LIFT-UP: Lifted First-Order Planning Under Uncertainty

Steffen Hölldobler and Olga Skvortsova
International Center for Computational Logic
Technische Universität Dresden, Dresden, Germany
{sh,skvortsova}@iccl.tu-dresden.de

Abstract

We present a new approach for solving first-order Markov decision processes combining first-order state abstraction and heuristic search. In contrast to existing systems, which start with propositionalizing the decision process and then perform state abstraction on its propositionalized version we apply state abstraction directly on the decision process avoiding propositionalization. Secondly, guided by an admissible heuristic, the search is restricted to those states that are reachable from the initial state. We demonstrate the usefulness of the above techniques for solving first-order Markov decision processes within a domain dependent system called FluCaP which participated in the probabilistic track of the 2004 International Planning Competition. Working toward a domain independent implementation we present novel approaches to θ-subsumption involving literal and object contexts.

1 Introduction

We are interested in solving probabilistic planning problems, i. e. planning problems, where the execution of an action leads to the desired effects only with a certain probability. For such problems, Markov decision processes have been adopted as a representational and computational model in much recent work, e.g., by [BBS95]. They are usually solved using the so-called dynamic programming principle [BDH99] employing a value iteration algorithm. Classical dynamic programming algorithms explicitly enumerate the state space and are thus exponenetial. In recent years several methods have been developed which avoid an explicit enuration of the state space. The most prominent are state abstraction [BDH99], heuristic search (e. g. [BBS95, DKKN95]) and a combination of both as used, for example, in symbolic LAO* [FH02].

A common feature of these approaches is that a Markov decision process is propositionalized before state abstraction techniques and heuristic search algorithms are applied within a value iteration algorithm. Unfortunately, the propositionalization step itself may increase the problem significantly. To overcome this problem, it was first proposed in [BRP01] to solve first-order Markov decision processes by applying a first-order value iteration algorithm and first-order state abstraction techniques. Whereas this symbolic dynamic programming approach was rooted in a version of the Situation Calculus [Rei91], we have reformulated and extended these ideas in a variant of the fluent calculus [HS04]. In this system, which is now called LIFT-UP, lifted first-order planning under uncertainty can be performed.

In the LIFT-UP system, states and actions are expressed in the language of the fluent calculus [HS90], which is slightly extended to handle probabilities. In addition, value functions and policies are represented by constructing first-order formulas which partition the state space into clusters, referred to as *abstract states*. Then, value iteration can be performed on top of these clusters, obviating the need for explicit state enumeration. This allows the solution of first-order Markov decision processes without requiring explicit state enumeration or propositionalization. In addition, heuristics are used to guide the search and normalization techniques are applied to eliminate redundant states. The LIFT-UP approach can thus be viewed as a first-order generalization of symbolic LAO* or, alternatively, as symbolic dynamic programming enhanced by heuristic search and state space normalization.

To evaluate the LIFT-UP system we have developed a domain-dependent implementation called FluCaP. It can solve probabilistic blocksword problems as they appeared, for example, in the colored blocksworld domain of the 2004 International Planning Competition. FluCaP is quite successful and outperforming other systems on truly first-order problems. On the other hand and working towards a domain-independent implementation we have studied θ-subsumption algorithms. θ-subsumption problems arise at various places in the LIFT-UP system: The normalization process requires to check whether one abstract state subsumes another one; the check whether an action is applicable to some abstract state and the computation of set of the successor or predecessor states also requires subsumption. One should observe that the latter application requires to compute a complete set of substitutions.

In this paper we give an overview of the LIFT-UP approach.

2 First-order Markov Decision Processes

A *Markov decision process*, is a tuple $(\mathcal{Z}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{C})$, where $\mathcal{Z}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, and $\mathcal{P} : \mathcal{Z} \times \mathcal{Z} \times \mathcal{A} \rightarrow [0, 1]$, written $\mathcal{P}(z'|z, a)$, specifies transition probabilities. In particular, $\mathcal{P}(z'|z, a)$ denotes the probability of ending up at state z' given that the agent was in state z and action a was executed. $\mathcal{R} : \mathcal{Z} \rightarrow \mathbb{R}$ is a real-valued reward function associating with each state z its immediate utility $\mathcal{R}(z)$. $\mathcal{C} : \mathcal{A} \rightarrow \mathbb{R}$ is a real-valued cost function associating a cost $\mathcal{C}(a)$ with each action a. A *sequential decision problem* consists of a Markov decision process and is the problem of finding a policy $\pi : \mathcal{Z} \rightarrow \mathcal{A}$ that maximizes the total expected discounted reward received when executing the policy π over an infinite (or indefinite) horizon. A Markov decision process is said to be *first-order* if the expressions used to define $\mathcal{Z}$, $\mathcal{A}$ and $\mathcal{P}$ are first-order.

The *value* $V_\pi(z)$ of a state z with respect to the policy π is defined as

$$V_\pi(z) = \mathcal{R}(z) + \mathcal{C}(\pi(z)) + \gamma \sum_{z' \in \mathcal{Z}} \mathcal{P}(z'|z, \pi(z)) V_\pi(z'),$$

where $0 \leq \gamma \leq 1$ is a discount factor. We take γ equal to 1 for indefinite-horizon problems only, i. e. when a goal is reached the system enters an absorbing state in which no further rewards or costs are accrued. A value function V is set to be *optimal* if it satisfies

$$\mathcal{R}(z) + \max_{a \in \mathcal{A}} \{\mathcal{C}(a) + \gamma \sum_{z' \in \mathcal{Z}} \mathcal{P}(z'|z,a) V^*(z')\} ,$$

for each $z \in \mathcal{Z}$; in this case the value function is usually denoted by $V^*(z)$. The optimal policy is extracted from the optimal value function.

We assume that planning problems meet the following requirements:

1. Each problem has a goal statement, identifying a set of absorbing goal states.

2. A positive reward is associated with each action ending in a goal state; otherwise it is 0.

3. A cost is associated with each action.

4. A "done" action is available in all states.

The "done" action can be used to end any further accumulation of reward. Together, these conditions ensure that an MDP model of a planning problem is a positive bounded model as described by [Put94]. Such planning problems are also often called stochastic shortest path problems.

3 Probabilistic Fluent Calculus

States, actions, transition probabilities, cost and reward function are specified in a probabilistic and sorted extension of the fluent calculus [HS90, Thi98].

Fluents and States Let Σ denote a set of function symbols containing the binary function symbol $\circ$ and the nullary function symbol 1. $\circ$ is an AC1-symbol with 1 as unit element. Let $\Sigma^- = \Sigma \setminus \{\circ, 1\}$. Non-variable Σ^--terms are called *fluents*. Let $f(t_1, \ldots, t_n)$ be a fluent. The terms t_i, $1 \le i \le n$ are called *objects*. A *state* is a finite set of ground fluents. Let $\mathcal{D}$ be the set of all states.

Fluent Terms and Abstract States *Fluent terms* are defined inductively as follows: 1 is a fluent term; each fluent is a fluent term; if G_1 and G_2 are fluent terms, then so is $G_1 \circ G_2$. Let $\mathcal{F}$ be the set of all fluent terms. We assume that each fluent term obeys the *singularity condition*: each fluent may occur at most once in a fluent term. Because of the latter, there is a bijection $\cdot^M$ between ground fluent terms and states. Some care must be taken when instantiating a non-ground fluent term F by a substitution θ because $F\theta$ may violate the singularity condition. A substitution θ is *allowed* for fluent term F if $F\theta$ meets the singularity condition.

Abstract states are expressions of the form F or $F \circ X$, where F is a fluent term and X is a variable of sort fluent term. Let $\mathcal{S}$ denote the set of abstract states. Abstract states denote sets of states as defined by the mapping $\cdot^I : \mathcal{S} \to 2^{\mathcal{D}}$: Let Z be an abstract state. Then

$$[Z]^I = \{[Z\theta]^M \mid \theta \text{ is an allowed grounding substitution for } Z\}.$$

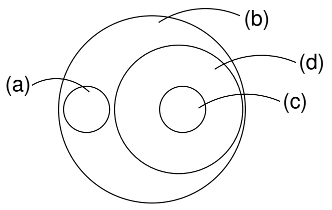

Figure 1: The interpretations of the abstract states (a) $Z_1 = on(X_1, a) \circ on(a, table)$, (b) $Z_2 = on(X_2, a) \circ on(a, table) \circ Y_2$, (c) $Z_3 = on(X_3, a) \circ on(a, table) \circ clear(X_3)$ and (d) $Z_4 = on(X_4, a) \circ on(a, table) \circ clear(X_4) \circ Y_4$, where a is an object denoting a block, *table* is an object denoting a table, X_1, X_2, X_3 and X_4 are variables of sort object, Y_2 and Y_4 are variables of sort fluent term, $on(X_i, a)$, $i = 1 \dots 4$, is a fluent denoting that some block X_i is on a and $clear(X_i)$, $i = 3, 4$, is a fluent denoting that block X_i is clear.

This is illustrated in Figure 1. In other words, abstract states are characterized by means of positive conditions that must hold in each ground instance thereof and, thus, they represent clusters of states. In this way, abstract states embody a form of state space abstraction, which is called *first-order state abstraction*.

As a running example, we consider problems taken from the colored Blocksworld scenario, which is an extension of the classical Blocksworld scenario in the sense that along with the unique identifier, each block is now assigned a specific color. Thus, a state description provides an arrangement of colors instead of an arrangement of blocks. For example, a state Z defined as a fluent term:

$$\begin{aligned} Z = red(X_0) \circ green(X_1) \circ blue(X_2) \circ red(X_3) \circ red(X_4) \circ \\ red(X_5) \circ green(X_6) \circ green(X_7) \circ Tower(X_0, \dots, X_7)\ , \end{aligned}$$

specifies a tower that is comprised of eigth colored blocks.

Subsumption Let Z_1 and Z_2 be abstract states. Then Z_1 is *subsumed* by Z_2, in symbols $Z_1 \sqsubseteq Z_2$, if there exists an allowed substitution θ such that $Z_2\theta =_{AC1} Z_1$. Intuitively, Z_1 is subsumed by Z_2 iff $Z_1^I \subseteq Z_2^I$. In the LIFT-UP system we are often concerned with the problem of finding a complete set of allowed substitutions solving the AC1-matching problem $Z_2\theta =_{AC1} Z_1$.

For example, consider the abstract states mentioned in Figure 1. Then, $Z_1 \sqsubseteq Z_2$ with $\theta = \{X_2 \mapsto X_1, Y_2 \mapsto 1\}$, $Z_3 \sqsubseteq Z_2$ with $\theta = \{X_2 \mapsto X_3, Y_2 \mapsto clear(X_3)\}$. However, $Z_1 \not\sqsubseteq Z_3$ and $Z_3 \not\sqsubseteq Z_1$.

Actions Let Σ_a denote a set of action names, where $\Sigma_a \cap \Sigma = \emptyset$. An *action space* $\mathcal{A}$ is a set of expressions of the form $(a(X_1, \dots, X_n), C, E)$, where $a \in \Sigma_a$, X_i, $1 \leq i \leq n$, are variables or constants, $C \in \mathcal{F}$ called *precondition* and $E \in \mathcal{F}$ called *effect* of the *action* $a(X_1, \dots, X_n)$. E.g., a pickup-action in the blockworld can be specified by

$$(pickup\,(X, Y),\ on(X, Y) \circ clear(X) \circ empty,\ holding(X) \circ clear(Y)),$$

where *empty* denotes that the robot arm is empty and $holding(X)$ that the block X is in the gripper. For simplicity, we will often supress parameters, preconditions and effects of an action $(a(X_1, \dots, X_n), C, E)$ and refer to it as a instead.

Nature's Choice and Probabilities In analogy to the approach in [BRP01] stochastic actions are decomposed into deterministic primitives under nature's control, referred to as *nature's choices*. It can be modelled with the help of a binary relation symbol *choice* as follows: Consider the action $\mathit{pickup}(X,Y)$:

$$\mathit{choice}(\mathit{pickup}(X,Y),a) \leftrightarrow \ (a = \mathit{pickupS}(X,Y) \vee a = \mathit{pickupF}(X,Y)),$$

where *pickupS* and *pickupF* define two nature's choices for action *pickup*, viz., that it succeeds or fails. For simplicity, we denote the set of nature's choices of an action a as $\mathit{Ch}(a) := \{a_j | \mathit{choice}(a,a_j)\}$.

For each of nature's choices a_j associated with an action a we define the probability $\mathit{prob}(a_j,a,Z)$ denoting the probability with which one of nature's choices a_j is chosen in a state Z. For example,

$$\mathit{prob}(\mathit{pickupS}(X,Y),\mathit{pickup}(X,Y),Z) = .75$$

states that the probability for the successful execution of the *pickup* action in state Z is .75. We require that for each action the probabilities of all its nature's choices sum up to 1.

Rewards and Costs Reward and cost functions are defined for abstract states using the unary relation symbols *reward* and *cost*. For example, we might want to give a reward of 500 to all states in which some block X is on block a and 0, otherwise:

$$\begin{array}{lcl} \mathit{reward}(Z) = 500 & \leftrightarrow & Z \sqsubseteq (on(X,a),\emptyset), \\ \mathit{reward}(Z) = 0 & \leftrightarrow & Z \not\sqsubseteq (on(X,a),\emptyset). \end{array}$$

In other words, the state space is divided into two abstract states depending on whether or not, a block X is on block a. Likewise, value functions can be specified with respect to the abstract states only. Action costs can be analogously defined. E. g., with

$$\mathit{cost}(\mathit{pickup}(X,Y)) = 3$$

the execution of the *pickup*-action is penalized with 3.

Forward and Backward Application of Actions An action $(a(X_1,\ldots,X_n),C,E)$ is *forward applicable with* θ to an abstract state $Z \in \mathcal{S}$, denoted as $\mathit{forward}(Z,a,\theta)$, if $(C \circ U)\theta =_{AC1} Z$, where U is a new variable of sort fluent term and θ is an allowed substitution. If applicable, then the action *progresses to* or *yields* the state $(E \circ U)\theta$. In this case, $(E \circ U)\theta$ is called *successor state* of Z and denoted as $\mathit{succ}(Z,a,\theta)$.

An action $(a(X_1,\ldots,X_n),C,E)$ is *backward applicable with* θ to an abstract state $Z \in \mathcal{S}$, denoted as $\mathit{backward}(Z,a,\theta)$, if $(E \circ U)\theta =_{AC1} Z$, where U is a new variable of sort fluent term and θ is an allowed substitution. If applicable, then the action *regresses to* the state $(C \circ U)\theta$. In this case, $(C \circ U)\theta$ is called *predecessor state* of Z and denoted as $\mathit{pred}(Z,a,\theta)$.

One should observe that the AC1-matching problems involved in the application of actions are subsumption problems, viz. $Z \sqsubseteq (C \circ U)$ and $Z \sqsubseteq (E \circ U)$. Moreover, in order to determine all possible successor or predecessor states of some state with respect to some action we have to compute complete sets of allowed substitutions solving the corresponding subsumption problems.

```
$policyExpansion(\pi, \mathcal{S}^0, G)$
 $E := F := \emptyset$
 $from := \mathcal{S}^0$
 repeat
  $to := \bigcup_{Z \in from} \bigcup_{a_j \in Ch(a)} \{succ(Z, a_j, \theta)\}$,
  where $(a, \theta) := \pi(Z)$
  $F := F \cup (to - G)$
  $E := E \cup from$
  $from := to \cap G - E$
 until $(from = \emptyset)$
 $E := E \cup F$
 $G := G \cup F$
 return $(E, F, G)$

FOVI$(E, \mathcal{A}, prob, reward, cost, \gamma, V)$
 repeat
  $V' := V$
  loop for each $Z \in E$
   loop for each $a \in \mathcal{A}$
    loop for each $\theta$ such that $forward(Z, a, \theta)$
     $Q(Z, a, \theta) := reward(Z) + cost(a) +$
          $\gamma \sum_{a_j \in Ch(a)} prob(a_j, a, Z) \cdot V'(succ(Z, a_j, \theta))$
    end loop
   end loop
   $V(Z) := \max_{(a,\theta)} Q(Z, a, \theta)$
  end loop
  $V := normalize(V)$
  $r := \|V - V'\|$
 until stopping criterion
 $\pi := extractPolicy(V)$
 return $(V, \pi, r)$

FOLAO$^*(\mathcal{A}, prob, reward, cost, \gamma, \mathcal{S}^0, h, \varepsilon)$
 $V := h$
 $G := \emptyset$
 For each $Z \in \mathcal{S}^0$, initialize $\pi$ with an arbitrary action
 repeat
   $(E, F, G) := policyExpansion(\pi, \mathcal{S}^0, G)$
   $(V, \pi, r) :=$ FOVI$(E, \mathcal{A}, prob, reward, cost, \gamma, V)$
 until $(F = \emptyset)$ and $r \leq \varepsilon$
 return $(\pi, V)$
```

Figure 2: LIFT-UP algorithm.

4 LIFT-UP Algorithm

In order to solve first-order MDPs, we have developed a new algorithm that combines heuristic search and first-order state abstraction techniques.

Our algorithm, referred to as LIFT-UP, can be seen as a generalization of the symbolic LAO* algorithm by [FH02]. Given an initial state, LIFT-UP uses an admissible heuristic to focus computation on the parts of the state space that are reachable from the initial state. Moreover, it specifies MDP components, value functions, policies, and admissible heuristics using a first-order language of the Probabilistic Fluent Calculus. This allows LIFT-UP to manipulate abstract states instead of individual states. The algorithm itself is presented in Figure 2.

As symbolic LAO*, LIFT-UP has two phases that alternate until a complete solution is found, which is guaranteed to be optimal. First, it expands the best partial policy and evaluates the states on its fringe using an admissible heuristic function. Then it performs dynamic programming on the states visited by the best partial policy, to update their values and possibly revise the current best partial policy. We note that we focus on partial policies that map a subcollection of states into actions.

In the policy expansion step, we perform reachability analysis to find the set F of states that have not yet been expanded, but are reachable from the set $\mathcal{S}^0$ of initial states by following the partial policy π. The set of states G contains states that have been expanded so far. By expanding a partial policy we mean that it will be defined for a larger set of states in the dynamic programming step.

In symbolic LAO*, reachability analysis is performed on propositional algebraic decision diagrams (ADDs). Therefore, an additional preprocessing of a first-order MDP is required at the outset of any solution attempt. This preprocessing involves propositionalization of the first-order structure of an MDP, viz., instantiation of the MDP components with all possible combinations of domain objects. Whereas, LIFT-UP relies on the lifted first-order reasoning, that is, computations are kept on the first-order level avoiding propositionalization. In particular, action applicability check and computation of successors as well as predecessors are accomplished on abstract states directly.

In the dynamic programming step of LIFT-UP, we employ a modified first-order value iteration algorithm (FOVI) that computes the value only on those states which are reachable from the initial states. More precisely, we call FOVI on the set E of states that are visited by the best current partial policy. In this way, we improve the efficiency of the original FOVI algorithm by [HS04] by using symbolic dynamic programming together with reachability analysis.

Given a FOMDP and a value function represented in PFC, FOVI returns the best partial value function V, the best partial policy π and the residual r. In order to update the values of the states Z in E, we assign the values from the current value function to the successors of Z. We compute successors with respect to all nature's choices a_j. The residual r is computed as the absolute value of the largest difference between the current and the newly computed value functions V' and V, respectively. We note that the newly computed value function V is taken in its normalized form, i.e., as a result of the *normalize* procedure that will be described in Section 4.2.1. Extraction of a best partial policy π is straightforward: One simply needs to extract the maximizing actions from the best partial value function V.

As with symbolic LAO*, LIFT-UP converges to an ε-optimal policy when three conditions are met: (1) its current policy does not have any unexpanded states, (2) the residual r is less than the predefined threshold ε, and (3) the value function is initialized with an admissible heuristic. The original convergence proofs for LAO* and symbolic LAO* by [HZ01] carry over in a straightforward way to LIFT-UP.

When calling LIFT-UP, we initialize the value function with an admissible heuristic function h that focuses the search on a subset of reachable states. A simple way to create an admissible heuristic is to use dynamic programming to compute an approximate value function. Therefore, in order to obtain an admissible heuristic h in LIFT-UP, we perform several iterations of the original FOVI. We start the algorithm on an initial value function that is admissible. Since each step of FOVI preserves admissibility, the resulting value function is admissible as well. The initial value function assigns the goal reward to each state thereby overestimating the optimal value, since the goal reward is the maximal possible reward.

Since all computations in LIFT-UP are performed on abstract states instead of individual states, FOMDPs are solved avoiding explicit state and action enumeration and propositionalization. Lifted first-order reasoning leads to better performance of LIFT-UP in comparison to symbolic LAO*, as shown in Section 5.2.

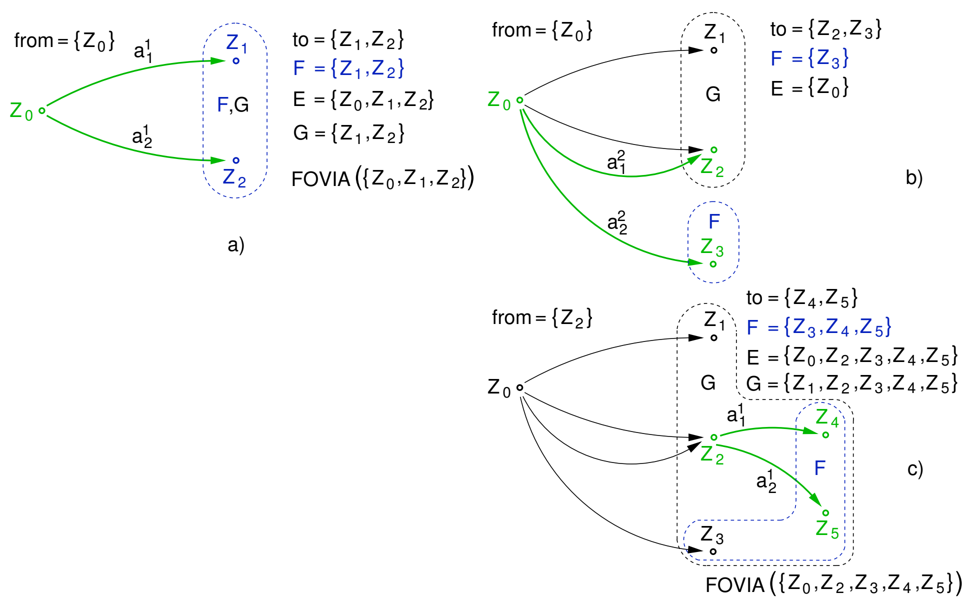

Figure 3: Policy Expansion.

4.1 Policy Expansion

We illustrate the policy expansion procedure in LIFT-UP by means of an example. Assume that we start from the initial state Z_0 and two nondeterministic actions a^1 and a^2 are applicable in Z_0, each having two outcomes a^1_1, a^1_2 and a^2_1, a^2_2, respectively. Without loss of generality, we assume that the current best policy π chooses a^1 as an optimal action at state Z_0. We construct the successors Z_1 and Z_2 of Z_0 with respect to both outcomes a^1_1 and a^1_2 of the action a^1. The fringe set F as well as the set G of states expanded so far contain the states Z_1 and Z_2 only, whereas, the set E of states visited by the best current partial policy gets the state Z_0 in addition. See Figure 3a. In the next step, FOVI is performed on the set E. We assume that the values have been updated in such a way that a^2 becomes an optimal action in Z_0. Thus, the successors of Z_0 have to be recomputed with respect to the optimal action a^2. See Figure 3b.

One should observe that one of the a^2-successors of Z_0, namely Z_2, is an element of the set G and thus, it has been contained already in the fringe F during the previous expansion step. Hence, the state Z_2 should be expanded and its value recomputed. This is shown in Figure 3c, where states Z_4 and Z_5 are a^1-successors of Z_2, under assumption that a^1 is an optimal action in Z_2. As a result, the fringe set F contains the newly discovered states Z_3, Z_4 and Z_5 and we perform FOVI on $E = \{Z_0, Z_2, Z_3, Z_4, Z_5\}$. The state Z_1 is not contained in E, because it does not belong to the best current partial policy, and the dynamic programming step is performed only on the states that were visited by the best current partial policy.

N	Number of states		Time, msec		Runtime, msec	Runtime w/o norm, msec
	$\mathcal{S}_{\mathsf{update}}$	$\mathcal{S}_{\mathsf{norm}}$	Update	Norm		
0	9	6	144	1	145	144
1	24	14	393	3	396	593
2	94	23	884	12	896	2219
3	129	33	1377	16	1393	13293
4	328	39	2079	46	2125	77514
5	361	48	2519	51	2570	805753
6	604	52	3268	107	3375	n/a
7	627	54	3534	110	3644	n/a
8	795	56	3873	157	4030	n/a
9	811	59	4131	154	4285	n/a

Table 1: Representative timing results for the first ten iterations of the first-order value iteration with the normalization procedure switched on or off.

4.2 First-order Value Iteration

The first-order value iteration algorithm (FOVI) produces a first-order representation of the optimal value function and policy by exploiting the logical structure of a first-order MDP. Thus, FOVI can be seen as a first-order counterpart of the classical value iteration algorithm by [Bel57].

In LIFT-UP, the first-order value iteration algorithm serves two purposes: First, we perform several iterations of FOVI in order to create an admissible heuristic h in LIFT-UP. Second, in the dynamic programming step of LIFT-UP, we apply FOVI on the states visited by the best partial policy in order to update their values and possibly revise the current best partial policy.

4.2.1 Normalization

It was already mentioned by several authors that value iteration adds a dramatic computational overhead to a solution technique for first-order MDPs if no care about redundant computations is taken [BRP01, HS04].

Recently, there have been proposed an automated normalization procedure that, given a state space, delivers an equivalent one that contains no redundancy [HS04]. This procedure, referred to as *normalize* in the LIFT-UP algorithm, is always called before the value function is transmitted to the next iteration step, thereby preventing the propagation of redundancy to the next computation steps. The technique employs the notion of the subsumption relation defined in Section 3. Informally, given two abstract states Z_1 and Z_2 such that $Z_1 \sqsubseteq Z_2$ and the values associated to states are identical, Z_1 can be easily removed from the state space because it contains redundant information.

Table 1 illustrates the importance of the normalization algorithm by providing some representative timing results for the first ten iterations of the first-order value iteration. The experiments were carried out on the problem taken from the colored Blocksworld scenario consisting of ten blocks. Even on such a relatively simple problem FOVI with the normalization switched off does not scale beyond the sixth iteration.

The results in Table 1 demonstrate that the normalization during some iteration of FOVI dramatically shrinks the computational effort during the next iterations. The columns labelled $\mathcal{S}_{\mathsf{update}}$ and $\mathcal{S}_{\mathsf{norm}}$ show the size of the state space after performing the value updates and the normalization, respectively. For example, the normalization factor, i.e., the ratio between the number $\mathcal{S}_{\mathsf{update}}$ of states obtained after performing one update step and the number $\mathcal{S}_{\mathsf{norm}}$ of states obtained after performing the normalization step, at the seventh iteration is 11.6. This means that more than ninety percent of the state space contained redundant information. The fourth and fifth columns in Table 1 contain the time Update and Norm spent on performing value updates and on the normalization, respectively. The total runtime Runtime, when the normalization is switched on, is given in the sixth column. The seventh column labelled Runtime w/o norm depicts the total runtime of FOVI when the normalization is switched off. If we would sum up all values in the seventh column and the values in the sixth column up to the sixth iteration inclusively, subtract the latter from the former and divide the result by the total time Norm needed for performing normalization during the first six iterations, then we would obtain the normalization gain of about three orders of magnitude.

5 The Planning System FluCaP

To evaluate the LIFT-UP approach we have developed a domain-dependent implementation called FluCaP. It can solve probabilistic Blocksworld problems as they appeared, for example, in the colored Blocksworld domain of the 2004 International Planning Competition.

5.1 Domain-dependent Optimizations

So far, we have presented a general theory of LIFT-UP for finding solutions in uncertain planning environments which are represented as first-order MDPs. However, several domain-driven optimizations or relaxations have been posed on the general theory. As a result, FluCaP has demonstrated a competitive computational behaviour.

Action Applicability Since in the Blocksworld domain, all states were fully specified, abstract states were described as fluent terms only. This allows to relax the forward and backward action applicability conditions. Since the cases are symmetric, we will concentrate on the forward action applicability condition that was initially defined as: An action $(a(X_1, \ldots, X_n), C, E)$ is *forward applicable with* θ to an abstract state $Z \in \mathcal{S}$, denoted as $\mathit{forward}(Z, a, \theta)$, if $(C \circ U)\theta =_{AC1} Z$, where U is a new variable of sort fluent term and θ is an allowed substitution. Since C and Z are fluent terms under the singularity condition, the aforementioned AC1-matching problem can be transformed into the θ-subsumption problem [Rob65].

Moreover, we optimize the obtained θ-subsumption problem further. Since a state description in a colored Blocksworld represents a number of towers, we compare towers of blocks and their color distributions instead of matching respective fluent terms. The experiments have shown that it is much faster to manipulate with towers rather than with fluent terms. For example, assume that an action precondition contains a fluent $\mathit{clear}(X)$. Let a state describe three towers of blocks. By inspecting the uppermost blocks in the towers, we conclude that there are only three blocks, which satisfy the precondition. It would be interesting to check whether this optimization technique can be successfully applied in other planning domains as well.

Meanwhile, in Section 6, we present some results towards efficient domain-independent solution methods for the θ-subsumption problem.

Normalization The similar situation occurs in the case of normalization which relies on the subsumption relation defined in Section 3. The AC1-matching problem underlying the subsumption relation reduces to the θ-subsumption problem.

Again, it is much faster to operate on towers rather than on fluent terms. For example, assume that one state describes two towers of four and three blocks, respectively. Another state also describes two towers but of five and two blocks, respectively. In order to decide, whether one state subsumes another one we try to match the corresponding towers and their color distributions. As experiments have shown, this optimization speeds up the normalization immensely.

5.2 Experimental Evaluation

We demonstrate the advantages of combining the heuristic search together with first-order state abstraction on a FluCaP system, that has successfully entered the domain-dependent track of the probabilistic part of the 2004 International Planning Competition (IPC'2004). The experimental results were all obtained using RedHat Linux running on a 3.4GHz Pentium IV machine with 3GB of RAM.

In Table 2, we present the performance comparison of FluCaP together with symbolic LAO* on examples taken from the colored Blocksworld (BW) scenario. Our main objective was to investigate whether first-order state abstraction using logic could improve the computational behaviour of a planning system for solving FOMDPs. The colored BW problems were our main interest since they were the only ones represented in first-order terms and hence the only ones that allowed us to make use of the first-order state abstraction.

At the outset of solving a colored BW problem, symbolic LAO* starts by propositionalizing its components, namely, the goal statement and actions. Only after that, the abstraction using propositional ADDs is applied. In contrast, FluCaP performs first-order abstraction on a colored BW problem directly, avoiding unnecessary grounding. In the following, we show how an abstraction technique affects the computation of a heuristic function. To create an admissible heuristic, FluCaP performs twenty iterations of FOVI and symbolic LAO* performs twenty iterations of an approximate value iteration algorithm similar to APRICODD by [SAHB00]. The columns labelled H.time and NAS show the time needed for computing a heuristic function and the number of abstract states it covers, respectively. In comparison to FluCaP, symbolic LAO* needs to evaluate fewer abstract states in the heuristic function but takes considerably more time. One can conclude that abstract states in symbolic LAO* enjoy more complex structure than those in FluCaP.

We note that, in comparison to FOVI, FluCaP restricts the value iteration to a smaller state space. Intuitively, the value function, which is delivered by FOVI, covers a larger state space, because the time that is allocated for the heuristic search in FluCaP is now used for performing additional iterations in FOVI. The results in the column

Problem		Total av. reward				Total time, sec.				H.time, sec.		NAS			NGS, $\times 10^3$		%
B	C	LAO*	FluCaP	FOVI	FluCaP$^-$	LAO*	FluCaP	FOVI	FluCaP$^-$	LAO*	FluCaP	LAO*	FluCaP	FOVI	LAO*	FluCaP	FluCaP
5	4	494	494	494	494	22.3	22.0	23.4	31.1	8.7	4.2	35	410	1077	0.86	0.82	2.7
	3	496	495	495	496	23.1	17.8	22.7	25.1	9.5	1.3	34	172	687	0.86	0.68	2.1
	2	496	495	495	495	27.3	11.7	15.7	16.5	12.7	0.3	32	55	278	0.86	0.66	1.9
6	4	493	493	493	493	137.6	78.5	261.6	285.4	76.7	21.0	68	1061	3847	7.05	4.24	3.1
	3	493	492	493	492	150.5	33.0	119.1	128.5	85.0	9.3	82	539	1738	7.05	6.50	2.3
	2	495	494	495	496	221.3	16.6	56.4	63.3	135.0	1.2	46	130	902	7.05	6.24	2.0
7	4	492	491	491	491	1644	198.1	2776	n/a	757.0	171.3	143	2953	12014	65.9	23.6	3.5
	3	494	494	494	494	1265	161.6	1809	2813	718.3	143.6	112	2133	7591	65.9	51.2	2.4
	2	494	494	494	494	2210	27.3	317.7	443.6	1241	12.3	101	425	2109	65.9	61.2	2.0
8	4	n/a	490	n/a	n/a	n/a	1212	n/a	n/a	n/a	804.1	n/a	8328	n/a	n/a	66.6	4.1
	3	n/a	490	n/a	n/a	n/a	598.5	n/a	n/a	n/a	301.2	n/a	3956	n/a	n/a	379.7	3.0
	2	n/a	492	n/a	n/a	n/a	215.3	1908	n/a	n/a	153.2	n/a	2019	7251	n/a	1121	2.3
15	3	n/a	486	n/a	n/a	n/a	1809	n/a	n/a	n/a	1733	n/a	7276	n/a	n/a	$1.2 \cdot 10^7$	5.7
17	4	n/a	481	n/a	n/a	n/a	3548	n/a	n/a	n/a	1751	n/a	15225	n/a	n/a	$2.5 \cdot 10^7$	6.1

Table 2: Performance comparison of FluCaP (denoted as FluCaP) and symbolic LAO* (denoted as LAO*), where the cells n/a denote the fact that a planner did not deliver a solution within the time limit of one hour. NAS and NGS are number of abstract and ground states, respectively.

labelled % justify that the harder the problem is (that is, the more colors it contains), the higher the percentage of runtime spent on normalization. Almost on all test problems, the effort spent on normalization takes three percent of the total runtime on average.

In order to compare the heuristic accuracy, we present in the column labelled NGS the number of ground states which the heuristic assigns non-zero values to. One can see that the heuristics returned by FluCaP and symbolic LAO* have similar accuracy, but FluCaP takes much less time to compute them. This reflects the advantage of the plain first-order abstraction in comparison to the marriage of propositionalization with abstraction using propositional ADDs. In some examples, we gain several orders of magnitude in H.time.

The column labelled Total time presents the time needed to solve a problem. During this time, a planner must execute 30 runs from an initial state to a goal state. A one-hour block is allocated for each problem. We note that, in comparison to FluCaP, the time required by heuristic search in symbolic LAO* (i.e., difference between Total time and H.time) grows considerably faster in the size of the problem. This reflects the potential of employing first-order abstraction instead of abstraction based on propositional ADDs during heuristic search.

The average reward obtained over 30 runs, shown in column Total av. reward, is the planner's evaluation score. The reward value close to 500 (which is the maximum possible reward) simply indicates that a planner found a reasonably good policy. Each time the number of blocks B increases by 1, the running time for symbolic LAO* increases roughly 10 times. Thus, it could not scale to problems having more than seven blocks. This is in contrast to FluCaP which could solve problems of seventeen blocks. We

B	Total av. reward, $\leq$500	Total time, sec.	H.time, sec.	NAS	NGS $\times 10^{21}$
20	489.0	137.5	56.8	711	1.7
22	487.4	293.8	110.2	976	1.1×10^3
24	492.0	757.3	409.8	1676	1.0×10^6
26	482.8	817.0	117.2	1141	4.6×10^8
28	493.0	2511.3	823.3	2832	8.6×10^{11}
30	491.2	3580.4	1174.0	4290	1.1×10^{15}
32	476.0	3953.8	781.8	2811	7.4×10^{17}
34	475.6	3954.1	939.4	3248	9.6×10^{20}
36	n/a	n/a	n/a	n/a	n/a

Table 3: Performance of FluCaP on larger instances of one-color Blocksworld problems, where the cells n/a denote the fact that a planner did not deliver a solution within the time limit.

note that the number of colors C in a problem affects the efficiency of an abstraction technique. In FluCaP, as C decreases, the abstraction rate increases which, in turn, is reflected by the dramatic decrease in runtime. The opposite holds for symbolic LAO*.

In addition, we compare FluCaP with two variants. The first one, denoted as FOVI, performs no heuristic search at all, but rather, employs FOVI to compute the ε-optimal total value function from which a policy is extracted. The second one, denoted as FluCaP$^-$, performs 'trivial' heuristic search starting with an initial value function as an admissible heuristic. As expected, FluCaP that combines heuristic search and FOVI demonstrates an advantage over plain FOVI and trivial heuristic search. These results illustrate the significance of heuristic search in general (FluCaP vs. FOVI) and the importance of heuristic accuracy, in particular (FluCaP vs. FluCaP$^-$). FOVI and FluCaP$^-$ do not scale to problems with more than seven blocks.

Table 3 presents the performance results of FluCaP on larger instances of one-color BW problems with the number of blocks varying from twenty to thirty four. We believe that FluCaP does not scale to problems of larger size because the implementation is not yet well optimized. In general, we believe that the FluCaP system should not be as sensitive to the size of a problem as propositional planners are.

Our experiments were targeted at the one-color problems only because they are, on the one hand, the simplest ones for us and, on the other hand, the bottleneck for propositional planners. The structure of one-color problems allows us to apply first-order state abstraction in its full power. For example, for a 34-blocks problem FluCaP operates on about 3.3 thousand abstract states that explode to 9.6×10^{41} individual states after propositionalization. A propositional planner must be highly optimized in order to cope with this non-trivial state space.

We note that additional colors in larger instances (more than 20 blocks) of BW problems cause dramatic increase in computational time, so we consider these problems as being unsolved. One should also observe that the number of abstract states NAS increases with the number of blocks non-monotonically because the problems are generated randomly. For example, the 30-blocks problem happens to be harder than the 34-blocks one. Finally, we note that all results that appear in Tables 2 and 3 were obtained by using the new version of the evaluation software that does not rely on propositionalization in contrast to the initial version that was used during the competition.

The competition domains and results are available in [YLWA05].

6 Domain-independent Methods for θ-subsumption

Given two fluent terms Z_1 and Z_2 under singularity condition, Z_1 θ-subsumes Z_2, written $Z_1 \vdash_{\theta}^{AC1} Z_2$, iff there exists an allowed substitution θ such that $(Z_1 \circ U)\theta =_{AC1} Z_2$.

Initially, θ-subsumption was defined on clauses. Given two clauses C and D, C θ-subsumes D iff there exists a substitution θ such that $C\theta \subseteq D$ [Rob65]. In general, θ-subsumption is NP-complete [KN86]. In the domain-dependent implementation of the LIFT-UP approach, that was described in the previous section, we have employed domain-driven optimization techniques that have allowed to reduce the complexity of θ-subsumption. This section is devoted to the efficient domain-independent solution methods for θ-subsumption which cope with its NP-completeness.

One approach to cope with the NP-completeness of θ-subsumption is deterministic subsumption. A state is said to be determinate if there is an ordering of fluents, such that in each step there is a fluent which has exactly one match that is consistent with the previously matched fluents [KL94]. However, in practice, there may be only few fluents, or none at all, that can be matched deterministically. Recently, in [SHW96], it was developed another approach, which we refer to as literal context, LitCon, for short, to cope with the complexity of θ-subsumption. The authors propose to reduce the number of matching candidates for each fluent by using the contextual information. The method is based on the idea that fluents may only be matched to those fluents that possess the same relations up to an arbitrary depth in a clause. As a result, a certain superset of determinate states can be tested for subsumption in polynomial time.

Unfortunately, as it was shown in [KRS06], LitCon does not scale very well up to large depth. Because in some planning problems, the size of state descriptions can be relatively large, it might be necessary to compute the contextual information for large values of the depth parameter. Therefore, we are strongly interested in a technique that scales better than LitCon. In this section, we present an approach, referred to as object context, ObjCon, for short, which demonstrates better computational behaviour than LitCon. Based on the idea of ObjCon, we develop a new θ-subsumption algorithm and compare it with the LitCon-based approach.

6.1 Object Context

In general, a fluent f in a state Z_1 can be matched with several fluents in a state Z_2, that are referred to as matching candidates of f. LitCon is based on the idea that fluents in Z_1 can be only matched to those fluents in Z_2, the context of which include the context of the fluents in Z_1 [SHW96]. The context is given by occurrences of identical objects (variables $\mathit{Vars}(Z)$ and constants $\mathit{Const}(Z)$) or chains of such occurrences and is defined up to some fixed depth. In effect, matching candidates that do not meet the above context condition can be effortlessly pruned. In most cases, such pruning results in deterministic subsumption, thereby considerably extending the tractable class of states.

The computation of the context itself is dramatically affected by the depth parameter: The larger the depth is, the longer the chains of objects' occurrences are, and thus, more effort should be devoted to build them. Unfortunately, LitCon does not scale very well up to large depth [KRS06]. For example, consider a state

$$Z = on(X, Y) \circ on(Y, table) \circ r(X) \circ b(Y) \circ h(X) \circ h(Y) \circ w(X) \circ l(Y)$$

that can be informally read as: A block X is on the block Y which is on the table, and both blocks enjoy various properties, like color (red r or blue b) or weight (heavy h or light l), they can be wet w. Z contains eight fluents and only three objects. In LitCon, the context should be computed for each of eight fluents in order to keep track of all occurrences of identical objects. What if we were to compute the context for each object instead? In our running example, we would need to perform computations only three times, in this case.

Herein, we propose a more efficient approach, referred to as ObjCon, for computing the contextual information and incorporate it into a new context-based θ-subsumption algorithm. More formally, we build the object occurrence graph $\mathcal{G}_Z = (V, E, \ell)$ for a state Z, where vertices are objects of Z, denoted as $Obj(Z)$, and edges $E = \{(o_1, \pi_1, f, \pi_2, o_2)|$ Z contains $f(t_1, \ldots, t_n)$ and $o_1 = t_{\pi_1}$ and $o_2 = t_{\pi_2}\}$ with $o_1, o_2 \in Obj(Z)$, $f(t_1, \ldots, t_n)$ being a fluent and π_1, π_2 being positions of objects o_1, o_2 in f. The labeling function $\ell(o) = \{f|Z \text{ contains } f(o)\}$ associates each object o with a unary fluent name f this object belongs to. The object occurrence graph for the state Z from our running example will contain three vertices X, Y and $table$ with labels $\{r, h, w\}$, $\{b, h, l\}$ and $\{\}$, resp., and two edges $(X, 1, on, 2, Y)$ and $(Y, 1, on, 2, table)$.

The object context $\text{ObjCon}(o, Z, d)$ of depth $d > 0$ is defined for each object o of a state Z as a chain of labels: $\ell(o) \xrightarrow{\pi_1^1 \cdot f^1 \cdot \pi_2^1} \ell(o_1) \xrightarrow{\pi_1^2 \cdot f^2 \cdot \pi_2^2} \ldots \xrightarrow{\pi_1^d \cdot f^d \cdot \pi_2^d} \ell(o_d) \in \text{ObjCon}(o, Z, d)$ *iff* $o \xrightarrow{\pi_1^1 \cdot f^1 \cdot \pi_2^1} o_1 \xrightarrow{\pi_1^2 \cdot f^2 \cdot \pi_2^2} \ldots \xrightarrow{\pi_1^d \cdot f^d \cdot \pi_2^d} o_d$ is a path in $\mathcal{G}_Z$ of length d starting at o. In our running example, $\text{ObjCon}(X, Z, 1)$ of depth 1 of the variable X in Z contains one chain $\{\{r, h, w\} \xrightarrow{1 \cdot on \cdot 2} \{b, h, l\}\}$.

Following the ideas of [SHW96], we define the embedding of object contexts for states Z_1 and Z_2, which serves as a pruning condition for reducing the space of matching candidates for Z_1 and Z_2. Briefly, let $OC_1 = \text{ObjCon}(o_1, Z_1, d)$, $OC_2 = \text{ObjCon}(o_2, Z_2, d)$. Then OC_1 is embedded in OC_2, written $OC_1 \preccurlyeq OC_2$, iff for every chain of labels in OC_1 there exists a chain of labels in OC_2 which preserves the positions of objects in fluents and the labels for each object in OC_1 are included in the respective labels in OC_2 up to the depth d. Finally, if $\text{ObjCon}(X, Z_1, d) \not\preccurlyeq \text{ObjCon}(o, Z_2, d)$ then there exists no θ such that $(Z_1 \circ U)\mu\theta =_{AC1} Z_2$, where $\mu = \{X \mapsto o\}$ and U is a new variable of sort fluent term. In other words, a variable X in Z_1 cannot be matched against an object o in Z_2 within a globally consistent match, if the variable's context cannot be embedded in the object's context. Therefore, the substitutions that meet the above condition can be effortlessly pruned from the search space. For any context depth $d > 0$, the context inclusion is an additional condition that reduces the number of candidates, and hence there exists more often at most one remaining matching candidate.

Based on the idea of the object context, we describe a new θ-subsumption algorithm in Algorithm 1. Please note that this algorithm provides a complete set of all allowed substitutions which is used later on for determining the set of all possible successors or predecessors of some state with respect to some action. Due to the lack of space, we

Input: Two fluent terms Z_1, Z_2.
Output: A complete set of substitutitons θ such that $Z_1 \vdash_{\theta}^{AC1} Z_2$.

1. Deterministically match as many fluents of Z_1 as possible to fluents of Z_2. Substitute Z_1 with the substitution found. If some fluent of Z_1 does not match any fluent of Z_2, decide $Z_1 \nvdash_{\theta}^{AC1} Z_2$.
2. ObjCon-based deterministically match as many fluents of Z_1 as possible to fluents of Z_2. Substitute Z_1 with the substitution found. If some fluent of Z_1 does not match any fluent of Z_2, decide $Z_1 \nvdash_{\theta}^{AC1} Z_2$.
3. Build the substitution graph (V, E) for Z_1 and Z_2 with nodes $v = (\mu, i) \in V$, where μ is a matching candidate for Z_1 and Z_2, i.e., matches some fluent at position i in Z_1 to some fluent in Z_2 and $i \geq 1$ is referred to as a layer of v. Two nodes (μ_1, i_1) and (μ_2, i_2) are connected with an edge iff $\mu_1\mu_2 = \mu_2\mu_1$ and $i_1 = i_2$. Delete all nodes (μ, i) with $X\mu = o$, for some $X \in \mathit{Vars}(Z_1)$ and $o \in \mathit{Obj}(Z_2)$, and ObjC∅n$(X, Z_1, d) \preccurlyeq$ ObjCon(o, Z_2, d) for some d. Find all cliques of size $|Z_1|$ in (V, E).

Algorithm 1: ObjCon-alltheta.

omit the algorithm for computing all cliques in a substitution graph. However, it can be found in [KRS06].

6.2 Experimental Evaluation

Figure 4 depicts the comparison timing results between the LitCon-based subsumption reasoner, referred to as AllTheta, and its ObjCon-based opponent, referred to as FluCaP. The results were obtained using RedHat Linux running on a 2.4GHz Pentium IV machine with 2GB of RAM.

We demonstrate the advantages of exploiting the object-based context information on problems that stem from the colored Blocksworld and Pipesworld planning scenarios. The Pipesworld domain models the flow of oil-derivative liquids through pipeline segments connecting areas, and is inspired by applications in the oil industry. Liquids are modeled as batches of a certain unit size. A segment must always contain a certain number of batches (i.e., it must always be full). Batches can be pushed into pipelines from either side, leading to the batch at the opposite end "falling" into the incident area. Batches have associated product types, and batches of certain types may never be adjacent to each other in a pipeline. Moreover, areas may never have constraints on how many batches of a certain product type they can hold.

For each problem, there have been done 1000 subsumption tests. The time limit of 100 minutes has been allocated. The results show that FluCaP scales better than AllTheta. It is best to observe on the problems of forteen-, twenty-, and thirty-blocks. As empirical results demonstrate, the optimal value of the depth parameter for Blocksworld and Pipesworld is four.

The main reason for the computational gain of FluCaP is that it is less sensitive to the growth of the depth parameter. Under the condition that the number of objects in a state is strictly less than the number of fluents and other parameters are fixed, the amount of object-based context information is strictly less than the amount of the literal-based context information. Moreover, on the Pipesworld problems, FluCaP requires two orders of magnitude less time than AllTheta.

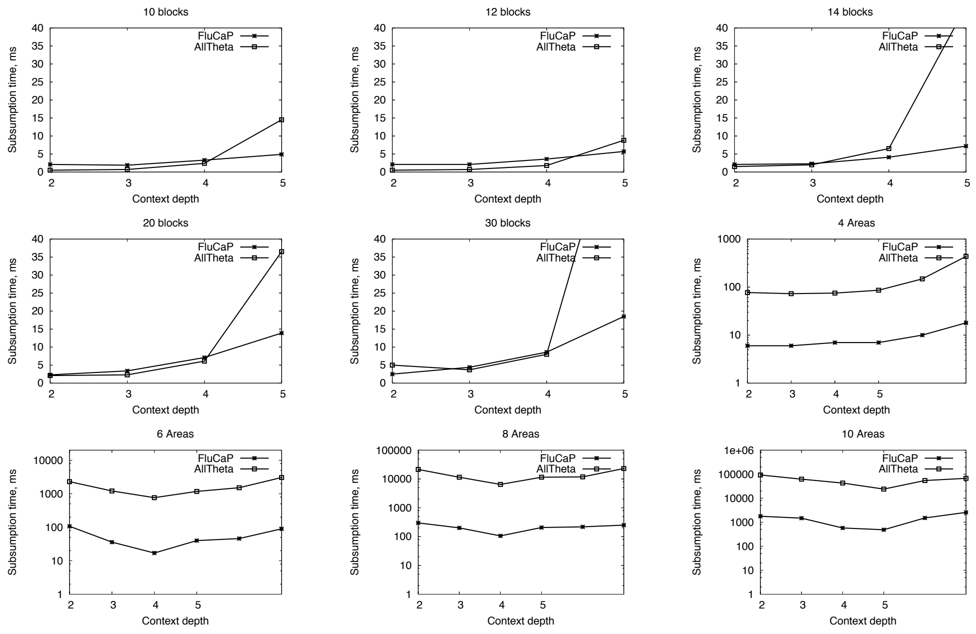

Figure 4: Comparison timing results for FluCaP and AllTheta. The results present the average time needed for one subsumption test. Please note that the plots for Pipesworld are shown in logscale. Therefore small differences in the plot may indicate a substantial difference on runtimes.

7 Related Work

We follow the symbolic dynamic programming (SDP) approach within Situation Calculus (SC) of [BRP01] in using first-order state abstraction for FOMDPs. In the course of first-order value iteration, a state space may contain redundant abstract states that dramatically affect the algorithm's efficiency. In order to achieve computational savings, normalization must be performed to remove this redundancy. However, in the original work by [BRP01] this was done by hand. To the best of our knowledge, the preliminary implementation of the SDP approach within SC uses human-provided rewrite rules for logical simplification. In contrast, [HS04] have developed an automated normalization procedure for FOVI brings the computational gain of several orders of magnitude. Another crucial difference is that our algorithm uses heuristic search to limit the number of states for which a policy is computed.

The ReBel algorithm by [KvOdR04] relates to LIFT-UP in that it also uses a representation language that is simpler than Situation Calculus. This feature makes the state space normalization computationally feasible.

All the above algorithms can be classified as deductive approaches to solving FOMDPs. They can be characterized by the following features: (1) they are model-based, (2) they aim at exact solutions, and (3) logical reasoning methods are used to compute abstractions. We should note that FOVI aims at exact solution for a FOMDP, whereas LIFT-UP, due to the heuristic search that avoids evaluating all states, seeks for an approximate solution. Therefore, it would be more appropriate to classify LIFT-UP as an approximate deductive approach to FOMDPs.

In another vein, there is some research on developing inductive approaches to solving FOMDPs, e.g., by [FYG03]. The authors propose the approximate policy iteration (API) algorithm, where they replace the use of cost-function approximations as policy representations in API with direct, compact state-action mappings, and use a standard relational learner to learn these mappings. A recent approach by [GT04] proposes an inductive policy construction algorithm that strikes a middle-ground between deductive and inductive techniques.

8 Conclusions

We have proposed a new approach that combines heuristic search and first-order state abstraction for solving first-order MDPs more efficiently. In contrast to existing systems, which start with propositionalizing the decision problem at the outset of any solution attempt, we perform lifted reasoning on the first-order structure of an MDP directly. However, there is plenty remaining to be done. For example, we are interested in the question of to what extent the optimization techniques applied in modern propositional planners can be combined with first-order state abstraction.

Acknowledgements

We thank anonymous reviewers for their comments on the previous versions of this paper. We also thank Zhengzhu Feng for fruitful discussions and for providing us with the executable of the symbolic LAO* planner. Olga Skvortsova was supported by a grant within the Graduate Programme GRK 334 "Specification of discrete processes and systems of processes by operational models and logics" under auspices of the Deutsche Forschungsgemeinschaft (DFG).

References

[BBS95] A. G. Barto, S. J. Bradtke, and S. P Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138, 1995.

[BDH99] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[Bel57] R. E. Bellman. *Dynamic programming.* Princeton University Press, Princeton, NJ, USA, 1957.

[BRP01] C. Boutilier, R. Reiter, and B. Price. Symbolic Dynamic Programming for First-Order MDPs. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Conference on Artificial Intelligence (IJCAI'2001)*, pages 690–700. Morgan Kaufmann, 2001.

[DKKN95] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76:35–74, 1995.

[FH02] Z. Feng and E. Hansen. Symbolic heuristic search for factored Markov Decision Processes. In R. Dechter, M. Kearns, and R. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'2002)*, pages 455–460, Edmonton, Canada, 2002. AAAI Press.

[FYG03] A. Fern, S. Yoon, and R. Givan. Approximate policy iteration with a policy language bias. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Proceedings of the Seventeenth Annual Conference on Neural Information Processing Systems (NIPS'2003)*, Vancouver, Canada, 2003. MIT Press.

[GT04] C. Gretton and S. Thiebaux. Exploiting first-order regression in inductive policy selection. In M. Chickering and J. Halpern, editors, *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI'2004)*, Banff, Canada, July 2004. Morgan Kaufmann.

[HS90] S. Hölldobler and J. Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990.

[HS04] S. Hölldobler and O. Skvortsova. A Logic-Based Approach to Dynamic Programming. In *Proceedings of the Workshop on "Learning and Planning in Markov Processes–Advances and Challenges" at the Nineteenth National Conference on Artificial Intelligence (AAAI'04)*, pages 31–36, San Jose, CA, July 2004. AAAI Press.

[HZ01] E. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001.

[KL94] J.-U. Kietz and M. Lübbe. An efficient subsumption algorithm for inductive logic programming. In *Proceedings of the Eleventh International Conference on MAchine Learning*, pages 130–138, 1994.

[KN86] D. Kapur and P. Narendran. NP-completeness of the set unification and matching problems. In J.H.Siekman, editor, *Proceedings of the Conference on Automated Deduction*, pages 489–495. Springer, Berlin, 1986. Lecture Notes in Computer Science 230.

[KRS06] E. Karabaev, G. Rammé, and O. Skvortsova. Efficient symbolic reasoning for first-order MDPs. In *Proceedings of the Workshop on "Planning, Learning and Monitoring with Uncertainty and Dynamic Worlds" at the Seventeenth European Conference on Artificial Intelligence (ECAI'2006)*, Riva del Garda, Italy, 2006. To appear.

[KvOdR04] K. Kersting, M. van Otterlo, and L. de Raedt. Bellman goes relational. In C. E. Brodley, editor, *Proceedings of the Twenty-First International Conference in Machine Learning (ICML'2004)*, pages 465–472, Banff, Canada, July 2004. ACM.

[Put94] M. L. Puterman. *Markov Decision Processes - Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, 1994.

[Rei91] R. Reiter. The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, 1991.

[Rob65] J.A. Robinson. A machine-learning logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, 1965.

[SAHB00] R. St-Aubin, H. Hoey, and C. Boutilier. APRICODD: Approximate policy construction using decision diagrams. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Proceedings of the Fourteenth Annual Conference on Neural Information Processing Systems (NIPS'2000)*, pages 1089–1095, Denver, 2000. MIT Press.

[SHW96] T. Scheffer, R. Herbrich, and F. Wysotzki. Efficient θ-subsumption based on graph algorithms. In *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 212–228, Berlin, August 1996. volume 1314 of LNAI.

[Thi98] M. Thielscher. Introduction to the fluent calculus. *Electronic Transactions on Artificial Intelligence*, 2(3-4):179–192, 1998.

[YLWA05] H.L.S. Younes, M.L. Littman, D. Weissman, and J. Asmuth. The first probabilistic track of the International Planning Competition. *Journal of Artificial Intelligence Research*, 24:851–887, 2005.