Guided Symbolic Universal Planning[*]

Rune M. Jensen, Manuela M. Veloso and **Randal E. Bryant**
Computer Science Department,Carnegie Mellon University,
5000 Forbes Avenue, Pittsburgh,PA 15213-3891, USA
{runej,mmv,bryant}@cs.cmu.edu

Abstract

Symbolic universal planning based on the reduced Ordered Binary Decision Diagram (OBDD) has been shown to be an efficient approach for planning in non-deterministic domains. To date, however, no guided algorithms exist for synthesizing universal plans. In this paper, we introduce a general approach for guiding universal planning based on an existing method for heuristic symbolic search in deterministic domains. We present three new sound and complete algorithms for best-first strong, strong cyclic, and weak universal planning. Our experimental results show that guiding the search dramatically can reduce both the computation time and the size of the generated plans.

Introduction

Universal planning handles non-determinism in a planning domain by computing a plan that for each reachable domain state associates a set of relevant actions for achieving the goal (Schoppers 1987). A major challenge is to represent universal plans compactly. It has been shown that even for a flexible circuit representation of universal plans in domains with n Boolean state variables, the fraction of randomly chosen universal plans with polynomial size in n decreases exponentially with n (Ginsberg 1989). However, universal plans encountered in practice are normally far from randomly distributed. Often real-world planning problems and their universal plan solutions are regularly structured. A primary objective is therefore to develop efficient techniques for exploiting such structure.

A particularly successful approach uses the reduced Ordered Binary Decision Diagram (OBDD, Bryant 1986) to represent and synthesize universal plans (Daniele, Traverso, & Vardi 1999). A universal plan is generated by a breadth-first backward search from the goal states to the initial state. The state space is implicitly represented and searched using efficient OBDD techniques originally developed for *symbolic model checking* (McMillan 1993). Three major classes of universal plans have been studied: strong, strong cyclic, and weak. An execution of a strong universal plan is guaranteed to reach states covered by the plan and terminate in a goal state after a finite number of steps. An execution of a strong cyclic plan is also guaranteed to reach states covered by the plan and terminate in a goal state, if it terminates. However, a goal state may never be reached due to cycles. An execution of a weak plan may reach states not covered by the plan, it only guarantees that some execution exists that reaches the goal from each state covered by the plan.

A substantial limitation of the current OBDD-based universal planning algorithms is that they are *unguided*. The search frontier is blindly expanded in all directions from the goal states and the final universal plan may cover a large number states that are unreachable from the initial states. In this paper, we show that the approach used by SetA* (Jensen, Bryant, & Veloso 2002) for heuristic OBDD-based search in deterministic domains also can be applied to guide universal planning algorithms. The basic idea is to split the states of the backward search frontier with respect to a *heuristic estimate* of the minimum number of actions needed to reach an initial state and perform a best-first search on the resulting search tree.

We develop guided versions of the three basic OBDD-based universal planning algorithms. Each algorithm is sound and complete. Thus, the algorithms will return a solution if it exists, and otherwise return failure. The algorithms have been empirically validated in two non-deterministic domains: a non-deterministic version of the 8-puzzle and a steel plant of SIDMAR (Fehnker 1999). Our results show that guided OBDD-based universal planning in these domains consistently outperforms the previous blind OBDD-based universal planning algorithms both in terms of the planning time and the size of the OBDD representing the universal plan.

Heuristic symbolic search was first suggested to guide formal verification algorithms toward error states (Yuan *et al.* 1997; Yang & Dill 1996). These algorithms can handle non-deterministic domains, but they only decide whether error states are reachable. Heuristic symbolic search has also been studied for classical search in deterministic domains (Edelkamp & Reffel 1998; Hansen, Zhou, & Feng 2002;

[*]This research is sponsored in part by the Danish Research Agency and the United States Air Force under Grants Nos F30602-00-2-0549 and F30602-98-2-0135. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, the Air Force, or the US Government.

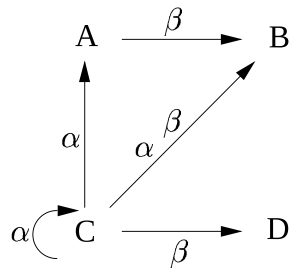
$$\begin{tikzcd}
A \arrow[r, "\beta"] & B \\
C \arrow[u, "\alpha"] \arrow[ur, "\alpha", "\beta"'] \arrow[r, "\beta"'] \arrow[loop left, "\alpha"] & D
\end{tikzcd}$$

Figure 1: An example universal planning domain.

Jensen, Bryant, & Veloso 2002). However, these algorithms do not handle non-deterministic domains. Closer related works are a recent guided symbolic approach for solving Markov decision processes (Feng & Hansen 2002) and a guided controller synthesis approach for timed automata (Hune, Larsen, & Pettersson 2001). However, in both cases, the domain model is more general and the developed techniques are not efficiently applicable to universal planning.

The reminder of the paper is organized as follows. We first describe universal planning and OBDD-based universal planning. We then introduce the search frontier splitting approach and the guided strong, strong cyclic, and weak universal planning algorithms. Finally, we present empirical results and draw conclusions.

Universal Planning

A universal planning domain is a tuple $\langle V, S, Act, \rightarrow \rangle$ where V is a finite set of Boolean state variables, $S \subseteq 2^V$ is a set of states, Act is a finite set of actions, and $\rightarrow \subseteq S \times Act \times S$ is a non-deterministic transition relation. Instead of $(s, a, s') \in \rightarrow$ we write $s \xrightarrow{a} s'$. Further, let $\textsc{Next}(s, a) = \{s' \mid s \xrightarrow{a} s'\}$ denote the set of possible next states of action a applied in state s. A simple domain with two actions α and β and two state variables v_1 and v_2 yielding four states $A = \neg v_1, \neg v_2$, $B = \neg v_1, v_2$, $C = v_1, \neg v_2$ and, $D = v_1, v_2$ is shown in Figure 1. A *universal planning problem* is a tuple $\langle \mathcal{D}, s_0, G \rangle$ where $\mathcal{D}$ is a universal planning domain, $s_0 \in S$ is a single initial state, and $G \subseteq S$ is a set of goal states. A *universal plan* is a set of state-action pairs (SAs) $\mathcal{U} \subseteq \{\langle s, a \rangle \mid \exists s' . s \xrightarrow{a} s'\}$ that associates states with relevant actions to apply in the state for reaching a goal state. The set of states *covered* by a universal plan $\mathcal{U}$ is $\textsc{States}(\mathcal{U}) = \{s \mid \exists a \in Act . \langle s, a \rangle \in \mathcal{U}\}$. An *execution* of $\mathcal{U}$ starting in s_1 is a possibly infinite sequence of states $s_1, s_2, \cdots$ where, for each adjacent pair of states (s, s'), an action a exists such that $\langle s, a \rangle \in \mathcal{U}$ and $s \xrightarrow{a} s'$. An execution terminates in s, iff no action a exists with $\langle s, a \rangle \in \mathcal{U}$. Let $\textsc{Closure}(\mathcal{U}) = \{s' \notin \textsc{States}(\mathcal{U}) \mid \langle s, a \rangle \in \mathcal{U}, s \xrightarrow{a} s'\} \cup G$. We say that $\mathcal{U}$ is *total* iff $\textsc{Closure}(\mathcal{U}) = G$.

We use the definition of strong, strong cyclic, and weak plans introduced by (Daniele, Traverso, & Vardi 1999). This definition relies on the branching time temporal logic CTL (Emerson & Srinivasan 1989). CTL provides universal and existential path quantifiers and temporal operators like "eventually" and "always". CTL formulas are defined starting from the finite set $\mathcal{P}$ of propositions, the Boolean connectives, the temporal connectives $\mathtt{X}$ ("next-time") and $\mathtt{U}$ ("until"), and the path quantifiers $\mathtt{E}$ ("exists") and $\mathtt{A}$ ("for all"). Given a finite set $\mathcal{P}$ of propositions, CTL formulas are inductively defined as follows

- Each element of $\mathcal{P}$ is a formula
- $\neg\psi$, $\psi \vee \phi$, $\mathtt{EX}\,\psi$, $\mathtt{AX}\,\psi$, $\mathtt{E}(\phi\,\mathtt{U}\,\psi)$, and $\mathtt{A}(\phi\,\mathtt{U}\,\psi)$ are formulas if ϕ and ψ are.

CTL semantics is given with respect to *Kripke structures*. A Kripke structure K is a triple $\langle W, T, L \rangle$ where W is a set of worlds, $T \subseteq W \times W$ is a total transition relation, and $L : W \rightarrow 2^{\mathcal{P}}$ is a labeling function. A *path* π in K is a sequence $w_1 w_2 \cdots$ of worlds in W such that, for $i > 0$, $T(w_i, w_{i+1})$. In the following inductive definition of the semantics of CTL, $K, w \models \psi$ denotes that ψ holds in the world w of K

- $K, w_1 \models p$ iff $p \in L(w_1)$, for $p \in \mathcal{P}$
- $K, w_1 \models \neg\psi$ iff $K, w_1 \not\models \psi$
- $K, w_1 \models \psi \vee \phi$ iff $K, w_1 \models \psi$ or $K, w_1 \models \phi$
- $K, w_1 \models \mathtt{EX}\,\psi$ iff there exists a path $w_1 w_2 \cdots$ such that $K, w_2 \models \psi$
- $K, w_1 \models \mathtt{AX}\,\psi$ iff for all paths $w_1 w_2 \cdots$ we have $K, w_2 \models \psi$
- $K, w_1 \models \mathtt{E}(\phi\,\mathtt{U}\,\psi)$ iff there exists a path $w_1 w_2 \cdots$ and $i > 0$ such that $K, w_i \models \psi$ and, for all $0 < j < i$, $K, w_j \models \phi$
- $K, w_1 \models \mathtt{A}(\phi\,\mathtt{U}\,\psi)$ iff for all paths $w_1 w_2 \cdots$ there exists $i > 0$ such that $K, w_i \models \psi$ and, for all $0 < j < i$, $K, w_j \models \phi$.

We introduce the usual abbreviations $\mathtt{AF}\,\psi \equiv \mathtt{A}(\mathbf{true}\,\mathtt{U}\,\psi)$ ($\mathtt{F}$ stands for "future" or "eventually"), $\mathtt{EF}\,\psi \equiv \mathtt{E}(\mathbf{true}\,\mathtt{U}\,\psi)$, $\mathtt{AF}\,\psi \equiv \neg\mathtt{EF}\neg\psi$ ($\mathtt{G}$ stands for "globally" or "always"), and $\mathtt{EG}\,\psi \equiv \neg\mathtt{AF}\neg\psi$.

The executions of a universal plan $\mathcal{U}$ for the planning problem P can be encoded as paths of Kripke structure $K_{\mathcal{U}}^P$ *induced* by $\mathcal{U}$

- $W_{\mathcal{U}}^P = \textsc{States}(\mathcal{U}) \cup \textsc{Closure}(\mathcal{U})$
- $T_{\mathcal{U}}^P(s, s')$ iff $\langle s, a \rangle \in \mathcal{U}$ and $s \xrightarrow{a} s'$, or $s = s'$ and $s \in \textsc{Closure}(\mathcal{U})$
- $L_{\mathcal{U}}^P(s) = s$.

A *strong* plan for a planning problem P is a total universal plan $\mathcal{U}$ such that $s_0 \in W_{\mathcal{U}}^P$ and $K_{\mathcal{U}}^P, s_0 \models \mathtt{AF}\,G$. A *strong cyclic* plan for a planning problem P is a total universal plan $\mathcal{U}$ such that $s_0 \in W_{\mathcal{U}}^P$ and $K_{\mathcal{U}}^P, s_0 \models \mathtt{AGEF}\,G$.[1] A *weak* plan for a planning problem P is a universal plan $\mathcal{U}$ such that $s_0 \in W_{\mathcal{U}}^P$ and $K_{\mathcal{U}}^P, s_0 \models \mathtt{EF}\,G$.

Consider a universal planning problem for our example domain with C as initial state ($s_0 = \mathrm{C}$) and B as a single goal state ($G = \{\mathrm{B}\}$). There does not exist a strong

[1]This definition can be further improved by excluding counter productive transitions (Daniele, Traverso, & Vardi 1999). It is trivial to extend the guided strong cyclic algorithm presented in this paper with a post optimization producing such solutions.

universal plan for this problem since any action applied in C may cause a cycle or reach a dead end. However, $\{\langle \mathrm{C}, \alpha\rangle, \langle \mathrm{A}, \beta\rangle\}$ is a valid strong cyclic universal plan for the problem, and $\{\langle \mathrm{C}, \alpha\rangle, \langle \mathrm{C}, \beta\rangle, \langle \mathrm{A}, \beta\rangle\}$ is a valid weak universal plan.

Universal plans can be synthesized by a backward breadth-first search from the goal states to the initial state. The search algorithm is shown in Figure 2. In each step (l.2-7), a precomponent U_p of the plan is computed from the current states covered by the plan C. If the precomponent is empty, a fixed point of U has been reached that does not cover the initial state and *failure* is returned. Otherwise, the precomponent is added to the universal plan and the states in the precomponent are added to the set of covered states (l.6-7). Strong, Strong cyclic, and weak universal plans only

```
function BLINDUNIVERSALPLANNING($s_0, G$)
1  $U \leftarrow \emptyset$; $C \leftarrow G$
2  while $s_0 \notin C$
3      $U_p \leftarrow$ PRECOMP($C$)
4      if $U_p = \emptyset$ then return failure
5      else
6          $U \leftarrow U \cup U_p$
7          $C \leftarrow C \cup$ STATES($U_p$)
8  return $U$
```

Figure 2: A generic algorithm for synthesizing universal plans.

differ in the definition of the precomponent. Let the preimage of a set of states C be defined by

$$\textsc{PreImg}(C) = \{\langle s, a\rangle \mid \textsc{Next}(s, a) \cap C \neq \emptyset\}.$$

The weak precomponent PRECOMPW(C) is the set of SAs

$$\{\langle s, a\rangle \mid \langle s, a\rangle \in \textsc{PreImg}(C),\ s \notin C\}.$$

The strong precomponent PRECOMPS(C) is the set of SAs

$$\{\langle s, a\rangle \mid \langle s, a\rangle \in \textsc{PreImg}(C) \setminus \textsc{PreImg}(\overline{C}),\ s \notin C\}$$

where $\overline{C}$ denotes the complement of C. The strong cyclic precomponent PRECOMPSC(C) can be generated by iteratively extending a set of candidate SAs (wSA) and pruning it until a fixed point is reached.

```
function PRECOMPSC($C$)
1  $wSA \leftarrow \emptyset$
2  repeat
3      $OldwSA \leftarrow wSA$
4      $wSA \leftarrow$ PREIMG(STATES($wSA$) $\cup C$)
5      $wSA \leftarrow wSA \setminus (C \times Act)$
6      $scSA \leftarrow$ SCPLANAUX($wSA, C$)
7  until $scSA \neq \emptyset \ \vee\ wSA = OldwSA$
8  return $scSA$
```

```
function SCPLANAUX($startSA, C$)
1  $SA \leftarrow startSA$
2  repeat
3      $OldSA \leftarrow SA$
4      $SA \leftarrow$ PRUNEOUTGOING($SA, C$)
5      $SA \leftarrow$ PRUNEUNCONNECTED($SA, C$)
6  until $SA = OldSA$
7  return $SA$
```

```
function PRUNEOUTGOING($SA, C$)
1  $NewSA \leftarrow SA \setminus$ PREIMG($\overline{C \cup \textsc{States}(SA)}$)
2  return $NewSA$
```

```
function PRUNEUNCONNECTED($SA, C$)
1  $NewSA \leftarrow \emptyset$
2  repeat
3      $OldSA \leftarrow NewSA$
4      $NewSA \leftarrow SA \cap$ PREIMG($C \cup$ STATES($NewSA$))
5  until $NewSA = OldSA$
6  return $NewSA$
```

It can be shown that PRECOMPS(C) (PRECOMPSC(C), PRECOMPW(C)) includes a strong (strong cyclic, weak) plan for each state it covers for reaching C. On the other hand, if a strong (strong cyclic, weak) plan exists for reaching C from a state $s \notin C$ then PRECOMPS(C) (PRECOMPSC(C), PRECOMPW(C)) will be non-empty. Thus, BLINDUNIVERSALPLANNING(s_0, G) is *sound* and *complete* for both strong, strong cyclic, and weak universal planning.

OBDD-based Universal Planning

An OBDD is a rooted directed acyclic graph representing a Boolean function. It has one or two terminal nodes 0 and 1 and a set of internal nodes associated with the variables of the function. Each internal node has a $high$ and a low edge. For a particular assignment of the variables, the value of the function is determined by traversing the OBDD from the root node to a terminal node by recursively following the high edge, if the associated variable is true, and the low edge, if the associated variable is false. The value of the function is $true$, if the reached terminal node is 1 and otherwise $false$. Every path in the graph respects a linear ordering of the variables. An OBDD is reduced such that no two distinct nodes are associated with the same variable and have identical low and high successors, and no variable node has identical low and high successors. Due to these reductions, the OBDD of a regularly structured function is often much smaller than its disjunctive normal form (DNF). Another advantage is that the reductions make the OBDD a canonical representation such that equality check can be performed in constant time. In addition, OBDDs are easy to manipulate. Any Boolean operation $f \star g$ on two OBDDs f and g can be carried out in $O(|f||g|)$. For symbolic search, a large number of OBDDs need to be stored and manipulated. In this case, a multi-rooted OBDD can be efficiently used to share structure between the OBDDs. The size of an OBDD is sensitive to the variable ordering. To find an optimal variable ordering is a co-NP-complete problem in itself (Bryant 1986), but as illustrated in Figure 3, a good heuristic for choosing an ordering is to locate related variables near each other (Clarke, Grumberg, & Peled 1999).

By encoding states and actions as bit vectors, OBDDs can be used to represent the *characteristic function* of a set of states and the transition relation. To make this clear, assume

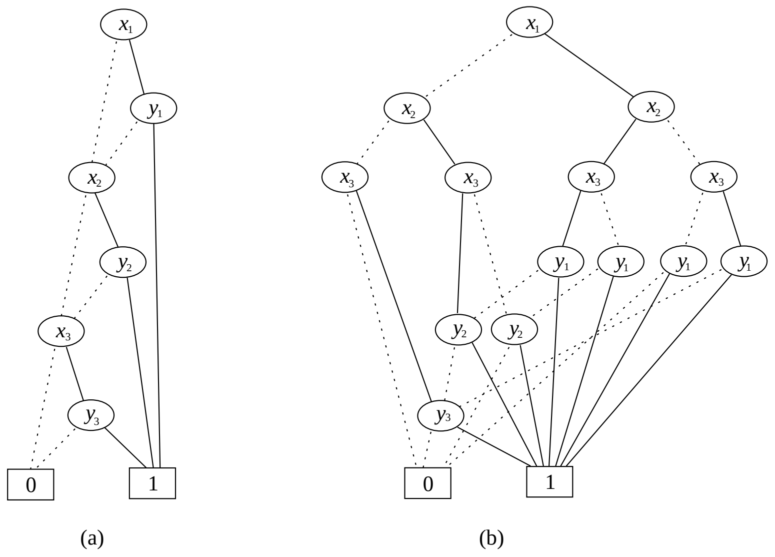

Figure 3: Two OBDDs representing the function $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee (x_3 \wedge y_3)$. The OBDD in (a) only grows linearly with the number of variables in the expression, while the OBDD in (b) grows exponentially. Low and high edges are drawn dashed and solid, respectively.

that we represent the states of the example domain with a two dimensional bit vector $\vec{v} = (v_1, v_2)$ and the actions with a one dimensional bit vector $\vec{a} = (a)$. A possible encoding of the domain could then be as shown in Figure 4. The initial

$$\begin{tikzcd}
(0,0) \arrow[r, "1"] & (0,1) \\
(1,0) \arrow[u, "0"] \arrow[ur, "1", "0"'] \arrow[r, "1"'] \arrow[loop left, "0"] & (1,1)
\end{tikzcd}$$

Figure 4: The example domain encoded with bit vectors.

state is given by the characteristic function $s_0(\vec{v}) = v_1 \wedge \neg v_2$, and similarly we have $G(\vec{v}) = \neg v_1 \wedge v_2$. To encode the transition relation, we construct an OBDD of its characteristic function $T(\vec{v}, \vec{a}, \vec{v}\,')$. The unprimed and primed variables are called current and next state variables, respectively. For the example domain, the DNF of this function is

$$\begin{aligned} T(\vec{v}, \vec{a}, \vec{v}\,') &= \neg v_1 \wedge \neg v_2 \wedge a \wedge \neg v_1' \wedge v_2' \\ &\vee\ v_1 \wedge \neg v_2 \wedge \neg a \wedge \neg v_1' \\ &\vee\ v_1 \wedge \neg v_2 \wedge \neg a \wedge v_1' \wedge \neg v_2' \\ &\vee\ v_1 \wedge \neg v_2 \wedge a \wedge v_2'. \end{aligned}$$

The main idea in OBDD-based universal planning is to perform the search entirely at the OBDD level. Thus, all computations of the algorithm in Figure 2 must be implemented with OBDD operations. This is straight forward for set operations since union, intersection, and complement translates to disjunction, conjunction and negation on the characteristic functions of the sets. In the reminder of this paper, we will not distinguish between set operations and their corresponding OBDD operations. The remaining question is how to compute precomponents symbolically. An inspection of the precomponent defi nitions reveals that we only need to defi ne the symbolic preimage computation. It is given by the expression

$$\textsc{PreImg}(C) = \exists \vec{v}\,' \,.\, T(\vec{v}, \vec{a}, \vec{v}\,') \wedge C(\vec{v}\,').$$

Consider computing the preimage of the goal state of our example domain. We have $C(\vec{v}\,') = \neg v_1' \wedge v_2'$. Thus,

$$\begin{aligned} \textsc{PreImg}(C) &= \exists \vec{v}\,' \,.\, T(\vec{v}, \vec{a}, \vec{v}\,') \wedge \neg v_1' \wedge v_2' \\ &= \neg v_1 \wedge \neg v_2 \wedge a \\ &\vee\ \ v_1 \wedge \neg v_2 \wedge \neg a \\ &\vee\ \ v_1 \wedge \neg v_2 \wedge a. \end{aligned}$$

A common problem when computing the preimage is that the intermediate OBDDs tend to be large compared to the OBDD representing the result. Another problem is that the transition relation may be very large if it is represented by a single OBDD. In symbolic model checking, one of the most successful approaches to solve this problem is *transition relation partitioning*. For universal planning problems, where each transition normally only modifi es a small subset of the state variables, the suitable partitioning technique is *disjunctive partitioning* (Clarke, Grumberg, & Peled 1999). In a disjunctive partitioning, unmodifi ed next state variables are unconstrained in the transition expressions and the abstracted transition expressions are partitioned such that each partition only modifi es a small subset of the variables. Let $\vec{m}_i$ denote the modifi ed next state variables of partition P_i in a partition $P_1, P_2, \cdots, P_n$. The preimage computation may now skip the quantification of unchanged variables and operate on smaller expressions

$$\textsc{preImg}(C) = \bigvee_{i=1}^{n} \Big(\exists \vec{m}_i' \,.\, P_i(\vec{v}, \vec{m}_i') \wedge C(\vec{v})[\vec{m}_i/\vec{m}_i']\Big),$$

where $C(\vec{v})[\vec{m}_i/\vec{m}_i']$ substitutes $\vec{m}_i$ with $\vec{m}_i'$ in $C(\vec{v})$.

Guided Symbolic Universal Planning

In guided symbolic universal planning, the precomponents are divided according to a heuristic function h. For a state s, $h(s)$ estimates the minimum number of actions necessary to reach s from the initial state. Each partition of the precomponent contains states with identical h-value. It has been shown how this can be accomplished by associating each transition with its change δh in the value of the heuristic function (in forward direction) and constructing a disjunctive partitioning where each partition only contains transitions with identical δh (Jensen, Bryant, & Veloso 2002).

Let δh_i denote δh of partition P_i. Further, let C be a set of states with identical h-value h_c. By computing the preimage individually for each partition

$$\textsc{preImg}_i(C) = \exists \vec{m}_i' \,.\, P_i(\vec{v}, \vec{m}_i') \wedge C(\vec{v})[\vec{m}_i/\vec{m}_i'],$$

we split the preimage of C into n components $\textsc{preImg}_1$,$\cdots$,$\textsc{preImg}_n$ where the value of the heuristic function for all states in $\textsc{preImg}_i$ equals $h_c - \delta h_i$. To illustrate this, assume we define a heuristic function for our example problem, as shown in Figure 5. For the example

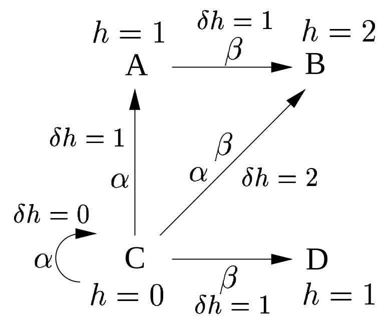

Figure 5: The example problem with a heuristic function estimating the sum of the horizontal and vertical distance to the initial state.

problem, we would need at least 3 partitions since the possible values of δh are 0,1, and 2. Splitting the preimage of the goal state B would give $\textsc{preImg}_1 = \{\langle \mathrm{A}, \beta\rangle\}$ with $h = 1$ and $\textsc{preImg}_2 = \{\langle \mathrm{C}, \alpha\rangle, \langle \mathrm{C}, \beta\rangle\}$ with $h = 0$.

The preimage splitting makes it possible to define a search tree that can be used to explore the state space in a best-first manner. For weak universal planning this approach is straight forward. Each node in the search tree contains a set of SAs with identical h-value. Similar to heuristic deterministic search algorithms, we use a queue to store the unexpanded leaf nodes of the search tree. The queue is prioritized with highest priority given to nodes with lowest h-value. The algorithm is shown in Figure 6.[2] The current

```
function WeakGuided($s_0, G, h_g$)
1  $U \leftarrow \emptyset$; $C \leftarrow G$; $Q \leftarrow emptyQueue$
2  for $i = 1$ to $n$
3      $cSA \leftarrow$ PreCompW($i, C, G$)
4      if $cSA \neq \emptyset$ then
5          $Q \leftarrow$ Insert($Q, \langle cSA, h_g - \delta h_i\rangle$)
6  while $s_0 \notin C$
7      if $|Q| = 0$ then return failure
8      $\langle pSA, h\rangle \leftarrow$ RemoveTop($Q$)
9      $pS \leftarrow$ States($pSA$)
10     $U \leftarrow U \cup pSA$; $C \leftarrow C \cup pS$
11     for $i = 1$ to $n$
12         $cSA \leftarrow$ PreCompW($i, C, pS$)
13         if $cSA \neq \emptyset$ then
14             $Q \leftarrow$ Insert($Q, \langle cSA, h - \delta h_i\rangle$)
15 return $U$
```

Figure 6: The algorithm for guided weak universal planning.

universal plan and the states covered by the plan are stored in U and C respectively (l.1). Initially, the child precomponents of the goal states are computed and inserted in the search queue Q (l.2-5). If a node already exists in the search queue with the same h-value as a child, the SAs of the child are added to the existing node. The weak precomponent is defined by

$$\textsc{PreCompW}(i, C, S) = \textsc{preImg}_i(S) \wedge \neg C(\vec{v}).$$

That is, the subset of the preimage of S for partition i that is not already covered by the plan. In each iteration (l.6-14) the top node of Q is removed and added to the plan (l. 8-10). The precomponents of its children are then computed and inserted in the search queue (l. 11-14). The algorithm terminates with failure if the search queue at some point becomes empty (l.7). Otherwise, the algorithm terminates with success when the initial state is covered by the universal plan (l.15).

The algorithm is *sound* due to the correctness of the weak precomponent computation. The algorithm always terminates since the number of states is finite and a search node only can be inserted in Q if it contains a non-empty set of states not already covered by the plan. Furthermore, it can not terminate with failure if a solution exists, since the only child states that are pruned from a search node are states for which precomponents already have been generated. Thus, the algorithm is also *complete*. An execution example of the algorithm is shown in Figure 7. In this example, we assume that the h-value of all goal states is 5. Initially, two weak precomponents with h-value 4 and 6 are inserted in the search queue. In the first iteration, the weak precomponent with h-value 4 is removed from the top of the queue and added to the current plan. Its children with h-value 3 and 4 are then inserted in the search queue. In the next iteration, the weak precomponent with h-value 3 is added to the plan in a similar way.

[2]To simplify the presentation of the guided universal planning algorithms, we assume that all goal states have identical h-value. It is trivial to generalize the algorithms to accept goal states with

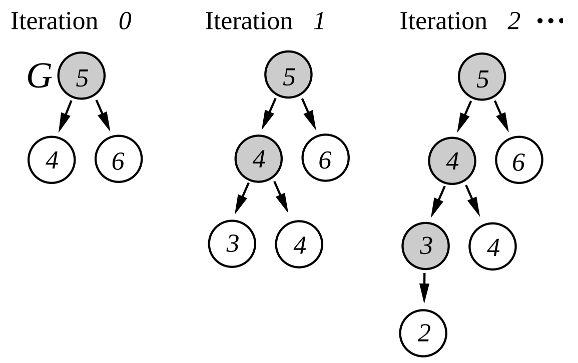

Figure 7: The search tree of an execution example of WEAKGUIDED. The numbers in the search nodes are the h-values of the states. Shaded nodes are covered by the plan.

The strong and strong cyclic guided universal planning algorithms build a new search tree in each iteration. The strong algorithm is particularly simple since it only produces the first level of the search tree. The algorithm is shown in Figure 8. As usual, the current universal plan and the

```
function STRONGGUIDED($s_0, G, h_g$)
1  $U \leftarrow \emptyset$; $\mathbf{C}[h_g] \leftarrow G$; $C \leftarrow G$
2  while $s_0 \notin C$
3      $Q \leftarrow emptyQueue$
4      for $j = 1$ to $|\mathbf{C}|$
5          for $i = 1$ to $n$
6              $cSA \leftarrow$ PRECOMPS($i, C, \mathbf{C}[h_j]$)
7              if $cSA \neq \emptyset$ then
8                  $Q \leftarrow$ INSERT($Q, \langle cSA, h_j - \delta h_i \rangle$)
9      if $|Q| = 0$ then return failure
10     $\langle pSA, h \rangle \leftarrow$ REMOVETOP($Q$)
11     $pS \leftarrow$ STATES($pSA$)
12     $U \leftarrow U \cup pSA$; $\mathbf{C}[h] \leftarrow \mathbf{C}[h] \cup pS$; $C \leftarrow C \cup pS$
13 return $U$
```

Figure 8: The algorithm for guided strong universal planning.

states covered by the plan are stored in U and C. In addition, however, the covered states are divided with respect to their h-values in a map $\mathbf{C}$ with h-values as keys. In each iteration (l.2-12), a queue is generated with strong precomponents of the states in $\mathbf{C}$ (l.3-8). The strong precomponent is defined by

$$\text{PRECOMPS}(i, C, S) = \text{PREIMG}_i(S) \wedge \neg\text{PREIMG}(\overline{C}) \wedge \neg C(\vec{v}).$$

That is, the subset of the preimage of S for partition i that is not already covered by the plan and has no transitions leading outside the covered states. The top node is then removed from Q and added to the plan (l.10-12). The algorithm terminates with failure if the search queue at some point becomes empty (l.9). Otherwise, the algorithm terminates with success when the initial state is covered by the universal plan (l.13).

The algorithm is *sound* due to the correctness of the strong precomponent computation. Similarly to the weak algorithm, the strong algorithm always terminates since the number of states is finite and a search node only can be inserted in Q, if it contains a non-empty set of states not already covered by the plan. The algorithm is also *complete* since it terminates and computes a complete strong search frontier in each iteration (thus Q is only empty if no such frontier exists). An execution example of the algorithm is shown in Figure 9. Again, we assume that the h-value of all goal states is 5. In the first iteration, the strong precomponent of the goal states is divided according to the h-value of the states in the precomponent. We get two partitions with h-value 4 and 6, respectively. The partition with least h-value is added to the plan. In the second iteration, the same procedure is repeated, however this time, the covered states have two distinct h-values.

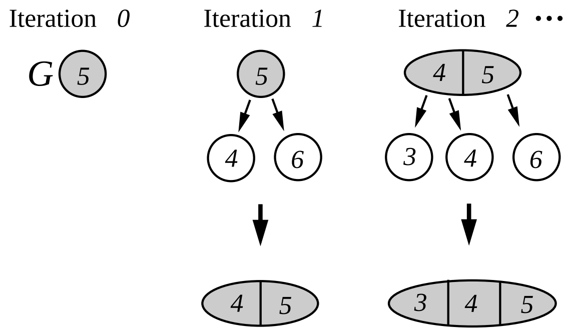

Figure 9: An execution example of STRONGGUIDED. The numbers in the search nodes is the h-value of the states. Shaded nodes are covered by the plan.

The strong cyclic guided universal planning algorithm uses a search tree of preimages to generate a candidate set of SAs for a strong cyclic precomponent. The algorithm is shown in Figure 10. The variables U, C, and $\mathbf{C}$ have their usual meaning. The search queue Q stores the leaf nodes of a search tree of weak precomponents generated from the states in the current plan. Each node is associated with its h-value as usual, however, for this algorithm, we also associate a node with its depth d in the search tree. In addition, the highest priority is given to nodes with smallest sum of h and d. In case of a tie the highest priority is given to the node with smallest depth. When inserting a new node in Q, it will only be merged with an existing node in Q if this node has identical h and d value. In each iteration of the outer loop (l. 2-27), a new search tree is generated and a strong cyclic precomponent is added to the plan. First, the weak precomponents of the current plan are added to Q (l.4-8). These are all at depth 1 in the tree. Then, the candidate SAs wSA for the strong cyclic precomponent and auxiliary variables are initialized (l.9). The inner loop (l.10-23) performs a guided version of the expansion and pruning of wSA

different h-values.

```
function StrongCyclicGuided($s_0, G, h_g$)
1  $U \leftarrow \emptyset$; $\mathbf{C}[h_g] \leftarrow G$; $C \leftarrow G$
2  while $s_0 \notin C$
3     $Q \leftarrow emptyQueue$
4     for $j = 1$ to $|\mathbf{C}|$
5        for $i = 1$ to $n$
6           $cSA \leftarrow$ PreCompW($i, C, \mathbf{C}[h_j]$)
7           if $cSA \neq \emptyset$ then
8              $Q \leftarrow$ Insert($Q, \langle cSA, 1, h_j - \delta h_i \rangle$)
9     $wSA \leftarrow \emptyset$; $\mathbf{wS} \leftarrow emptyMap$
10    repeat
11       if $|Q| = 0$ then return failure
12       $\langle pSA, d, h \rangle \leftarrow$ RemoveTop($Q$)
13       $pSA \leftarrow pSA \setminus wSA$
14       if $pSA \neq \emptyset$ then
15          $pS \leftarrow$ States($pSA$)
16          $\mathbf{wS}[h] \leftarrow \mathbf{wS}[h] \cup pS$
17          for $i = 1$ to $n$
18             $cSA \leftarrow$ PreCompW($i, C, pS$)
19             if $cSA \neq \emptyset$ then
20                $Q \leftarrow$ Insert($Q, \langle cSA, d+1, h - \delta h_i \rangle$)
21          $wSA \leftarrow wSA \cup pSA$
22          $scSA \leftarrow$ SCPlanAux($wSA, C$)
23    until $scSA \neq \emptyset$
24    $scS \leftarrow$ States($scSA$)
25    $U \leftarrow U \cup scSA$; $C \leftarrow C \cup scS$
26    for $k = 1$ to $|\mathbf{wS}|$
27       $\mathbf{C}[h_k] \leftarrow \mathbf{C}[h_k] \cup (\mathbf{wS}[h_k] \cap scS)$
28 return $U$
```

Figure 10: The algorithm for guided strong cyclic universal planning.

carried out by PreCompSC(C). When a non-empty strong cyclic precomponent $scSA$ is found, it is added to the current plan (l. 24-27). The algorithm terminates with success when the plan includes the initial state.

The purpose of taking the depth of search nodes into account when expanding the search tree is to avoid a deep but too narrow candidate set for the strong cyclic precomponent. The tie breaking rule of the search queue further ensures that nodes at depth i are expanded before nodes at depth $i + 1$ with same priority.

The algorithm is *sound* due to the correctness of the approach for computing the strong cyclic precomponent. Given that the inner loop terminates the outer loop will also terminate since the number of states covered by the plan grows in each iteration such that the initial state eventually will be covered. The inner loop terminates, since the number of SAs covered by the tree of weak precomponents grows in each iteration and there only is a finite number of SAs. If the algorithm terminates with failure, the inner loop has generated the largest possible set of states for which a weak plan exists for reaching the states in the current plan. Since a strong cyclic precomponent must be a subset of this set of states, the algorithm can only terminate with failure if no strong cyclic plan exists. Thus, the algorithm is also *complete*. An execution example of the strong cyclic algorithm is shown in Figure 11. As usual, we assume that the h-value of all goal states is 5. Since these states form the root of our search tree their depth is 0. In the first iteration, the candidate set for the strong cyclic precomponent is grown by iteratively adding nodes from a tree of weak precomponents. The nodes added has the current least sum of d and h. Thus, we first add the node $(1, 4)$ and then (because of the tie breaking rule) $(1, 5)$. It is assumed that the SAs of these two nodes include a non-empty strong cyclic precomponent which is then added to the plan. The algorithm continues in a similar fashion in the second iteration. The only difference is that the set of covered states now is partitioned into two sets of states.

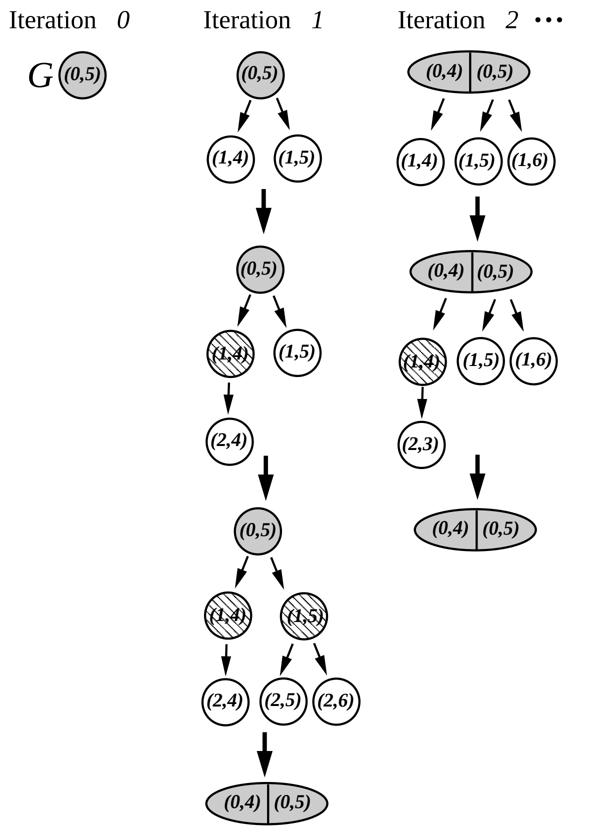

Figure 11: An execution example of StrongCyclicGuided showing the iterations of the outer loop. The numbers in the search nodes is the d and h-value of the states. Shaded nodes are covered by the plan. Striped nodes form the strong cyclic precomponent candidate.

Experimental Results

The performance of the guided universal planning algorithms have been evaluated in two non-deterministic domains. The first of these domains is a non-deterministic version of the well known 8-puzzle problem. The 8-puzzle consists of a 3×3 board with 8 numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space. The goal is to reach a configuration where the tiles are ordered $1, 2, 3$; $4, 5, 6$; and $7, 8$ in row 1,2, and 3, respectively. To make the domain non-deterministic, we assume that up and down moves of the blank space may move left and right as well, as shown in Figure 12. However, to ensure that the sum of Manhattan distances of the tiles to their initial position remains an underestimating heuristic, we assume that a single move at most will cause this heuristic to be reduced by one. Left and right moves are deterministic in order to

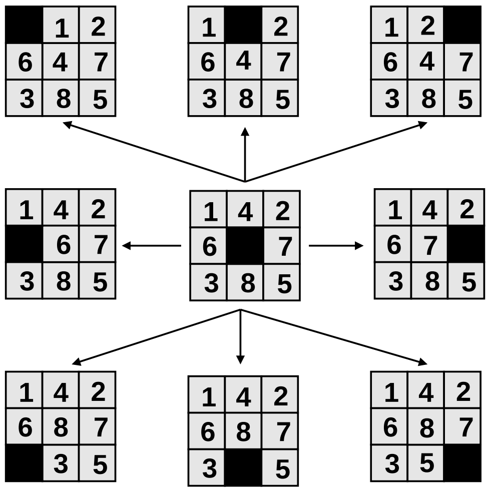

Figure 12: An example move of the blank space and its possible outcomes.

ensure that a strong universal plan exists for any reachable initial state. The second domain is an abstract model of a real-world steel producing plant of SIDMAR in Ghent, Belgium used as an Esprit case study (Fehnker 1999). The layout of the steel plant is shown in Figure 13. The goal is to

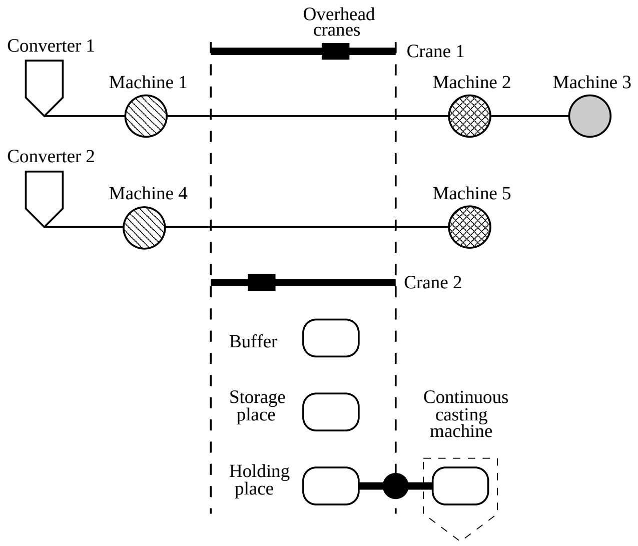

Figure 13: Layout of the SIDMAR steel plant.

cast steel of different qualities. Pig iron is poured portionwise in ladles by the two converter vessels. The ladles can move autonomously on the two east-west tracks. However, two ladles can not pass each other and there can at most be one ladle between machines. Ladles are moved in the north-south direction by the two overhead cranes. The pig iron must be treated differently to obtain steel of different qualities. Before empty ladles are moved to the storage place the steel is cast by the continuous casting machine. A ladle can only leave the casting machine if there already is a filled ladle at the holding place. The actions of machine 1,2,4, and 5 are non-deterministic. They may either cause the steel in the ladles to be treated or the machine to break. To ensure that a strong universal plan exists, actions have been added to fix failed machines.

The guided and blind universal planning algorithms described in this paper have been implemented in C++/STL using the BuDDy OBDD package (Lind-Nielsen 1999). All experiments are carried out on a Linux RedHat 7.1 PC with kernel 2.4.16, 500 MHz Pentium III CPU, 512 KB L2 cache and 512 MB RAM. For the non-deterministic 8-puzzle, we consider problems where the minimum length of a path from the initial state to the goal state grows linearly from 8 to 23. The sum of Manhattan distances is used as heuristic function. For the SIDMAR problem, we consider producing steel from two ladles. They both need an initial treatment on machine 1 or 4 and 2 or 5. One of the ladles in addition need a treatment on machine 3 and a final treatment on machine 2 or 5 before being cast. Non-determinism is caused by machines failures. We consider 6 problems where the goal states correspond to situations with growing distances from the initial state during the production of these two ladles. The number of completed treatments is used as heuristic function. Notice that this heuristic for the SIDMAR problem is weaker than the sum of Manhattan distances for the 8-puzzle. The reason is that it underestimates the distance to the initial state more relative to the sum of Manhattan distances.

For the non-deterministic 8-puzzle experiment, the OBDD package was initialized with 4000000 free OBDD nodes in its node table and 700000 free OBDD nodes in its operator cache. Memory allocation and transition relation construction took 1.56 and 1.34 seconds respectively for all experiments. For the SIDMAR experiment the OBDD package was initialized with 8000000 free OBDD nodes in its node table and 700000 free OBDD nodes in its operator cache. Memory allocation and transition relation construction took 2.34 and 0.22 seconds respectively for all experiments. The results of the non-deterministic 8-puzzle and SIDMAR experiment are shown in Figure 14. For the 8-puzzle, each datapoint is the average of 3 computational results. The results consistently show that guided universal planning may reduce the computation time and the size of the produced plans dramatically. This may be somewhat surprising for guided strong and strong cyclic universal planning. These algorithms apparently repeat a large number of computations. The previous results of such recomputations, however, will often be stored in the operator cache of the OBDD package and may therefore not cause a significant computation overhead. The problem for the blind universal

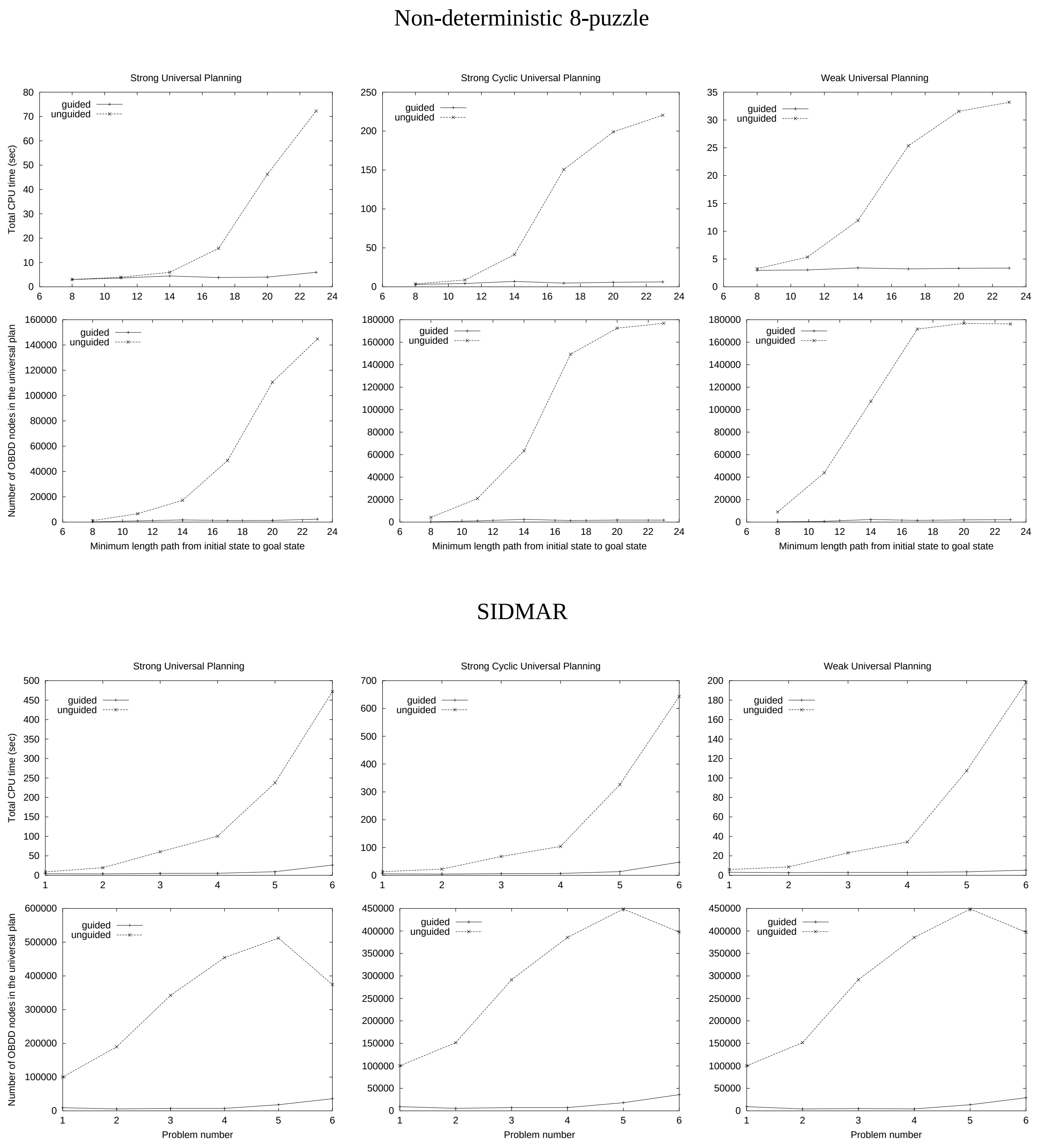

Figure 14: Results of the non-deterministic 8-puzzle and SIDMAR experiments.

planning algorithms is that the blind precomponent grows fast with the search depth. In addition, the plan returned by these algorithms may cover states that are never reached by an execution of the plan from the initial state.

Conclusion

In this paper, we have shown how the approach of SetA* for guiding OBDD-based deterministic search can be applied to universal planning as well. We have developed sound and complete guided versions of the three main symbolic universal planning algorithms and studied their performance in a non-deterministic version of the 8-puzzle and a real-world domain of a steel producing plant. The results consistently show a dramatic performance improvement compared to the previous blind algorithms both in terms of the total CPU time and the size of the produced plans.

Other versions of the algorithms are obvious to consider. A best-first search approach may be too aggressive if the non-determinism in the domain is less local or if multiple initial states are considered. Another issue is how to control the quality of the produced plans. When using breadth-first search to produce strong plans for instance, the solutions have minimum worst-case execution length. Best-first search does not provide similar guarantees.

References

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

Clarke, E.; Grumberg, O.; and Peled, D. 1999. *Model Checking*. MIT Press.

Daniele, M.; Traverso, P.; and Vardi, M. Y. 1999. Strong cyclic planning revisited. In *Proceedings of the Fifth European Conference on Planning (ECP'99)*, 35–48. Springer-Verlag.

Edelkamp, S., and Reffel, F. 1998. OBDDs in heuristic search. In *Proceedings of the 22nd Annual German Conference on Advances in Artificial Intelligence (KI-98)*, 81–92. Springer.

Emerson, E. A., and Srinivasan, J. 1989. Branching time temporal logic. In Bakker, J. W.; Roever, W. P.; and Rozenberg, G., eds., *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*. Berlin: Springer. 123–172.

Fehnker, A. 1999. Scheduling a steel plant with timed automata. In *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*. IEEE Computer Society Press.

Feng, Z., and Hansen, E. 2002. Symbolic LAO* search for factored markov decision processes. In *Proceedings of the AIPS-02 Workshop on Planning via Model Checking*, 49–53.

Ginsberg, M. L. 1989. Universal planning: An (almost) universal bad idea. *AI Magazine* 10(4):40–44.

Hansen, E.; Zhou, R.; and Feng, Z. 2002. Symbolic heuristic search using decision diagrams. In *Symposium on Abstraction, Reformulation and Approximation SARA'02*.

Hune, T.; Larsen, K. G.; and Pettersson, P. 2001. Guided synthesis of control programs using UPPALL. *Nordic Journal of Computing* 8(1):43–64.

Jensen, R. M.; Bryant, R. E.; and Veloso, M. M. 2002. SetA*: An efficient BDD-based heuristic search algorithm. In *Proceedings of 18th National Conference on Artificial Intelligence (AAAI'02)*, 668–673.

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

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

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

Yang, C. H., and Dill, D. L. 1996. Spotlight: Best-first search of FSM state space. IEEE International High Level Design Validation and Test Workshop.

Yuan, J.; Shen, J.; Abraham, J.; and Aziz, A. 1997. Formal and informal verification. In *Conference on Computer Aided Verification (CAV'97)*, 376–387.