A Logic-based Approach to Dynamic Programming

Steffen Hölldobler and **Olga Skvortsova***
Computer Science Department
Technische Universität Dresden
Dresden, Germany
{sh,skvortsova}@inf.tu-dresden.de

Abstract

We present a first-order value iteration algorithm that addresses the scalability problem of classical dynamic programming techniques by *logically* partitioning the state space. An MDP is represented in the Probabilistic Fluent Calculus, that is a first-order language for reasoning about actions. Moreover, we develop a normalization algorithm that discovers and prunes redundant states. We have implemented our approach and describe some experimental results.

Introduction

Markov decision processes (MDPs) have been adopted as a representational and computational model for decision-theoretic planning problems in much recent work, e.g., (Barto, Bradtke, & Singh 1995). However, classical dynamic programming (DP) algorithms for solving MDPs require explicit state and action enumeration. Therefore these algorithms do not scale up to large domains. Recently, following the idea of symbolic DP within the Situation Calculus (SC) by Boutilier and colleagues (Boutilier, Reiter, & Price 2001), we have developed an algorithm, that we refer to as first-order value iteration algorithm (FOVIA), that addresses the above scalability problem by dividing a state space into clusters, called *abstract states*, and computing the value functions for them thereafter (Großmann, Hölldobler, & Skvortsova 2002). The dynamics of an MDP is formalized in the Probabilistic Fluent Calculus, that extends the original version of the Fluent Calculus (Hölldobler & Schneeberger 1990) by introducing probabilistic effects. The Fluent Calculus (FC) is a first-order equational language for specifying actions, states and causality. Our approach constructs a first-order representation of value functions and policies by exploiting the logical structure of the MDP. Thus, FOVIA can be seen as a symbolic (logical) counterpart of the classical value iteration algorithm (Bellman 1957).

Symbolic dynamic programming approaches in SC as well as in FC rely on the normalization of the state space. Such normalization was done by hand so far. In (Skvortsova 2003) an automated normalization procedure has been developed that, given a state space, delivers an equivalent one that contains no redundancies. The technique employs the notion of a subsumption relation determining the redundant states which can be removed from the state space.

Our current prototypical implementation of FOVIA along with the normalization algorithm, referred to as FCPlanner, is tailored to the domains and problems that were specifically designed for the probabilistic track of the International Planning Competition'2004. The domains themselves together with the competition results will be released at the 14th International Conference on Automated Planning and Scheduling. Preliminary experiments which are described later indicate that on symbolic problems (where the goals are specified in a non-ground form) FCPlanner may outperform propositional MDP solvers that do rely on full domain and problem groundization. Whereas, on grounded problems, we expect that FCPlanner will not be as competitive as modern propositional MDP solvers like, e.g., SPUDD (Hoey *et al.* 1999) which employs a very efficient logical reasoning software. But the actual comparison results will be available only after the competition.

Probabilistic planning domains and problems in the aforementioned competition are expressed in PPDDL language (Younes & Littman 2003), that is an extension of PDDL (McDermott 1998) specifically designed to incorporate decision-theoretic notions. On the other hand, the dynamics of an MDP is specified within the Probabilistic Fluent Calculus (PFC). In order to incorporate these language discrepancies, we have developed a translation procedure, that given a generic PPDDL domain/problem description returns a PFC one, and implemented it in the FCPlanner.

Reasoning about Actions within FC

The Fluent Calculus, much like the Situation Calculus, is a logical approach to modelling dynamically changing systems based on first-order logic. One could indeed argue that the Fluent and the Situation Calculus have very much in common. But the latter has the following disadvantage: Knowledge of the current state is represented indirectly via the initial conditions and the actions which the agent has performed up to a point. As a consequence, each time a condition is evaluated in an agent program, the entire history of actions is involved in the computation (Thielscher

*Supported by the grant from GRK 334/3 (DFG). Corresponding author.

2004). The Fluent Calculus overcomes the aforementioned unfolding problem by providing an explicit state representation. The information about fluents in the current state of the world is effortlessly extracted from the state description without tracing back to the initial state.

Following the idea by (Green 1969) of planning as first-order theorem proving, FC was originally set up as a first-order logic program with equality using SLDE-resolution as sole inference rule (Hölldobler & Schneeberger 1990). In the meantime, FC has been revised as a predicate logic specification language using constraint handling rules for reasoning (Thielscher 2004). The original version allows for backward as well as forward reasoning. Classical DP algorithms for solving MDPs are intimately related to regression, and from this prospective, the original version of the Fluent Calculus appears to have a natural mechanism for approaching this problem.

In FC, functions whose values vary from state to state are called *fluents* and are denoted by function symbols. Throughout the paper, we will use examples taken from Blocksworld scenario. For instance, the fluent $on(a, b)$ denotes the fact that the block a is on the block b. A *state* is a multiset of fluents represented as a term, called *fluent term*, using a constant 1 denoting the empty multiset and a binary AC1-function symbol $\circ$ denoting multiset. For example, a state in which blocks a and b are on the table and a block c is on a is represented by

$$on(a, table) \circ on(b, table) \circ on(c, a) \ .$$

Constants are denoted by small letters, variables by capital ones and substitutions by θ or σ. All changes to the world are the result of *actions*. An action is represented using a predicate symbol *action* (P, N, E), whose arguments denote the preconditions (P), the name (N), and the effects (E) of an action, respectively. Similar to states, preconditions and effects are multisets of fluents represented as terms. As an example, consider the *move* action:

$$\textit{action}\,(on(X, Y), move(X, Y), holding(X)) \ .$$

Causality is represented using a predicate symbol *causes*$/3$, whose arguments denote a state, a sequence of actions, and a successor state, respectively. Intuitively, an atom such as *causes* (Z, P, Z') is to be understood as: The execution of a plan P transforms a state Z into a state Z'. The predicate *causes*$/3$ is defined recursively on the structure of plans, which are lists of actions.

The definition for *causes* together with the formulae representing actions as well as the AC1 equational theory comprise the FC theory.

Symbolic Dynamic Programming

Abstract states are characterized by means of conditions that must hold in each ground instance thereof and, thus, they represent sets of real-world states. Informally, abstract states can be specified by stating that particular fluent terms do or do not hold. We refer to such abstract states as *CN-states*, where C stands for conjunction and N for negation, respectively.

Formally, let $\mathcal{L}$ be a set of fluent terms. A *CN-state* is a pair $(P, \mathcal{N})$, where $P \in \mathcal{L}$, $\mathcal{N} \in 2^{\mathcal{L}}$. Let $\cdot^M$ be a mapping from fluent terms to multisets of fluents, which can be formally defined as follows: $1^M = \dot{\{\}}$ or $F^M = \dot{\{}F\dot{\}}$, if F is a fluent, or $(F \circ G)^M = F^M \,\dot\cup\, G^M$, where F, G are fluent terms and $\dot\cup$ is a multiset union. Let $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ be an interpretation, whose domain Δ is a set of all finite multisets of ground fluents and every *CN-state* $Z = (P, \mathcal{N})$ is mapped onto

$$Z^{\mathcal{I}} = \{d \in \Delta \mid \exists\theta.\, (P\theta)^M \,\dot\subseteq\, d \wedge \forall N \in \mathcal{N}.\forall\sigma.((N\theta)\sigma)^M \,\dot\not\subseteq\, d\} \ ,$$

where $\dot\subseteq$ is a submultiset relation.

In other words, the P-part of a state Z describes properties that a real-world state should satisfy, whereas $\mathcal{N}$-part specifies the properties that are not allowed to fulfil. For example, the *CN-state*

$$Z = (on(X, table) \circ red(X), \{on(Y, X)\})$$

represents all states in which there exists a red object that is on the table and clear, viz., none of other objects covers it.

Thus, the real-world state

$$z = \dot{\{}on(a, table), red(a), on(b, table), green(b)\dot{\}}$$

is specified by Z. Whereas,

$$z' = \dot{\{}on(a, table), red(a), on(b, a)\dot{\}}$$

is not.

Please note that *CN-states* should be thought of as incomplete state descriptions, i.e., the properties that are not listed in either P- or $\mathcal{N}$-part can hold or not.

Herein, we present the Probabilistic Fluent Calculus (PFC) that extends the original Fluent Calculus by decision-theoretic notions. For lack of space, we will only concentrate on the representation of stochastic actions in PFC. The technique used here is to decompose a stochastic action into deterministic primitives under nature's control, referred to as *nature's choices*. We use a relation symbol *choice*$/2$ to model nature's choice. Consider the action *move* (X, Y):

$$\textit{choice}\,(\textit{move}\,(X, Y), A) \leftrightarrow (A = \textit{moveS}\,(X, Y) \vee A = \textit{moveF}\,(X, Y)) \ ,$$

where *moveS* (X, Y) and *moveF* (X, Y) define two nature's choices for action *move* (X, Y), viz., that it is successfully executed or fails. For each of nature's choices $a_j(\overline{X})$ associated with an action $a(\overline{X})$ with parameters $\overline{X}$ we define the probability *prob* $(a_j(\overline{X}), a(\overline{X}), Z)$ denoting the probability with which one of nature's choices $a_j(\overline{X})$ is chosen in a state Z. For example,

$$\textit{prob}\,(\textit{moveS}\,(X, Y), \textit{move}\,(X, Y), Z) = .75$$

states that the probability for the successful execution of the *move* action in state Z is $.75$.

FOVIA is an iterative approximation algorithm for constructing optimal policies. The difference to the classical case is that it produces a first-order representation of optimal policies by utilizing the logical structure of MDP. The algorithm itself can be found in (Großmann, Hölldobler, & Skvortsova 2002).

Normalizing State Descriptions

The regression in PFC is realized by using a fluent unification problem of the form $(F \circ X)\sigma =_{AC1} G\sigma$, where F, G are fluent terms, AC1 is an equational theory and X is a variable not occurring in F or G. Since it is known that a fluent unification problem may have more than one solution (Große *et al.* 1992), the regression of a single abstract state may yield several abstract states. Moreover, the state space obtained after regression may contain several redundancies that proliferate exponentially many unnecessary computations in subsequent steps of FOVIA. We have developed a normalization algorithm that normalizes the state space obtained after regression and delivers an equivalent state space that is free of redundancies.

Consider a set S of *CN-states* and let $Z \in S$. Informally, Z is redundant iff the set of states represented by Z is contained in the set of states represented by another member of S. This kind of redundancy can be captured by subsumption: Let P_1, P_2 be fluent terms. P_1 is said to *subsume* P_2, written $P_2 \sqsubseteq_f P_1$, iff $\exists\sigma.\, (P_1\sigma)^M \doteq P_2^M$. Let $Z_1 = (P_1, N_1)$ and $Z_2 = (P_2, N_2)$ be *CN-states*. Then Z_1 is said to *subsume* Z_2, written $Z_2 \sqsubseteq Z_1$, iff $P_2 \sqsubseteq_f P_1$ and $\forall F \in N_1.\, \exists G \in N_2.\, F \sqsubseteq_f G$. One should note that our notion of $\sqsubseteq$ bears the intuition of the truth ordering $\leq_t$ in a bilattice $\langle C \times D, \leq_k, \leq_t\rangle$, where C and D are complete lattices and $\leq_k$ is knowledge ordering (Ginsberg 1988).

In order to illustrate the definition of the subsumption relation on *CN-states* we use an example taken from logistics scenario, because it is more representative than a blocksworld example. Consider two *CN-states* $Z_1 = (P_1, N_1)$ with $P_1 = on(R, T)$ and $N_1 = \{rin(f, m) \circ rain\}$ as well as $Z_2 = (P_2, N_2)$ with $P_2 = on(f, T) \circ tin(T, m)$ and $N_2 = \{rin(f, C)\}$. In this case, $P_2 \sqsubseteq_f P_1$ with the help of $\sigma = \{R \mapsto f\}$ and $F_1 \sqsubseteq_f F_2$ with the help of $\theta = \{C \mapsto m\}$, where $F_1 = rin(f, m) \circ rain$ and $F_2 = rin(f, C)$. Thus, $Z_2 \sqsubseteq Z_1$.

We use the notion of system to represent in a compact way *CN-states* along with their values. Formally, a *system* $\mathcal{S}$ is a multiset of pairs $\langle Z, \alpha\rangle$, where Z is a *CN-state* and α is its value. For instance, in our logistics example, a system $\mathcal{S}$ defined as

$$\dot{\{}\langle Z_1' = (rin(f, m), \emptyset),\ 10\rangle,\ \langle Z_2' = (1, \{rin(f, m)\}),\ 0\rangle\dot{\}}$$

describes symbolically a goal state space that consists of two *CN-states* Z_1' and Z_2' together with their values: States in which the Ferrari (f) is in Monte Carlo (m)receive a reward of 10; all other states receive a reward of 0. The regression of system S through the action $unload(R, T)$ results in a system $\mathcal{S}_1$:

$$\begin{array}{l}\dot{\{}\ \langle Z_1 = (on(f, T) \circ tin(T, m), \{rin(f, m)\}),\ 0\rangle,\\ \quad\langle Z_2 = (on(R, T) \circ tin(T, C) \circ rin(f, m), \emptyset),\ 10\rangle,\\ \quad\langle Z_3 = (on(R, T) \circ tin(T, C),\\ \qquad\{rin(f, m), on(f, T) \circ tin(T, m)\}),\ 0\rangle,\\ \quad\langle Z_4 = (on(R, T) \circ tin(T, C) \circ rin(f, m), \emptyset),\ 10\rangle,\\ \quad\langle Z_5 = (on(R, T) \circ tin(T, C), \{rin(f, m)\}),\ 0\rangle\ \dot{\}}.\end{array}$$

System $\mathcal{S}_1$ contains several redundancies: $Z_2 \sqsubseteq Z_4$ (as well as $Z_4 \sqsubseteq Z_2$); $Z_1 \sqsubseteq Z_5$; $Z_3 \sqsubseteq Z_5$. Hence, Z_1, Z_2 and Z_3 are subsumed and, thus, are redundant. They should be removed from the system. Please note that the comparison of *CN-states* makes only sense in case when the respective values are identical. For instance, the states Z_1 and Z_2 in $\mathcal{S}_1$ are not comparable because their values are different. The aforementioned intuition for determining redundant *CN-states* is reflected in the first rule in Figure 1. Namely, if two *CN-states* Z, Z' have identical values and $Z' \sqsubseteq Z$ then the pair $\langle Z', \alpha\rangle$ ought to be removed from the system. The second rule in Figure 1 removes redundancies in negative parts of single abstract states. Intuitively, the application of a simplification rule should not alter the meaning of a system. Systems $\mathcal{S}_1$ and $\mathcal{S}_2$ are said to be *equivalent* iff for each pair $\langle Z_i, \alpha_i\rangle \in \mathcal{S}_1$ there exists a pair $\langle Z_j, \alpha_i\rangle \in \mathcal{S}_2$ such that $Z_i \sqsubseteq Z_j$, and vice versa. In addition, we want to apply the simplification rules as long as possible: A system is said to be in *normal form* if none of the rules shown in Figure 1 is applicable.

Let $\mathcal{S}$ be a system such that $\langle Z, \alpha\rangle, \langle Z', \alpha\rangle \in \mathcal{S}$.

1. $$\frac{\langle Z, \alpha\rangle \quad \langle Z', \alpha\rangle}{\langle Z, \alpha\rangle}\ Z' \sqsubseteq Z$$

2. Let $Z = (P, \{F_1, F_2, \ldots, F_k\})$.
$$\frac{\langle (P, \{F_1, F_2, \ldots, F_k\}), \alpha\rangle}{\langle (P, \{F_2, \ldots, F_k\}), \alpha\rangle}\ F_1 \sqsubseteq_f F_2$$

Figure 1: Simplification rules for *CN-states*.

Returning to the running example, after applying the normalization algorithm to the system $\mathcal{S}_1$ we obtain the system $\mathcal{S}_2$:

$$\begin{array}{l}\dot{\{}\ \langle Z_4 = (on(R, T) \circ tin(T, C) \circ rin(f, m), \emptyset),\ 10\rangle,\\ \quad\langle Z_5 = (on(R, T) \circ tin(T, C), \{rin(f, m)\}),\ 0\rangle\ \dot{\}}.\end{array}$$

Some useful properties of the normalization algorithm, including termination, correctness, completeness as well as the uniqueness result have been proven (see (Skvortsova 2003)).

PPDDL and the Probabilistic Fluent Calculus

We have implemented the first-order value iteration algorithm FOVIA including the normalization procedure that was presented in the previous section. Our current implementation, that is referred to as FCPlanner, is targeted to the domains that were designed for the probabilistic track of the International Planning Competition'2004. The domains themselves are represented in PPDDL (Younes & Littman 2003) and will be released at the 14th International Conference on Automated Planning and Scheduling.

Because the dynamics of an MDP, that is an input of FOVIA, is formalized within PFC, a translation procedure, that given a PPDDL description of an MDP returns an equivalent PFC one, is required. For some PPDDL constructs, e.g., objects, sorts, predicates, the translation is quite

`Input:` PPDDL formula F
`Output:` $\mathcal{S}$, a set of *CN-states*

1. $\mathcal{S} := \emptyset$.
2. Convert F into the negation normal form F'.
3. Convert F' into the prenex normal form F''.
4. Convert the matrix of F'' into DNF $D = d_1 \vee d_2 \vee \ldots \vee d_n$.
5. Let d be the first disjunct of D.
6. If d contains bad variables then eliminate them from D via groundization obtaining D'. $F := D'$. Go to 2.
7. Otherwise, construct a *CN-state* $Z = (P, N)$, where P is a fluent term that is built from the positive conjuncts in d and N is a set of fluent terms that correspond to the negative conjuncts in d.
8. $\mathcal{S} := \mathcal{S} \cup \{Z\}$.
9. If $n > 1$ then $D := d_2 \vee \ldots \vee d_n$. Go to 5.
10. Return $\mathcal{S}$.

Figure 2: Translation procedure for PPDDL formulae.

straightforward. Whereas goal descriptions as well as action preconditions and effects should be treated more carefully.

We start with presenting how a PPDDL goal description can be translated into a respective PFC one. Given a PPDDL formula F, that is a first-order formula without function symbols, plus domain objects with sorts, the translation procedure delivers a set of *CN-states* $\mathcal{S}$, such that $\mathcal{S}$ is equivalent to F in the sense that $\mathcal{M}$ is a model for F iff $\mathcal{M}$ is a model for some Z from $\mathcal{S}$, and is minimal wrt the set inclusion. The translation procedure itself is depicted on Figure 2.

In the sixth step of the translation procedure, a disjunct d, which is a conjunction of literals, is checked against bad variables. The 'bad variables check' is performed as follows. Let V^+ denote variables that occur in positive literals of a disjunct and V^- contain all variables that occur in negative literals and do not occur in positive ones. Variables of V^+ that are existentially quantified are marked as good. All other variables of V^+ are marked as bad. A variable from V^-, that is bounded universally and, most importantly, after all good variables from V^+, is marked as good. All other variables in V^- are marked as bad. For example, consider a disjunct $d = \exists X.(on(X, table) \wedge \forall Y.\neg on(Y, X))$. Both variables X and Y will be marked as good. As it was intended, the intuition behind the 'bad variable check' precisely coincides with the semantics of variables in *CN-states* that is defined in section on Symbolic Dynamic Programming.

Because action preconditions are expressed as PPDDL formulae, the translation procedure on Figure 2 can be directly applied for them. Action effects, however, need an additional treatment. PPDDL actions may have several kinds of effects. First of all, the effect can be an atomic formula. Secondly, conjunctive effects are allowed, viz., of the form `<effect1> and <effect2>`. Thirdly, the effect may represent a negation of the atomic formula. The fourth, fifth, and sixth cases correspond to conditional, universal, and probabilistic effects, respectively. The first three cases are obvious. Conditional effects of the form:

:action Name
Pre : P
Eff : `when` C $E_1 \wedge$ E_2

will be translated into two actions

:action Name$_1$	:action Name$_2$
Pre : $P \wedge C$	Pre : $P \wedge \neg C$
Eff : $E_1 \wedge E_2$	Eff : E_2 .

For the sake of readability, we use an informal syntax which is different from PPDDL syntax. Each of the universal quantifiers should be eliminated via groundization.

Probabilistic effects of the form:

:action Name
Pre : P
Eff : $E_0 \wedge$ `probabilistic` p_1 E_1 p_2 E_2

are translated into three actions:

:action Name$_1$	:action Name$_2$	:action Name$_3$
Pre : P	Pre : P	Pre : P
Eff : $E_0 \wedge E_1$	Eff : $E_0 \wedge E_2$	Eff : E_0

The effects of actions Name$_1$, Name$_2$, and Name$_3$ will occur with probabilities p_1, p_2, and $1 - p_1 - p_2$, respectively.

As a result, instead of an action with complex effects we obtain a set of actions with primitive effects, where 'primitive' stands for conjunction of literals. Positive literals are additive effects, whereas negative literals are subtractive ones. The current version of FCPlanner implements the aforementioned translation procedure that enables to process a generic PPDDL domain/problem specification.

Some Experimental Results

The experimental results described in this section were all obtained using a Linux Red Hat machine running at 2.7GHz Pentium IV with 2Gb of RAM.

The colored Blocksworld scenario was one of the symbolic domains that were made available to participants prior to the competition and where a goal is specified in a non-ground form. Herein, we present some timing results that characterize the best- and worst-case computational behaviour of our planner on the examples taken from the colored Blocksworld scenario. In the colored Blocksworld scenario, along with the unique number, each block is assigned a specific color. A goal formula specifies an arrangement of colors instead of an arrangement of blocks. In other words, a symbolic goal state description represents an equivalence class of the grounded goal state descriptions that have the same color distribution.

For example, consider the initial situation that contains two blue blocks b_1, b_2, and one red block b_3 that are on the table. The symbolic state description, that is of the form

$(\exists X_1.\mathit{is\text{-}blue}(X_1) \wedge$
$(\exists X_2.\mathit{is\text{-}red}(X_2) \wedge on(X_1, X_2) \wedge$
$(\exists X_3.\mathit{is\text{-}blue}(X_3) \wedge on(X_2, X_3) \wedge on(X_3, table))))$,

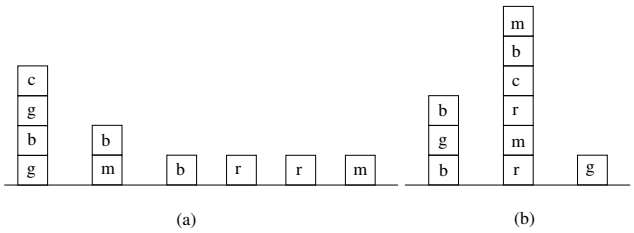

Figure 3: Descriptions of initial (a) and goal (b) states. Each block is assigned one of five colors, namely red (r), cyan (c), magenta (m), green (g), or blue (b).

represents an equivalence class that consists of two distinct grounded state descriptions that resemble the same color arrangement as in the symbolic representation, namely

$$\textit{is-blue}(b_1) \ \wedge \textit{is-red}(b_3) \ \wedge \textit{on}(b_1, b_3) \wedge \\ \textit{is-blue}(b_2) \wedge \textit{on}(b_3, b_2) \wedge \textit{on}(b_2, \textit{table})$$

and

$$\textit{is-blue}(b_2) \ \wedge \textit{is-red}(b_3) \ \wedge \textit{on}(b_2, b_3) \wedge \\ \textit{is-blue}(b_1) \wedge \textit{on}(b_3, b_1) \ \wedge \textit{on}(b_1, \textit{table}) \ .$$

Such sort of problems, where the goal descriptions do not put restrictions on the objects of the domain, but rather represent a combination of object properties (like, e.g., colors), requires the first-order expressiveness of the language for describing an MDP.

Why do we think that FCPlanner will demonstrate the better computational behaviour on the symbolic problems, e.g., colored Blocksworld? The dynamics of an MDP in FCPlanner is represented within PFC. The first-order nature of the Fluent Calculus enables to specify the (goal) states symbolically, namely the clusters of ground states are represented as *CN-states*, and hence, avoid the full state space propositionalization. On the other hand, in order to solve a symbolically represented problem, a propositional solver would require to consider all possible ground combinations of the problem. For example, in the scenario, where both the initial and goal state specifications contain ten red blocks, a symbolic description of a state that represents a pyramid of ten red blocks results into $10! = 3628800$ its ground instances.

Because our implementation is tailored to the competition domains and they were made known in advance, we were able to tune our normalization algorithm based on this information. For example, the goal formulae generated by the competition problem generator contain the information about all blocks from the domain. The state descriptions, that are obtained after regression, contain no negative literals either. Thus, the negative parts of *CN-states* can be omitted. Consequently, the second rule on Figure 1 becomes inapplicable, which means that the computation process at the normalization step of FOVIA will be much simpler. One should note that it is not true in general. An arbitrary state description may contain positive as well as negative literals which makes the second rule of the normalization algorithm applicable again.

Moreover, in the (colored) Blocksworld scenario, the states are described in such a way that makes it possible to apply a sort of structural subsumption algorithm of quadratic worst-case complexity instead of general subsumption algorithm which is based on solving NP-hard submultiset matching problem. This results in considerable computational savings on the normalization step of FOVIA.

In general, as a worst-case example for FCPlanner, one could take an example, where the number of blocks is equal to the number of colors and no two blocks have the same color. This example corresponds to extreme case, when a colored (symbolic) blocksworld problem degenerates into its non-colored (grounded) counterpart. Such an example will not serve as a representative for analyzing the worst-case behaviour of our planner on the *colored* Blocksworld problems. Therefore, as a worst-case example we have chosen an example with five (maximum number for the competition domains) colors and ten blocks. In order to illustrate the best-case behaviour of FCPlanner on the colored Blocksworld domains, we use an example with ten blocks of the same color. The initial and goal state descriptions for the worst-case scenario are depicted on Figure 3.

Some representative timing results for the first ten iterations of the FOVIA algorithm can be found in Table 1. For each iteration, the size of the state space at the regression step $\mathcal{S}$, after the regression step $\mathcal{S}_{\text{REGR}}$, and after the normalization step $\mathcal{S}_{\text{NORM}}$ as well as the timing results for the regression REGR, normalization NORM and v-values computation VALUES procedures are depicted. In each row, the results for both best- and worst-case examples are presented; the first line of each Ith iteration corresponds to the best-case scenario, whereas the second - to the worst case.

In the best-case example, the state space growth stabilizes with the number of iterations. Whereas for the worst-case, the state space grows exponentially in a number of iterations. Normalization grants the decrease in the size of the state space $\mathcal{S}_{\text{REGR}}$ obtained after the regression. E.g., on the 7th iteration in the best-case example the normalization coefficient γ, i.e., $\frac{\mathcal{S}_{\text{REGR}}}{\mathcal{S}_{\text{NORM}}}$, is equal to 11. Whereas, for the worst-case scenario it approaches 4. In addition, for the best-case scenario, γ increases as the state space grows. E.g., on 5th iteration, γ is equal to 8, whereas on the 9th it already approaches 14.

If we calculate the total time (REGR plus NORM plus VALUES) that FCPlanner with the normalization switched on has to spend during the first six iterations (for the worst-case scenario) and compare it with the total time (REGR plus VALUES) that FCPlanner with the normalization switched off has to spend during the same six iterations, then it turns out that the normalization brings the gain of about three orders of magnitude.

At the time of writing, we can make the following conclusions about the best- and worst-case computational behaviour of FCPlanner. First, having two problems with the same number of colors, the one with the larger number of blocks is harder to solve. This conclusion is quite obvious and therefore we present no representative timing results. Second, having two problems with the same number of blocks, the one with the larger number of colors is harder to solve. Table 1 illustrates exactly this case.

I	Number of states			Time, msec		
	$\mathcal{S}$	$\mathcal{S}_{\text{REGR}}$	$\mathcal{S}_{\text{NORM}}$	REGR	NORM	VALUES
0	1	9	6	47	1	97
	1	9	6	47	1	97
1	6	24	14	204	3	189
	6	24	14	204	3	189
2	14	94	23	561	12	323
	14	94	39	561	11	561
3	23	129	33	885	16	492
	39	203	82	1491	29	1250
4	33	328	39	1473	46	606
	82	652	208	3584	167	3301
5	39	361	48	1740	51	779
	208	1449	434	8869	614	7839
6	48	604	52	2340	107	928
	434	3634	962	19359	2981	22299
7	52	627	54	2573	110	961
	962	7608	2029	44512	12166	89378
8	54	795	56	2799	157	1074
	2029	18090	4407	104567	54747	279512
9	56	811	59	2965	154	1166
	4407	36720	9415	245647	238697	894438

Table 1: Representative timing results for first ten iterations of FOVIA.

Questions like 'if one problem contains more blocks than the another one, and the second problem contains more colors than the first, on which of these problems FCPlanner will demonstrate the better behaviour?', have not yet been answered but are considered as a next step in evaluating the FCPlanner. Another interesting class of problems that FCPlanner should be tested on are the problems where the goal contains (much) less objects than are present in the domain. Our preliminary investigations show that it would require the introduction of new variables in the domain. In general, FCPlanner supports this feature, but for the sake of the competition (competition problems introduce no new variables), it was disabled. We believe, that FCPlanner may outperform modern propositional MDP solvers on problems that require new variables, but an extensive analysis is involved at this point. We expect to obtain some of these and other (especially, comparison with similar approaches) evaluation results after the competition.

Acknowledgements

We would like to thank Axel Großmann for his valuable comments on the previous versions of the paper. We also deeply appreciate Eldar Karabaev for his hard work in designing and coding the FCPlanner. Many thanks to all anonymous referees for their helpful suggestions.

References

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to Act Using Real-Time Dynamic Programming. *Artificial Intelligence* 72(1-2):81–138.

Bellman, R. E. 1957. *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press.

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

Ginsberg, M. 1988. Multivalued Logics: A Uniform Approach to Inference in Artificial Intelligence. *Computational Intelligence* 4(3).

Green, C. 1969. Application of theorem proving to problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 219–239. Morgan Kaufmann Publishers.

Große, G.; Hölldobler, S.; Schneeberger, J.; Sigmund, U.; and Thielscher, M. 1992. Equational logic programming, actions, and change. In Apt, K., ed., *IJCSLP*, 177–191. MIT Press.

Großmann, A.; Hölldobler, S.; and Skvortsova, O. 2002. Symbolic Dynamic Programming within the Fluent Calculus. In Ishii, N., ed., *Proceedings of the IASTED International Conference on Artificial and Computational Intelligence*, 378–383. Tokyo, Japan: ACTA Press.

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

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

McDermott, D. 1998. PDDL - the planning domain definition language. Technical Report 1165, Yale University, Department of Computer Science.

Skvortsova, O. 2003. Towards Automated Symbolic Dynamic Programming. Master's thesis, TU Dresden.

Thielscher, M. 2004. FLUX: A logic programming method for reasoning agents. *Theory and practive of Logic Programming*.

Younes, H., and Littman, M. 2003. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling*. To appear.