Efficient Symbolic Reasoning for First-Order MDPs

Eldar Karabaev and **Georg Rammé** and **Olga Skvortsova**[1]

Abstract. We propose an algorithm, referred to as ALLTHETA, for performing efficient domain-independent symbolic reasoning in a planning system FLUCAP 1.1 that solves first-order MDPs. The computation is done avoiding vicious state and action grounding.

1 INTRODUCTION

Markov Decision Processes (MDPs) are de facto standard representational and computational model for decision-theoretic planning problems. Recently, several compact representations for propositionally-factored MDPs have been proposed, including dynamic Bayesian networks [1] and algebraic decision diagrams [6]. For instance, the SPUDD algorithm [6] has been used to solve MDPs with hundreds of millions of states optimally, producing logical descriptions of value functions that involve only hundreds of distinct values. This technique, referred to as state abstraction, demonstrates that large MDPs, described in a logical fashion, can often be solved optimally by exploiting the logical structure of the problem.

Meanwhile, many realistic planning domains are best specified in first-order terms. However, most existing implemented solutions for first-order MDPs (FOMDPs) rely on grounding, i.e., eliminate all variables at the outset of a solution attempt by instantiating terms with all possible combinations of domain objects, e.g., [4]. This technique is very impractical because the number of propositions grows considerably with the number of domain objects and relations. This has a dramatic impact on the complexity of the algorithms that depends directly on the number of propositions. Moreover, as soon as the universe of objects is infinite, these algorithms cannot be made work. Finally, systems for solving FOMDPs that rely on state grounding also perform action grounding which is problematic in first-order domains, because the number of ground actions also grows drastically with domain size.

To address these difficulties, we have recently proposed a first-order generalization of LAO* algorithm [9], referred to as FOLAO*, in which our contribution was to show how to perform heuristic search for FOMDPs, circumventing their grounding. In order to ensure first-order reasoning without descending to the propositional level, a planning system should be equipped with highly-optimized domain-independent inference algorithms that compute sets of successor and predecessor states of a given state wrt. a given action. Such inference algorithms rely on non-trivial symbolic computations as, e.g., unification or subsumption problem under some equational theory between two states specified as first-order terms.

In this paper, we develop an algorithm, referred to as ALLTHETA, that solves the subsumption problem under AC1[2] equational theory and delivers all possible substitutions. The computation is done avoiding aggressive grounding. ALLTHETA has been recently integrated into the FLUCAP 1.1 planning system that is a successor of FLUCAP 1.0 [7].

[1] International Center for Computational Logic, Technische Universität Dresden, email: {eldar,skvortsova}@iccl.tu-dresden.de, georg.ramme@gmail.com.

[2] A - associative, C - commutative, 1 - unit element.

2 FIRST-ORDER REPRESENTATION OF MDPs

As a first step in the quest of designing a planning system for solving FOMDPs avoiding their grounding, we propose a concise representation of FOMDPs within the Probabilistic Fluent Calculus ($\mathcal{PFC}$) language. $\mathcal{PFC}$ is a logical approach to modelling dynamically changing and uncertain environments based on first-order logic [7, 8].

2.1 MDPs

An MDP is a tuple $(\mathcal{Z}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{C})$, where $\mathcal{Z}$ and $\mathcal{A}$ are finite sets of states and actions, respectively. $\mathcal{P} : \mathcal{Z} \times \mathcal{Z} \times \mathcal{A} \to [0, 1]$, written $\mathcal{P}(z'|z, a)$, specifies transition probabilities. In particular, $\mathcal{P}(z'|z, a)$ denotes the probability of ending up at a state z' given that the agent was in a state z and an action a was executed. $\mathcal{R} : \mathcal{Z} \to \Re$ is a real-valued reward function associating with each state z its immediate utility $\mathcal{R}(z)$. $\mathcal{C} : \mathcal{A} \to \Re$ is a real-valued cost function associating a cost $\mathcal{C}(a)$ to each action a.

A sequential decision problem consists of an MDP and is the problem of finding a policy $\pi : \mathcal{Z} \to \mathcal{A}$ that maximizes the total expected discounted reward received when executing the policy π over an infinite (or indefinite) horizon. The value of a state z with respect to a policy π is defined recursively 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. The optimal value function V^* satisfies:

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

for each $z \in \mathcal{Z}$.

2.2 Probabilistic Fluent Calculus

Formally, let Σ denote a set of function symbols. We distinguish two function symbols in Σ, namely $\circ/2$ which is associative (A), commutative (C), and admits the unit element, and a constant 1. Let $\Sigma_- = \Sigma \setminus \{\circ, 1\}$. Non-variable Σ_--terms are called fluents. Let $\mathcal{F}$ denote the set of fluents. Fluent terms are defined inductively as follows: 1 is a fluent term; each fluent is a fluent term; $F \circ G$ is a fluent term, if F and G are fluent terms.

A *state* is a fluent term. We assume that each fluent may occur at most once in a state, i.e., states of the form $euro \circ euro$ are disallowed. For example, a state $Z = on(X', Y') \circ on(Y', t) \circ cl(X') \circ e$ denotes that some clear block X' is on the block Y', which is on the table, the gripper is empty and something else might be also true. We note that the negation can be effortlessly included in the language [7].

The interpretation over $\mathcal{F}$, denoted as $\mathcal{I}$, is the pair $(\Delta, \cdot^{\mathcal{I}})$, where the domain Δ is a set of all finite sets of ground fluents from $\mathcal{F}$; and an interpretation function $\cdot^{\mathcal{I}}$ which assigns to each state Z a set

$$Z^{\mathcal{I}} = \{d \in \Delta | \exists \theta . (Z \circ U)\theta =_{AC1} d\} ,$$

where θ is a substitution and U is a new AC1-variable. Thus, states in $\mathcal{PFC}$ represent clusters of individual states. In this way, they embody a form of state space abstraction, referred to as first-order state abstraction, and, hence, can be treated as abstract states. E.g, the state $z_1 = on(b, c) \circ on(c, t) \circ cl(b) \circ e \circ cl(f)$, where t stands for table and b, c and f are blocks, is represented by the abstract state Z above; whereas $z_2 = on(b, c)$ is not, since other three 'mandatory' fluents of Z are missing in z_2. In essence, abstract states are defined under incomplete semantics, viz., other fluents that are not explicitly present in the state description might also hold, as e.g., $cl(f)$ appears in the state $z_1 \in Z^{\mathcal{I}}$.

Actions are first-order terms leading with an action function symbol. For example, the action of picking up some block X from another block Y might be denoted as *pickup* (X, Y). Stochastic actions are described via decomposition into deterministic primitives under nature's control, referred to as nature's choices. E.g., action *pickup* (X, Y) can be defined by means of successful *pickupS* (X, Y) and failure *pickupF* (X, Y) nature's choices. Preconditions and effects of an action a, denoted as *Pre* (a) and *Eff* (a), respectively, are abstract states. E.g., for preconditions and effects of the action *pickupS* (X, Y), we have:

$$\begin{aligned}&\mathit{Pre}\,(\mathit{pickupS}\,(X, Y)) := on(X, Y) \circ cl(X) \circ e\\ &\mathit{Eff}\,(\mathit{pickupS}\,(X, Y)) := h(X) ,\end{aligned}$$

where $h(X)$ stands for the fact of holding a block X. Probabilities of each nature's choice, rewards and action costs can be defined in an obvious way.

3 SYMBOLIC REASONING FOR FOMDPs

Systems for solving FOMDPs that rely on state grounding also perform action grounding which is problematic in first-order domains, because the number of ground actions grows drastically with domain size. Herein, we show how to perform inferences, i.e., compute successors and predecessors of a given abstract state, with action schemata directly, avoiding unnecessary grounding.

For this, an inference problem of finding all a-successors (all a-predecessors) of an abstract state Z is represented in terms of the AC1-unification problem[3], referred to as AC1-UNIFY(Z_1, Z_2), where Z_1 represents the preconditions (effects) of a and $Z_2 = Z$. AC1-UNIFY(Z_1, Z_2) is defined by:

$$\exists \theta .\ (Z_1 \circ U)\theta =_{AC1} (Z_2 \circ W)\theta ,$$

where U and W are new AC1-variables.

Intuitively, an action a is applicable to an abstract state Z iff it is applicable to *all* individual states that constitute $Z^{\mathcal{I}}$. In order to determine all fragments of Z, an action a is applicable to, we compute all solutions for the following AC1-unification problem:

$$(\mathit{Pre}\,(a) \circ U)\theta =_{AC1} (Z \circ W)\theta . \qquad (1)$$

In this way, the bindings for W define the fragments $Z^i = (Z \circ W)\theta$ of Z, an action a is applicable to. Moreover, the bindings for U allow us to construct the successors of Z^i, i.e., $Z^i_{succ} := (\mathit{Eff}\,(a) \circ U)\theta$. In essence, in order to compute the set of all a-successors of all fragments of Z, a is applicable to, it is enough to find all solutions θ for the AC1-unification problem given by Equation 1.

In this work, we present a restricted case of AC1-unification, where $(Z_2 \circ W)\theta = Z_2$. Thus, the AC1-unification problem AC1-UNIFY(Z_1, Z_2) can be simplified into the AC1-subsumption problem AC1-SUBSUME(Z_1, Z_2), i.e., a subsumption problem under the equational theory AC1:

$$\exists \theta .\ (Z_1 \circ U)\theta =_{AC1} Z_2 .$$

We write $Z_1 \vdash^{AC1}_{\theta} Z_2$ for 'Z_1 subsumes Z_2 under AC1'. There are at least two applications of AC1-SUBSUME(Z_1, Z_2) in the FOLAO* algorithm. First, for detecting a more specific abstract state between Z_1 and Z_2, that can be removed from the state space thereafter. Second, for computing a set of *all* states that are reachable from the initial state Z_2 wrt. all actions, where Z_2 is a ground fluent term specified under Closed World Assumption. In both cases, the computation is performed on action schemata directly preventing their grounding.

In the following, we exploit the fact that the AC1-subsumption problem is a specialization of the θ-subsumption problem on general clauses, since abstract states are Horn clauses with empty head [13]. The θ-subsumption problem for clauses C and D is a problem of whether there exists a substitution θ such that $C\theta \subseteq D$ (or, in our terms, $(C \circ U)\theta =_{AC1} D$).

In general, θ-subsumption is NP-complete [13]. It is known that deterministic subsumption, i.e., when there exists 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, can be solved in polynomial time [11]. Unfortunately, in general, there are only few, or, in some cases, none at all, fluents in a state that can be matched deterministically. The complexity grows exponentially with the number of remaining non-determinate fluents.

In [13], there have been developed two approaches to reduce the complexity of non-deterministic θ-subsumption. Both approaches have been reconciled in an algorithm, referred to as THETA, that solves θ-subsumption and hence, AC1-subsumption, and delivers a single substitution, in a positive case.

3.1 Approach one: context-based subsumption

One approach is context-based matching candidate elimination. In general, a fluent f in an abstract state Z_1 can be matched with several fluents in an abstract state Z_2, that are referred to as matching candidates of f. The approach 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. The context is given by occurrences of identical variables 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 abstract states. Deterministic subsumption that exploits the context information is referred to as context-based deterministic subsumption.

[3] AC1-unification problem is a unification problem under the equational theory AC1.

For example, two abstract states $Z_1 = on(X,Y) \circ on(Y,t)$ (one tower of two blocks) and $Z_2 = on(a,b) \circ on(b,c) \circ on(c,t) \circ on(d,t)$ (two towers of two and one blocks, respectively) cannot be subsumed deterministically because each fluent in Z_1 has more than one matching candidate in Z_2. However, exploiting the context information already at depth 1 enables us to conclude that Z_1 subsumes Z_2.

At depth 1, the context of the first fluent $on(X,Y)$ contains the path $on \cdot 2 \to 1 \cdot on$, i.e., a variable Y appears at position 2 in $on(X,Y)$ and at position 1 in $on(Y,t)$. The context of the second fluent $on(Y,t)$ contains the path $on \cdot 1 \to 2 \cdot on$, i.e., the variable Y appears at position 2 in $on(X,Y)$ and at position 1 in $on(Y,t)$. The contexts of the fluents in Z_2 are $\{on \cdot 2 \to 1 \cdot on\}$, $\{on \cdot 1 \to 2 \cdot on, on \cdot 2 \to 1 \cdot on\}$, $\{on \cdot 1 \to 2 \cdot on, on \cdot 2 \to 2 \cdot on\}$ and $\{on \cdot 2 \to 2 \cdot on\}$, respectively. The fluent $on(Y,t)$ has two matching candidates, viz., $on(c,t)$ and $on(d,t)$. Since the context of $on(Y,t)$ can only be embedded in the context of $on(c,t)$, written $con(on(Y,t), Z_1, 1) \subseteq con(on(c,t), Z_2, 1)$, the matching candidate $on(d,t)$ is excluded and $on(Y,t)$ can be matched deterministically. Then, the matching substitution $\mu_1 = \{Y \mapsto c\}$ is applied to Z_1. As a result, the fluent $on(X,Y)\mu_1 = on(X,c)$ can be matched deterministically to $on(b,c)$ with $\mu_2 = \{X \mapsto b\}$. Hence, both fluents can be matched deterministically and the substitution $\theta = \mu_1\mu_2$ was found without backtracking.

The context depth is a very crucial parameter. If its value is overestimated, then considerable effort is devoted for computing the context itself and the efficiency of pruning is potentially increased. Alternatively, if depth is underestimated, we save time and space for constructing the context but end up with a larger search space. For example, in CBW and RANDOM datasets from Section 4, the optimal depth has the value of 2.

3.2 Approach two: ALL-CLIQUES

In some cases, however, after performing the context-based deterministic subsumption, there still remain some fluents that cannot be matched deterministically. Thus, a remaining space of matching candidates has to be searched for a substitution. For this, a second approach that reduces the complexity of non-deterministic AC1-subsumption, has been presented in [13].

It exploits a well-known correspondance result between the AC1-subsumption problem and the clique problem, i.e., a problem of finding a clique[4] of the fixed size in a graph. More precisely, an abstract state Z_1 subsumes an abstract state Z_2 iff there is a clique of size $|Z_1|$ in the space of matching candidates for fluents in Z_1. The size $|Z|$ of an abstract state Z is equal to the number of fluents comprising it. Exploiting the above relationship, an algorithm, referred to as CLIQUE, that finds a clique in the space of matching candidates, has been developed. The candidates that do not form a clique can be effortlessly excluded from the search space. Moreover, the authors describe some methods that allow to reduce the certain amount of search space already a priori.

However, further pruning techniques can be applied in order to alleviate the search for a clique. Based on CLIQUE, we present an algorithm, referred to as ALL-CLIQUES, that computes *all* paths of size $|Z_1|$ that form cliques in the space of matching candidates for Z_1. ALL-CLIQUES is given in Algorithm 1.

Following ideas of [13], we construct a substitution graph (V,E) for abstract states 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 $i_1 \neq i_2$. The former condition is referred to as strong compatibility of substitutions, i.e., no variable is assigned different terms in μ_1 and μ_2.

[4] A clique in a graph is a set of pairwise adjacent nodes.

Function `findPath`(V, E, *Paths*, v, *currPath*, i)

```
1  if valid($v$) then
2      $currPath := currPath \cup \{v\}$
3      if $i = |Z_1|$ then
4          $Paths := Paths \cup \{currPath\}$
5      else
6          foreach $u = (\mu', i+1) \in V$ with $(v,u) \in E$ do
7              if clique($u$, $currPath$) then
8                  findPath($V$, $E$, $Paths$, $u$, $currPath$, $i+1$)
9  else $V := V \setminus \{v\}$
10 return $Paths$
```

Algorithm 1: ALL-CLIQUES.

```
Input: A substitution graph $(V,E)$ for abstract states $Z_1$, $Z_2$.
Output: All paths in $(V,E)$ that form cliques of size $|Z_1|$.
$Paths := currPath := \emptyset$
foreach $v = (\mu, 1) \in V$ do
    $Paths :=$ findPath($V$, $E$, $Paths$, $v$, $currPath$, $1$)
end
return $Paths$
```

ALL-CLIQUES returns all paths *Paths* in the graph (V,E) that start at the first layer and form a clique of size $|Z_1|$. In essence, ALL-CLIQUES performs depth-first search in the substitution graph. The validity check `valid`(v) in line 1 of function `findPath` is successful iff v has at least one edge to each layer. In this case, v is added to the current path *currPath*. If v is located at the last layer, i.e., $i = |Z_1|$, then the current path can be already added to *Paths*. Otherwise, if a next-layer neighbour u of v forms a clique with the nodes in *currPath*, i.e., `clique`(u, *currPath*) holds in line 7, then `findPath` is called recursively for u. In line 9, invalid nodes are excluded from V.

In the following, we discuss major pruning techniques that are currently integrated in ALL-CLIQUES.

First, in contrast to [13], we organize the substitution graph in layers, i.e., each node $v = (\mu, i) \in V$ belongs to a layer i. The layers should be visited in the order of their appearance. For example, the third layer cannot be visited before the second layer. The layered architecture of the substitution graph is a natural way to avoid duplicate occurrences of the same clique in the set of all cliques. For example, for the graph on Figure 1(a), the clique $\{(\mu_1, 1), (\mu_4, 3), (\mu_2, 2)\}$ is not counted because the node $(\mu_4, 3)$, that lies on the third layer, is visited immediately after the node $(\mu_1, 1)$, that lies on the first layer. Whereas, the set $\{(\mu_1, 1), (\mu_2, 2), (\mu_4, 3)\}$ is a valid clique, because the order is preserved. In the non-layered case, both cliques will be considered.

However, in some negative cases, it might take more time to detect that there exists no clique of desired size in a layered graph. For example, the graph on Figure 1(b) contains no clique of size 4. Following the prescribed order, the ALL-CLIQUES algorithm detects this case only after traversing through all four layers. Since there is no edge between the second and the fourth layer, the node $(\mu_4, 4)$ becomes invalid by the condition in line 1 of the ALL-CLIQUES algorithm. Whereas, in the non-layered counterpart of the graph, we could obey the order in which the nodes are visited, jump immediately to the node $(\mu_4, 4)$ and detect that the graph does not have any

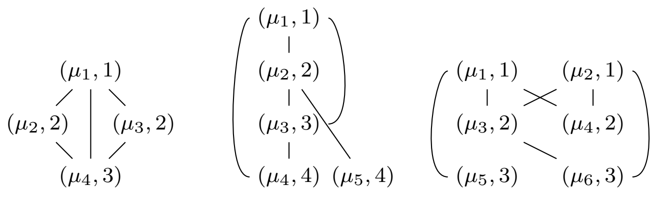

Figure 1. Several pruning techniques in ALL-CLIQUES.

cliques of size 4.

Second, once it is detected that a node has no edge to some layer, it is completely removed together with the respective edges from the substitution graph. In this way, ALL-CLIQUES assures that invalid nodes will be visited only once. Whereas CLIQUE will perform multiple visits. For example, in the graph on Figure 1(c), the node $(\mu_4, 2)$ will be visited twice by CLIQUE, because this node is a successor of both nodes on the first layer. Whereas the ALL-CLIQUES algorithm will hit the node $(\mu_4, 2)$ only once. After detecting that it has no edge to the third layer and, thus, is invalid by condition in line 1, this node will be deleted from the graph in line 9.

Third, in cases, when a layer contains a single node v only, the substitution graph can be pruned further. Since v is included in all cliques, those nodes, that are not strongly compatible[5] with v, should be removed.

Fourth, we have developed an idea of dynamic graph construction, i.e., the next layer is built only in the case, when the current layer contains some valid nodes. Because otherwise, we could immediately conclude that there exists no clique.

Fifth, graph context is applied in order to shrink the substitution graph further. Namely, all nodes $(\mu, i) \in V$, such that, for some $f \in Z_1$ and $g \in Z_2$, $f\mu = g$, and the context of f at some depth d is not included in the context of g at the same d, are removed from the substitution graph.

The context-based determinacy and ALL-CLIQUES are combined into an algorithm, referred to as ALLTHETA. In contrast to THETA, ALLTHETA delivers all substitutions for the AC1-SUBSUME(Z_1, Z_2) problem by employing ALL-CLIQUES instead of the CLIQUE algorithm. In more detail, ALL-CLIQUES is a modified version of CLIQUE, where additional pruning techniques have been developed in order to alleviate the search for substitutions. ALLTHETA is summarized in Algorithm 2. ALLTHETA inherits termination, correctness and completeness properties of ALL-CLIQUES. Its computational behaviour is evaluated in Section 4.

4 EXPERIMENTAL EVALUATION

We demonstrate the advantages of using the context information for efficient domain-independent symbolic reasoning in FOMDPs on a system, referred to as ALLTHETA. ALLTHETA has been recently integrated as a module into the FLUCAP 1.1 planning system, that is a successor of FLUCAP 1.0 [7] that has entered the probabilistic track of the International Planning Competition IPC'2004.

The experimental results were all obtained using a Linux RedHat machine running at 2.4 GHz Intel Celeron with 1 Gb of RAM.

In the comparison, we have used two datasets. One, referred to as CBW, stems from the colored Blocksworld planning scenario that

[5] Nodes are strongly compatible if the associated substitutions are.

Algorithm 2: ALLTHETA.

Input: Two abstract states Z_1, Z_2.
Output: All 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 any fluent of Z_1 does not match any fluent of Z_2, decide $Z_1 \nvdash_{\theta}^{AC1} Z_2$.
2. Context-based deterministically match as many fluents of Z_1 as possible to fluents of Z_2. Substitute Z_1 with the substitution found. If any 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. Delete all nodes (μ, i) with $f\mu = g$, for some $f \in Z_1$ and $g \in Z_2$, and $con(f, Z_1, d) \not\subseteq con(g, Z_2, d)$ for some d. Apply ALL-CLIQUES to search for all cliques of size $|Z_1|$ in (V, E).

was first introduced during the IPC'2004. CBW is, currently, one of a few probabilistic scenarios that are represented in first-order terms and, hence, enable to make use of grounding-free symbolic reasoning. The CBW problems differ from the classical ones in that, along with the unique identifier, each block is assigned a specific color. A goal formula, specified in first-order terms, provides an arrangement of colors instead of an arrangement of blocks.

The second dataset, referred to as RANDOM, consists of tests generated with a random states generator that has the following parameters: The number of states, the maximal number of fluents in a clause, the number of different predicates with varying arity, the number of constants and variables and some other.

Figure 2 presents the performance comparison results of ALLTHETA with the system FASTTHETA [5] on CBW and RANDOM datasets, respectively. FASTTHETA, that is motivated by Inductive Logic Programming (ILP), can be applied to compute all solutions of the AC1-subsumption problem. Therefore, FASTTHETA is an appropriate candidate for performing comparison.

However, there are some discrepancies in the input formats. Namely, FASTTHETA admits only those clauses C and D, where C is constant-free and D is variable-free. In order to cope with more general clauses C and D the authors recommend to accomplish several auxiliary operations (from private communication). First, to skolemize each variable X in D by replacing it with a new constant c_X. Second, for each constant c appearing in C, replace c with a new variable X_c and add a new literal $c(X_c)$ to C. Since the presence of constants in C brings FASTTHETA an additional overhead, in the RANDOM dataset we have generated clauses with a few number of constants. All problems in the CBW dataset contain a single constant t that stands for table.

In the following, we show how the usage of the context information in an abstract state affects the computation of all solutions of the AC1-subsumption problem. More precisely, we motivate the importance of the context depth parameter that a user may vary for a given problem.

There is a well-known tradeoff. The deeper inside the abstract state we look, thus devoting the considerable effort for computing the context itself, the higher the pruning rate is. Alternatively, if the depth value is underestimated, we save time and space for constructing the context but end up with a larger search space. We observe that for both CBW and RANDOM datasets, the optimal depth value varies from 2 to 3. It was also noticed in [13], that THETA has shown the best results at depth 2 on the mesh design dataset, that is a classical ILP benchmark framework.

Altogether, there are 100 abstract states that lead to 10000 subsumption tests. In RANDOM case, each problem is described by a number N of fluents in an abstract state. Whereas, in CBW case, two

B	C	Total time, sec.						
			AllTheta					
		FTheta	d=0	d=1	d=2	d=3	d=4	d=5
5	3	0.5	2.9	0.4	0.3	0.3	0.4	1.0
	4	0.4	2.0	0.3	0.2	0.2	0.3	0.6
	5	0.4	1.7	1.3	0.2	0.2	0.2	0.5
10	3	1.5	44.7	1.1	0.5	0.5	1.0	4.3
	4	1.1	22.4	1.1	0.4	0.4	0.5	1.4
	5	0.9	13.5	1.0	0.5	0.5	0.8	3.1
15	3	3.9	n/a	2.3	0.9	0.9	1.7	7.7
	4	3.5	243.3	2.4	0.8	0.9	2.0	10.6
	5	2.8	84.7	2.0	0.7	0.7	1.2	4.9
20	3	8.7	n/a	10.1	4.6	3.1	4.2	15.7
	4	9.2	n/a	3.3	1.1	1.0	1.8	8.5
	5	7.3	n/a	3.0	1.0	1.1	2.1	11.6
25	3	16.5	n/a	7.2	2.0	1.8	4.1	28.3
	4	17.1	n/a	7.8	1.8	1.7	4.2	30.7
	5	15.7	n/a	7.3	1.7	1.8	4.2	34.0
50	3	164.9	n/a	n/a	38.8	29.5	28.6	52.2
	4	201.1	n/a	186.8	33.0	26.0	27.9	42.7
	5	175.1	n/a	140.4	30.8	26.3	29.1	57.7
75	5	702.5	n/a	240.8	58.0	47.2	52.3	121.8
100	5	n/a	n/a	452.6	96.7	78.1	74.0	155.0

(a)

N	Total time, sec.						
		AllTheta					
	FTheta	d=0	d=1	d=2	d=3	d=4	d=5
20	0.9	5.2	1.2	1.0	1.6	5.6	52.5
30	4.5	3.9	1.1	0.9	1.2	4.2	35.5
40	356.5	23.6	1.4	1.3	1.8	6.6	58.3
50	n/a	7.9	1.7	1.7	2.1	6.4	52.4
100	n/a	1456.3	31.1	27.0	27.8	39.1	126.6
200	n/a	n/a	116.9	111.4	112.3	131.7	311.6
300	n/a	n/a	448.2	404.6	381.9	429.5	694.5
400	n/a	n/a	1139.5	1071.4	1083.0	1175.1	1519.1
500	n/a	n/a	n/a	n/a	n/a	n/a	n/a

(b)

Figure 2. Performance comparison of ALLTHETA (denoted as AllTheta) with FASTTHETA (denoted as FTheta) on (a) CBW and (b) RANDOM datasets.

parameters, i.e., a number of blocks B and a number of colors C, represent each problem. Since there are only two fluents in the CBW domain, i.e., $on(X, Y)$ that can be read as 'a block X is on a block Y', and a color fluent, say, $red(X)$, that states that X is red, there are twice as much as B fluents in an abstract state.

In both tables, the column labelled Total time presents the time in seconds needed to solve all of 10000 subsumption tests. A 30-mins block is allocated for each problem. Such time limit is fair enough. For example, during the IPC'2004, every contestant has been given a 15-mins block for 30 runs of each problem. The problems, runtime of which have exceeded the limit, are marked as n/a in a table.

In CBW case, on small problems of size up to 25 blocks, the depth parameter d posesses the optimal value of 2. Whereas, on larger problems, this value grows. This reflects the necessity to store an additional context information about the fluents in an abstract state. ALLTHETA employs the context information twice: Once, during the context-based deterministic subsumption test and secondly, while constructing the substitution graph for ALL-CLIQUES. For both cases, we defined a depth parameter d_1 and d_2. For the purpose of evaluation, we have chosen d_1=d_2=d, where d=0 means that no context information is taken into consideration.

In comparison to ALLTHETA, the runtime of FASTTHETA grows considerably faster in the size of a problem. For example, at depth of 2, for the five-colored 15, 25 and 75 problems, FASTTHETA is by factor of 4, 8 and 15 slower. As a result, it could scale to problems up to the size of 75 blocks only. Whereas, the limit of ALLTHETA comprises 360 blocks.

Neither FASTTHETA nor ALLTHETA are sensitive to the number of colors in a problem. In contrast, grounding-based reasoners are severely affected by this parameter. The timing results for a special case of d=0 demonstrate the dramatic loss in runtime in comparison even with the case of d=1, where the context information about the direct neighbours of a fluent is counted.

Additional tests for the depth of 2, that are not included in this paper due to the lack of space, have been carried out, in order to analyze the importance of the context information during the context-based deterministic subsumption and ALL-CLIQUES phases of ALLTHETA. The results show that the context knowledge is more vital for the former than for the latter. Namely, disregarding the context of fluents in the context-based deterministic subsumption phase yields an increase of the runtime by factor of 10.

Most importantly, present results indicate that the domain-independent inference algorithm ALLTHETA performs symbolic reasoning for first-order MDPs in about the same time as the domain-specific subsumption solver that was integrated in the FLUCAP 1.0 planning system [7]. We note that the latter reduces the AC1-subsumption problem to a quadratic variant of the subset problem. Whereas, the former solves the general case, which is NP-complete. For example, for a single subsumption test at depth of 2 in the problem of 15 blocks and 3 colors, ALLTHETA requires of about 92 microseconds. Whereas, for its domain-specific counterpart, the runtime comprises 85 microseconds.

In RANDOM case, the similar computational behaviour of ALLTHETA can be observed. However, random abstract states are more tricky in the sense, that for a fluent there is no guarantee at all of whether or not the context of a non-zero depth is non-empty. Whereas, CBW always provides such a guarantee.

During the ALL-CLIQUES phase of ALLTHETA, it is important that the construction of the substitution graph takes less time. This is assured by using the modern structure sharing techniques in building the context itself.

We have not performed the comparison with the original THETA algorithm because THETA delivers only a single solution for the AC1-subsumption problem. Whereas ALLTHETA computes all answers.

Finally, it is worth to pinpoint that FASTTHETA has outperformed ALLTHETA by a factor of four, on the Mutagenesis dataset that is a classical ILP benchmark that stems from the field of organic chemistry [5].

5 RELATED WORK

In [7], a domain-specific subsumption checker has been developed. Whereas, in this work, its domain-independent version is presented. In using symbolic reasoning for solving FOMDPs, our approach is related to that by Boutilier et al. [2] who employ the situation calculus language. To the best of our knowledge, due to the complexity of the language, neither complete implementation nor automated experiments have been reported. In contrast, our method is simpler and, thus, fully automated.

Some related approaches for AC1-subsumption are known. For example, Django [12] is, nowadays, the fastest θ-subsumption, and thus, AC1-subsumption, checker that is based on the constraints satisfaction. Yet, it returns a binary answer 'yes/no' only and provides no solutions, even in the positive case.

To the best of our knowledge, apart from ALLTHETA and FASTTHETA, there exists only one more approach, referred to as ReBel, that delivers a set of all solutions for the AC1-subsumption problem [10]. For this, the authors employ a generalized AC1-subsumption framework [3]. Moreover, our methods rely on different state semantics. ReBel considers abstract states as sets of fluents. Whereas, ALLTHETA can cope with abstract states that are provided with the multiset semantics, if the assumption, that each fluent may occur at most once in a state, is relaxed.

6 CONCLUSION AND FUTURE WORK

We have presented an algorithm, referred to as ALLTHETA, for performing automated domain-independent symbolic reasoning in FOMDPs. ALLTHETA is currently integrated in an algorithm for solving FOMDPs, referred to as FOLAO*[9], and has been recently included in a planning system FLUCAP 1.1 based on FOLAO*.

The usage of ALLTHETA for the purpose of symbolic reasoning and for FOLAO*, in particular, should be seen as two-fold. First, during the normalization phase, ALLTHETA is employed for detecting a more specific abstract state that can be pruned from the state space. Second, during the heuristic search, it is used to compute a set of all states that are reachable from an initial state wrt. all actions. In both cases, the computation is performed on action schemata directly avoiding their grounding.

In comparison to existing methods for symbolic reasoning that find all solutions for the AC1-subsumption problem, e.g., FASTTHETA, our approach scales better on larger FOMDPs. As results indicate, the domain-independent inference algorithm ALLTHETA performs symbolic reasoning for FOMDPs in about the same time as the domain-specific subsumption solver that was integrated in the FLUCAP 1.0 planning system [7]. This result is particularly valuable since the former solves the general AC1-subsumption problem, which is NP-complete.

However, there is plenty remaining to be done. One of our primary interests is extending the idea of using the context information for solving the AC1-unification problem that is thoroughly applied in FOLAO* for computing predecessor abstract states of a goal abstract state.

Planning domains with resources become more and more popular these days. To cover such domains, we should allow for multiple occurrences of fluents in an abstract state Z. In this case, each fluent f would be assigned with the so-called $degree(f) = k \geq 1$, which would mean that f occurs k times in Z. Then, each solution θ of ALLTHETA(Z_1, Z_2) should satisfy an additional condition: The degree of each fluent f in Z_2 is not exceeded by the total degree of all fluents g in Z_1 such that $g\theta = f$.

State constraints are most naturally specified by disequalities. To support these, both Z_1, Z_2 become equipped with sets E_1, E_2 of disequalities of the form $X \neq r$, where X is a variable and r is either a variable or a constant. Then a potential solution θ should satisfy the condition of that $E_1\theta$ does not contain a disequality $r \neq r$ for some r. Due to the graph nature of the approach, the substitutions are built incrementally. This, in turn, allows to detect incorrect θ's well before they are completely constructed. Moreover, we note that neither new data structures nor non-trivial add-on's are required for integrating multisets or disequalities.

ACKNOWLEDGEMENTS

We thank anonymous reviewers for useful comments. Many thanks to Stefano Ferilli and Nicola Di Mauro for the code of FASTTHETA and for their help in getting it work. Our joint discussions have greatly influenced the earlier versions of this paper. Olga Skvortsova was supported by the grant from the Graduate School GRK 334 under auspices of the German Research Foundation.

References

[1] C. Boutilier, T. Dean, and S. Hanks, 'Decision-theoretic planning: Structural Assumptions and Computational Leverage', *Journal of Artificial Intelligence Research*, **11**, 1–94, (1999).

[2] C. Boutilier, R. Reiter, and B. Price, 'Symbolic Dynamic Programming for First-Order MDPs', in *Proceedings of the Seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, ed., Bernhard Nebel, pp. 690–700. Morgan Kaufmann, (2001).

[3] W. Buntine, 'Generalized subsumption and its applications to induction and redundancy', *Artificial Intelligence*, **36**, 149–176, (1988).

[4] Z. Feng and E. Hansen, 'Symbolic heuristic search for factored markov decision processes', in *Eighteenth National Conference on Artificial Intelligence (AAAI-02*, pp. 455–460, Edmonton, Alberta, Canada, (2002).

[5] S. Ferilli, N. Di Mauro, T.M.A. Basile, and F. Esposito, 'A complete subsumption algorithm', in *AI*IA 2003: Advances in Artificial Intelligence*, eds., A. Cappelli and F. Turini, volume 2829 of *LNCS*, pp. 1–13. Springer Verlag, (2003).

[6] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier, 'SPUDD: Stochastic Planning using Decision Diagrams', in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 279–288, Stockholm, (1999).

[7] S. Hölldobler, E. Karabaev, and O. Skvortsova, 'FLUCAP: A heuristic search planner for first-order MDPs', *JAIR*, (2006). To appear.

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

[9] E. Karabaev and O. Skvortsova, 'A Heuristic Search Algorithm for Solving First-Order MDPs', in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI'2005)*, eds., F. Bacchus and T. Jaakkola, pp. 292–299, Edinburgh, Scotland, (July 2005). AUAI Press. ISBN-0-9749039-1-4.

[10] K. Kersting, M. van Otterlo, and L. de Raedt, 'Bellman goes relational', in *International Conference on Machine Learning*, pp. 465 – 472, Banff, (July 2004).

[11] J.-U. Kietz and M. Lübbe, 'An efficient subsumption algorithm for inductive logic programming', in *Proceedings of the Eleventh International Conference on MAchine Learning*, pp. 130–138, (1994).

[12] J. Maloberti and M. Sebag, 'Fast theta-subsumption with constraint satisfaction algorithms', *Machine Learning*, **55**(2), 137–174, (2004).

[13] 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*, pp. 212–228, Berlin, (August 1996). volume 1314 of LNAI.