knowledge, and proceeds towards the goal only thereafter. The algorithm selects an initial target knowledge from a problem analysis, and may subsequently increase it during the search.

Second, the acquisition of knowledge is based on the identification of *Knowledge Subgoals* (KSs). Intuitively, a KS for a certain boolean formula ϕ is a belief state Bs where (a) ϕ holds, and (b) Bs can be reached by some action α from a belief Bs' where ϕ does not hold. The detection of KSs allows us to reduce knowledge acquisition to a form of directed search, similar to the one used to direct towards the goal.

Third, the algorithm is based on the alternation of two main modalities, which implement goal-directed search (*ground level* search), and knowledge acquisition via KS-directed search (*knowledge level* search). The ground search modality greedily directs towards the goal, exploiting a notion of goal distance, and is active when open search nodes exists for which enough knowledge is available to reach the goal. When no such open node exist, instead, the knowledge level search is adopted, which directs towards KSs associated with knowledge formulae being searched for. These two modes alternate on the basis of a dynamic re-evaluation of the current target knowledge (which is estimated as necessary to reach the goal), and of the level of knowledge which is currently available. It is possible for the algorithm to decide that a certain target knowledge cannot be reached, based on the failure of previous knowledge-level attempts. In this case, the ground mode acts as a backup mode, so to preserve completeness of the search.

Finally, *quantitative information* about knowledge formulae is used to enhance directed search, both at the ground and knowledge levels. In particular, the fraction of states in a belief state Bs which satisfy a given boolean formula ϕ can be used as an estimate of how close Bs is to knowing ϕ. This is crucial to deal with problems where knowledge can only be gathered ‘incrementally’ according to such a quantitative measure, in the course of several plan execution steps.

7.1. The conformant planning algorithm

The algorithm is presented in Fig. 33. The top level routine is KACMBP (Knowledge Acquisition Conformant MBP). KACMBP assumes that the domain and problem are globally available to the underlying routines. An open and a closed list of nodes are also available, similarly to those used in Section 3. The algorithm is structured in three main phases. An aggressive and fast goal directed search is initially performed (line 2): this is a greedy hill-climbing over a quantitative measure of goal achievement, and no backtracking is allowed. If this fails, a preprocessing of the problem is carried out (lines 6–9) to detect

```
 1  function KACMBP();
 2     res := GreedyPreprocessing();
 3     if (Success(res)) then
 4        return GetPlan(res);
 5     endif
 6     ComputeGoalDistanceLayers();
 7     ComputeKSs();
 8     ComputeKSsDistanceLayers();
 9     SelectTargetKnowledge();
10     if (EnoughKnowledgeAvailable()) then
11        return ReachGoalMode();
12     else
13        return AcquireKnowledgeMode();
14     endif;
15  end;

 1  function ReachGoalMode()
 2     res := ReachGoalSearch();
 3     if (Success(res)) then
 4        return GetPlan(res);
 5     else if (EmptyOpenList) then
 6        return Fail;
 7     else if (TargetKnowledgeLimitNotReached()) then
 8        IncreaseTargetKnowledge();
 9        return AcquireKnowledgeMode();
10     else
11        return ReachGoalMode();
12     endif
13  end;

 1  function AcquireKnowledgeMode()
 2     res := AcquireKnowledgeSearch();
 3     if (KnowledgeNotAcquired(res)) then
 4        SetTargetKnowledgeLimitReached();
 5     endif
 6     return ReachGoalMode();
 7  end;
```

Fig. 33. The algorithms for planning with knowledge acquisition.

information useful to the main search, i.e., goal distances, KSs, distances from KSs, and the initial target knowledge. The main search is then started, which alternates between the ground level search (implemented by ReachGoalMode) and knowledge level search (implemented by AcquireKnowledgeMode).

EnoughKnowledgeAvailable compares the currently available knowledge with the current target knowledge: if the available knowledge is considered sufficient, then the algorithm performs an attempt to reach the goal with ReachGoalMode. In the other case, it tries to acquire knowledge first, by the AcquireKnowledgeMode mode (lines 10–14). The mutually recursive routines ReachGoalMode and Acquire-KnowledgeMode implement the main search loop.

ReachGoalMode performs a goal-directed, greedy search, limited to open beliefs satisfying the current target knowledge. In particular, the ReachGoalSearch procedure combines two ideas: it selects for expansion open belief states "closer" to the goal; and, when the goal distance does not decrease, it increases a quantitative measure of goal achievement. ReachGoalSearch is exited when a solution is found, when the search space is exhausted, or when the search ends up in a situation where, locally, neither the distance to the goal can decrease, nor the quantitative measure of goal achievement can increase. In the first two cases (lines 3–6), either a solution plan or failure are returned, respectively. In the last case (line 7), the ground level search has failed at the current level of knowledge, possibly because the current knowledge is in fact not sufficient. In turn, this may be a consequence of the fact that the current target knowledge is an optimistic estimate of the necessary knowledge. Thus, unless the current target knowledge has reached a maximal bound, it is increased by IncreaseTargetKnowledge, and control is switched over to the AcquireKnowledgeMode in order to acquire the new target knowledge. If, instead, the target knowledge has reached the maximal knowledge bound, the search is continued by recurring on ReachGoalMode. Therefore, once ground search has failed for every selected target knowledge, it is no longer exited, until a solution is found or and all open nodes are recursively expanded.

In AcquireKnowledgeMode, AcquireKnowledgeSearch tries to acquire the current target knowledge, i.e., to construct a belief state *Bs* which satisfies the current target knowledge. The search in AcquireKnowledgeSearch is based on the notion of *active* KS, i.e., knowledge formulae for which the current knowledge level is deemed not sufficient (we call these *active* knowledge formulae). The search proceeds by greedily reducing the distance from active KSs. Similarly to what happens in the ground search, when distance cannot be decreased, the algorithm tries to increase a quantitative measure of achievement for the knowledge formulae. Notice that several KSs may be active at the same time; in this case, the algorithm tries to diminish the distance at the same time to as many KSs as possible. Knowledge-level search may either succeed or fail; in both cases, it returns the control to ReachGoalMode. In the case of failure, it would be pointless, in the future, to repeat trying to achieve the currently required knowledge levels, or higher ones, for the currently active knowledge formulae. For this reason, SetTargetKnowledgeLimitReached is called to set the maximum reachable knowledge bound for the current active knowledge formulae, so to inhibit future useless invocations of AcquireKnowledgeMode by ReachGoalMode (see ReachGoalMode, lines 7–11).

The algorithm enjoys the soundness, completeness and termination properties proved for the algorithms in Section 3. This is a trivial consequence of the following facts: first, the storage mechanisms and the expansion primitives are the same; second, the search modes only affect the order in which the search space is expanded, without ever pruning the search.

7.2. Description of the search procedures

We now provide a more detailed description of the concepts intuitively presented above, and of the data structures and primitives used in the algorithm. In the description we ignore the details related to the actual implementation, such as a lazy computation of expensive data structures—(for instance, the distance layers associated to a given KS are computed only when that KS is first determined to be active within AcquireKnowledgeSearch).

Truth percentage. The notion of *truth percentage* provides a quantitative estimate of how far is a *Bs* from knowing a certain boolean function. This quantitative information is used both by the initial greedy search at line 2 of KACMBP, and (as a tie-breaking criteria) in the ground and knowledge level search, in order to try and achieve a given condition, possibly over several expansion steps.

Definition 12 (*Truth percentage*). The truth percentage of a boolean formula ϕ in a belief state *Bs*, written $\textsc{TruthPerc}(\phi, Bs)$, is defined as

$$\textsc{TruthPerc}(\phi, Bs) \doteq \frac{|Bs \cap [\![\phi]\!]|}{|Bs|}.$$

The BDD-based implementation of this notion relies on a standard BDD operation that returns the number of models (i.e., states) associated with it. Although this operation may appear to be complex, the computation is efficiently carried out by means of a traversal on the structure of the BDD, and is quite feasible in practice.

GreedyPreprocessing. GreedyPreprocessing realizes a greedy search exploiting the truth percentage concept. In particular, it tries to construct a sequence of belief states $Bs_0, Bs_1, Bs_2, \ldots, Bs_n$, such that $Bs_0 = \mathcal{I}$, and $\textsc{TruthPerc}(\mathcal{G}, Bs_i) \leqslant \textsc{TruthPerc}(\mathcal{G}, Bs_{i+1})$, and $Bs_n \subseteq \mathcal{G}$, and admits no backtracking. In case of success, the solution plan associated with the result node is constructed.

ComputeGoalDistanceLayers. Since the accurate computation of V^*_{dp} may be expensive in practice, the distance of a given belief to the goal is presented as a vector of distances, each element being associated with one of the domain variables.

The ComputeGoalDistanceLayers primitive builds, for each variable x, the corresponding *goal distance layers*, a sequence of increasingly large sets of states, denoted $L_x[0], \ldots, L_x[n]$. Each layer $L_x[i]$ is associated with a distance i, and is computed, starting from $L_x[0] = \textsc{Proj}(Goal, x)$, by the following V^*_{dp}-like iteration:

$$L_x[i+1] = \textsc{Proj}\left(\bigcup_{\alpha} \textsc{Exec}^{-1}[\alpha]\big(L_x[i]\big), x\right) \cup L_x[i].$$

The distance layers are stored, once for all, and support the search routines in computing goal distances. In particular, given a *Bs*, for each x, the corresponding entry of the distance vector from the goal is computed by a sequence of set inclusion operations; the result is the least i such that $Bs \subset L_x[i]$.

Knowledge formulae. Several choices are possible for the knowledge formulae considered for a domain. KACMBP restricts its analysis to atomic knowledge formulae of the form $\bigvee_{v \in \mathcal{V}(x)} \mathcal{K}(x = v)$, denoted ϕ_x, where x is a domain variable. (In the case of a boolean fluent p, ϕ_p corresponds to $\mathcal{K}p \vee \mathcal{K}\neg p$.) For many domains, knowing the value of domain variables is a natural measure of knowledge, and for several others, it conveniently approximates more refined knowledge formulae.

Knowledge subgoals. For every knowledge formula ϕ_x considered by KACMBP, COMPUTEKSS computes a belief state representing the associated KS. The computation focuses on the states for which $\mathcal{K}\phi_x$ holds both before and after a given action α is performed; if some of such states may originate (via α) from a state where ϕ_x does not hold, then a KS is found: we obtain the KS by collecting all the states for which $\mathcal{K}\phi_x$ holds both before and after α is performed.

More in detail, for every knowledge formula ϕ_x, KACMBP considers singularly each disjoint $\mathcal{K}(x = v)$, denoted with $\phi_{(x,v)}$, in the selected knowledge formulae, building an associated subgoal for each of them, and finally making the union the results. The subgoal for $\phi_{(x,v)} = \mathcal{K}(x = v)$ is built as

$$KS(\phi_{(x,v)}) = \bigcup_{\alpha:\ v \in \textsc{Proj}(\textsc{Exec}^{-1}[\alpha](\llbracket\phi_x\rrbracket),x)} \big(\textsc{Exec}[\alpha]\big(\textsc{Exec}^{-1}[\alpha]\big(\llbracket\phi_x\rrbracket\big)\big) \cap \textsc{Exec}^{-1}[\alpha]\big(\llbracket\phi_x\rrbracket\big)\big).$$

That is, for every action α that leads to knowing the value v of the variable from not knowing it, we consider the states where v is known after executing α, and that are also possible before α is executed. This simplification yields speed-ups without significantly worsening the level of information of the search. Notice that, of course, such a subgoal may not exist, indicating that there is no single action that may achieve, in general, the required knowledge formula.

Computation of knowledge subgoals layers. The COMPUTEKSSDISTANCELAYERS performs a task similar to COMPUTEGOALDISTANCELAYERS, building distance layers for each of the KSs formerly identified by COMPUTEKSS. That is, the distance from each KS is projected onto each of the domain variables, and layers are computed via a V^*_{dp}-like computation. As a result, each KS is associated to a set of lists of distance layers, where a single list of distance layers is associated with a domain variable.

Target knowledge, current knowledge, knowledge bound. The target knowledge (i.e., the level of knowledge deemed necessary to achieve the goal) and the "current knowledge" (i.e., the level of knowledge that is currently available in some of the open nodes) are represented as vectors, that are initialized by SELECTTARGETKNOWLEDGE and dynamically updated from then on. Each component in a vector refers to a specific knowledge formula, and describes to which extent is that formula known. Given a belief, its associated knowledge for each formula is estimated as the cardinality of the belief projection over the formula: smaller cardinalities indicate less uncertainty, i.e., a higher degree of knowledge.

Since the knowledge formulae used by KACMBP are directly associated with state variables, this evaluation amounts to the size of the projection of belief over the domain variables. Thus, SELECTTARGETKNOWLEDGE effectively computes $|\textsc{Proj}(Goal, x)|$ and

$|\text{PROJ}(\mathit{Init}, x)|$ for every variable x of the domain, where $|Bs|$ indicates the number of states in *Bs*, and

$$\text{PROJ}(Bs, x_i) \doteq \exists x_0, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n : Bs.$$

The "knowledge bound", used to prevent useless knowledge-level searches, is also represented as a vector whose components refer to the considered knowledge formulae. This vector is initialized by SELECTTARGETKNOWLEDGE, and indicates the maximal knowledge achievable during the search for each ϕ_x. Its initial value indicates that perfect knowledge might be reached over each ϕ_x; the values for each active component are updated during the search, when a knowledge-level search attempt fails, by SETTARGETKNOWLEDGELIMITREACHED.

REACHGOALSEARCH. REACHGOALSEARCH is an iterative greedy, distance-decreasing procedure that only considers open beliefs that satisfy the current knowledge target, and expands a belief chosen according to the following criteria:

(1) The closer belief to the goal is preferred. Distance comparison amongst beliefs is based on the vector-based representation of goal distances. In particular, notice that two distance vectors A, B are non-comparable iff for some component i, $A[i] < B[i]$, and for some other component j, $A[j] > B[j]$.
(2) When two beliefs have equal or non-comparable goal distances, their truth percentage w.r.t. the goal is evaluated, and acts as a tie-breaking mechanism, choosing the belief with a higher truth percentage.
(3) As a further tie-breaking mechanism, beliefs are compared considering their knowledge level w.r.t. the knowledge formulae considered by KACMBP. Higher knowledge beliefs are of course preferred.

The search exits when either a solution is found, when the search space is exhausted, or when no belief expansion improves the current state of the search according to criteria 1 or 2.

ACQUIREKNOWLEDGESEARCH. ACQUIREKNOWLEDGESEARCH performs a search similar to that performed by REACHGOALSEARCH, but with different selection criteria. In particular, the search consists in an iterative greedy, distance decreasing procedure, where the belief state to be expanded is chosen according to the following criteria:

(1) The belief state which is closer to the active KSs. Again, distance comparison is based a vector representation, and distances may be non-comparable. Moreover, since the knowledge formulae considered by KACMBP are associated to domain variables, each vector component is associated to a knowledge formula, which may be active or not. Distance vector components related to inactive knowledge formulae are not taken into account in the comparison of the distances of two belief states.
(2) When two belief states have equal or non-comparable goal distances, as a tie-breaking mechanism, we choose the belief with a higher truth percentage of the active KSs.

(3) As a further tie-breaking mechanism, the belief state which is closer to the goal is preferred.

The search exits when either the current target knowledge has been achieved, when the search space is exhausted, or when no beliefs expansion improves the current state of the search according to criteria 1 or 2.

8. Related work

In recent years, conformant planning has received a significant amount of attention. Here we describe, in chronological order, the most significant approaches.

CGP. Historically, the first efficient conformant planner was CGP [44]. The approach underlying CGP is based on the construction of a planning graph for each initial state, and tries to propagate constraints among these planning graphs to show the conformance of the plan. The approach encompasses the case of nondeterministic action effects, at the expense of branching a planning graph for each nondeterministic action effect. The main strength of CGP is the ability to exploit the parallelism of the domain, when available. The main weakness appears to be in the enumerative nature of the algorithms (for instance, in the CUBE domain of size 10, CGP would try to generate 10^3 planning graphs). CGP cannot detect unsolvability of a problem, since it relies on a bounded-depth search.

GPT. Bonet and Geffner [6] reformulate conformant planning as search in the belief space. The approach, implemented within the GPT system, relies on the use of the V^*_{dp} heuristics to drive an A^*-style forward search algorithm. GPT is able to deal with nondeterministic action effects, and to detect when the problem is unsolvable; it is guaranteed to return optimal plans. The implementation is based on non-symbolic machinery.

QBFPLAN. Rintanen [43] extends the planning as satisfiability approach to the case of nondeterministic domains. The case of conformant planning is reduced to the problem of checking the satisfiability of a QBF formula in exists-forall form. Intuitively, the formula states that “there exists a plan such that all the associated runs are valid and result in a goal”. QBFPLAN tackles the problem of bounded-length conformant planning, i.e., it looks for a conformant solution of specified length l. When this does not exist, it iteratively increases l until a solution is found or a specified limit is reached. QBFPLAN is unable to detect when the problem is unsolvable. The use of a logic-based approach is sometimes able to mitigate the problems of an enumerative approach. However, it is difficult for QBFPLAN to scale up to large problems, possibly for the limited maturity of the technology for QBF satisfiability.

CMBP. CMBP [20] tackles conformant planning by means of symbolic, BDD-based data structures. There are however significant differences with the work presented in this paper. These stems from the fact that the whole search frontier is represented and expanded breadth-first, in a fully symbolic way, and additional BDD variables are dynamically introduced during search. In this way, the approach is guaranteed to find optimal plans. CMBP detects when a problem is unsolvable, i.e., it is not limited to bounded-length conformant planning. Despite the breadth-first style of the search, CMBP is able to outperform CGP, GPT and QBFPLAN (see [20] for a comparison).

HSCP. The Heuristic-Symbolic Conformant Planner (HSCP) described in [4] is a preliminary step towards the results presented in this paper. Basically HSCP performs a backward search in belief space based on the symbolic machinery presented in this paper, which provides an efficient way to expand and store belief states. The main difference from the present work concerns the way in which the search is guided. HSCP is driven by a simple selection criterion, based on the cardinality of the belief state. This turns out to be ineffective in many cases, since reachability information and knowledge acquisition information are both disregarded.

CPLAN. CPLAN [14] is similar in spirit to QBFPLAN. Rather than generating QBF encodings, however, CPLAN implements a "generate-and-test" specialized solver, based on traditional SAT techniques: a (modified) propositional solver runs at the top level to "enumerate" the possible candidate plans; "testing" for conformance is then reduced to a SAT validity check. If no solution is found, the length of the plan is then increased. The approach is limited to the case of bounded-length conformant planning. An interesting idea in CPLAN is a pruning mechanism that discards candidate plans at step $j + 1$ based on the information learned at step j.

FragPlan. Kurien and Smith [35] propose the *fragment-based* approach to conformant planning. The idea is to find a plan that works for one world (an initial state), and then to extend it to a full conformant plan. The core of the approach is SAT-based planning; however, rather than a blind generate-and-test procedure, the candidate solution is constructed incrementally, by combining "fragments" of plans that work only for a limited number of initial states. The approach is limited to the case of bounded-length conformant planning, and does not encompass nondeterministic action effects. The preliminary experimental results reported in [35], though quite promising, have not been further refined.

DVLK. DLV^k [29] reduces conformant planning to answer set programming, by exploiting the Disjunctive Datalog-based system DVL. The produced answer set is to be interpreted as a (parallel) plan. The domain description language of DLV^k is $\mathcal{K}$, where it is possible to express incomplete information, action nondeterminism, and initial uncertainty; in particular, in $\mathcal{K}$ it is possible to express transitions between knowledge states, where predicates can be three-valued (known true, known false, unknown). DLV^k can produce conformant plans by requiring the underlying DLV^k engine to perform "secure" reasoning, which amounts to iteratively producing weak plans and checking their security, similarly to CPLAN. DLV^k tackles bounded conformant planning problems, i.e., the length of plans must be provided to the system.

CAltAlt. CAltAlt [12] combines two interesting ideas: a backward search procedure, where belief states are represented as propositional formulae in clausal form; and the use of planning graphs to provide heuristics. As also shown in this paper, the simple combination of reachability heuristics is often unable to provide detailed guidance. Therefore, an attempt is carried out to combine information obtained from different planning graphs. The presented results are still somewhat preliminary.

Conformant FF (CFF). The most recent and impressive approach to conformant planning is implemented in CFF [8]. CFF is the first conformant planner able to fully exploit the ability of state-of-the-art classical planners to deal with large deterministic domains. CFF searches in forward the space of conformant plans, with a control structure similar to FF: it first performs an enforced hill climbing, switching to best-first search when this fails.The selection function is based on planning graph information, enhanced with efficient reasoning to detect literals that are known to (or not to) hold. CFF avoids the explicit construction of belief states; the belief state associated with an action sequence is implicitly represented as a CNF in the variables at different time instants, just like in SatPlan. The corresponding set of models is not explicitly constructed, as in our approach. Rather, these models are queried by means of a SAT engine: this enables checking if an action is applicable, if atomic knowledge formulae are known or not, and to implement a domination check between plans. By dispensing with an explicit representation of the belief state associated with a plan, CFF is sometimes able to avoid the explosion associated with the storage of the traversed belief space. However, explicit SAT checking is needed to detect (in an approximate way) if the plan is dominated by (i.e., results in a less informed belief state than) the plans that have been constructed up to this point. In our approach, domination check is limited to the case of plan equivalence, which can be carried out in constant time as equivalence checking between BDDs. Furthermore, quantitative measures (e.g., the truth percentage) are very hard to obtain by means of standard SAT-based technologies. CFF is currently unable to deal with nondeterministic action effects, and disjunctive goals are not handled in a fully general way. In [8], CFF is reported to outperform HSCP.

Other related work. Somewhat related is the approach presented in [30], where conformant planning can be codified as a form of Open World planning; the solution engine is based on the computation of prime implicates. The framework presented in [40] advocates the reformulation of planning with incomplete information at the level of knowledge, and enables tackling several forms of planning, including conformant planning. The search for a solution relies on user-defined control functions. However, this approach essentially implements a form of abstraction. This may endanger the ability to find a solution, since reasoning by cases is sometimes no longer possible.

Finally, the work presented in this paper is set in the framework of Planning via Symbolic Model Checking [17], where symbolic techniques are also used to tackle conditional planning under full observability [19,22,23,28], under partial observability [3, 5], and for temporally extended goals [41,42]. Some of the ideas presented in this paper have been presented in a preliminary way in [1,4,21].

9. Experimental comparison

9.1. Set-up of the comparison

Conformant planners. In this section we compare KACMBP with some of the most advanced competitor systems, focusing on DLV^k, CPLAN, and CFF. We do not to directly include other relevant systems such as CGP, GPT, QBFPLAN, CMBP, and HSCP. An indirect comparison with these systems, showing that they are somewhat less efficient than the ones considered here, can be derived from the analyses in [1,4,14,20,29]. The behavior of GPT (in terms of expanded nodes) can be directly obtained by the plots of the forward search driven by V^*_{dp} in Section 5.

The test set. The comparison covers the problems described in Section 5.2, and others that have been used to evaluate conformant planners in the past. Given the different expressivity of the compared systems, we do not compare all the systems on all the problems. We also remark that this is not a “competition”, which would require at least that every system takes in input the same problem description. However, we tried to highlight the significant differences, if any, in the encodings of the problems.

For each of the systems, we attempted to maximize the performance by using hand-crafted encodings provided by the developers (when available). For KACMBP, the encodings are written in SMV language. Similar to CPLAN and DLV^k, we use scalar variables, that are logarithmically encoded in booleans. We remark that the static analysis tools like Stan [31] and Discoplan [32] would be able (at least in deterministic domains) to detect and eliminate constant predicates, and partition atoms into groups of mutually exclusive facts. CFF uses propositional encodings written in a variation of PDDL. DLV^k is able to express parallel encodings, and to express incomplete knowledge by an appropriate “knowledge-based” modeling. These two modeling styles, when applicable, can significantly improve the performance with respect to serial, state-based encodings. Finally, notice that the compared systems are not solving the same problems. For instance, CPLAN and DLV^k tackle the *bounded* case of conformant planning and are therefore unable to decide whether the problem is solvable. CPLAN returns optimal plans by iteratively increasing the plan length. CFF is unable to deal with nondeterministic action effects and with full-fledged disjunctive goals.

Platform. All the experiments were run on a Pentium III Xeon 700MHz with 6GB RAM running Linux. For each run, we fix a memory limit of 512MB and a CPU time limit of 1 hour. For each problem instance, we report both the CPU time (in seconds) required to find a solution for the given planning problem, and the length of the generated plan.

For CFF and KACMBP we report total times. In the case of KACMBP this includes constructing the BDD-based representation of the automaton, and the computation of the heuristic information. For CPLAN and DLV^k the situation is slightly more complex. These systems are able to solve the problem for a given plan length. If k is the minimum solution length for a given problem instance, CPLAN has an internal iteration mechanism, which attempts all the instances up to a specified limit, and exits as soon as a solution is found (i.e., when it reaches k, or times out). For DLV^k, there is no iteration. Following [29], we report only the time needed to find a conformant plan at length k. Notice that in several cases, the time taken to prove that the problem cannot be solved with length $k-1$ may be much higher than that, as reported in the experiments. This makes the comparison between CPLAN and DLV^k somehow difficult. Concerning particular systems, we used the latest DLV release, from DLVs web page, August 1 2003. We report, for each experiment, the results for parallel encodings, using when possible the knowledge features of $\mathcal{K}$—that is, in general, the best encoding possible for the system. The timings we report differ, sometimes considerably, from those reported in [29]; these refer to a previous DLV release. It appears that some newly introduced DLV features and heuristics may to some degree affect DLVs performance when used as DLV^{k}s engine. The general scaling behavior of the various versions of DLV^k is however very similar.

9.2. Experiments and results

Bomb in the toilet domains. The results for the different versions of the Bomb in the Toilet domains are reported in Tables 1, 2, 4, 5.

For CPLAN and DLV^k, we report the times for the encodings provided with the respective distributions. We remark that these encodings *do not* capture the domain in its classical formulation (one Armed predicate), but rather the (easier) formulation proposed in [40]. For CPLAN, we also notice that the encodings used in the tests in [14] include a control strategy that forbids dunking a package twice. For CFF, we report the times on the hard formulation of the problem.

KACMBP easily solves all the problems in very short times, and easily scales up to larger problems. For instance, KACMBP is able to solve the BMTC(8,30) in 1.94 seconds, while the BMTUC(8,30) is solved in 2.53 seconds. DLV^k and CPLAN are able to solve the BT in its parallel encoding in no time, since the number of levels of the graph explored is very low. When they are required to explore higher levels, the problem becomes much harder. CFF easily outperforms CPLAN and DLV^k, even when tackling the hard version of the problem. In the BMTC, we notice that CFF is very effective in dealing with "larger" domains: when the number of toilets increases, we only notice a limited degrade in performance.

The time needed by DLV^k to determine unsatisfiability, when given an insufficient plan length, may become very large. As an example, considering plan lengths shorter than the optimal plans, in the case of the BTC(7), 58 seconds are required while for the BTC(8) the computation does not terminate in 10 minutes. Similar results are obtained for the BTUC and BMTC.

Square and cube. The results for the SQUARE and CUBE domains are reported in Tables 6 and 7. We notice that the CORNER problems are much easier than the other ones, for all the systems. CFF is able to solve the instance in the hill-climbing phase, meaning that the heuristics are very helpful. CFF is in general much more efficient than CPLAN and DLV^k. Again, the time required by DLV^k to tackle the largest unsatisfiable instance is orders of magnitude larger than the time reported in the table. For instance, it takes almost 10 minutes for the square CORNER of size 7, and 5 minutes for the cube CORNER of size 5. The difficulty increases with the EDGE, SIDE, and CENTER versions

Table 1
Results for the BT problems

	KACMBP		CPLAN		DLV^k	
	|PL|	*Time*	*|PL|*	*Time*	*|PL|*	*Time*
BT(4)	4	0.00	1	0.00	1	0.02
BT(5)	5	0.00	1	0.00	1	0.02
BT(6)	6	0.00	1	0.01	1	0.02
BT(7)	7	0.01			1	0.02
BT(8)	8	0.01			1	0.02
BT(9)	9	0.01			1	0.02
BT(10)	10	0.01			1	0.02
BT(14)	14	0.02			1	0.02
BT(18)	18	0.04			1	0.02
BT(22)	22	0.05			1	0.02
BT(26)	26	0.07			1	0.02
BT(30)	30	0.09			1	0.02
BT(35)	35	0.12			1	0.02
BT(40)	40	0.13			1	0.02
BT(50)	50	0.23			1	0.02
BT(60)	60	0.30			1	0.03
BT(70)	70	0.45			1	0.03
BT(80)	80	0.53			1	0.03
BT(90)	90	0.76			1	0.03
BT(100)	100	0.89			1	0.03
BT(120)	120	1.27			1	0.04
BT(140)	140	1.98			1	0.05
BT(160)	160	2.50			1	0.05
BT(180)	180	3.54			1	0.05
BT(200)	200	4.38			1	0.06
BT(240)	240	6.76			1	0.07
BT(280)	280	10.59			1	0.09

of the problem. None of the systems but KACMBP is able to deal efficiently with complex configurations of knowledge.

Ring domains. The RING domains (Tables 8 and 9) turn out to be quite hard. CPLAN and DLV^k are unable to deal with the serial nature of the problem and with the significant length of the plans. As in other domains, DLV^k requires a significant time to detect unsatisfiability at length $l - 1$, being l the optimal length, e.g., more than 10 minutes for rings of size 5.

CFF also faces difficulties, as also explained in [8], due to the failure of the binary clauses simplification mechanism. KACMBP is able to tackle the DET-RING quite efficiently, but the performance degrades with the DET-RING-KEY. The problem for KACMBP lies in the fact that the necessary knowledge of having the key is not straightforward to single out, and in addition no knowledge acquisition points are available for it. In the case of the nondeterministic version, which cannot be represented in CFF, the behavior of CPLAN and DLV^k degrades even further. Interestingly, KACMBP is able to do a slightly better job: sometimes higher uncertainty conditions result into more manageable BDDs.

Table 2
Results for the BTC problems

	KACMBP		CFF		CPLAN		DLV^k	
	$\|PL\|$	*Time*	$\|PL\|$	*Time*	$\|PL\|$	*Time*	$\|PL\|$	*Time*
BTC(4)	8	0.01			7	0.88	7	0.04
BTC(5)	10	0.01	10	0.02	9	21.81	9	0.04
BTC(6)	12	0.01			11	622.03	11	0.05
BTC(7)	14	0.01				$T.O.$	13	0.06
BTC(8)	16	0.02					15	0.08
BTC(9)	18	0.02					17	0.11
BTC(10)	20	0.03	20	0.33			19	0.14
BTC(14)	28	0.05					27	0.41
BTC(18)	36	0.08					35	1.08
BTC(20)	40	0.10	40	13.33			39	1.66
BTC(22)	44	0.12					43	2.35
BTC(26)	52	0.19					51	4.76
BTC(30)	60	0.25	60	124.02			59	8.82
BTC(35)	70	0.40					69	17.67
BTC(40)	80	0.55	80	645.41			79	32.88
BTC(50)	100	1.06		$T.O.$			99	102.03
BTC(60)	120	1.68					119	282.87
BTC(70)	140	2.71					139	729.48
BTC(80)	160	3.97					159	1654.21
BTC(90)	180	5.79					179	3338.58
BTC(100)	200	8.22						$T.O.$

Table 3
Results for the BTUC problems

	KACMBP		DLV^k	
	$\|PL\|$	*Time*	$\|PL\|$	*Time*
BTUC(4)	8	0.01	7	0.07
BTUC(5)	10	0.01	9	0.05
BTUC(6)	12	0.00	11	0.07
BTUC(7)	14	0.01	13	0.09
BTUC(8)	16	0.02	15	0.10
BTUC(9)	18	0.02	17	0.15
BTUC(10)	20	0.03	19	0.24
BTUC(14)	28	0.05	27	0.43
BTUC(18)	36	0.09	35	1.12
BTUC(22)	44	0.13	43	2.44
BTUC(26)	52	0.21	51	4.94
BTUC(30)	60	0.28	59	9.10
BTUC(35)	70	0.45	69	18.10
BTUC(40)	80	0.61	79	33.36
BTUC(50)	100	1.18	99	103.22
BTUC(60)	120	1.93	119	288.11
BTUC(70)	140	3.10	139	745.35
BTUC(80)	160	4.60	159	1703.74
BTUC(90)	180	6.79	179	3373.04
BTUC(100)	200	9.40		$T.O.$

Table 4
Results for the BMTC problems (up to 5 toilets)

	KACMBP		CFF		CPLAN		DLV^k	
	|PL|	*Time*	*|PL|*	*Time*	*|PL|*	*Time*	*|PL|*	*Time*
BMTC(2,04)	8	0.01			3	0.46	3	0.07
BMTC(2,05)	10	0.02	10	0.02	5	18.54	5	0.04
BMTC(2,06)	12	0.02			5	256.33	5	0.10
BMTC(2,07)	14	0.02				$T.O.$	7	0.97
BMTC(2,08)	16	0.03					7	25.34
BMTC(2,09)	18	0.05					9	426.50
BMTC(2,10)	20	0.05	20	0.47				$T.O.$
BMTC(2,14)	28	0.10						
BMTC(2,18)	36	0.14						
BMTC(2,20)	40	0.16	40	16.95				
BMTC(2,22)	44	0.21						
BMTC(2,26)	52	0.28						
BMTC(2,30)	60	0.36	60	154.31				
BMTC(2,40)	80	0.60	80	799.90				
BMTC(2,50)	100	0.97		$T.O.$				
BMTC(3,04)	8	0.01			3	0.89	3	0.04
BMTC(3,05)	10	0.02	10	0.02	3	2.11	3	0.04
BMTC(3,06)	12	0.02			3	4.25	3	0.07
BMTC(3,07)	14	0.03				$T.O.$	5	0.19
BMTC(3,08)	16	0.04					5	0.76
BMTC(3,09)	18	0.05					5	21.87
BMTC(3,10)	20	0.07	20	0.66			7	334.73
BMTC(3,14)	28	0.14						$T.O.$
BMTC(3,18)	36	0.25						
BMTC(3,20)	40	0.32	40	20.98				
BMTC(3,22)	44	0.40						
BMTC(3,26)	52	0.58						
BMTC(3,30)	60	0.85	60	188.41				
BMTC(3,40)	80	1.94	80	981.92				
BMTC(4,04)	8	0.04			1	0.02	1	0.05
BMTC(4,05)	10	0.05	10	0.03	3	26.95	3	0.07
BMTC(4,06)	12	0.05			3	74.63	3	0.08
BMTC(4,07)	14	0.07				$T.O.$	3	0.11
BMTC(4,08)	16	0.08					3	0.21
BMTC(4,09)	18	0.11					5	1.27
BMTC(4,10)	20	0.12	20	0.75			5	29.84
BMTC(4,14)	28	0.20						$T.O.$
BMTC(4,18)	36	0.31						
BMTC(4,20)	40	0.35	40	25.29				
BMTC(4,22)	44	0.43						
BMTC(4,26)	52	0.54						
BMTC(4,30)	60	0.71	60	225.56				
BMTC(4,40)	80	1.15	80	1174.54				

(*continued on next page*)

Table 4 (*continued*)

	KACMBP		CFF		CPLAN		DLV^k	
	|PL|	*Time*	*|PL|*	*Time*	*|PL|*	*Time*	*|PL|*	*Time*
BMTC(5,04)	8	0.05			1	0.01	1	0.06
BMTC(5,05)	10	0.08			1	0.01	1	0.05
BMTC(5,06)	12	0.08			3	920.36	3	0.09
BMTC(5,07)	14	0.10				$T.O.$	3	0.09
BMTC(5,08)	16	0.12					3	0.15
BMTC(5,09)	18	0.16					3	0.28
BMTC(5,10)	20	0.17					3	2.10
BMTC(5,14)	28	0.31						$T.O.$
BMTC(5,18)	36	0.47						
BMTC(5,22)	44	0.64						
BMTC(5,26)	52	0.89						
BMTC(5,30)	60	1.14						
BMTC(5,40)	80	1.90						

Harder navigation domains. The harder navigation domains were not attempted with DLV^k and CPLAN, given their difficulties with the simple navigation domains. The experiments show that KACMBP is very effective in solving even larger instances of such problems (Table 10). This is due to the presence of knowledge-based heuristics. CFF seems to perform quite well for the small instances. However, it seems not to scale-up as well as KACMBP. In the SQUARE-ALLEY problem instances, CFF was run on a simplified problem whose goal entails the original one because of its inability to express generic disjunctive goals. In the TREASURE(11) instance CFF reports reaching a hard-coded plan bound ($M.P.L$). In the cube obstacle, CFF was run on a simplified bidimensional version of the problem, but run out of time.

Pigeon-hole scattered obstacles. In [14], a bidimensional navigation domain is proposed to evaluate CPLAN. Each grid of side n contains n one-square fixed obstacles, that are scattered according to the pigeon-hole (PH) principle (i.e., at least one per row, and no more than one per column), and fixed. The position of d of these n obstacles is unknown. Differently from the other navigation domains described above, robots must avoid bumping into the walls and into the obstacles. We consider the problem, called OPH(n, d), where the robot is initially in $(0, 0)$, and has to reach any location with $x = n$. Each instance is determined by choosing any PH-consistent configuration for the fixed obstacles.

Since a conformant plan must avoid all possible obstacles, the problem requires solving a pigeon-hole formula. Notice that not all the problem instances are solvable, depending on the way fixed obstacles are scattered.

In CFF it is impossible to express the PH principle directly, since the description of uncertainty conditions is subject to specific restrictions. For this reason, the position of unknown obstacles is statically encoded not to clash with the obstacles with known position. In essence, this reduces the size of the PH problem tackled from n to d. In CFF, we have a boolean proposition for each location that may contain an obstacle with unknown position, thus resulting in d^2 propositions. In KACMBP, we exploit boolean fluents by

Table 5
Results for the BMTUC problems (up to 5 toilets)

	KACMBP		DLV^k	
	$\|PL\|$	*Time*	$\|PL\|$	*Time*
BMTUC(2,04)	8	0.02	3	0.04
BMTUC(2,05)	10	0.02	5	0.05
BMTUC(2,06)	12	0.03	5	0.19
BMTUC(2,07)	14	0.03	7	2.23
BMTUC(2,08)	16	0.03	7	66.26
BMTUC(2,09)	18	0.05	9	2095.39
BMTUC(2,10)	20	0.06		$T.O.$
BMTUC(2,14)	28	0.09		
BMTUC(2,18)	36	0.14		
BMTUC(2,22)	44	0.21		
BMTUC(2,26)	52	0.29		
BMTUC(2,30)	60	0.36		
BMTUC(3,04)	8	0.01	3	0.04
BMTUC(3,05)	10	0.01	3	0.04
BMTUC(3,06)	12	0.02	3	0.06
BMTUC(3,07)	14	0.03	5	0.17
BMTUC(3,08)	16	0.01	5	0.88
BMTUC(3,09)	18	0.05	5	41.48
BMTUC(3,10)	20	0.05	7	917.33
BMTUC(3,14)	28	0.12		$T.O.$
BMTUC(3,18)	36	0.20		
BMTUC(3,22)	44	0.32		
BMTUC(3,26)	52	0.48		
BMTUC(3,30)	60	0.65		
BMTUC(4,04)	8	0.04	1	0.06
BMTUC(4,05)	10	0.05	3	0.06
BMTUC(4,06)	12	0.06	3	0.07
BMTUC(4,07)	14	0.06	3	0.10
BMTUC(4,08)	16	0.08	5	0.59
BMTUC(4,09)	18	0.12	5	1.19
BMTUC(4,10)	20	0.14	7	65.70
BMTUC(4,14)	28	0.20		$T.O.$
BMTUC(4,18)	36	0.36		
BMTUC(4,22)	44	0.53		
BMTUC(4,26)	52	0.72		
BMTUC(4,30)	60	0.95		
BMTUC(5,04)	8	0.04	1	0.06
BMTUC(5,05)	10	0.08	1	0.05
BMTUC(5,06)	12	0.09	3	0.08
BMTUC(5,07)	14	0.11	3	0.10
BMTUC(5,08)	16	0.14	3	0.15
BMTUC(5,09)	18	0.17	3	1.12
BMTUC(5,10)	20	0.19	3	14.06
BMTUC(5,14)	28	0.36		$T.O.$
BMTUC(5,18)	36	0.53		
BMTUC(5,22)	44	0.78		
BMTUC(5,26)	52	1.10		
BMTUC(5,30)	60	1.48		

Table 6
Results for the SQUARE problems

	KACMBP		CFF		CPLAN		DLV^k	
	|PL|	*Time*	*|PL|*	*Time*	*|PL|*	*Time*	*|PL|*	*Time*
SQUARE-corner(3)	6	0.00	4	0.00	6	1.55	3	0.00
SQUARE-corner(4)	8	0.00			8	20.55	4	0.04
SQUARE-corner(5)	10	0.00	8	0.03	10	2278.88	5	0.07
SQUARE-corner(6)	12	0.00				$T.O.$	6	0.10
SQUARE-corner(7)	14	0.00	12	0.23			7	0.15
SQUARE-corner(9)	18	0.00	16	0.64			9	0.31
SQUARE-corner(11)	22	0.00	20	1.96			11	0.57
SQUARE-corner(13)	26	0.00					13	1.00
SQUARE-corner(15)	30	0.00					15	1.75
SQUARE-corner(17)	34	0.01					17	2.77
SQUARE-corner(19)	38	0.01					19	4.53
SQUARE-corner(25)	50	0.02					25	16.34
SQUARE-corner(35)	70	0.02					35	79.30
SQUARE-side(3)	7	0.00	6	0.01	7	16.55	4	0.07
SQUARE-side(4)	12	0.00				$T.O.$	6	2.73
SQUARE-side(5)	12	0.01	16	0.25			7	22.33
SQUARE-side(7)	17	0.01	18	6.09			10	2772.95
SQUARE-side(9)	22	0.01	35	46.26				$T.O.$
SQUARE-side(11)	27	0.02		$T.O.$				
SQUARE-side(13)	32	0.02						
SQUARE-side(15)	37	0.03						
SQUARE-side(17)	42	0.04						
SQUARE-side(19)	47	0.04						
SQUARE-side(25)	62	0.07						
SQUARE-side(35)	87	0.11						
SQUARE-center(3)	10	0.00	10	0.01	8	2776.23	4	0.24
SQUARE-center(4)	14	0.00				$T.O.$	6	43.79
SQUARE-center(5)	18	0.02	23	9.42			7	1950.37
SQUARE-center(7)	25	0.02		$T.O.$				$T.O.$
SQUARE-center(9)	32	0.04						
SQUARE-center(11)	39	0.03						
SQUARE-center(13)	46	0.05						
SQUARE-center(15)	53	0.04						
SQUARE-center(17)	60	0.07						
SQUARE-center(19)	67	0.09						
SQUARE-center(25)	88	0.12						
SQUARE-center(35)	123	0.19						

introducing two position variables for each of the d unknown obstacles; these variables are stated not to change over time, and the initial assignments are all the PH-consistent ones.

We ran the experiments by sampling the space of the possible instances, with increasing dimension, and increasing number of unknown position obstacles. CPLAN was run from dimension 6 to dimension 15, with step size 1. CFF and KACMBP were run from 6 to 20 with step size 1, and then up to 150 with step size 10. For each configuration, 4 instances (with different random seeds) were run. The key factor in performance is the number of

Table 7
Results for the CUBE problems

	KACMBP		CFF		CPLAN		DLV^k	
	|PL|	*Time*	*|PL|*	*Time*	*|PL|*	*Time*	*|PL|*	*Time*
CUBE-corner(3)	9	0.00	6	0.00		$T.O.$	3	0.05
CUBE-corner(5)	15	0.01	12	0.17			5	0.16
CUBE-corner(7)	21	0.00	18	1.33			7	0.44
CUBE-corner(9)	27	0.01	24	6.12			9	1.07
CUBE-corner(11)	33	0.01	30	19.11			11	2.39
CUBE-corner(13)	39	0.02					13	4.74
CUBE-corner(15)	45	0.02					15	9.07
CUBE-corner(17)	51	0.03					17	17.27
CUBE-corner(19)	57	0.02					19	31.77
CUBE-corner(25)	75	0.04					25	185.87
CUBE-corner(35)	105	0.07					35	1433.91
CUBE-edge(3)	10	0.01	8	0.04		$T.O.$	4	1.61
CUBE-edge(5)	17	0.02	20	1.73				$T.O.$
CUBE-edge(7)	24	0.02	24	54.53				
CUBE-edge(9)	31	0.04		$T.O.$				
CUBE-edge(11)	38	0.04						
CUBE-edge(13)	45	0.05						
CUBE-edge(15)	52	0.05						
CUBE-edge(17)	59	0.11						
CUBE-edge(19)	66	0.11						
CUBE-edge(25)	87	0.19						
CUBE-edge(35)	122	0.32						
CUBE-side(3)	13	0.02	12	0.14		$T.O.$	4	0.30
CUBE-side(5)	23	0.04		$T.O.$			7	523.73
CUBE-side(7)	32	0.03						$T.O.$
CUBE-side(9)	41	0.08						
CUBE-side(11)	50	0.10						
CUBE-side(13)	59	0.13						
CUBE-side(15)	68	0.11						
CUBE-side(17)	77	0.19						
CUBE-side(19)	86	0.23						
CUBE-side(25)	113	0.36						
CUBE-side(35)	158	0.63						
CUBE-center(3)	14	0.02	15	0.13		$T.O.$		10.33
CUBE-center(5)	25	0.04		$T.O.$				$T.O.$
CUBE-center(7)	35	0.04						
CUBE-center(9)	45	0.09						
CUBE-center(11)	55	0.10						
CUBE-center(13)	65	0.16						
CUBE-center(15)	75	0.15						
CUBE-center(17)	85	0.25						
CUBE-center(19)	95	0.27						
CUBE-center(25)	125	0.43						
CUBE-center(35)	175	0.79						
CUBE-center(45)	225	1.22						
CUBE-center(55)	275	1.50						
CUBE-center(65)	325	2.46						
CUBE-center(75)	375	3.10						

Table 8
Results for the RING domains

	KACMBP		CFF		CPLAN		DLV^k	
	|PL|	*Time*	*|PL|*	*Time*	*|PL|*	*Time*	*|PL|*	*Time*
DET-RING(2)	5	0.00	7	0.05	5	5.24	5	0.04
DET-RING(3)	8	0.00	15	4.60		*T.O.*	8	0.06
DET-RING(4)	11	0.00	26	183.48			11	0.09
DET-RING(5)	14	0.01		*T.O.*			14	0.13
DET-RING(6)	17	0.01					17	0.29
DET-RING(7)	20	0.02					20	3.31
DET-RING(8)	23	0.02					23	78.46
DET-RING(9)	26	0.03					26	2422.98
DET-RING(10)	29	0.03						*T.O.*
DET-RING(11)	32	0.04						
DET-RING(12)	35	0.06						
DET-RING(13)	38	0.08						
DET-RING(14)	41	0.08						
DET-RING(15)	44	0.11						
DET-RING(16)	47	0.13						
DET-RING(17)	50	0.16						
DET-RING(18)	53	0.18						
DET-RING(19)	56	0.21						
DET-RING(20)	59	0.25						
DET-RING(25)	74	0.51						
DET-RING(30)	89	0.85						
DET-RING(35)	104	1.40						
DET-RING(40)	119	2.17						
DET-RING-KEY(2)	13	0.02	11	0.24		*T.O.*	8	1.46
DET-RING-KEY(3)	15	0.02	16	4.22			13	649.08
DET-RING-KEY(4)	25	0.08		*T.O.*				*T.O.*
DET-RING-KEY(5)	34	0.18						
DET-RING-KEY(6)	41	0.36						
DET-RING-KEY(7)	56	0.56						
DET-RING-KEY(8)	63	1.69						
DET-RING-KEY(9)	85	2.40						
DET-RING-KEY(10)	99	12.82						
DET-RING-KEY(11)	117	14.36						
DET-RING-KEY(12)	129	121.24						
DET-RING-KEY(13)	156	130.78						
DET-RING-KEY(14)	159	2060.09						
DET-RING-KEY(15)	198	2110.56						
DET-RING-KEY(16)		*T.O.*						

obstacles with unknown position. For this class of problems, we report the performance of the A* algorithm driven by the MAXSDIST heuristics. It is interesting to note that, for this problem class, this is the most effective conformant planner, with very stable behavior, and it returns optimal plans. Each of the plots in Table 12 shows the performance of the systems for a given number of obstacles with unknown position. We see that CFF and KACMBP outperform CPLAN by orders of magnitude. While the behavior of V^*_{dp} is

Table 9
Results for the ND-RING domains

	KACMBP		CPLAN		DLV^k	
	|PL|	*Time*	*|PL|*	*Time*	*|PL|*	*Time*
ND-RING(2)	5	0.00	5	43.42	5	0.04
ND-RING(3)	8	0.00		$T.O.$	8	0.06
ND-RING(4)	11	0.01			11	10.37
ND-RING(5)	14	0.01			14	443.03
ND-RING(6)	17	0.01				$T.O.$
ND-RING(7)	20	0.02				
ND-RING(8)	23	0.02				
ND-RING(9)	26	0.03				
ND-RING(10)	29	0.04				
ND-RING(11)	32	0.05				
ND-RING(12)	35	0.06				
ND-RING(13)	38	0.07				
ND-RING(14)	41	0.10				
ND-RING(15)	44	0.12				
ND-RING(16)	47	0.14				
ND-RING(17)	50	0.16				
ND-RING(18)	53	0.20				
ND-RING(19)	56	0.23				
ND-RING(20)	59	0.24				
ND-RING(25)	74	0.52				
ND-RING(30)	89	0.88				
ND-RING(35)	104	1.48				
ND-RING(40)	119	2.31				
ND-RING-KEY(2)	9	0.00		$T.O.$	8	1.12
ND-RING-KEY(3)	15	0.03			13	551.74
ND-RING-KEY(4)	21	0.08				$T.O.$
ND-RING-KEY(5)	29	0.13				
ND-RING-KEY(6)	41	0.24				
ND-RING-KEY(7)	48	0.37				
ND-RING-KEY(8)	57	0.71				
ND-RING-KEY(9)	69	1.01				
ND-RING-KEY(10)	79	2.26				
ND-RING-KEY(11)	86	2.85				
ND-RING-KEY(12)	98	6.80				
ND-RING-KEY(13)	112	8.30				
ND-RING-KEY(14)	131	24.28				
ND-RING-KEY(15)	139	28.37				
ND-RING-KEY(16)	153	78.99				
ND-RING-KEY(17)	156	91.81				
ND-RING-KEY(18)	181	341.61				
ND-RING-KEY(19)	189	361.30				
ND-RING-KEY(20)	206	1293.55				
ND-RING-KEY(25)		$T.O.$				
ND-RING-KEY(30)						
ND-RING-KEY(35)						
ND-RING-KEY(40)						

Table 10
Results for the Harder Navigation domains (I)

	KACMBP		CFF	
	$\|PL\|$	*Time*	$\|PL\|$	*Time*
TREASURE(2)			6	0.01
TREASURE(3)	3	0.01	7	0.05
TREASURE(4)	6	0.01	7	0.00
TREASURE(5)	6	0.01	10	0.40
TREASURE(6)	9	0.02		
TREASURE(7)	9	0.02		
TREASURE(8)	12	0.02		
TREASURE(9)	12	0.02		
TREASURE(10)	15	0.03	16	0.02
TREASURE(11)	15	0.03		$M.P.L$
TREASURE(12)	18	0.03		
TREASURE(13)	18	0.04		
TREASURE(14)	21	0.05		
TREASURE(15)	21	0.04		
TREASURE(16)	24	0.06		
TREASURE(18)	27	0.07		
TREASURE(20)	30	0.08		
TREASURE(22)	33	0.10		
TREASURE(24)	36	0.11		
TREASURE(26)	39	0.12		
TREASURE(28)	42	0.13		
TREASURE(30)	45	0.16		
SQUARE-ALLEY(3)	10	0.01	7	0.03
SQUARE-ALLEY(4)	12	0.01		$T.O.$
SQUARE-ALLEY(5)	38	0.04		
SQUARE-ALLEY(6)	40	0.06		
SQUARE-ALLEY(7)	43	0.07		
SQUARE-ALLEY(8)	42	0.11		
SQUARE-ALLEY(9)	46	0.15		
SQUARE-ALLEY(10)	79	0.24		
SQUARE-ALLEY(11)	69	0.21		
SQUARE-ALLEY(12)	59	0.39		
SQUARE-ALLEY(13)	67	0.36		
SQUARE-ALLEY(14)	102	0.75		
SQUARE-ALLEY(15)	85	0.81		
SQUARE-WALL-HOLE(3)	6	0.00	6	0.03
SQUARE-WALL-HOLE(4)	7	0.01		
SQUARE-WALL-HOLE(5)	10	0.01	9	0.10
SQUARE-WALL-HOLE(6)	11	0.01		
SQUARE-WALL-HOLE(7)	14	0.01	13	0.39
SQUARE-WALL-HOLE(8)	15	0.01		
SQUARE-WALL-HOLE(9)	18	0.02		$T.O.$
SQUARE-WALL-HOLE(10)	19	0.01		
SQUARE-WALL-HOLE(11)	22	0.01		
SQUARE-WALL-HOLE(12)	23	0.02		
SQUARE-WALL-HOLE(13)	26	0.01		
SQUARE-WALL-HOLE(14)	27	0.02		
SQUARE-WALL-HOLE(15)	30	0.02		

Table 11
Results for the Harder Navigation domains (II)

	KACMBP		CFF	
	$\|PL\|$	*Time*	$\|PL\|$	*Time*
CUBE-OBSTACLE(3)		0.07		
CUBE-OBSTACLE(4)	17	0.04		$T.O.$SQUARE
CUBE-OBSTACLE(5)	17	0.04		
CUBE-OBSTACLE(6)	11	0.03		
CUBE-OBSTACLE(7)	11	0.02		
CUBE-OBSTACLE(8)	17	0.05		
CUBE-OBSTACLE(9)	17	0.05		
CUBE-OBSTACLE(10)	23	0.08		
CUBE-OBSTACLE(11)	23	0.08		
CUBE-OBSTACLE(12)	29	0.10		
CUBE-OBSTACLE(13)	29	0.10		
CUBE-OBSTACLE(14)	35	0.13		
CUBE-OBSTACLE(15)	35	0.12		
CUBE-OBSTACLE(16)	41	0.17		
CUBE-OBSTACLE(17)	41	0.18		
CUBE-OBSTACLE(18)	47	0.22		
CUBE-OBSTACLE(19)	47	0.20		
CUBE-OBSTACLE(20)	53	0.26		
CUBE-OBSTACLE(21)	53	0.25		
CUBE-OBSTACLE(22)	59	0.28		
CUBE-OBSTACLE(23)	59	0.27		
CUBE-OBSTACLE(24)	65	0.35		
CUBE-OBSTACLE(25)	65	0.33		
CUBE-OBSTACLE(26)	71	0.40		
CUBE-OBSTACLE(27)	71	0.36		
CUBE-OBSTACLE(28)	77	0.42		
CUBE-OBSTACLE(29)	77	0.47		
CUBE-OBSTACLE(30)	83	0.52		
CUBE-OBSTACLE(50)	143	1.42		
CUBE-OBSTACLE(70)	203	3.03		
CUBE-OBSTACLE(100)	293	6.89		
CUBE-OBSTACLE(150)	443	19.45		
CUBE-OBSTACLE(200)	593	41.04		
CUBE-OBSTACLE(250)	743	69.20		
CUBE-OBSTACLE(300)	893	23.77		

quite stable, KACMBP and CFF shows some variance in the results. The main bottleneck for KACMBP and V_{dp}^* is the construction of the domain: the BDD representation of the pigeon-hole formulae is a hard problem.

Logistics. The Logistic conformant planning problems from [8] are a modification of the logistic domains for classical planning. The uncertainty lies in that the initial position of each package is unknown (with uncertainty limited to the loading locations within a city). Loading a package onto a truck has the expected effect only when the package is at the same location as the truck, otherwise it has no effect. The uncertainty on the position of packages can be eliminated by performing a sequence of load actions in all the locations

Table 12
The results of the search on the PHO domains

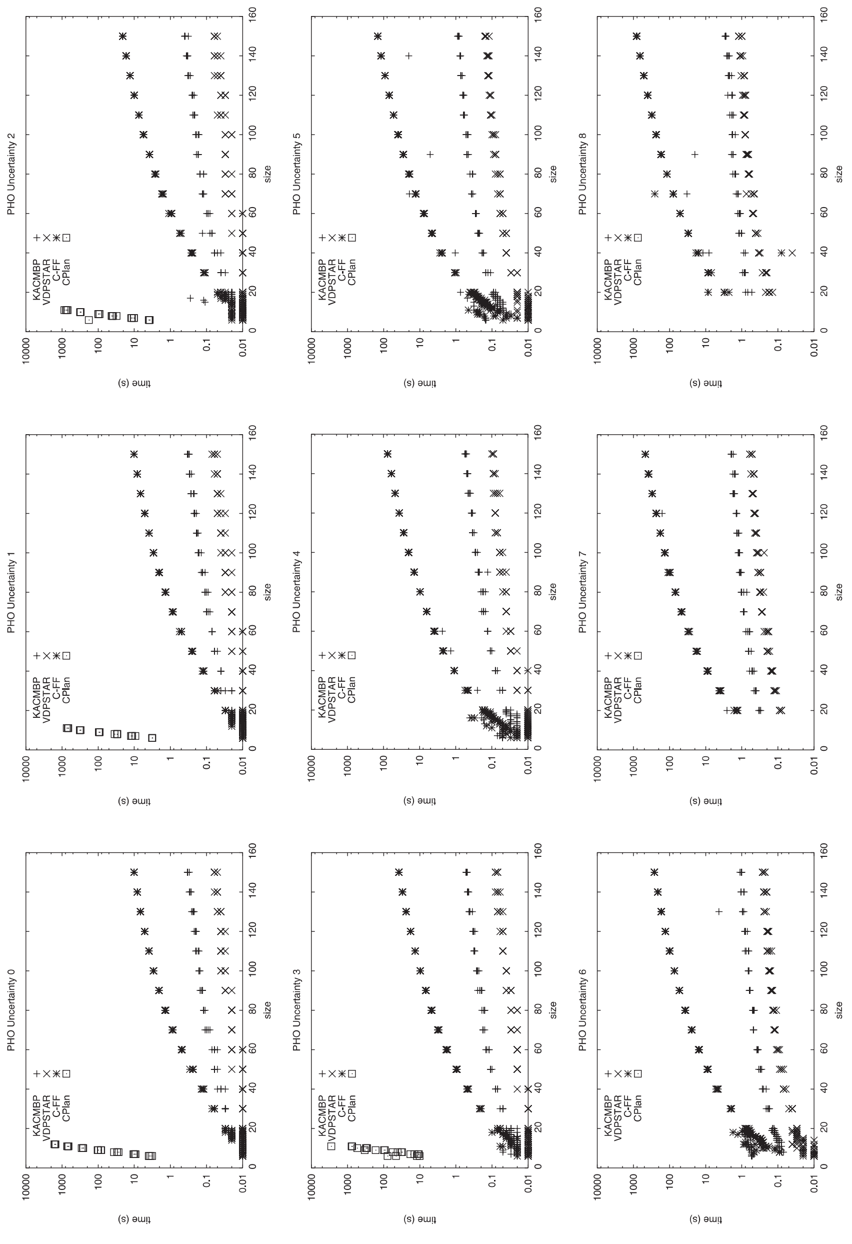


Table 13
Results for the LOGISTICS domains

	KACMBP		CFF	
	$\|PL\|$	*Time*	$\|PL\|$	*Time*
LOGISTIC(2-2-2)	14	0.34	16	0.02
LOGISTIC(2-2-4)			26	0.03
LOGISTIC(2-3-2)			17	0.01
LOGISTIC(2-3-3)	30	236.97	24	0.02
LOGISTIC(2-4-4)		$T.O.$		
LOGISTIC(2-10-10)			80	7.38
LOGISTIC(3-2-2)	17	3.63	20	0.02
LOGISTIC(3-2-4)			33	0.06
LOGISTIC(3-3-2)			28	0.05
LOGISTIC(3-3-3)	33	0.49	34	0.13
LOGISTIC(3-10-10)			108	96.84
LOGISTIC(4-2-2)	21	44.09	19	0.01
LOGISTIC(4-2-4)			40	0.26
LOGISTIC(4-3-2)			23	0.03
LOGISTIC(4-3-3)		$T.O.$		
LOGISTIC(4-10-10)			121	123.34
LOGISTIC(5-2-2)	47	491.50		
LOGISTIC(6-2-2)		$T.O.$		

where the package could be located. We considered the problem instances as provided within the available distribution of CFF. LOGISTIC(l, c, p) consists of a problem instance with c cities, with l locations per city and p packs. There is one airport, one plane, and a truck per city. The goal is to have all the packages at a given location. The results of the experiments are reported in Table 13 were we compare CFF with KACMBP. The results show that CFF is quite effective in finding a solution for this class of problems. KACMBP suffers from the extreme difficulty in detecting the knowledge acquisition points for the given representation.

Blocksworld. The results for the BLOCKSWORLD domain are reported in Table 14. In this conformant version of the domain, it is possible to move a block to the table even if a pile of other blocks is stacked onto it, in which case the whole pile will move along. This makes it possible to reach final configurations in spite of initial uncertainty. No action nondeterminism appears, which makes the problem amenable to CFF. For this domain, CFF is extremely efficient due to its clever use of graph-based heuristics. KACMBP does not scale up as well; this is due almost exclusively to the computation time for the distance-based heuristics, growing with the size of the problem. Once the layered structures for the heuristics are computed, though, KACMBP performs an extremely efficient search.

9.3. *Analysis of results*

KACMBP is the only planner able to tackle the whole range of problems. The quality of the returned plans is in general not too far from the optimal. The main difficulty for KACMBP is the sheer size of the domains, as shown by the performance in LOGISTIC

Table 14
Results for the BLOCKSWORLD domains

	KACMBP		CFF	
	$\|PL\|$	*Time*	$\|PL\|$	*Time*
BW(2-5-0)	5	0.03	7	0.01
BW(2-6-0)	10	0.27	15	0.05
BW(2-7-0)	12	0.58	13	0.03
BW(2-13-0)		$T.O.$	20	0.29
BW(2-20-0)			30	9.41
BW(3-5-0)	7	0.03	7	0.00
BW(3-6-0)	13	0.31	12	0.03
BW(3-7-0)	25	0.50	15	0.03
BW(3-13-0)		$T.O.$	31	1.02
BW(3-20-0)			41	6.22
BW(4-5-0)	7	0.01	7	0.00
BW(4-6-0)	13	0.30	19	0.34
BW(4-7-0)	15	1.23	23	0.60
BW(4-13-0)		$T.O.$	43	264.25
BW(4-20-0)			47	9.01

and BLOCKSWORLD. Notice however that the main bottleneck is not in the automaton construction, nor in the search, but rather in the pre-computation of the layers for the heuristics. Once the distance layers are computed, the LOGISTIC and BLOCKSWORLD problems are solved very efficiently. This leaves the hope of improving the performance by means of less costly heuristics, for instance planning graph technology, similarly to what is used in CFF or in CAltAlt. An open issue is the construction of planning graphs for nondeterministic action effects.

DLV^k gains significantly in performance when the encoding can exploit parallel actions and knowledge-level descriptions. Even so, the performance of DLV^k does not scale up well when longer plans are needed. A part of the problem may be related to the fact that the underlying DLV engine exploits general-purpose heuristics which have not been designed with planning issues in mind.

CPLAN is uniformly outperformed by the other systems. In certain domains, it is unable to solve even the simplest instances. This can be explained by considering that the "generate-and-test" approach is bound to fail when possible (but invalid) plans have lengths significantly lower than optimal solutions. The system, in this case, has to evaluate and rule them out before increasing the length of the search so that a solution can be found. One may consider that CPLAN analyzes plans without constructing the associated belief states. This might be seen as an advantage in terms of memory requirements. However, saving belief states is an extremely effective way to cache previous computations—our approach is able to prune plans associated with previously constructed belief states. Furthermore, CPLAN is unable to drive the search based on the level of knowledge associated with a plan. This seems to suggest that search in the belief space is a much more promising approach, as also shown by the good performance of CFF.

CFF is extremely effective in tackling a number of large problems, such as the LOGISTIC and the BLOCKSWORLD domains, by taking full advantage of classical planning technology. However, its performance degrades in several domains, for different reasons. When actions are present with multiple preconditions, as in the (deterministic) RING domain, the simplification mechanism based on binary implications used by CFF becomes ineffective, as discussed in [8]. In the case of SQUARE and CUBE, the performance degrades because of the difficulty in dealing with knowledge. Notice also that the use of propositional encodings for representing functions (e.g., the robot position) is intrinsically a bottleneck.

10. Conclusions and future work

In this paper, we addressed the problem of conformant planning, making two main contributions. First, we extended techniques from symbolic model checking to provide an efficient mechanism to store and expand the belief space traversed during the search. Second, we enhanced heuristic search in the belief space by taking into explicit account the knowledge associated with belief states. Based on the notion of necessary knowledge, we derived admissible and more informed selection functions, and proposed a planning algorithm able to extract them and exploit them. The conformant planning algorithm implemented in MBP, though suboptimal, is shown to outperform the most efficient competing conformant planners in a thorough experimental evaluation.

In the future, our first priority is to investigate planning graphs techniques to overcome the bottleneck associated with the BDD-based computation of heuristic information. Then, we will investigate the possibility of deriving (and exploit) invariants on knowledge that can be used as pruning conditions, and to identify knowledge acquisition points more accurately. Finally, we plan to lift the principles discussed in this paper to the case of partial observability [5], along the lines previously stated in [3].

Acknowledgements

We would like to thank Enrico Giunchiglia and Claudio Castellini for providing CPLAN; Ronen Brafman and Joerg Hoffman for providing CFF and the problem set; Axel Polleres for providing scripts for the comparisons with DLV^k; Blai Bonet, Hector Geffner, Jim Kurien and David Smith for many useful discussions.

Appendix A. Proofs for the forward algorithm

Definition A.1 (*Set of visited belief states*). Let $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ be a planning problem. The set of visited belief states at the ith iteration of the *while* loop of the $\textsc{HeurConformantFwd}(\mathcal{I}, \mathcal{G})$, written $\textsc{BsVisited}(i)$, is the set

$$\textsc{BsVisited}(i) \doteq \textsc{PoolBStates}(\mathit{Open}, i) \cup \textsc{PoolBStates}(\mathit{Closed}, i),$$

where $\textsc{PoolBStates}(\mathit{Open}, i) = \{Bs \mid \langle Bs . \pi\rangle \in \mathit{Open}\}$, and $\textsc{PoolBStates}(\mathit{Closed}, i) = \{Bs \mid \langle Bs . \pi\rangle \in \mathit{Closed}\}$ denote the set of belief states in *Open* and in *Closed* at the ith iteration of the *while* loop.

At each iteration i of the *while* loop of the algorithm the set of belief states contained in *Open* and *Closed* are disjoint.

Lemma A.1. *Let $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G}\rangle$ be a planning problem. At the ith iteration of the* while *loop of* $\textsc{HeurConformantFwd}(\mathcal{I}, \mathcal{G})$ *it holds*

$$\textsc{PoolBStates}(\mathit{Open}, i) \cap \textsc{PoolBStates}(\mathit{Closed}, i) = \emptyset.$$

Proof. The proof is by induction on i.
Case $i = 0$. Initially $\mathit{Open} = \{\langle \mathcal{I} . \varepsilon\rangle\}$ and $\mathit{Closed} = \emptyset$, thus $\{\mathcal{I}\} \cap \emptyset = \emptyset$.
Case $i = i' + 1$. By the induction hypothesis we have that

$$\textsc{PoolBStates}(\mathit{Open}, i') \cap \textsc{PoolBStates}(\mathit{Closed}, i') = \emptyset.$$

At iteration i one element is removed from *Open* (line 6) and is added to *Closed* (line 7). At line 14, BsP pairs which are neither in *Open* nor in *Closed* are added to *Open* (at line 12 the call to $\textsc{PruneBsExpansion}$ guarantes the removal of already visited belief states). Hence, the relation among the belief states in both pools is maintained. □

The set of visited belief states is monotonically increasing and there is a bound on the size of the set of visited belief states.

Lemma A.2. *Let $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G}\rangle$ be a planning problem. At the ith iteration of the* while *loop of* $\textsc{HeurConformantFwd}(\mathcal{I}, \mathcal{G})$,

(1) $\textsc{BsVisited}(i) \subseteq \textsc{BsVisited}(i + 1)$,
(2) *There exists an index i such that* $\textsc{BsVisited}(i) = \textsc{BsVisited}(i + j)$ *for all* $j \geqslant 0$.

Proof. (1) At each iteration i of the *while* loop, if *Open* is not empty, a *BsP* pair $\langle Bs . \pi\rangle$ is removed from *Open* and added to *Closed* (lines 9–10). No other removal may take place on *Open* or *Closed*. Thus if $Bs \in \textsc{BsVisited}(i)$, $Bs \in \textsc{BsVisited}(i + 1)$. Moreover, for each BsP pair $\langle Bs . \alpha\rangle$ in *BsPList* a $\langle Bs . \pi; \alpha\rangle$ is added to *Open* at $i + 1$. BsP pairs contained in *BsPList* are such that they have not been visited at previous iterations. Hence, $Bs \notin \textsc{BsVisited}(i)$. If $\mathit{BsPList} = \emptyset$ then the only change is that an element is removed from *Open* and added to *Closed*, thus $\textsc{BsVisited}(i) = \textsc{BsVisited}(i + 1)$. Otherwise, a *BsP* pair $\langle Bs . \pi; \alpha\rangle$, such that $Bs \notin \textsc{BsVisited}(i)$, is added to *Open*, and thus there exists a belief state $Bs \notin \textsc{BsVisited}(i)$ and $Bs \in \textsc{BsVisited}(i + 1)$.

(2) The proof relies on the consideration that for all i, $\textsc{BsVisited}(i) \subseteq \mathit{Pow}(\mathcal{S})$. By (1) and by the finiteness of $\mathcal{S}$ (and thus of $\mathit{Pow}(\mathcal{S})$) it follows that i can grow at most to $|\mathit{Pow}(\mathcal{S})| + 1$. □

Lemma A.3. *Let $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G}\rangle$ be a planning problem. Let i be the minimum integer such that* $\textsc{BsVisited}(i) = \textsc{BsVisited}(i + j)$ *for all* $j \geqslant 0$.

(1) *For all* $k \geqslant i$,

$$\textsc{PoolBStates}(\mathit{Open}, k) \supset \textsc{PoolBStates}(\mathit{Open}, k+1).$$

(2) *There exists an index* $k \geqslant i$ *such that*

$$\textsc{PoolBStates}(\mathit{Open}, k) = \emptyset.$$

Proof. (1) Since $\textsc{BsVisited}(i) = \textsc{BsVisited}(i + j)$ for all $j \geqslant 0$ it is easy to show that no new belief states are added to *Open*. At each iteration $k \geqslant i$ an element $\langle Bs \,.\, \pi\rangle$ is removed from *Open* and added to *Closed*. Hence there is a belief state *Bs* such that $Bs \in \textsc{PoolBStates}(\mathit{Open}, k)$ and $Bs \notin \textsc{PoolBStates}(\mathit{Open}, k+1)$.

(2) Since $\textsc{BsVisited}(i) = \textsc{BsVisited}(i + j)$ for all $j \geqslant 0$ then no new belief states are added to *Open* (see (1)). Let $l = |\textsc{PoolBStates}(\mathit{Open}, i)|$ be the cardinality of the open pool at the ith iteration. At each iteration $k \geqslant i$ an element is removed from *Open* and added to *Closed*. Thus, $|\textsc{PoolBStates}(\mathit{Open}, k+1)| = |\textsc{PoolBStates}(\mathit{Open}, k)| - 1$. Hence, at most $l + 1$ iterations after i will be performed, and at $k = i + l + 1$ *Open* is empty. □

Theorem A.1. *Let* $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G}\rangle$ *be a planning problem. Procedure* $\textsc{HeurConformantFwd}(\mathcal{I}, \mathcal{G})$ *always terminates.*

Proof. The only possible cause of non-termination of the HeurConformantFwd algorithm is the *while* loop at lines 5–17. After a bounded number of iterations the set of visited belief states cannot grow anymore (Lemma A.2(2)). Moreover, by Lemma A.2(2) after a further bounded number k of iterations, $\textsc{PoolBStates}(\mathit{Open}, k)$ will be empty, causing the algorithm to terminate. □

Lemma A.4. *Let* $\mathcal{D}$ *be a planning domain*; *let* $S_1 \subseteq S_2 \subseteq \mathcal{S}$ *be two sets of states*; *let* π *be a plan applicable in* S_2. *Then* $\textsc{Exec}[\pi](S_1) \subseteq \textsc{Exec}[\pi](S_2)$.

Proof. Trivial from the definition of $\textsc{Exec}[\pi](S)$. □

Lemma A.5. *Let* $\mathcal{D}$ *be a planning domain*; *let* $\mathcal{I} \subseteq \mathcal{S}$ *be a set of states*; *let* $\langle Bs \,.\, \pi\rangle$ *be a BsP pair such that* $\bot \neq \textsc{Exec}[\pi](\mathcal{I}) \subseteq Bs$; *let* $BsExp = \textsc{FwdExpandBs}(Bs)$. *Then for all* $\langle Bs_i \,.\, \alpha\rangle \in BsExp$ *we have* $\pi' = \pi; \alpha$ *is a solution for the conformant planning problem* $\langle \mathcal{D}, \mathcal{I}, Bs_i\rangle$.

Proof. By definition each $\langle Bs_i \,.\, \alpha\rangle \in BsExp = \textsc{FwdExpandBs}(Bs)$ is such that

$$\textsc{Exec}[\alpha](Bs) \subseteq Bs_i.$$

By hypothesis we have $\textsc{Exec}[\pi](\mathcal{I}) \subseteq Bs$, and applying α to both sides we have:

$$\textsc{Exec}[\alpha]\big(\textsc{Exec}[\pi](\mathcal{I})\big) \subseteq \textsc{Exec}[\alpha](Bs).$$

Since $\textsc{Exec}[\alpha](\textsc{Exec}[\pi](\mathcal{I})) = \textsc{Exec}[\pi; \alpha](\mathcal{I})$ it follows that $\textsc{Exec}[\pi; \alpha](\mathcal{I}) \subseteq Bs_i$. Hence, $\pi' = \pi; \alpha$ is a solution for the conformant planning problem $P = \langle \mathcal{D}, \mathcal{I}, Bs_i\rangle$. □

Lemma A.6. *Let $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ be a planning problem; each BsP pair $\langle Bs . \pi \rangle \in Open$ at the ith iteration of* HEURCONFORMANTFWD$(\mathcal{I}, \mathcal{G})$ *is such that* EXEC$[\pi](\mathcal{I}) \subseteq Bs$.

Proof. Trivial by Lemma A.5. $\square$

Theorem A.2. *Let $P = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ be a planning problem and let π be the value returned by procedure* HEURCONFORMANTFWD$(\mathcal{I}, \mathcal{G})$.

- *If $\pi \neq Fail$, then π is a conformant solution for the planning problem P.*
- *If $\pi = Fail$, instead, then there is no conformant solution for the planning problem P.*

Proof. Let us assume $\pi \neq Fail$. This means that there is an action α_i, a belief state Bs_i and a *BsP* pair $\langle Bs' . \pi' \rangle$ such that $\pi = \pi'; \alpha_i$, $Bs_i \subseteq \mathcal{G}$, $\langle Bs' . \pi' \rangle \in Open$ and

$$\langle Bs_i . \alpha_i \rangle \in \text{FWDEXPANDBS}(Bs').$$

By Lemma A.6 we have that

$$\text{EXEC}[\pi'](\mathcal{I}) \subseteq Bs'.$$

By Lemma A.5 we have that

$$\text{EXEC}[\pi'; \alpha_i](\mathcal{I}) \subseteq Bs_i.$$

Since $Bs_i \subseteq \mathcal{G}$ it follows that

$$\text{EXEC}[\pi'; \alpha_i](\mathcal{I}) \subseteq \mathcal{G}.$$

Let us assume $\pi = Fail$. Then there is an index i such that at the ith iteration of the *while* loop POOLBSTATES$(Open, i) = \emptyset$ and there is no $Bs \in$ BSVISITED(i) such that $Bs \subseteq \mathcal{G}$. Let us assume for contradiction that there is a conformant solution π'. There is an index $j \leqslant i$ such that at the ith iteration of the *while* loop there is an action α_j, a belief state Bs_j and a *BsP* pair $\langle Bs' . \pi' \rangle$ such that $\pi = \pi'; \alpha_j$, $Bs_j \subseteq \mathcal{G}$, $\langle Bs' . \pi' \rangle \in Open$ and

$$\langle Bs_j . \alpha_j \rangle \in \text{FWDEXPANDBS}(Bs').$$

Thus, $Bs_j \in$ BSVISITED$(j) \subseteq$ BSVISITED(i) and $Bs_j \subseteq \mathcal{G}$ which is absurd. $\square$

Appendix B. Proofs of some properties of MAXWDIST and MAXSDIST

Let us consider the MAXWDIST (MAXSDIST) algorithm and, let $WD[i]$ ($SD[i]$) be the set SOLVED$[i]$ at iteration i of the algorithm, *MAXWD* (*MAXSD*) be the number of *while* iterations performed by the algorithm, and let $WD[j] = WD[MAXWD]$ for all $j > MAXWD$ ($SD[j] = SD[MAXSD]$ for all $j > MAXSD$).

Lemma B.1. *Let $\mathcal{D} = \langle \mathcal{X}, \mathcal{V}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ and let Bs be a belief state, the following holds*:

- STRONGPREIMAGE$(Bs) \subseteq$ WEAKPREIMAGE(Bs).
- *If $\mathcal{R}$ is deterministic then* STRONGPREIMAGE$(Bs) =$ WEAKPREIMAGE(Bs).

Proof. It is a direct consequence of the definitions of STRONGPREIMAGE and WEAKPREIMAGE that if $s \in \text{STRONGPREIMAGE}(Bs)$ then $s \in \text{WEAKPREIMAGE}(Bs)$. Let us assume that $\text{WEAKPREIMAGE}(Bs) \subset \text{STRONGPREIMAGE}(Bs)$; then there exists a state $s \in \text{STRONGPREIMAGE}(Bs) \setminus \text{WEAKPREIMAGE}(Bs)$. By definition of STRONGPREIMAGE there is an action α such that $\text{EXEC}[\alpha](s) \subseteq Bs$. Then, by definition of WEAKPREIMAGE s must belong to WEAKPREIMAGE as well, which contradicts the assumption.

The proof is a direct consequence of the definition of WEAKPREIMAGE and STRONGPREIMAGE. □

Lemma B.2. *The following statements hold*:

(1) $\forall i \geqslant 0.\ WD[i] \subseteq WD[i+1]$.
(2) $\forall i \geqslant 0.\ SD[i] \subseteq SD[i+1]$.
(3) $\forall i \geqslant 0.\ SD[i] \subseteq WD[i]$.
(4) *If $\mathcal{R}$ is deterministic then* $\forall i \geqslant 0.\ SD[i] = WD[i]$.

Proof. (1) The proof is by induction on i noticing that

$$WD[i+1] = WD[i] \cup \big(\text{WEAKPREIMAGE}\big(WD[i]\big) \setminus WD[i]\big).$$

(2) The proof is by induction on i noticing that

$$SD[i+1] = SD[i] \cup \big(\text{WEAKPREIMAGE}\big(SD[i]\big) \setminus SD[i]\big).$$

(3) The proof is by induction on i, using Lemma B.1.

(4) Since $\mathcal{R}$ is deterministic by Lemma B.1 and from the definition of $SD[i]$ and of $WD[i]$ it is obvious that $SD[i] = WD[i]$. □

Theorem B.1. *Let $\mathcal{D} = \langle \mathcal{X}, \mathcal{V}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ and let Bs_1 and Bs_2 be two belief states. Then*

(1) $\text{MAXWDIST}(Bs_1, Bs_2) \leqslant \text{MAXSDIST}(Bs_1, Bs_2)$.
(2) *If $\mathcal{R}$ is deterministic, then* $\text{MAXWDIST}(Bs_1, Bs_2) = \text{MAXSDIST}(Bs_1, Bs_2)$.

Proof. (1) From Lemma B.2 it follows that

$$\begin{array}{ccccccccc} WD[0] & \subseteq & WD[1] & \subseteq & \cdots & \subseteq & WD[i] & \subseteq & \cdots \\ \cup\!| & & \cup\!| & & & & \cup\!| & & \\ SD[0] & \subseteq & SD[1] & \subseteq & \cdots & \subseteq & SD[i] & \subseteq & \cdots \end{array}$$

Let $m \doteq \text{MAXWDIST}(Bs_1, Bs_2) \neq \infty$ and $M \doteq \text{MAXSDIST}(Bs_1, Bs_2) \neq \infty$, m being the minimum index such that $Bs_1 \subseteq WD[m]$ and M the minimum index such that $Bs_1 \subseteq SD[M]$. Let us suppose that $M < m$. Then $Bs_1 \subseteq SD[M]$ and $Bs_1 \nsubseteq WD[M]$, so that $SD[M] \nsubseteq WD[M]$, which contradicts Lemma B.2, and thus $M \geqslant m$. If $m = \infty$, then there is no index i such that $Bs_1 \subseteq WD[i]$ and since $\forall i\,.\,SD[i] \subseteq WD[i]$ then it follows that there is no index j such that $Bs_1 \subseteq SD[j]$, and thus $M = \infty$.

(2) From Lemma B.2 it follows that

$$\begin{array}{ccccccccc} WD[0] & \subseteq & WD[1] & \subseteq & \cdots & \subseteq & WD[i] & \subseteq & \cdots \\ \parallel & & \parallel & & & & \parallel & & \\ SD[0] & \subseteq & SD[1] & \subseteq & \cdots & \subseteq & SD[i] & \subseteq & \cdots \end{array}$$

The proof is trivial given that for any i, $WD[i] = SD[i]$, and given the definition of MAXSDIST and MAXWDIST. □

Appendix C. Admissibility of V^*_{dp}

Lemma C.1. *The* STRONGPREIMAGE *operator is monotone, i.e., if $Bs_1 \subseteq Bs_2$, then* STRONGPREIMAGE$(Bs_1) \subseteq$ STRONGPREIMAGE(Bs_2).

Proof. The proof is a direct consequence of the definition of STRONGPREIMAGE. □

Theorem C.1. *Let the optimal cost function h^* for conformant planning be defined as follows*:

$$h^*(Bs, Bs') = \begin{cases} \infty & \textit{if the problem } \langle Bs, Bs' \rangle \textit{ admits no conformant solution}, \\ |\pi| & \textit{where } \pi \textit{ is a conformant solution of optimal length} \\ & \textit{for} \langle Bs, Bs' \rangle. \end{cases}$$

Then, MAXSDIST$(Bs, Bs') \leqslant h^*(Bs, Bs')$.

Proof. First, we prove that the theorem holds when the problem is unsolvable. In fact, $h^*(Bs, Bs') = \infty$, and MAXSDIST(Bs, Bs') either returns a positive integer or ∞. Second, we prove that MAXSDIST$(Bs, \mathcal{G}) \leqslant h^*(Bs, \mathcal{G})$ when MAXSDIST$(Bs, \mathcal{G}) \neq \infty$ and $h^*(Bs, \mathcal{G}) \neq \infty$, by induction on the value of $h^*(Bs, \mathcal{G})$.

Base case. When $h^*(Bs, \mathcal{G}) = 0$, the 0-length plan ε is the only possible optimal one. Therefore, $Bs \subseteq \mathcal{G}$, and MAXSDIST$(Bs, \mathcal{G}) = 0$.

Induction step. The thesis is that if $h^*(Bs, \mathcal{G}) = i + 1$, then MAXSDIST$(Bs, \mathcal{G}) \leqslant i + 1$. The inductive hypothesis is that, for all Bs', if $h^*(Bs', \mathcal{G}) = j \leqslant i$, then MAXSDIST$(Bs', \mathcal{G}) \leqslant j$. We proceed by contradiction. Assume that (a) there exists a Bs such that $h^*(Bs, \mathcal{G}) = i + 1$, and that (b) MAXSDIST$(Bs, \mathcal{G}) > i + 1$. Then, there exists an optimal conformant solution, say $\alpha; \pi$, for $\langle Bs, \mathcal{G} \rangle$, with π of length i. Let Bs' be EXEC$[\alpha](Bs)$. Then, π is an optimal conformant solution for $(Bs', \mathcal{G})$, and $h^*(Bs', \mathcal{G}) = i$. From the inductive hypothesis, MAXSDIST$(Bs', \mathcal{G}) \leqslant i$, that is $Bs' \subseteq$ SOLVED$[i]$. From the monotonicity of STRONGPREIMAGE (Lemma C.1), we obtain STRONGPREIMAGE$(Bs') \subseteq$ STRONGPREIMAGE(SOLVED$[i]$). From the algorithm, STRONGPREIMAGE (SOLVED$[i]$) $\subseteq$ STRONGPREIMAGE(SOLVED$[i]$) $\cup$ SOLVED$[i]$ $=$ SOLVED$[i + 1]$. From $Bs \subseteq$ STRONGPREIMAGE(EXEC$[\alpha](Bs)$) $=$ STRONGPREIMAGE(Bs'), and from the transitivity of $\subseteq$ it follows that $Bs \subseteq$ SOLVED$[i + 1]$, that is MAXSDIST$(Bs, \mathcal{G}) \leqslant i + 1$, which contradicts (b).

Finally, we prove that if MAXSDIST$(Bs, \mathcal{G}) = \infty$, then $h^*(Bs, \mathcal{G}) = \infty$. We prove the converse implication: if $h^*(Bs, \mathcal{G}) \neq \infty$, then MAXSDIST$(Bs, \mathcal{G}) \neq \infty$. We reason by induction on the value of $h^*(Bs, \mathcal{G})$, i.e., on the length of the optimal conformant solution. We show that if $h^*(Bs, \mathcal{G}) = i$, where $i = |\pi|$ and π is an optimal conformant solution for the problem $\langle Bs, \mathcal{G}\rangle$, then $Bs \subseteq \textsc{Solved}[i]$. The thesis follows from the fact that if $Bs \subseteq \textsc{Solved}[i]$ then $\textsc{MaxSDist}(Bs, \mathcal{G}) \leqslant i \neq \infty$.

Base case. $|\pi| = 0$, from which $\pi = \varepsilon$, and $Bs \subseteq \mathcal{G} = \textsc{Solved}[0]$.

Induction step. The thesis is that if $\alpha; \pi$ is an optimal conformant solution such that $|\alpha; \pi| = i + 1$, then $Bs \subseteq \textsc{Solved}[i + 1]$. Then, π is an optimal conformant solution of length i to the problem $\langle Bs', \mathcal{G}\rangle$, where $Bs' = \textsc{Exec}[\alpha](Bs)$. From the inductive hypothesis, it follows that $Bs' \subseteq \textsc{Solved}[i]$. From the monotonicity of $\textsc{StrongPreImage}$, it follows that $\textsc{StrongPreImage}(Bs') \subseteq \textsc{StrongPreImage}(\textsc{Solved}[i])$. A fortiori, $\textsc{StrongPreImage}(Bs') \subseteq \textsc{StrongPreImage}(\textsc{Solved}[i]) \cup \textsc{Solved}[i] = \textsc{Solved}[i + 1]$. From $Bs \subseteq \textsc{StrongPreImage}(\textsc{Exec}[\alpha](Bs)) = \textsc{StrongPreImage}(Bs')$, it follows that $Bs \subseteq \textsc{Solved}[i + 1]$. □

Appendix D. Some properties of V_ψ

Theorem D.1. *If ψ is a necessary knowledge formula for the planning problem $\langle \mathcal{I}, \mathcal{G}\rangle$, then*

- *$V_\psi(Bs, \mathcal{G})$ is admissible, i.e., $V_\psi(Bs, \mathcal{G}) \leqslant h^*(Bs, \mathcal{G})$.*
- *$V_\psi(Bs, \mathcal{G})$ is at least as informed as $V^*_{dp}(Bs, \mathcal{G})$.*

Proof. The proof of the first statement is by cases on the definition of V_ψ.

- If ψ holds in Bs or if ψ is not necessary to reach G, then
$$V_\psi(Bs, \mathcal{G}) = V^*_{dp}(Bs, \mathcal{G}) \leqslant h^*(Bs, \mathcal{G})$$
from the admissibility of V^*_{dp}.
- Otherwise $V_\psi(Bs, \mathcal{G}) = \max(V^*_{dp}(Bs, \mathcal{G}),\ \min_{Bs' \in [\![\psi]\!]_k}(V^*_{dp}(Bs, Bs') + \textsc{MinSDist}(Bs', \mathcal{G})))$.
  - If $V^*_{dp}(Bs, \mathcal{G}) \geqslant \min_{Bs' \in [\![\psi]\!]_k}(V^*_{dp}(Bs, Bs') + \textsc{MinSDist}(Bs', \mathcal{G}))$, then
  $$V_\psi(Bs, \mathcal{G}) = V^*_{dp}(Bs, \mathcal{G}) \leqslant h^*(Bs, \mathcal{G})$$
  from the admissibility of V^*_{dp}.
  - Otherwise, $V_\psi(Bs, \mathcal{G}) = \min_{Bs' \in [\![\psi]\!]_k}(V^*_{dp}(Bs, Bs') + \textsc{MinSDist}(Bs', \mathcal{G}))$. From admissibility of V^*_{dp} and of $\textsc{MinSDist}$ it follows that:
  $$\min_{Bs' \in [\![\psi]\!]_k}\big(V^*_{dp}(Bs, Bs') + \textsc{MinSDist}(Bs', \mathcal{G})\big) \\ \leqslant \min_{Bs' \in [\![\psi]\!]_k}\big(h^*(Bs, Bs') + h^*(Bs', \mathcal{G})\big).$$
  Let us prove $\min_{Bs' \in [\![\psi]\!]_k}(h^*(Bs, Bs') + h^*(Bs', \mathcal{G})) \leqslant h^*(Bs, \mathcal{G})$. Since ψ is a necessary formula, all the solution plans π_i are such that there is a prefix π'_i of π_i such that $\textsc{Exec}[\pi'_i](Bs) \in [\![\psi]\!]_k$. Let us consider an optimal plan $\pi = \pi_1; \pi_2$, such

that $\text{EXEC}[\pi_1](Bs) = Bs' \subseteq [\![\psi]\!]_k$. It is easy to see that π_1 and π_2 are optimal plans for the conformant planning problems $\langle Bs, Bs'\rangle$ and $\langle Bs', \mathcal{G}\rangle$, respectively. Thus, $h^*(Bs, \mathcal{G}) = h^*(Bs, Bs') + h^*(Bs', \mathcal{G}) \geqslant \min_{Bs_j \in [\![\psi]\!]_k}(h^*(Bs, Bs_j) + h^*(Bs_j, \mathcal{G}))$.

The proof of the second statement is a trivial consequence of the definition of V_ψ. □

References

[1] P. Bertoli, A. Cimatti, Improving heuristics for planning as search in belief space, in: Proceedings of Sixth International Conference on AI Planning and Scheduling, AIPS-02, Toulouse, France, 2002, pp. 83–92.

[2] P. Bertoli, A. Cimatti, U. Dal Lago, M. Pistore, Extending PDDL to nondeterminism, limited sensing and iterative conditional, in: Proceedings of ICAPS-03 Workshop on PDDL, Trento, Italy, 2003.

[3] P. Bertoli, A. Cimatti, M. Roveri, Conditional planning under partial observability as heuristic-symbolic search in belief space, in: Proceedings of the Sixth European Conference on Planning, ECP-01, Toledo, Spain, 2001, pp. 379–384.

[4] P. Bertoli, A. Cimatti, M. Roveri, Heuristic search + symbolic model checking = efficient conformant planning, in: B. Nebel (Ed.), Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI-2001, Seattle, WA, Morgan Kaufmann, San Mateo, CA, 2001, pp. 467–472.

[5] P. Bertoli, A. Cimatti, M. Roveri, P. Traverso, Planning in nondeterministic domains under partial observability via symbolic model checking, in: B. Nebel (Ed.), Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI-2001, Seattle, WA, Morgan Kaufmann, San Mateo, CA, 2001, pp. 473–478.

[6] B. Bonet, H. Geffner, Planning with incomplete information as heuristic search in belief space, in: S. Chien, S. Kambhampati, C.A. Knoblock (Eds.), Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling, Breckenridge, CO, AAAI Press, Menlo Park, CA, 2000, pp. 52–61.

[7] K.S. Brace, R.L. Rudell, R.E. Bryant, Efficient implementation of a BDD package, in: Proceedings of the 27th ACM/IEEE Design Automation Conference, Orlando, FL, ACM/IEEE, IEEE Computer Society Press, 1990, pp. 40–45.

[8] R.I. Brafman, J. Hoffmann, Conformant planning via heuristic forward search, in: Proceedings of ICAPS-03 Workshop on Planning under Uncertainty and Incomplete Information, Trento, Italy, 2003.

[9] R.E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Trans. Comput. 35 (8) (1986) 677–691.

[10] R.E. Bryant, On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication, IEEE Trans. Comput. 40 (2) (1991) 205–213.

[11] R.E. Bryant, Symbolic boolean manipulation with ordered binary-decision diagrams, ACM Comput. Surveys 24 (3) (1992) 293–318.

[12] D. Bryce, S. Kambhampati, Conformant planning via heuristic forward search, in: Proceedings of ICAPS-03 Workshop on Planning under Uncertainty and Incomplete Information, Trento, Italy, 2003.

[13] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang, Symbolic model checking: 10^{20} states and beyond, Inform. and Comput. 98 (2) (1992) 142–170.

[14] C. Castellini, E. Giunchiglia, A. Tacchella, SAT-based planning in complex domains: Concurrency, constraints and nondeterminism, Artificial Intelligence 147 (1, 2) (2003) 85–117.

[15] A. Cimatti, E.M. Clarke, F. Giunchiglia, M. Roveri, NUSMV: A new symbolic model checker, Internat. J. Software Tools Technol. Transfer (STTT) 2 (4) (2000).

[16] A. Cimatti, E.M. Clarke, F. Giunchiglia, M. Roveri, NUSMV: A new symbolic model verifier, in: N. Halbwachs, D. Peled (Eds.), Proceedings Eleventh Conference on Computer-Aided Verification, CAV-99, Trento, Italy, in: Lecture Notes in Comput. Sci., vol. 1633, Springer, Berlin, 1999, pp. 495–499.

[17] A. Cimatti, E. Giunchiglia, F. Giunchiglia, P. Traverso, Planning via model checking: A decision procedure for $\mathcal{AR}$, in: S. Steel, R. Alami (Eds.), Proceeding of the Fourth European Conference on Planning, Toulouse, France, in: Lecture Notes in Artif. Intell., vol. 1348, Springer, Berlin, 1997, pp. 130–142. Also: ITC-IRST Technical Report 9705-02, ITC-IRST Trento, Italy.

[18] A. Cimatti, E. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, Integrating BDD-based and SAT-based symbolic model checking, in: A. Armando (Ed.), Proceedings of the Fourth International Workshop on Frontiers of Combining Systems, in: Lecture Notes in Artif. Intell., vol. 2309, Springer, Berlin, 2002, pp. 49–56.

[19] A. Cimatti, M. Pistore, M. Roveri, P. Traverso, Weak, strong, and strong cyclic planning via symbolic model checking, Artificial Intelligence 147 (1, 2) (2003) 35–84.

[20] A. Cimatti, M. Roveri, Conformant planning via symbolic model checking, J. Artificial Intelligence Res. 13 (2000) 305–338.

[21] A. Cimatti, M. Roveri, P. Bertoli, Searching powerset automata by combining explicit-state and symbolic model checking, in: T. Margaria, W. Yi (Eds.), Proceedings of the Seventh International Conference on Tools and Algorithms for the Construction of Systems, in: Lecture Notes in Comput. Sci., vol. 2031, Springer, Berlin, 2001, pp. 313–327.

[22] A. Cimatti, M. Roveri, P. Traverso, Automatic OBDD-based generation of universal plans in non-deterministic domains, in: Proceeding of the Fifteenth National Conference on Artificial Intelligence, AAAI-98, Madison, WI, AAAI Press, 1998. Also IRST-Technical Report 9801-10, Trento, Italy.

[23] A. Cimatti, M. Roveri, P. Traverso, Strong planning in non-deterministic domains via model checking, in: Proceeding of the Fourth International Conference on Artificial Intelligence Planning Systems, AIPS-98, Pittsburgh, PA, AAAI Press, 1998.

[24] E.M. Clarke, J.M. Wing, Formal methods: State of the art and future directions, ACM Comput. Surveys 28 (4) (1996) 626–643.

[25] E.M. Clarke, E.A. Emerson, A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, ACM Trans. Programming Languages Systems 8 (2) (1986) 244–263.

[26] O. Coudert, J.C. Madre, H. Touati, TiGeR Version 1.0 User Guide, Digital Paris Research Lab, 1993.

[27] U. Dal Lago, M. Pistore, P. Traverso, Planning with a language for extended goals, in: Proceedings of the Eighteenth National Conference on Artificial Intelligence, AAAI-02, Edmonton, AB, AAAI Press/The MIT Press, 2002, pp. 447–454.

[28] M. Daniele, P. Traverso, M.Y. Vardi, Strong cyclic planning revisited, in: S. Biundo (Ed.), Proceedings of the Fifth European Conference on Planning, Durham, United Kingdom, in: Lecture Notes in Artif. Intell., vol. 1809, Springer, Berlin, 1999, pp. 35–48.

[29] T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres, A logic programming approach to knowledge-state planning, II: The DLVK system, Artificial Intelligence 144 (1–2) (2003) 157–211.

[30] A. Filzi, F. Pirri, R. Reiter, Open world planning in the situation calculus, in: Proceedings of Seventeenth National Conference on Artificial Intelligence, AAAI-00, Austin, TX, AAAI Press, 2000.

[31] M. Fox, D. Long, The automatic inference of state invariants in TIM, J. Artificial Intelligence Res. 9 (1998) 367–421.

[32] A. Gerevini, L.K. Schubert, Discovering state constraints in DISCOPLAN: Some new results, in: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, Austin, TX, 2000, pp. 761–767.

[33] E. Giunchiglia, G.N. Kartha, V. Lifschitz, Representing action: Indeterminacy and ramifications, Artificial Intelligence 95 (2) (1997) 409–438.

[34] P. Haslum, P. Jonsson, Some results on the complexity of planning with incomplete information, in: Proceedings of the 5th European Conference on Planning, in: Lecture Notes in Artif. Intell., vol. 1809, Springer, Berlin, 1999, pp. 308–318.

[35] J. Kurien, P. Pandurang Nayak, D.E. Smith, Fragment-based conformant planning, in: Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, Toulouse, France, 2002.

[36] D. McDermott, A critique of pure reason, Comput. Intelligence 3 (3) (1987) 151–237.

[37] K.L. McMillan, Symbolic Model Checking, Kluwer Academic, Dordrecht, 1993.

[38] D. Michie, Machine intelligence at Edinburgh, in: On Machine Intelligence, Edinburgh University Press, 1974, pp. 143–155.

[39] N. Nilsson, Principles of Artificial Intelligence, Morgan Kaufmann, Los Altos, CA, 1980.

[40] R. Petrick, F. Bacchus, A knowledge-based approach to planning with incomplete information and sensing, in: Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling, AIPS-02, Toulouse, France, 2002.

[41] M. Pistore, R. Bettin, P. Traverso, Symbolic techniques for planning with extended goals in non-deterministic domains, in: Proceedings of the Sixth European Conference on Planning, ECP'01, Toledo, Spain, 2001, pp. 253–264.

[42] M. Pistore, P. Traverso, Planning as model checking for extended goals in non-deterministic domains, in: B. Nebel (Ed.), Proceedings of the Seventh International Joint Conference on Artificial Intelligence, IJCAI-01, Seattle, WA, Morgan Kaufmann, San Mateo, CA, 2001, pp. 479–486.

[43] J. Rintanen, Constructing conditional plans by a theorem-prover, J. Artificial Intelligence Res. 10 (1999) 323–352.

[44] D.E. Smith, D.S. Weld, Conformant graphplan, in: Proceedings of the 15th National Conference on Artificial Intelligence, AAAI-98 and of the 10th Conference on Innovative Applications of Artificial Intelligence, IAAI-98, Menlo Park, AAAI Press, 1998, pp. 889–896.

[45] F. Somenzi, CUDD: CU Decision Diagram package—release 2.1.2, Department of Electrical and Computer Engineering—University of Colorado at Boulder, April 1997.

[46] B. Yang, R.E. Bryant, D.R. O'Hallaron, A. Biere, O. Coudert, G. Janssen, R.K. Ranjan, F. Somenzi, A performance study of BDD-based model checking, in: Proceedings of the Formal Methods on Computer-Aided Design, 1998, pp. 255–289.