

PROBABILISTIC PLANNING

PROBABILISTIC PLANNING

ABSTRACTION, TRANSFER

Generalizing Plans to New Environments in Relational MDPs

Carlos Guestrin Daphne Koller Chris Gearhart Neal Kanodia

Computer Science Department, Stanford University

{guestrin, koller, cmg33, nkanodia}@cs.stanford.edu

Abstract

A longstanding goal in planning research is the ability to generalize plans developed for some set of environments to a new but similar environment, with minimal or no replanning. Such generalization can both reduce planning time and allow us to tackle larger domains than the ones tractable for direct planning. In this paper, we present an approach to the generalization problem based on a new framework of *relational Markov Decision Processes (RMDPs)*. An RMDP can model a set of similar environments by representing objects as instances of different classes. In order to generalize plans to multiple environments, we define an approximate value function specified in terms of classes of objects and, in a multiagent setting, by classes of agents. This class-based approximate value function is optimized relative to a sampled subset of environments, and computed using an efficient linear programming method. We prove that a polynomial number of sampled environments suffices to achieve performance close to the performance achievable when optimizing over the entire space. Our experimental results show that our method generalizes plans successfully to new, significantly larger, environments, with minimal loss of performance relative to environment-specific planning. We demonstrate our approach on a real strategic computer war game.

1 Introduction

Most planning methods optimize the plan of an agent in a fixed environment. However, in many real-world settings, an agent will face multiple environments over its lifetime, and its experience with one environment should help it to perform well in another, even with minimal or no replanning.

Consider, for example, an agent designed to play a strategic computer war game, such as the *Freecraft* game shown in Fig. 1 (an open source version of the popular *Warcraft* game). In this game, the agent is faced with many scenarios. In each scenario, it must control a set of agents (or units) with different skills in order to defeat an opponent. Most scenarios share the same basic elements: *resources*, such as gold and wood; *units*, such as peasants, who collect resources and build structures, and footmen, who fight with enemy units; and *structures*, such as barracks, which are used to train footmen. Each scenario is composed of these same basic building blocks, but they differ in terms of the map layout, types of units available, amounts of resources, etc. We would like the agent to learn from its experience with playing some scenarios, enabling it to tackle new scenarios without significant amounts of replanning. In particular, we would like the agent to generalize from simple scenarios, allowing it to deal with other scenarios that are too complex for any effective planner.

The idea of generalization has been a longstanding goal in Markov Decision Process (MDP) and reinforcement learning

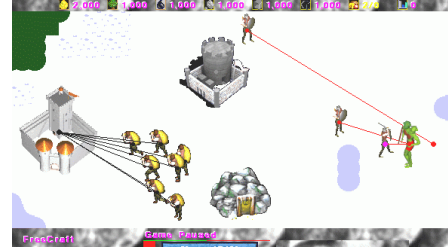


Figure 1: Freecraft strategic domain with 9 peasants, a barrack, a castle, a forest, a gold mine, 3 footmen, and an enemy, executing the generalized policy computed by our algorithm.

research [15; 16], and even earlier in traditional planning [5]. This problem is a challenging one, because it is often unclear how to translate the solution obtained for one domain to another. MDP solutions assign values and/or actions to states. Two different MDPs (e.g., two Freecraft scenarios), are typically quite different, in that they have a different set (and even number) of states and actions. In cases such as this, the mapping of one solution to another is not well-defined.

Our approach is based on the insight that many domains can be described in terms of objects and the relations between them. A particular domain will involve multiple objects from several classes. Different tasks in the same domain will typically involve different sets of objects, related to each other in different ways. For example, in Freecraft, different tasks might involve different numbers of peasants, footmen, enemies, etc. We therefore define a notion of a *relational MDP (RMDP)*, based on the *probabilistic relational model (PRM)* framework [10]. An RMDP for a particular domain provides a general schema for an entire suite of environments, or worlds, in that domain. It specifies a set of classes, and how the dynamics and rewards of an object in a given class depend on the state of that object and of related objects.

We use the class structure of the RMDP to define a value function that can be generalized from one domain to another. We begin with the assumption that the value function can be well-approximated as a sum of *value subfunctions* for the different objects in the domain. Thus, the value of a global Freecraft state is approximated as a sum of terms corresponding to the state of individual peasants, footmen, gold, etc. We then assume that individual objects in the same class have a very similar value function. Thus, we define the notion of a *class-based value function*, where each class is associated with a *class subfunction*. All objects in the same class have the value subfunction of their class, and the overall value function for a particular environment is the sum of value subfunctions for the individual objects in the domain.

A set of value subfunctions for the different classes imme-

diately determines a value function for any new environment in the domain, and can be used for acting. Thus, we can compute a set of class subfunctions based on a subset of environments, and apply them to another one without replanning.

We provide an optimality criterion for evaluating a class-based value function for a distribution over environments, and show how it can, in principle, be optimized using a linear program. We can also “learn” a value function by optimizing it relative to a sample of environments encountered by the agent. We prove that a polynomial number of sampled environments suffice to construct a class-based value function which is close to the one obtainable for the entire distribution over environments. Finally, we show how we can improve the quality of our approximation by automatically discovering subclasses of objects that have “similar” value functions.

We present experiments for a computer systems administration task and two Freecraft tasks. Our results show that we can successfully generalize class-based value functions. Importantly, our approach also obtains effective policies for problems significantly larger than our planning algorithm could handle otherwise.

2 Relational Markov Decision Processes

A *relational MDP* defines the system dynamics and rewards at the level of a template for a task domain. Given a particular environment within that domain, it defines a specific MDP instantiated for that environment. As in the PRM framework of [10], the domain is defined via a *schema*, which specifies a set of *object classes* $\mathcal{C} = \{C_1, \dots, C_c\}$. Each class C is also associated with a set of *state variables* $S[C] = \{C.S_1, \dots, C.S_k\}$, which describe the state of an object in that class. Each state variable $C.S$ has a *domain* of possible values $\text{Dom}[C.S]$. We define S_C to be the set of possible states for an object in C , i.e., the possible assignments to the state variables of C .

For example, our Freecraft domain might have classes such as *Peasant*, *Footman*, *Gold*; the class *Peasant* may have a state variable *Task* whose domain is $\text{Dom}[\text{Peasant.Task}] = \{\text{Waiting}, \text{Mining}, \text{Harvesting}, \text{Building}\}$, and a state variable *Health* whose domain has three values. In this case, S_{Peasant} would have $4 \cdot 3 = 12$ values, one for each combination of values for *Task* and *Health*.

The schema also specifies a set of *links* $\mathcal{L}[C] = \{L_1, \dots, L_l\}$ for each class representing links between objects in the domain. Each link $C.L$ has a *range* $\rho[C.L] = C'$. For example, *Peasant* objects might be linked to *Barrack* objects — $\rho[\text{Peasant.BuildTarget}] = \text{Barrack}$, and to the global *Gold* and *Wood* resource objects. In a more complex situation, a link may relate C to many instances of a class C' , which we denote by $\rho[C.L] = \{C'\}$, for example, $\rho[\text{Enemy.My.Footmen}] = \{\text{Footman}\}$ indicates that an instance of the enemy class may be related to many footman instances.

A particular instance of the schema is defined via a *world* ω , specifying the set of objects of each class; we use $\mathcal{O}[\omega][C]$ to denote the objects in class C , and $\mathcal{O}[\omega]$ to denote the total set of objects in ω . The world ω also specifies the links between objects, which we take to be fixed throughout time. Thus, for each link $C.L$, and for each

$o \in \mathcal{O}[\omega][C]$, ω specifies a set of objects $o' \in \rho[C.L]$, denoted $o.L$. For example, in a world containing 2 peasants, we would have $\mathcal{O}[\omega][\text{Peasant}] = \{\text{Peasant1}, \text{Peasant2}\}$; if *Peasant1* is building a barracks, we would have that $\text{Peasant1.BuildTarget} = \text{Barrack1}$.

The dynamics and rewards of an RMDP are also defined at the schema level. For each class, the schema specifies an *action* $C.A$, which can take on one of several values $\text{Dom}[C.A]$. For example, $\text{Dom}[\text{Peasant.A}] = \{\text{Wait}, \text{Mine}, \text{Harvest}, \text{Build}\}$. Each class C is also associated with a *transition model* P^C , which specifies the probability distribution over the next state of an object o in class C , given the current state of o , the action taken on o , and the states and actions of all of the objects linked to o :

$$P^C(S'_C \mid S_C, C.A, S_{C.L_1}, C.L_1.A, \dots, S_{C.L_l}, C.L_l.A). \quad (1)$$

For example, the status of a barrack, $\text{Barrack.Status}'$, depends on its status in the previous time step, on the task performed by any peasant that could build it ($\text{Barrack.BuiltBy.Task}$), on the amount of wood and gold, etc.

The transition model is conditioned on the state of $C.L_i$, which is, in general, an entire set of objects (e.g., the set of peasants linked to a barrack). Thus we must now provide a compact specification of the transition model that can depend on the state of an unbounded number of variables. We can deal with this issue using the idea of *aggregation* [10]. In Freecraft, our model uses the *count* aggregator $\sharp$, where the probability that Barrack.Status transitions from *Unbuilt* to *Built* depends on $\sharp[\text{Barrack.BuiltBy.Task} = \text{Build}]$, the number of peasants in Barrack.BuiltBy whose *Task* is *Build*.

Finally, we also define rewards at the class level. We assume for simplicity that rewards are associated only with the states of individual objects; adding more global dependencies is possible, but complicates planning significantly. We define a reward function $R^C(S_C, C.A)$, which represents the contribution to the reward of any object in C . For example, we may have a reward function associated with the *Enemy* class, which specifies a reward of 10 if the state of an enemy object is *Dead*: $R^{\text{Enemy}}(\text{Enemy.State} = \text{Dead}) = 10$. We assume that the reward for *each object* is bounded by $R_{\max}$.

Given a world, the RMDP uniquely defines a *ground* factored MDP Π_ω , whose transition model is specified (as usual) as a dynamic Bayesian network (DBN) [3]. The random variables in this factored MDP are the state variables of the individual objects $o.S$, for each $o \in \mathcal{O}[\omega][C]$ and for each $S \in S[C]$. Thus, the state s of the system at a given point in time is a vector defining the states of the individual objects in the world. For any subset of variables X in the model, we define $s[X]$ to be the part of the instantiation s that corresponds to the variables X . The ground DBN for the transition dynamics specifies the dependence of the variables at time $t+1$ on the variables at time t . The parents of a variable $o.S$ are the state variables of the objects o' that are linked to o . In our example with the two peasants, we might have the random variables Peasant1.Task , Peasant2.Task , Barrack1.Status , etc. The parents of the time $t+1$ variable $\text{Barrack1.Status}'$ are the time t variables $\text{Barrack1.Status}'$, Peasant1.Task , Peasant2.Task , Gold1.Amount and Wood1.Amount .

The transition model is the same for all instances in the same class, as in (1). Thus, all of the $o.Status$ variables for

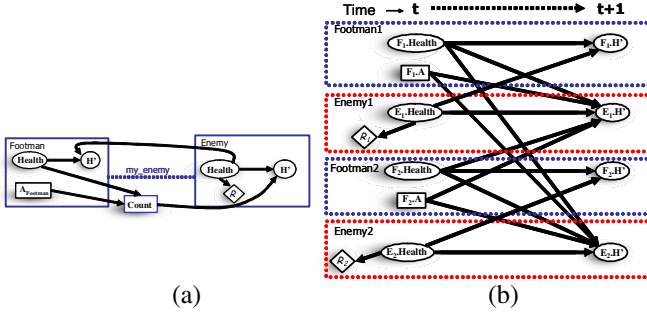


Figure 2: Freecraft tactical domain: (a) Schema; (b) Resulting factored MDP for a world with 2 footmen and 2 enemies.

barrack objects o share the same conditional probability distribution. Note, however, that each specific barrack depends on the particular peasants linked to it. Thus, the actual parents in the DBN of the status variables for two different barrack objects can be different.

The reward function is simply the sum of the reward functions for the individual objects:

$$R(s, a) = \sum_{C \in \mathcal{C}} \sum_{o \in \mathcal{O}[\omega][C]} R(s[S_o], a[o.A]).$$

Thus, for reward function for the Enemy class described above, our overall reward function in a given state will be 10 times the number of dead enemies in that state.

It remains to specify the actions in the ground MDP. The RMDP specifies a set of possible actions for every object in the world. In a setting where only a single action can be taken at any time step, the agent must choose both an object to act on, and which action to perform on that object. Here, the set of actions in the ground MDP is simply the union $\cup_{o \in \omega} \text{Dom}[o.A]$. In a setting where multiple actions can be performed in parallel (say, in a multiagent setting), it might be possible to perform an action on every object in the domain at every step. Here, the set of actions in the ground MDP is a vector specifying an action for every object: $\times_{o \in \omega} \text{Dom}[o.A]$. Intermediate cases, allowing degrees of parallelism, are also possible. For simplicity of presentation, we focus on the multiagent case, such as Freecraft, where, an action is an assignment to the action of every unit.

Example 2.1 (Freecraft tactical domain) Consider a simplified version of Freecraft, whose schema is illustrated in Fig. 2(a), where only two classes of units participate in the game: $\mathcal{C} = \{\text{Footman}, \text{Enemy}\}$. Both the footman and the enemy classes have only one state variable each, *Health*, with domain $\text{Dom}[\text{Health}] = \{\text{Healthy}, \text{Wounded}, \text{Dead}\}$. The footman class contains one single-valued link: $\rho[\text{Footman.My_Enemy}] = \text{Enemy}$. Thus the transition model for a footman's health will depend on the health of its enemy: $P^{\text{Footman}}(S'_{\text{Footman}} | S_{\text{Footman}}, S_{\text{Footman.My_Enemy}})$, i.e., if a footman's enemy is not dead, then the probability that a footman will become wounded, or die, is significantly higher. A footman can choose to attack any enemy. Thus each footman is associated with an action *Footman.A* which selects the enemy it is attacking.¹ As a consequence, an

enemy could end up being linked to a set of footmen, $\rho[\text{Enemy.My_Footmen}] = \{\text{Footman}\}$. In this case, the transition model of the health of an enemy may depend on the number of footmen who are not dead and whose action choice is to attack this enemy: $P^{\text{Enemy}}(S'_{\text{Enemy}} | S_{\text{Enemy}}, \#[\text{S}_{\text{Enemy.My_Footmen}}, \text{Enemy.My_Footmen.A}])$. Finally, we must define the template for the reward function. Here there is only a reward when an enemy is dead: $R^{\text{Enemy}}(S_{\text{Enemy}})$.

We now have a template to describe any instance of the tactical Freecraft domain. In a particular world, we must define the instances of each class and the links between these instances. For example, a world with 2 footmen and 2 enemies will have 4 objects: $\{\text{Footman1}, \text{Footman2}, \text{Enemy1}, \text{Enemy2}\}$. Each footman will be linked to an enemy: $\text{Footman1.My_Enemy} = \text{Enemy1}$ and $\text{Footman2.My_Enemy} = \text{Enemy2}$. Each enemy will be linked to both footmen: $\text{Enemy1.My_Footmen} = \text{Enemy2.My_Footmen} = \{\text{Footman1}, \text{Footman2}\}$. The template, along with the number of objects and the links in this specific ("2vs2") world yield a well-defined factored MDP, $\Pi_{2\text{vs}2}$, as shown in Fig. 2(b). ■

3 Approximately Solving Relational MDPs

There are many approaches to solving MDPs [15]. An effective one is based on linear programming (LP): Let $\mathcal{S}(\Pi)$ denote the states in an MDP Π and $\mathcal{A}(\Pi)$ the actions. If $\mathcal{S}(\Pi) = \{s_1, \dots, s_N\}$, our LP variables are $V_1, \dots, V_N$, where V_i represents $\mathcal{V}(s_i)$, the value of state s_i . The LP formulation is:

$$\begin{aligned} \text{Minimize:} \quad & \sum_i \alpha(s_i) V_i; \\ \text{Subject to:} \quad & V_i \geq R(s_i, a) + \gamma \sum_k P(s'_k | s_i, a) V_k \\ & \forall s_i \in \mathcal{S}(\Pi), a \in \mathcal{A}(\Pi). \end{aligned}$$

The *state relevance weights* $\alpha(s_1), \dots, \alpha(s_N)$ in the objective function are any set of positive weights, $\alpha(s_i) > 0$.

In our setting, the state space is exponentially large, with one state for each joint assignment to the random variables $o.S$ of every object (e.g., exponential in the number of units in the Freecraft scenario). In a multiagent problem, the number of actions is also exponential in the number of agents. Thus this LP has both an exponential number of variables and an exponential number of constraints. Therefore the exact solution to this linear program is infeasible.

We address this issue using the assumption that the value function can be well-approximated as a sum of *local value subfunctions* associated with the individual objects in the model. (This approximation is a special case of the factored linear value function approach used in [6].) Thus we associate a value subfunction $\mathcal{V}_o$ with every object in ω . Most simply, this local value function can depend only on the state of the individual object S_o . In our example, the local value subfunction V_{Enemy1} for enemy object *Enemy1* might associate a numeric value for each assignment to the variable *Enemy1.Health*. A richer approximation might associate a value function with pairs, or even small subsets, of closely related objects. Thus, the

world requires a small extension of our basic representation. We omit details due to lack of space.

¹A model where an action can change the link structure in the

function $\mathcal{V}_{Footman1}$ for *Footman1* might be defined over the joint assignments of *Footman1.Health* and *Enemy1.Health*, where *Footman1.My_Enemy* = *Enemy1*. We will represent the complete value function for a world as the sum of the local value subfunction for each individual object in this world. In our example world ($\omega = 2vs2$) with 2 footmen and 2 enemies, the global value function will be: $V_{2vs2}(F1.Health, E1.Health, F2.Health, E2.Health) = \mathcal{V}_{Footman1}(F1.Health, E1.Health) + \mathcal{V}_{Enemy1}(E1.Health) + \mathcal{V}_{Footman2}(F2.Health, E2.Health) + \mathcal{V}_{Enemy2}(E2.Health)$.

Let $\mathbf{T}_o$ be the *scope* of the value subfunction of object o , i.e., the state variables that $\mathcal{V}_o$ depends on. Given the local subfunctions, we approximate the global value function as:

$$\mathcal{V}_\omega(\mathbf{s}) = \sum_{o \in \mathcal{O}[\omega]} \mathcal{V}_o(\mathbf{s}[\mathbf{T}_o]). \quad (2)$$

As for any linear approximation to the value function, the LP approach can be adapted to use this value function representation [14]. Our LP variables are now the local components of the individual local value functions:

$$\{\mathcal{V}_o(\mathbf{t}_o) : o \in \mathcal{O}[\omega], \mathbf{t}_o \in \text{Dom}[\mathbf{T}_o]\}. \quad (3)$$

In our example, there will be one LP variable for each joint assignment of *F1.Health* and *E1.Health* to represent the components of $\mathcal{V}_{Footman1}$. Similar LP variables will be included for the components of $\mathcal{V}_{Footman2}$, $\mathcal{V}_{Enemy1}$, and $\mathcal{V}_{Enemy2}$.

As before, we have a constraint for each global state $\mathbf{s}$ and each global action $\mathbf{a}$:

$$\sum_o \mathcal{V}_o(\mathbf{s}[\mathbf{T}_o]) \geq \sum_o R^o(\mathbf{s}[\mathbf{S}_o], \mathbf{a}[o.A]) + \gamma \sum_{\mathbf{s}'} P_\omega(\mathbf{s}' | \mathbf{s}, \mathbf{a}) \sum_o \mathcal{V}_o(\mathbf{s}'[\mathbf{T}_o]); \forall \mathbf{s}, \mathbf{a}. \quad (4)$$

This transformation has the effect of reducing the number of free variables in the LP to n (the number of objects) times the number of parameters required to describe an object's local value function. However, we still have a constraint for each global state and action, an exponentially large number.

Guestrin, Koller and Parr [6] (GKP hereafter) show that, in certain cases, this exponentially large LP can be solved efficiently and exactly. In particular, this compact solution applies when the MDP is factored (i.e., represented as a DBN), and the approximate value function is decomposed as a weighted linear combination of local basis functions, as above. Under these assumptions, GKP present a decomposition of the LP which grows exponentially only in the *induced tree width* of a graph determined by the complexity of the process dynamics and the locality of the basis function.

This approach applies very easily here. The structure of the DBN representing the process dynamics is highly factored, defined via local interactions between objects. Similarly, the value functions are local, involving only single objects or groups of closely related objects. Often, the induced width of the resulting graph in such problems is quite small, allowing the techniques of GKP to be applied efficiently.

4 Generalizing Value Functions

Although this approach provides us with a principled way of decomposing a high-dimensional value function in certain types of domains, it does not help us address the generalization problem: A local value function for objects in a world ω

does not help us provide a value function for objects in other worlds, especially worlds with different sets of objects.

To obtain generalization, we build on the intuition that different objects in the same class behave similarly: they share the transition model and reward function. Although they differ in their interactions with other objects, their local contribution to the value function is often similar. For example, it may be reasonable to assume that different footmen have a similar long-term chance of killing enemies. Thus, we restrict our class of value functions by requiring that all of the objects in a given class share the same local value subfunction.

Formally, we define a *class-based local value subfunction* $\mathcal{V}_C$ for each class. We assume that the parameterization of this value function is well-defined for every object o in C . This assumption holds trivially if the scope of $\mathcal{V}_C$ is simply $\mathbf{S}_C$: we simply have a parameter for each assignment to $\text{Dom}[\mathbf{S}_C]$. When the local value function can also depend on the states of neighboring objects, we must define the parameterization accordingly; for example, we might have a parameter for each possible joint state of a linked footman-enemy pair. Specifically rather than defining separate subfunctions $\mathcal{V}_{Footman1}$ and $\mathcal{V}_{Footman2}$, we define a class-based subfunction $\mathcal{V}_{Footman}$. Now the contribution of *Footman1* to the global value function will be $\mathcal{V}_{Footman}(F1.Health, E1.Health)$. Similarly *Footman2* will contribute $\mathcal{V}_{Footman}(F2.Health, E2.Health)$.

A class-based value function defines a specific value function for each world ω , as the sum of the class-based local value functions for the objects in ω :

$$\mathcal{V}_\omega(\mathbf{s}) = \sum_{C \in \mathcal{C}} \sum_{o \in \mathcal{O}[\omega][C]} \mathcal{V}_C(\mathbf{s}[\mathbf{T}_o]). \quad (5)$$

This value function depends both on the set of objects in the world and (when local value functions can involve related objects) on the links between them. Importantly, although objects in the same class contribute the same function into the summation of (5), the argument of the function for an object is the state of that specific object (and perhaps its neighbors). In any given state, the contributions of different objects of the same class can differ. Thus, every footman has the same local value subfunction parameters, but a dead footman will have a lower contribution than one which is alive.

5 Finding Generalized MDP Solutions

With a class-level value function, we can easily generalize from one or more worlds to another one. To do so, we assume that a single set of local class-based value functions $\mathcal{V}_C$ is a good approximation across a wide range of worlds ω . Assuming we have such a set of value functions, we can act in any new world ω without replanning, as described in Step 3 of Fig. 3. We simply define a world-specific value function as in (5), and use it to act.

We must now optimize $\mathcal{V}_C$ in a way that maximizes the value over an entire set of worlds. To formalize this intuition, we assume that there is a probability distribution $P(\omega)$ over the worlds that the agent encounters. We want to find a single set of class-based local value functions $\{\mathcal{V}_C\}_{C \in \mathcal{C}}$ that is a good fit for this distribution over worlds. We view this task as one of optimizing for a single “meta-level” MDP Π^* , where

nature first chooses a world ω , and the rest of the dynamics are then determined by the MDP Π_ω . Precisely, the state space of Π^* is $\{s_0\} \cup \bigcup_\omega \{(\omega, s) : s \in \mathcal{S}(\Pi_\omega)\}$. The transition model is the obvious one: From the initial state s_0 , nature chooses a world ω according to $P(\omega)$, and an initial state in ω according to the initial starting distribution $P_\omega^0(s)$ over the states in ω . The remaining evolution is then done according to ω 's dynamics. In our example, nature will choose the number of footmen and enemies, and define the links between them, which then yields a well-defined MDP, e.g., Π_{2vs2} .

5.1 LP Formulation

The meta-MDP Π^* allows us to formalize the task of finding a generalized solution to an entire class of MDPs. Specifically, we wish to optimize the class-level parameters for $\mathcal{V}_C$, not for a single ground MDP Π_ω , but for the entire Π^* .

We can address this problem using a similar LP solution to the one we used for a single world in Sec. 3. The variables are simply parameters of the local class-level value subfunctions $\{\mathcal{V}_C(t_C) : C \in \mathcal{C}, t_C \in \text{Dom}[\mathbf{T}_C]\}$. For the constraints, recall that our object-based LP formulation in (4) had a constraint for each state s and each action vector $\mathbf{a} = \{\mathbf{a}_o\}_{o \in \mathcal{O}[\omega]}$. In the generalized solution, the state space is the union of the state spaces of all possible worlds. Our constraint set for Π^* will, therefore, be a union of constraint sets, one for each world ω , each with its own actions:

$$\mathcal{V}_\omega(s) \geq \sum_o R^o(s[\mathbf{S}_o], \mathbf{a}_o) + \gamma \sum_{s'} P_\omega(s' | s, \mathbf{a}) \mathcal{V}_\omega(s'); \quad \forall \omega, \forall s \in \mathcal{S}(\Pi_\omega), \mathbf{a} \in \mathcal{A}(\Pi_\omega); \quad (6)$$

where the value function for a world, $\mathcal{V}_\omega(s)$, is defined at the class level as in Eq. (5). In principle, we should have an additional constraint for the state s_0 . However, with a natural choice of state relevance weights α , this constraint is eliminated and the objective function becomes:

$$\text{Minimize: } \frac{1+\gamma}{2} \sum_\omega \sum_{s \in \mathcal{S}_\omega} P(\omega) P_\omega^0(s) \mathcal{V}_\omega(s); \quad (7)$$

if $P_\omega^0(s) > 0, \forall s$. In some models, the potential number of objects may be infinite, which could make the objective function unbounded. To prevent this problem, we assume that the $P(\omega)$ goes to zero sufficiently fast, as the number of objects tends to infinity. To understand this assumption, consider the following generative process for selecting worlds: first, the number of objects is chosen according to $P(\#)$; then, the classes and links of each object are chosen according to $P(\omega_\# | \#)$. Using this decomposition, we have that $P(\omega) = P(\#)P(\omega_\# | \#)$. The intuitive assumption described above can be formalized as: $\forall n, P(\# = n) \leq \kappa_\# e^{-\lambda_\# n}$; for some $\kappa_\# \geq \lambda_\# > 0$. Thus, the distribution $P(\#)$ over number of objects can be chosen arbitrarily, as long as it is bounded by some exponentially decaying function.

5.2 Sampling worlds

The main problem with this formulation is that the size of the LP — the size of the objective and the number of constraints — grows with the number of worlds, which, in most situations, grows exponentially with the number of possible objects, or may even be infinite. A practical approach to address this problem is to *sample* some reasonable number of worlds from the distribution $P(\omega)$, and then to solve the LP

for these worlds only. The resulting class-based value function can then be used for worlds that were not sampled.

We will start by sampling a set $\mathcal{D}$ of m worlds according to $P(\omega)$. We can now define our LP in terms of the worlds in $\mathcal{D}$, rather than all possible worlds. For each world ω in $\mathcal{D}$, our LP will contain a set of constraints of the form presented in Eq. (4). Note that in all worlds these constraints share the variables $\mathcal{V}_C$, which represent our class-based value function. The complete LP is given by:

$$\begin{aligned} \text{Variables: } & \{\mathcal{V}_C(t_C) : C \in \mathcal{C}, t_C \in \text{Dom}[\mathbf{T}_C]\}. \\ \text{Minimize: } & \frac{1+\gamma}{2m} \sum_{\omega \in \mathcal{D}} \sum_{C \in \mathcal{C}} \sum_{o \in \mathcal{O}[\omega][C]} \sum_{t_o \in \mathbf{T}_o} P_\omega^0(t_o) \mathcal{V}_C(t_o). \\ \text{Subject to: } & \sum_{C \in \mathcal{C}} \sum_{o \in \mathcal{O}[\omega][C]} \mathcal{V}_C(s[\mathbf{T}_o]) \geq \\ & \sum_{o \in \mathcal{O}[\omega][C]} R^C(s[\mathbf{S}_o], \mathbf{a}[o.A]) + \\ & \gamma \sum_{s'} P_\omega(s' | s, \mathbf{a}) \sum_{C \in \mathcal{C}} \sum_{o \in \mathcal{O}[\omega][C]} \mathcal{V}_C(s'[\mathbf{T}_o]); \\ & \forall \omega \in \mathcal{D}, \forall s \in \mathcal{S}(\Pi_\omega), \mathbf{a} \in \mathcal{A}(\Pi_\omega); \end{aligned} \quad (8)$$

where $P_\omega^0(\mathbf{T}_o)$ is the marginalization of $P_\omega^0(\mathbf{S}_o)$ to the variables in $\mathbf{T}_o$. For each world, the constraints have the same form as the ones in Sec. 3. Thus, once we have sampled worlds, we can apply the same LP decomposition techniques of GKP to each world to solve this LP efficiently. Our generalization algorithm is summarized in Step 2 of Fig. 3.

The solution obtained by the LP with sampled worlds will, in general, not be equal to the one obtained if all worlds are considered simultaneously. However, we can show that the quality of the two approximations is close, if a sufficient number of worlds are sampled. Specifically, with a *polynomial* number of sampled worlds, we can guarantee that, with high probability, the quality of the value function approximation obtained when sampling worlds is close to the one obtained when considering all possible worlds.

Theorem 5.1 *Consider the following class-based value functions (each with k parameters): $\hat{\mathcal{V}}$ obtained from the LP over all possible worlds by minimizing Eq. (7) subject to the constraints in Eq. (6); $\tilde{\mathcal{V}}$ obtained from the LP with the sampled worlds in (8); and $\mathcal{V}^*$ the optimal value function of the meta-MDP Π^* . For a number of sampled worlds m polynomial in $(1/\varepsilon, \ln 1/\delta, 1/(1-\gamma), k, \lambda_\#, 1/\kappa_\#)$, the error is bounded by:*

$$\|\mathcal{V}^* - \tilde{\mathcal{V}}\|_{1, P_\Omega} \leq \|\mathcal{V}^* - \hat{\mathcal{V}}\|_{1, P_\Omega} + \varepsilon R_{max};$$

with probability at least $1 - \delta$, for any $\delta > 0$ and $\varepsilon > 0$; where $\|\mathcal{V}\|_{1, P_\Omega} = \sum_{\omega, s \in \mathcal{S}_\omega} P(\omega) P_\omega^0(s) |\mathcal{V}_\omega(s)|$, and R_{max} is the maximum per-object reward. ■

The proof, which is omitted for lack of space (see online version of this paper), uses some of the techniques developed by de Farias and Van Roy [2] for analyzing constraint sampling in general MDPs. However, there are two important differences: First, our analysis includes the error introduced when sampling the objective, which in our case is a sum only over a subset of the worlds rather than over all of them as in the LP for the full meta-MDP. This issue was not previously addressed. Second, the algorithm of de Farias and Van Roy relies on the assumption that constraints are sampled according

to some “ideal” distribution (the stationary distribution of the optimal policy). Unfortunately, sampling from this distribution is as difficult as computing a near-optimal policy. In our analysis, after each world is sampled, our algorithm exploits the factored structure in the model to represent the constraints exactly, avoiding the dependency on the “ideal” distribution.

6 Learning Classes of Objects

The definition of a class-based value function assumes that all objects in a class have the same local value function. In many cases, even objects in the same class might play different roles in the model, and therefore have a different impact on the overall value. For example, if only one peasant has the capability to build barracks, his status may have a greater impact. Distinctions of this type are not usually known in advance, but are learned by an agent as it gains experience with a domain and detects regularities.

We propose a procedure that takes exactly this approach: Assume that we have been presented with a set $\mathcal{D}$ of worlds ω . For each world ω , an approximate value function $\mathcal{V}_\omega = \sum_{o \in \mathcal{O}[\omega]} \mathcal{V}_o$ was computed as described in Sec. 3. In addition, each object is associated with a set of features $\mathcal{F}_\omega[o]$. For example, the features may include local information, such as whether the object is a peasant linked to a barrack, or not, as well as global information, such as whether this world contains archers in addition to footmen. We can define our “training data” $\mathcal{D}$ as $\{(\mathcal{F}_\omega[o], \mathcal{V}_o) : o \in \mathcal{O}[\omega], \omega \in \mathcal{D}\}$.

We now have a well-defined learning problem: given this training data, we would like to partition the objects into classes, such that objects of the same class have similar value functions. There are many approaches for tackling such a task. We choose to use decision tree regression, so as to construct a tree that predicts the local value function parameters given the features. Thus, each split in the tree corresponds to a feature in $\mathcal{F}_\omega[o]$; each branch down the tree defines a subset of local value functions in $\mathcal{D}$ whose feature values are as defined by the path; the leaf at the end of the path is the average value function for this set. As the regression tree learning algorithm tries to construct a tree which is predictive about the local value function, it will aim to construct a tree where the mean at each leaf is very close to the training data assigned to that leaf. Thus, the leaves tend to correspond to objects whose local value functions are similar. We can thus take the leaves in the tree to define our subclasses, where each subclass is characterized by the combination of feature values specified by the path to the corresponding leaf. This algorithm is summarized in Step 1 of Fig. 3. Note that the mean subfunction at a leaf is not used as the value subfunction for the corresponding class; rather, the parameters of the value subfunction are optimized using the class-based LP in Step 2 of the algorithm.

7 Experimental results

We evaluated our generalization algorithm on two domains: computer network administration and Freecraft.

7.1 Computer network administration

For this problem, we implemented our algorithm in Matlab, using CPLEX as the LP solver. Rather than using the full LP decomposition of GKP [6], we used the constraint generation extension proposed in [13], as the memory requirements

1. Learning Subclasses:

- **Input:**
 - A set of training worlds $\mathcal{D}$.
 - A set of features $\mathcal{F}_\omega[o]$.
- **Algorithm:**
 - (a) For each $\omega \in \mathcal{D}$, compute an object-based value function, as described in Sec. 3.
 - (b) Apply regression tree learning on $\{(\mathcal{F}_\omega[o], \mathcal{V}_o) : o \in \mathcal{O}[\omega], \omega \in \mathcal{D}\}$.
 - (c) Define a subclass for each leaf, characterized by the feature vector associated with its path.

2. Computing Class-Based Value Function:

- **Input:**
 - A set of (sub)class definitions $\mathcal{C}$.
 - A template for $\{\mathcal{V}_C : C \in \mathcal{C}\}$.
 - A set of training worlds $\mathcal{D}$.
- **Algorithm:**
 - (a) Compute the parameters for $\{\mathcal{V}_C : C \in \mathcal{C}\}$ that optimize the LP in (8) relative to the worlds in $\mathcal{D}$.

3. Acting in a New World:

- **Input:**
 - A set of local value functions $\{\mathcal{V}_C : C \in \mathcal{C}\}$.
 - A set of (sub)class definitions $\mathcal{C}$.
 - A world ω .
- **Algorithm:** Repeat
 - (a) Obtain the current state s .
 - (b) Determine the appropriate class C for each $o \in \mathcal{O}[\omega]$ according to its features.
 - (c) Define $\mathcal{V}_\omega$ according to (5).
 - (d) Use the coordination graph algorithm of GKP to compute an action a that maximizes $R(s, a) + \gamma \sum_{s'} P(s' | s, a) \mathcal{V}_\omega(s')$.
 - (e) Take action a in the world.

Figure 3: The overall generalization algorithm.

were lower for this second approach. We experimented with the multiagent computer network examples in [6], using various network topologies and “pair” basis functions that involve states of neighboring machines (see [6]). In one of these problems, if we have n computers, then the underlying MDP has 9^n states and 2^n actions. However, the LP decomposition algorithm uses structure in the underlying factored model to solve such problems very efficiently [6].

We first tested the extent to which value functions are shared across objects. In Fig. 4(a), we plot the value each object gave to the assignment *Status* = *working*, for instances of the ‘three legs’ topology. These values cluster into three classes. We used *CART*[®] to learn decision trees for our class partition. In this case, the learning algorithm partitioned the computers into three subclasses illustrated in Fig. 4(b): ‘server’, ‘intermediate’, and ‘leaf’. In Fig. 4(a), we see that ‘server’ (third column) has the highest value, because a broken server can cause a chain reaction affecting the whole network, while ‘leaf’ value (first column) is lowest, as it cannot affect any other computer.

We then evaluated the generalization quality of our class-based value function by comparing its performance to that of planning specifically for a new environment. For each topology, we computed the class-based value function with 5 sampled networks of up to 20 computers. We then sampled a

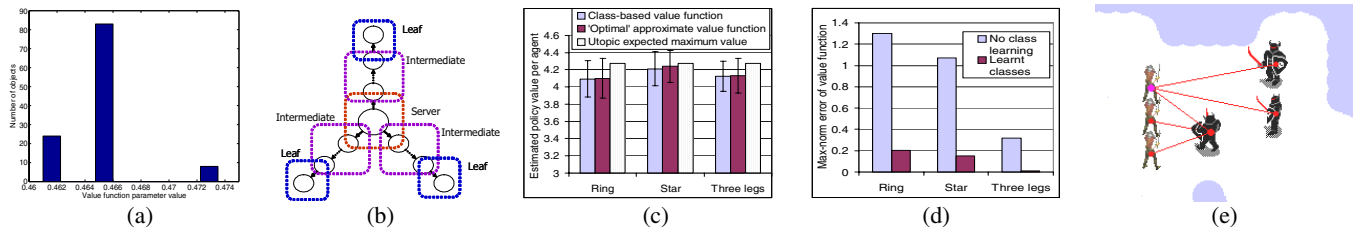


Figure 4: Network administrator results: (a) Training data for learning classes; (b) Classes learned for ‘three legs’; (c) Generalization quality (evaluated by 20 Monte Carlo runs of 100 steps); (d) Advantage of learning subclasses. Tactical Freecraft: (e) 3 footmen against 3 enemies.

new network and computed for it a value function that used the same factorization, but with no class restrictions. This value function has more parameters — different parameters for each object, rather than for entire classes, which are optimized for this particular network. This process was repeated for 8 sets of networks. The results, shown in Fig. 4(c), indicate that the value of the policy from the class-based value function is very close to the value of replanning, suggesting that we can generalize well to new problems. We also computed a utopic upper bound on the *expected* value of the optimal policy by removing the (negative) effect of the neighbors on the status of the machines. Although this bound is loose, our approximate policies still achieve a value close to it.

Next, we wanted to determine if our procedure for learning classes yields better approximations than the ones obtained from the default classes. Fig. 4(d) compares the max-norm error between our class-based value function and the one obtained by replanning. The graph suggests that, by learning classes using our decision trees regression tree procedure, we obtain a much better approximation of the value function we would have, had we replanned.

7.2 Freecraft

In order to evaluate our algorithm in the Freecraft game, we implemented the methods in C++ and used CPLEX as the LP solver. We created two tasks that evaluate two aspects of the game: long-term strategic decision making and local tactical battle maneuvers. Our Freecraft interface, and scenarios for these and other more complex tasks are publicly available at: <http://dags.stanford.edu/Freecraft/>. For each task we designed an RMDP model to represent the system, by consulting a “domain expert”. After planning, our policies were evaluated on the actual game. To better visualize our results, we direct the reader to view videos of our policies at a website:

<http://robotics.stanford.edu/~guestin/Research/Generalization/>.

This website also contains details on our RMDP model. It is important to note that, our policies were constructed relative to a very approximate model of the game, but evaluated against the real game.

In the tactical model, the goal is to take out an opposing enemy force with an equivalent number of units. At each time step, each footman decides which enemy to attack. The enemies are controlled using Freecraft’s hand-built strategy. We modelled footmen and enemies as each having 5 “health points”, which can decrease as units are attacked. We used a simple aggregator to represent the effect of multiple attackers. To encourage coordination, each footman is linked to a “buddy” in a ring structure. The local value functions include terms over triples of linked variables. We solved this model

for a world with 3 footmen and 3 enemies, shown in Fig. 4(e). The resulting policy (which is fairly complex) demonstrates successful coordination between our footmen: initially all three footmen focus on one enemy. When the enemy becomes injured, one footman switches its target. Finally, when the enemy is very weak, only one footman continues to attack it, while the others tackle a different enemy. Using this policy, our footmen defeat the enemies in Freecraft.

The factors generated in our planning algorithm grow exponentially in the number of units, so planning in larger models is infeasible. Fortunately, when executing a policy, we instantiate the current state at every time step, and action selection is significantly faster [6]. Thus, even though we cannot execute Step 2 in Fig. 3 of our algorithm for larger scenarios, we can generalize our class-based value function to a world with 4 footmen and enemies, without replanning using only Step 3 of our approach. The policy continues to demonstrate successful coordination between footmen, and we again beat Freecraft’s policy. However, as the number of units increases, the position of enemies becomes increasingly important. Currently, our model does not consider this feature, and in a world with 5 footmen and enemies, our policy loses to Freecraft in a close battle.

In the strategic model, the goal is to kill a strong enemy. The player starts with a few peasants, who can collect gold or wood, or attempt to build a barrack, which requires both gold and wood. All resources are consumed after each *Build* action. With a barrack and gold, the player can train a footman. The footmen can choose to attack the enemy. When attacked, the enemy loses “health points”, but fights back and may kill the footmen. We solved a model with 2 peasants, 1 barrack, 2 footmen, and an enemy. Every peasant was related to a “central” peasant and every footman had a “buddy”. The scope of our local value function included triples between related objects. The resulting policy is quite interesting: the peasants gather gold and wood to build a barrack, then gold to build a footman. Rather than attacking the enemy at once, this footman waits until a second footman is built. Then, they attack the enemy together. The stronger enemy is able to kill both footmen, but it becomes quite weak. When the next footman is trained, rather than waiting for a second one, it attacks the now weak enemy, and is able to kill him. Again, planning in large scenarios is infeasible, but action selection can be performed efficiently. Thus, we can use our generalized value function to tackle a world with 9 peasants and 3 footmen, without replanning. The 9 peasants coordinate to gather resources. Interestingly, rather than attacking with 2 footmen, the policy now waits for 3 to be trained before attacking. The 3 footmen kill the enemy, and only one of them dies. Thus,

we have successfully generalized from a problem with about 10^6 joint state-action pairs to one with over 10^{13} pairs.

8 Discussion and Conclusions

In this paper, we have tackled a longstanding goal in planning research, the ability to generalize plans to new environments. Such generalization has two complementary uses: First we can tackle new environments with minimal or no replanning. Second it allows us to generalize plans from smaller tractable environments to significantly larger ones, which could not be solved directly with our planning algorithm. Our experimental results support the fact that our class-based value function generalizes well to new plans and that the class and subclass structure discovered by our learning procedure improves the quality of the approximation. Furthermore, we successfully demonstrated our methods on a real strategic computer game, which contains many characteristics present in real-world dynamic resource allocation problems.

Several other papers consider the generalization problem. Several approaches can represent value functions in general terms, but usually require it to be hand-constructed for the particular task. Others [12; 8; 4] have focused on reusing solutions from isomorphic regions of state space. By comparison, our method exploits similarities between objects evolving in parallel. It would be very interesting to combine these two types of decomposition. The work of Boutilier *et al.* [1] on symbolic value iteration computes first-order value functions, which generalize over objects. However, it focuses on computing exact value functions, which are unlikely to generalize to a different world. Furthermore, it relies on the use of theorem proving tools, which adds to the complexity of the approach. Methods in deterministic planning have focused on generalizing from compactly described policies learned from many domains to incrementally build a first-order policy [9; 11]. Closest in spirit to our approach is the recent work of Yoon *et al.* [17], which extends these approaches to stochastic domains. We perform a similar procedure to discover classes by finding structure in the value function. However, our approach finds regularities in compactly represented value functions rather than policies. Thus, we can tackle tasks such as multiagent planning, where the action space is exponentially large and compact policies often do not exist.

The key assumption in our method is interchangeability between objects of the same class. Our mechanism for learning subclasses allows us to deal with cases where objects in the domain can vary, but our generalizations will not be successful in very heterogeneous environments, where most objects have very different influences on the overall dynamics or rewards. Additionally, the efficiency of our LP solution algorithm depends on the connectivity of the underlying problem. In a domain with strong and constant interactions between many objects (*e.g.*, Robocup), or when the reward function depends arbitrarily on the state of many objects (*e.g.*, Blocksworld), the solution algorithm will probably not be efficient. In some cases, such as the Freecraft tactical domain, we can use generalization to scale up to larger problems. In others, we could combine our LP decomposition technique with constraint sampling [2] to address this high connectivity issue. In general, however, extending these techniques to highly connected problems is still an open problem. Finally,

although we have successfully applied our class-value functions to new environments without replanning, there are domains where such direct application would not be sufficient to obtain a good solution. In such domains, our generalized value functions can provide a good initial policy, which could be refined using a variety of local search methods.

We have assumed that relations do not change over time. In many domains (*e.g.*, Blocksworld or Robocup), this assumption is false. In recent work, Guestrin *et al.* [7] show that *context-specific independence* can allow for dynamically changing coordination structures in multiagent environments. Similar ideas may allow us to tackle dynamically changing relational structures.

In summary, we believe that the class-based value functions methods presented here will significantly further the applicability of MDP models to large-scale real-world tasks.

Acknowledgements We are very grateful to Ron Parr for many useful discussions. This work was supported by the DoD MURI program, administered by the Office of Naval Research under Grant N00014-00-1-0637, and by Air Force contract F30602-00-2-0598 under DARPA's TASK program.

References

- [1] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In *IJCAI-01*, 2001.
- [2] D.P. de Farias and B. Van Roy. On constraint sampling for the linear programming approach to approximate dynamic programming. *Submitted to Math. of Operations Research*, 2001.
- [3] T. Dean and K. Kanazawa. Probabilistic temporal reasoning. In *AAAI-88*, 1988.
- [4] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.
- [5] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artf. Intel.*, 3(4):251–288, 1972.
- [6] C. E. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *NIPS-14*, 2001.
- [7] C. E. Guestrin, S. Venkataraman, and D. Koller. Context specific multiagent coordination and planning with factored MDPs. In *AAAI-02*, 2002.
- [8] M. Hauskrecht, N. Meuleau, L. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of Markov decision processes using macro-actions. In *UAI*, 1998.
- [9] R. Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113:125–148, 1999.
- [10] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *AAAI*, 1998.
- [11] M. Martin and H. Geffner. Learning generalized policies in planning using concept languages. In *KR*, 2000.
- [12] R. Parr. Flexible decomposition algorithms for weakly coupled markov decision problems. In *UAI-98*, 1998.
- [13] D. Schuurmans and R. Patrascu. Direct value-approximation for factored MDPs. In *NIPS-14*, 2001.
- [14] P. Schweitzer and A. Seidmann. Generalized polynomial approximations in Markovian decision processes. *J. of Mathematical Analysis and Applications*, 110:568 – 582, 1985.
- [15] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
- [16] S. Thrun and J. O'Sullivan. Discovering structure in multiple learning tasks: The TC algorithm. In *ICML-96*, 1996.
- [17] S. W. Yoon, A. Fern, and B. Givan. Inductive policy selection for first-order MDPs. In *UAI-02*, 2002.

SMDP Homomorphisms: An Algebraic Approach to Abstraction in Semi-Markov Decision Processes

Balaraman Ravindran and Andrew G. Barto

Department of Computer Science
University of Massachusetts
Amherst, MA, U. S. A.
{ravi|barto}@cs.umass.edu

Abstract

To operate effectively in complex environments learning agents require the ability to selectively ignore irrelevant details and form useful abstractions. In this article we consider the question of what constitutes a useful abstraction in a stochastic sequential decision problem modeled as a semi-Markov Decision Process (SMDPs). We introduce the notion of SMDP homomorphism and argue that it provides a useful tool for a rigorous study of abstraction for SMDPs. We present an SMDP minimization framework and an abstraction framework for factored MDPs based on SMDP homomorphisms. We also model different classes of abstractions that arise in hierarchical systems. Although we use the options framework for purposes of illustration, the ideas are more generally applicable. We also show that the conditions for abstraction we employ are a generalization of earlier work by Dietterich as applied to the options framework.

1 Introduction

The ability to form abstractions is one of the features that allows humans to operate effectively in complex environments. We systematically ignore information that we do not need for performing the immediate task at hand. While driving, for example, we may ignore details regarding clothing and the state of our hair. Researchers in artificial intelligence (AI), in particular machine learning (ML), have long recognized that applying computational approaches to operating in complex and real-world domains requires that we incorporate the ability to handle and form useful abstractions. In this article we consider the question of what constitutes a useful abstraction. Since this is a difficult problem when stated in general terms, much of the work in this field is specialized to particular classes of problems or specific modeling paradigms. In this work we will focus on Markov decision processes (MDPs), a formalism widely employed in modeling and solving stochastic sequential decision problems and semi-Markov decision processes (SMDPs), an extension to MDPs recently employed in modeling hierarchical systems [Sutton *et al.*, 1999; Dietterich, 2000; Parr and Russell, 1997].

Our objective in this article is to introduce the concept of an *SMDP homomorphism* and argue that it provides a unified view of key issues essential for a rigorous treatment of abstraction for stochastic dynamic decision processes. The concept of a homomorphism between dynamic systems, sometimes called a “dynamorphism” [Arbib and Manes, 1975], has played an important role in theories of abstract automata [Hartmanis and Stearns, 1966], theories of modeling and simulation [Zeigler, 1972], and it is frequently used by researchers studying model checking approaches to system validation [Emerson and Sistla, 1996]. Although those studying approximation and abstraction methods for MDPs and SMDPs have employed formalisms that implicitly embody the idea of a homomorphism, they have not made explicit use of the appropriate homomorphism concept. We provide what we claim is the appropriate concept and give examples of how it can be widely useful as the basis of the study of abstraction in a dynamic setting.

Informally, the kind of homomorphism we consider is a mapping from one dynamic system to another that eliminates state distinctions while preserving the system’s dynamics. We present a definition of homomorphisms that is appropriate for SMDPs. In earlier work [Ravindran and Barto, 2002] we developed an MDP abstraction framework based on our notion of an MDP homomorphism. This framework extended the MDP minimization framework proposed by [Dean and Givan, 1997] and enabled the accommodation of redundancies arising from symmetric equivalence of the kind illustrated in Figure 1. While we can derive reduced models with a smaller state set by applying minimization ideas, we do not necessarily simplify the description of the problem in terms of the number of parameters required to describe it. But MDPs often have additional structure associated with them that we can exploit to develop compact representations. By injecting structure in the definition of SMDP homomorphism we can model abstraction schemes for structured MDPs. In this paper we present one simple example of such an abstraction scheme and mention other factored abstraction methods that can be modeled by suitable structured homomorphisms.

In the second part of the paper we extend the notion of SMDP homomorphism to hierarchical systems. In particular, we apply homomorphisms in the options framework introduced by [Sutton *et al.*, 1999] and show that this facilitates employing different abstractions at different levels of

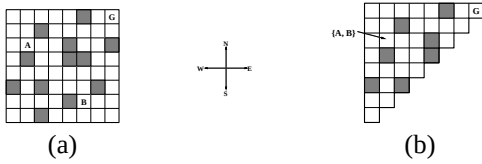


Figure 1: (a) A symmetric gridworld problem. The goal state is G and there are four deterministic actions. This gridworld is symmetric about the NE-SW diagonal. For example, states A and B are equivalent since for each action in A , there is an equivalent action in B . Taking action E , say, in state A is equivalent to taking action N in state B , in the sense that they go to equivalent states that are each one step closer to the goal. (b) An equivalent reduced model of the gridworld in (a). The states A and B in the original problem correspond to the single state $\{A, B\}$ in the reduced problem. A solution to this reduced gridworld can be used to derive a solution to the full problem.

a hierarchy. We also argue that homomorphisms allow us to model aspects of symbolic AI techniques in representing higher level task structure. Finally, we show that the SMDP homomorphism conditions generalize the “safe” state abstraction conditions for hierarchical systems introduced by [Dietterich, 2000].

After introducing some notation (Section 2), we define SMDP homomorphisms and discuss modeling symmetries (Section 3). Then we present a brief overview of our model minimization framework (Section 4), discuss abstraction in factored MDPs (Section 5) and abstraction in hierarchical systems (Section 6). We conclude with some discussion of directions for future research (Section 7).

2 Notation

A *Markov Decision Process* is a tuple $\langle S, A, \Psi, P, R \rangle$, where $S = \{1, 2, \dots, n\}$ is a set of states, A is a finite set of actions, $\Psi \subseteq S \times A$ is the set of admissible state-action pairs, $P : \Psi \times S \rightarrow [0, 1]$ is the transition probability function with $P(s, a, s')$ being the probability of transition from state s to state s' under action a , and $R : \Psi \rightarrow \mathbb{R}$ is the expected reward function, with $R(s, a)$ being the expected reward for performing action a in state s . Let $A_s = \{a | (s, a) \in \Psi\} \subseteq A$ denote the set of actions admissible in state s . We assume that for all $s \in S$, A_s is non-empty.

A discrete time semi-Markov decision process (SMDP) is a generalization of an MDP in which actions can take variable amounts of time to complete. As with an MDP, an SMDP is a tuple $\langle S, A, \Psi, P, R \rangle$, where S , A and Ψ are the sets of states, actions and admissible state-action pairs; $P : \Psi \times S \times \mathbb{N} \rightarrow [0, 1]$ is the transition probability function with $P(s, a, s', N)$ being the probability of transition from state s to state s' under action a in N time steps, and $R : \Psi \times \mathbb{N} \rightarrow \mathbb{R}$ is the expected discounted reward function, with $R(s, a, N)$ being the expected reward for performing action a in state s and completing it in N time steps.¹

A *stochastic policy*, π , is a mapping from Ψ to the real interval $[0, 1]$ with $\sum_{a \in A_s} \pi(s, a) = 1$ for all $s \in S$. For any

¹We are adopting the formalism of [Dietterich, 2000].

$(s, a) \in \Psi$, $\pi(s, a)$ gives the probability of executing action a in state s . The *value* of a state-action pair (s, a) under policy π is the expected value of the sum of discounted future rewards starting from state s , taking action a , and following π thereafter. When the SMDP has well defined terminal states, we often do not discount future rewards. In such cases an SMDP is equivalent to an MDP and we will ignore the transition times. The *action-value function*, Q^π , corresponding to a policy π is the mapping from state-action pairs to their values. The solution of an MDP is an *optimal policy*, π^* , that uniformly dominates all other possible policies for that MDP.

Let B be a partition of a set X . For any $x \in X$, $[x]_B$ denotes the block of B to which x belongs. Any function f from a set X to a set Y induces a partition (or equivalence relation) on X , with $[x]_f = [x']_f$ if and only if $f(x) = f(x')$.

3 SMDP Homomorphisms

A homomorphism from a dynamic system $\mathcal{M}$ to a dynamic system $\mathcal{M}'$ is a mapping that preserves $\mathcal{M}$ ’s dynamics, while in general eliminating some of the details of the full system $\mathcal{M}$. One can think of $\mathcal{M}'$ as a simplified model of $\mathcal{M}$ that is nevertheless a valid model of $\mathcal{M}$ with respect to the aspects of $\mathcal{M}$ ’s state that it preserves. The specific definition of homomorphism that we claim is most useful for MDPs and SMDPs is as follows:

Definition: An *SMDP homomorphism* h from an SMDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ to an SMDP $\mathcal{M}' = \langle S', A', \Psi', P', R' \rangle$ is a surjection from Ψ to Ψ' , defined by a tuple of surjections $\langle f, g_1, g_2, \dots, g_n \rangle$, with $h((s, a)) = (f(s), g_s(a))$, where $f : S \rightarrow S'$ and $g_s : A_s \rightarrow A'_{f(s)}$ for $s \in S$, such that $\forall s, s' \in S, a \in A_s$ and for all $N \in \mathbb{N}$:

$$P'(f(s), g_s(a), f(s'), N) = \sum_{s'' \in [s']_f} P(s, a, s'', N), \quad (1)$$

$$R'(f(s), g_s(a), N) = R(s, a, N). \quad (2)$$

We call $\mathcal{M}'$ the *homomorphic image* of $\mathcal{M}$ under h , and we use the shorthand $h(s, a)$ to denote $h((s, a))$. The surjection f maps states of $\mathcal{M}$ to states of $\mathcal{M}'$, and since it is generally many-to-one, it generally induces nontrivial equivalence classes of states s of $\mathcal{M}$: $[s]_f$. Each surjection g_s recodes the actions admissible in state s of $\mathcal{M}$ to actions admissible in state $f(s)$ of $\mathcal{M}'$. This *state-dependent* recoding of actions is a key innovation of our definition, which we discuss in more detail below. Condition (1) says that the transition probabilities in the simpler SMDP $\mathcal{M}'$ are expressible as sums of the transition probabilities of the states of $\mathcal{M}$ that f maps to that same state in $\mathcal{M}'$. This is the stochastic version of the standard condition for homomorphisms of deterministic systems that requires that the homomorphism commutes with the system dynamics [Hartmanis and Stearns, 1966]. Condition (2) says that state-action pairs that have the same image under h have the same expected reward. An MDP homomorphism is similar to an SMDP homomorphism except that the conditions (1) and (2) apply only to the states and actions and not to the transition times.

The state-dependent action mapping allows us to model symmetric equivalence in MDPs and SMDPs. For example,

if $h = \langle f, g_1, g_2, \dots, g_n \rangle$ is a homomorphism from the gridworld of Figure 1(a) to that of Figure 1(b), then $f(A) = f(B)$ is the state marked $\{A, B\}$ in Figure 1(b). Also $g_A(E) = g_B(N) = E$, $g_A(W) = g_B(S) = W$, and so on. Whereas [Zinkevich and Balch, 2001] defined symmetries of MDPs by employing equivalence relations on the state-action pairs, we explicitly formalize the notion of SMDP symmetries employing SMDP homomorphisms and group theoretic concepts.

Definitions: An SMDP homomorphism $h = \langle f, g_1, g_2, \dots, g_n \rangle$ from SMDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ to SMDP $\mathcal{M}' = \langle S', A', \Psi', P', R' \rangle$ is an *SMDP isomorphism* from $\mathcal{M}$ to $\mathcal{M}'$ if and only if f and g_s , $s \in S$, are bijective. $\mathcal{M}$ is said to be *isomorphic* to $\mathcal{M}'$ and vice versa. An SMDP isomorphism from an SMDP $\mathcal{M}$ to itself is an *automorphism* of $\mathcal{M}$.

The set of all automorphisms of an SMDP $\mathcal{M}$, denoted by $\text{Aut}\mathcal{M}$, forms a group under composition of homomorphisms. This group is the *symmetry group* of $\mathcal{M}$. In the gridworld example of Figure 1, the symmetry group consists of the identity map on states and actions, a reflection of the states about the NE-SW diagonal and a swapping of actions N and E and of actions S and W. Any subgroup of the symmetry group of an SMDP induces an equivalence relation on Ψ , which can also be induced by a suitably defined homomorphism [Ravindran and Barto, 2001]. Therefore we can model symmetric equivalence as a special case of homomorphic equivalence.

4 Minimization

The notion of homomorphic equivalence immediately gives us an SMDP minimization framework. In [Ravindran and Barto, 2001] we extended the minimization framework of Dean and Givan [Dean and Givan, 1997; Givan et al., 2003] to include state-dependent action recoding and showed that if two state-action pairs have the same image under a homomorphism, then they have the same optimal value. We also showed that when $\mathcal{M}'$ is a homomorphic image of an MDP $\mathcal{M}$, a policy in $\mathcal{M}'$ can induce a policy in $\mathcal{M}$ that is closely related. Specifically a policy that is optimal in $\mathcal{M}'$ can induce an optimal policy in $\mathcal{M}$. Thus we can solve the original MDP by solving a homomorphic image. It is easy to extend these results to SMDP models.

Thus the goal of minimization is to derive a *minimal image* of the SMDP, i.e., a homomorphic image with the least number of admissible state-action pairs. We also adapted an existing minimization algorithm to find minimal images employing state-action equivalence. Employing state-dependent action recoding allows us to achieve greater reduction in model size than possible with Dean and Givan's framework. For example, the gridworld in Figure 1(a) is minimal if we do not consider state-dependent action mappings.

5 Abstraction in Structured MDPs

SMDP homomorphisms can also be used to model various SMDP abstraction frameworks. The definition of homomorphism in Section 3 assumed a monolithic representation of the state set. While we can derive an equivalent MDP model with a smaller Ψ , it does not follow that the description of the

state set is necessarily simpler and hence might not lead to a simpler problem representation. Many classes of problems that are modeled as MDPs have some inherent structure. We define structured forms of homomorphisms that can exploit this structure in deriving simpler problem representations.

Factored MDPs are a popular way to model structure in MDPs. A factored MDP is described, as before, by the tuple $\langle S, A, \Psi, P, R \rangle$. The state set S is now given by M features or variables, $S \subseteq \prod_{i=1}^M S_i$, where S_i is the set of permissible values for feature i . Thus any $s \in S$ is of the form $s = \langle s_1, \dots, s_M \rangle$, where $s_i \in S_i$ for all i . The elements of S are still uniquely labeled 1 through n . The transition probabilities P are often described by a two-slice *temporal Bayesian network* (2-TBN) [Dean and Kanazawa, 1989]. A 2-TBN is a two layer directed acyclic graph, whose nodes are $\{s_1, \dots, s_M\}$ and $\{s'_1, \dots, s'_M\}$. Here s_i denotes the value of feature i in the present state and s'_i denotes the value of feature i in the resulting state. An arc from node s_i to node s'_j indicates that the s'_j depends on s_i . Many classes of structured problems may be modeled by a 2-TBN in which the arcs are restricted to go from nodes in the first set to those in the second. The state-transition probabilities can be factored as:

$$P(s, a, s') = \prod_{i=1}^M \text{Prob}(s'_i | \text{Pre}(s'_i, a)),$$

where $\text{Pre}(s'_i, a)$ denotes the parents of node s'_i in the 2-TBN corresponding to action a and each of the $\text{Prob}(s'_i | \text{Pre}(s'_i, a))$ is given by a conditional probability table (CPT) associated with node s'_i . The reward function may be similarly represented.

Structuring the state space representation allows us to consider morphisms that are structured, i. e. surjections from one structured set to another. An example of a structured morphism is a simple projection onto a subset of features. Here the state set of the homomorphic image is described by a subset of the features, while the rest are ignored. We introduce some notation, after [Zeigler, 1972], to make the following definitions easier. Given a structured set $X \subseteq \prod_{i=1}^M X_i$, the *i-th projection* of X is a mapping $\rho_i : X \rightarrow X_i$, defined by $\rho_i(\langle x_1, \dots, x_M \rangle) = x_i$. We extend this definition to that of a projection on a subset of features. Given a set $J \subseteq \{1, \dots, M\}$ the *J-projection* of X is a mapping $\rho_J : X \rightarrow \prod_{j \in J} X_j$, defined by $\rho_J = \prod_{j \in J} \rho_j$.

Definition: A *simple projection homomorphism* h from a structured MDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ to a structured MDP $\mathcal{M}' = \langle S', A', \Psi', P', R' \rangle$ is a surjection from Ψ to Ψ' , defined by a tuple of surjections $\langle f, g_1, g_2, \dots, g_n \rangle$, with $h(s, a) = (f(s), g_s(a))$, where $f = \rho_F : S \rightarrow S'$, where $F \subseteq \{1, \dots, M\}$ and $g_s : A_s \rightarrow A'_{f(s)}$ for $s \in S$, such that $\forall s, s' \in S, a \in A_s$:

$$P'(f(s), g_s(a), f(s')) = \prod_{j \in F} \text{Prob}(s'_j | \text{Pre}(s'_j, a)) \quad (3)$$

$$R'(f(s), g_s(a)) = R(s, a). \quad (4)$$

The first condition implies that the subset F should be chosen such that the features chosen are sufficient to describe the block transition dynamics of the system. In other words, the

subgraph of the 2-TBN described by the projection should include all the incoming arcs incident on the chosen nodes. The second condition requires that all the parents and the incoming arcs to the reward node are also included in the subgraph. To find such homomorphic projections, we just need to work back from the reward node including arcs and nodes, until we reach the desired subgraph. Such an algorithm will run in time polynomial in the number of features. It is evident that the space of such simple projections is much smaller than that of general maps and in general may not contain a homomorphism reducing a given MDP. Without suitable constraints, often derived from prior knowledge of the structure of the problem, searching for general structured homomorphisms results in a combinatorial explosion. Abstraction algorithms developed by Boutilier and colleagues can be modeled as converging to constrained forms of structured morphisms assuming various representations of the CPTs—when the space of morphisms is defined by boolean formulae of the features [Boutilier and Dearden, 1994], when it is defined by decision trees on the features [Boutilier *et al.*, 1995] and when it is defined by first-order logic formulae [Boutilier *et al.*, 2001].

6 Abstraction in Hierarchical Systems

In the previous section we showed that SMDP homomorphisms can model various abstraction schemes in “flat” MDPs and SMDPs. SMDP homomorphisms are a convenient and powerful formalism for modeling abstraction schemes in hierarchical systems as well. Before we explore various abstraction approaches we first introduce a hierarchical architecture that supports abstraction.

6.1 Hierarchical Markov Options

Recently several hierarchical reinforcement learning frameworks have been proposed [Parr and Russell, 1997; Sutton *et al.*, 1999; Dietterich, 2000] all of which use the SMDP formalism. In this article the hierarchical framework we adopt is the *options framework* [Sutton *et al.*, 1999], though the ideas developed here are more generally applicable. Options are actions that take multiple time steps to complete. They are usually described by: the policy to follow while the option is executing, the set of states in which the option can begin execution, and the termination function β which gives the probability with which the option can terminate in a given state. The resulting systems are naturally modeled as SMDPs with the transition time distributions induced by the option policies. We present an extension to the options framework that readily facilitates modeling abstraction at multiple levels of the hierarchy using SMDP homomorphisms.

We consider the class of options known as Markov options, whose policies satisfy the Markov property and that terminate on achieving a certain sub-goal. In such instances it is possible to implicitly define the option policy as the solution to an *option MDP*, or an option SMDP if the option has access to other options. Accordingly we define a hierarchical Markov sub-goal option:

Definition: A *hierarchical Markov sub-goal option* of an SMDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ is the tuple $O = \langle \mathcal{M}_O, \mathcal{I}, \beta \rangle$,

where $\mathcal{I} \subseteq S$ is the initiation set of the option, $\beta : S \rightarrow [0, 1]$, is the termination function and $\mathcal{M}_O$ is the option SMDP.

The state set of $\mathcal{M}_O$ is a subset of S and constitutes the *domain* of the option. The action set of $\mathcal{M}_O$ is a subset of A and may contain other options as well as “primitive” actions in A . The reward function of $\mathcal{M}_O$ is chosen to reflect the sub-goal of the option. The transition probabilities of $\mathcal{M}_O$ are induced by P and the policies of lower level options. We assume that the lower-level options are following fixed policies which are optimal in the corresponding option SMDPs. The option policy π is obtained by solving $\mathcal{M}_O$, treating it as an episodic task with the possible initial states of the episodes given by $\mathcal{I}$ and the termination of each episode determined by the option’s termination function β .

For example, in the gridworld task shown in Figure 2(a), an option to pick up the object and exit room 1 can be defined as the solution to the problem shown in 2(b), with a suitably defined reward function. The domain and the initiation set of the option are all states in the room and the option terminates on exiting the room with or without the object.

6.2 Option Specific Abstraction

The homomorphism conditions (1) and (2) are very strict and frequently we end up with trivial homomorphic images when deriving abstractions based on a non-hierarchical SMDP. But it is often possible to derive non-trivial reductions if we restrict attention to certain sub-problems, i.e., certain sub-goal options. In such cases we can apply the ideas discussed in Sections 4 and 5 to an option SMDP directly to derive abstractions that are specific to that option. The problem of learning the option policy is transformed to the usually simpler problem of learning an optimal policy for the homomorphic image.

6.3 Relativized Options

Consider the problem of navigating in the gridworld environment shown in Figure 2(a). The goal is to reach the central corridor after collecting all the objects in the environment. No non-trivial homomorphic image exists of the entire problem. But there are many similar components in the problem, namely, the five sub-tasks of getting the object and exiting *room_i*. We can model these similar components by a “partial” homomorphic image—where the homomorphism conditions are applicable only to states in the rooms. One such partial image is shown in Figure 2(b). Employing such an abstraction lets us compactly represent a related family of options, in this case the tasks of collecting objects and exiting each of the five rooms, using a single option MDP. We refer to this compact option as a *relativized option*. Such abstractions are an extension of the notion of relativized operators introduced by [Iba, 1989]. Formally we define a relativized option as follows:

Definition: A *relativized option* of an SMDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ is the tuple $O = \langle h, \mathcal{M}_O, \mathcal{I}, \beta \rangle$, where $\mathcal{I} \subseteq S$ is the initiation set, $\beta : S' \rightarrow [0, 1]$ is the termination function and $h = \langle f, g_1, g_2, \dots, g_n \rangle$ is a partial homomorphism from the SMDP $\langle S, A, \Psi, P, R_O \rangle$ to the option SMDP $\mathcal{M}_O$ with R_O chosen based on the sub-task.

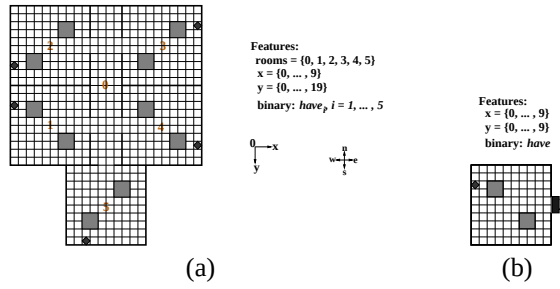


Figure 2: (a) A simple rooms domain with similar rooms and usual stochastic gridworld dynamics. The task is to collect all 5 objects (black diamonds) in the environment and reach the central corridor. The shaded squares are obstacles. (b) The option MDP corresponding to a *get-object-and-leave-room* option.

Here the state set of $\mathcal{M}_O$ is $S' = f(S_O)$, where S_O is the domain of the option, and the admissible state-action set is $h(\Psi)$. Going back to our example in Figure 2(a), we can now define a single *get-object-and-leave-room* relativized option using the option MDP of Figure 2(b). The policy learned in this option MDP can then be suitably lifted to $\mathcal{M}$ to provide different policy fragments in the different rooms. Figure 3 demonstrates the speed-up in performance when using a single relativized option as opposed to five regular options. In this experiment the option policies and the higher level policy were learned simultaneously.²

6.4 Modeling Higher Level Structure

SMDP homomorphisms have the power to model a broader class of abstractions, those not supported by the base level dynamics of the system but which can be induced at some intermediate levels of the hierarchy. For example, consider a multi-link assembly line robot arm that needs to move objects from one assembly line to another. The state of the system is described by the joint angles and velocities, *object-shape*, *object-orientation*, and Boolean variables indicating whether the object is *firmly-grasped* and whether the object is *at-target-location*. The primitive actions are various joint torques. Depending on the configuration of the arm and the shape and orientation of the object, this task requires different sequences of actions, or policies, even though conceptually the higher level task has the same “structure”—grasp object, move object, and place object.

We can model this higher level task structure using a relativized option. First, we define suitable grasp-object options, such as *grasp-cylinder*, *grasp-block* etc., and similar options for moving and placing objects. Then we form a partial homomorphic image of the resulting SMDP—the state set of the image is described by the two Boolean features and the action set consists of *grasp*, *move* and *place* actions. The partial homomorphism, which applies only to the admissible state-option pairs of the SMDP, consists of $f = \rho_J$, where $J = \{ \text{firmly-grasped}, \text{at-target-location} \}$ and $g(\langle \cdot, \text{object-shape} \rangle(\text{grasp-object-shape})) = \text{grasp}$ for all

²In [Ravindran and Barto, 2002] we have reported more detailed experiments in this setting.

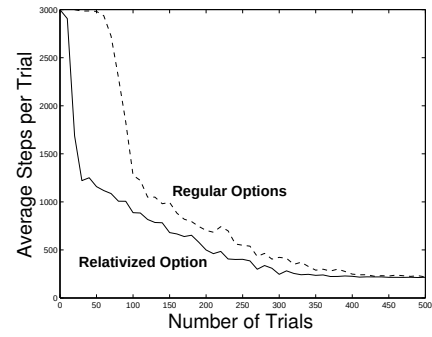


Figure 3: Comparison of performance of learning agents employing regular and relativized options on the task shown in Figure 2.

object-shape and similarly for the *move-object* and *place-object* options. A relativized option with this partial image and partial homomorphism as the option MDP and option homomorphism, respectively, captures the desired conceptual structure. Executing the optimal policy for this option results in action sequences of the form: *grasp*, *move*, *place*, ... Depending on the *object-shape* these abstract actions get bound to different lower level options. While techniques in symbolic AI have long been able to model such higher level task structure, the reinforcement learning community lacked an obvious mechanism to do the same. Our work gives us new tools to model conceptual task structure at various temporal and spatial scales.

6.5 Relation to MaxQ Safe State-abstraction

[Dietterich, 2000] introduced safe state-abstraction conditions for the MaxQ architecture, a hierarchical learning framework related to the options framework. These conditions ensure that the resulting abstractions do not result in any loss of performance. He assumes that the sub-problems at different levels of the hierarchy are specified by factored MDPs. The MaxQ architecture employs a form of value function decomposition and some of the safe abstraction conditions apply only to this form of decomposition. The following condition is more universal and applies to the hierarchical Markov options framework as well:

Definition: A Projection ρ_J is *safe* if: (i) for all (s, a) in Ψ and s' in S , $P(s, a, s', N) = \text{Prob}(\rho_J(s'), N | \rho_J(s), a) \times \text{Prob}(\rho_{M-J}(s') | s, a)$, and (ii) for all $(s, a), (t, a)$ in Ψ , if $\rho_J(s) = \rho_J(t)$, then $R(s, a, N) = R(t, a, N)$.

Condition (i) states that the transition probability can be expressed as a product of two probabilities, one of which describes the evolution of a subset of the features and depends only on that subset. Condition (ii) states that if two states project to the same abstract state, then they have the same immediate reward. From equations 1-4, it is evident that the above conditions are equivalent to the SMDP homomorphism conditions if we restrict our attention to simple projection homomorphisms and do not consider action remapping. Thus the SMDP homomorphism conditions introduced here generalize Dietterich’s safe state-abstraction condition as applicable to the hierarchical Markov option framework.

7 Discussion

The equivalence classes induced by SMDP homomorphisms satisfy the stochastic version of the substitution property [Hartmanis and Stearns, 1966]. This property is also closely related to *lumpability* in Markov chains [Kemeny and Snell, 1960] and *bisimulation homogeneity* [Givan et al., 2003] in MDPs. We chose the SMDP homomorphism as our basic formalism because we believe that it is a simpler notion and provides a more intuitive explanation of various abstraction schemes.

The homomorphism conditions (1) and (2) are very strict conditions that are often not met exactly in practice. One approach is to relax the homomorphism conditions somewhat and allow small variations in the block transition probabilities and rewards. We have explored this issue [Ravindran and Barto, 2002], basing our approximate homomorphisms on the concept of *Bounded-parameter MDPs* developed by [Givan et al., 2000]. We are currently working on extending approximate homomorphisms to hierarchical systems so as to accommodate variations in transition-time distributions.

Although SMDP homomorphisms are powerful tools in modeling abstraction, finding a minimal image of a given SMDP is an NP-hard problem. While taking advantage of structure allows us to develop efficient algorithms in special cases, much work needs to be done to develop efficient general purpose algorithms. Currently we are investigating methods that allow us to determine homomorphisms given a set of candidate transformations in a hierarchical setting.

In this article we presented a novel definition of SMDP homomorphism that employs state-dependent recoding of actions. This allows us to extend existing minimization and abstraction methods to a richer class of problems. We developed notions of equivalence specialized to factored representations and hierarchical architectures. We have shown that SMDP homomorphism can serve as the basis for modeling a variety of abstraction paradigms.

Acknowledgments

We wish to thank Dan Bernstein and Mike Rosenstein for many hours of useful discussion and Bob Givan and Matt Greig for clarifying certain ideas from their work. This material is based upon work supported by the National Science Foundation under Grant No. ECS-0218125 to Andrew G. Barto and Sridhar Mahadevan. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

- [Arbib and Manes, 1975] M. A. Arbib and E. G. Manes. *Arrows, Structures and Functors*. Academic Press, New York, NY, 1975.
- [Boutilier and Dearden, 1994] C. Boutilier and R. Dearden. Using abstractions for decision theoretic planning with time constraints. In *Proceedings of the AAAI-94*, pages 1016–1022. AAAI, 1994.
- [Boutilier et al., 1995] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proceedings of International Joint Conference on Artificial Intelligence 14*, pages 1104–1111, 1995.
- [Boutilier et al., 2001] Craig Boutilier, Ray Reiter, and Robert Price. Symbolic dynamic programming for first-order mdp. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 541–547, 2001.
- [Dean and Givan, 1997] T. Dean and R. Givan. Model minimization in Markov decision processes. In *Proceedings of AAAI-97*, pages 106–111. AAAI, 1997.
- [Dean and Kanazawa, 1989] Thomas Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computer Intelligence*, 5(3):142–150, 1989.
- [Dietterich, 2000] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Artificial Intelligence Research*, 13:227–303, 2000.
- [Emerson and Sistla, 1996] E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, 1996.
- [Givan et al., 2000] R. Givan, S. Leach, and T. Dean. Bounded-parameter Markov decision processes. *Artificial Intelligence*, 122:71–109, 2000.
- [Givan et al., 2003] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in Markov decision processes. To appear in *Artificial Intelligence*, 2003.
- [Hartmanis and Stearns, 1966] J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines*. Prentice-Hall, Englewood Cliffs, NJ, 1966.
- [Iba, 1989] Glenn A. Iba. A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3:285–317, 1989.
- [Kemeny and Snell, 1960] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. Van Nostrand, Princeton, NJ, 1960.
- [Parr and Russell, 1997] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Proceedings of Advances in Neural Information Processing Systems 10*, pages 1043–1049. MIT Press, 1997.
- [Ravindran and Barto, 2001] B. Ravindran and A. G. Barto. Symmetries and model minimization of Markov decision processes. Technical Report 01-43, University of Massachusetts, Amherst, 2001.
- [Ravindran and Barto, 2002] Balaraman Ravindran and Andrew G. Barto. Model minimization in hierarchical reinforcement learning. In Sven Koenig and Robert C. Holte, editors, *Proceedings of the Fifth Symposium on Abstraction, Reformulation and Approximation (SARA 2002), Lecture Notes in Artificial Intelligence 2371*, pages 196–211, New York, NY, August 2002. Springer-Verlag.
- [Sutton et al., 1999] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.
- [Zeigler, 1972] Bernard P. Zeigler. On the formulation of problems in simulation and modelling in the framework of mathematical system theory. In *Proceedings of the Sixth International Congress on Cybernetics*, pages 363–385. Association Internationale de Sybernétique, 1972.
- [Zinkevich and Balch, 2001] M. Zinkevich and T. Balch. Symmetry in Markov decision processes and its implications for single agent and multi agent learning. In *Proceedings of the 18th International Conference on Machine Learning*, pages 632–640, San Francisco, CA, 2001. Morgan Kaufmann.

PROBABILISTIC PLANNING

PROBABILISTIC PLANNING

Covariant Policy Search

J. Andrew Bagnell and Jeff Schneider

Robotics Institute

Carnegie-Mellon University

Pittsburgh, PA 15213

{dbagnell,schneide}@ri.cmu.edu

Abstract

We investigate the problem of non-covariant behavior of policy gradient reinforcement learning algorithms. The policy gradient approach is amenable to analysis by information geometric methods. This leads us to propose a natural metric on controller parameterization that results from considering the manifold of probability distributions over paths induced by a stochastic controller. Investigation of this approach leads to a covariant gradient ascent rule. Interesting properties of this rule are discussed, including its relation with actor-critic style reinforcement learning algorithms. The algorithms discussed here are computationally quite efficient and on some interesting problems lead to dramatic performance improvement over non-covariant rules.

1 Introduction

Much recent work in reinforcement learning and stochastic optimal control has focused on algorithms that search directly through a space of policies rather than building approximate value functions. Policy search has numerous advantages: domain knowledge may be easily encoded in a policy, the policy may require less representational power than a value-function approximation, there are simple extensions to the multi-agent domain, and convergent algorithms are known. Furthermore, policy search approaches have recently scored some encouraging successes [Bagnell and Schneider, 2001] [Baxter *et al.*, 1999] including helicopter control and game-playing.

One interesting aspect of existing gradient search algorithms, however, is that they are **non-covariant**; that is, a simple re-parameterization of the policy typically leads to a different gradient direction. (Not that found by applying the Jacobian of the re-parameterization to the gradient). This is an odd result; it is intuitively difficult to justify the gradient computation as actually indicating the direction of steepest descent. This problem is well recognized in the pattern-recognition and statistics communities and is a very active area of research. [Kakade, 2002] was, to the best of our knowledge the first to identify this problem in reinforcement learning and to suggest that techniques from information geometry may prove valuable in its solution.

Inspired by the work of [Amari and Nagaoka, 2000], Kakade proposed a “natural gradient” algorithm. In particular, [Kakade, 2002] proposed a scheme for generating a metric on parameter space that has interesting theoretical properties. Most convincingly, Kakade showed strong empirical evidence for the algorithm’s usefulness. In particular, [Kakade, 2002] applies the algorithm to the Tetris problem of [Bertsekas and Tsitsiklis, 1996]. This problem is particularly interesting as value function methods (as for example described in [Bertsekas and Tsitsiklis, 1996]) demonstrate non-monotone performance; first policy iteration improves the policy dramatically, but after a small number of iterations the policy gets much worse. Normal gradient methods, including second-order and conjugate methods also prove very ineffective on this problem; even after a tremendous number of rounds they only mildly increase the performance of the game-player. The method presented in [Kakade, 2002] shows rapid performance improvement and achieves a significantly better policy than value-function methods (at their peak) in comparable time.

However, despite recognizing an interesting defect in the general approach and intuiting an apparently powerful algorithm, Kakade concludes that his method also fails to be covariant leaving the problem open. We present here what we believe to be an appropriate solution.

In Kakade’s work, there is no proper probability manifold, but rather a collection of such (one for each state) based on the policy. As such, Kakade must rely on an ad-hoc method for generating a metric. Here instead we take a proper manifold, the distribution over paths induced by the controller, and compute a metric based on that. In the special case appropriate to the average reward RL formulation, Kakade’s scheme is shown to give rise to a bona-fide natural metric, despite the observation in the paper that the learning was non-covariant. We believe this is an artifact— perhaps of step-length. Further, we note that parametric invariance does **not** require this metric— there are numerous covariant metrics. Rather, a stronger probabilistically natural invariance demands the metric used. We describe this result to motivate our method.

Importantly, the notion of the metric on the path-distribution allows us to provide very simple and natural extensions to Kakade’s algorithm that cover the finite horizon, discounted start-state, and partially-observed reinforcement learning problems as well as the average reward one. Fi-

nally, for completeness, we describe the result discovered by Kakade relating the invariant metric and compatible actor-critic methods [Sutton *et al.*, 1999].

1.1 Problem Setup and Notation

A stochastic control problem consists of paths ξ (also called system trajectories) in a space Ξ , a distribution over path-space, $p(\xi)$, that is a function of a sequence of (finite) controls $(a_t)_{t \in \{0, \dots, T\}}$, indexed by time, from a space A . Throughout this paper we will be considering Partially Observed Markov Decision Problems, in which there are state-variables (x_t) of the system that compose a path ξ and render the past and future independent. In a POMDP, $p(\xi)$ is defined by an initial state and next state transition probabilities $p(x_t | x_{t-1}, a_{t-1})$. We also typically have a sequence of outputs (observations) $(y_t)_{t \in \{0 \dots T\}}$ measurable with respect to x_t . A controller (or policy), π , usually parameterized by θ , is a feedback-type strategy that maps the history of observations $(y_t)_{t \in \{0 \dots \tau\}}$ to a distribution over controls a_τ . Most of the derivations will be given in terms of memoryless stochastic controllers that map the current state to a distribution over actions, although they can be easily extended to finite window or recurrent representations that operate on only the observations. The goal in a control problem is to maximize the expected reinforcement $J(\theta) = \sum_{\xi} p_{\theta}(\xi) r(\xi)$ with respect to θ . The sum is always assumed to exist for all controllers. The reward function on paths in a POMDP is additive in time (or in the infinite time case, discounted or averaged), and a function $R(x)$ of state, although the algorithms discussed here are not necessarily predicated on that assumption. Reinforcement learning is the adaptive version of the control problem where we attempt to maximize the expected reinforcement by sampling trajectories and then improving our policy. We use the notation ∂_i throughout to denote $\frac{\partial}{\partial \theta_i}$ where the parameter θ should be clear from context. For vectors x and y , we use the notation $(x|y)_M$ to denote the inner product with respect to metric M . The notation $\langle f(x) \rangle_{p(x)}$ indicates the expected value of the function f with respect to the distribution p .

2 Covariance and Riemannian manifolds

2.1 Meaning of steepest ascent

For direct policy search methods, we are interested in finding the direction of *steepest ascent* with respect to our current parameters. That is, we wish to maximize the reward function $J(\theta + \Delta\theta)$ subject to $(\Delta\theta | \Delta\theta) = \epsilon$, an infinitesimal. If we reparameterize our controller in terms of, e.g. ζ , and express the same effective infinitesimal policy change in terms of these parameters, ΔJ will of course remain the same. However, if we measure lengths using the naive dot product, the same effective change will not have the same size. It is this problem that leads to the odd behavior of “steepest descent” rules.

2.2 Non-covariant learning rule

To demonstrate the problem we first consider a simple two state, two action MDP described in [Kakade02]. (Figure 1) In state 0 executing action 0 returns to the same state and gets a reward of 1. Executing action 1 get no reward but transits to

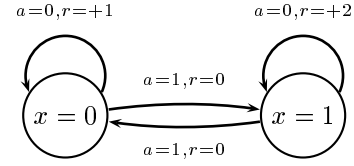


Figure 1: A two-state MDP. Each state has two controls one which self transitions (and earns the reward that labels the state) and another that simply transitions to the other state. Rewards occur only on the transitions between different states.

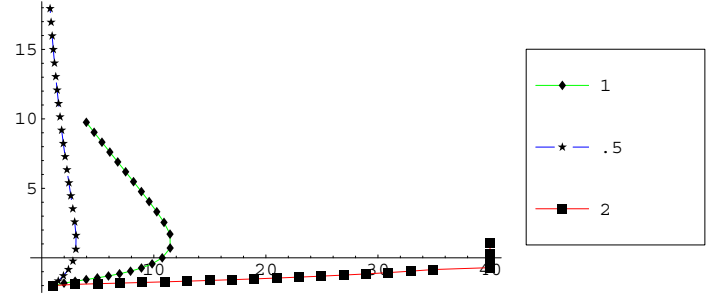


Figure 2: Log odds of actions for policies $\log p(a=0|x)/p(a=1|x)$ on the two-state MDP (horizontal axis corresponds to state 0). Different curves correspond to varying λ . Notice the non-covariant policy behavior.

state 1. In state 1, action 0 returns to state 1 and gets reward 2. Action 1 transits to state 0 and achieves no reward.

Consider parameterized probabilistic policies of the form $\pi(a=0|x; \theta) = \frac{1}{1 + e^{\lambda \theta_0 \delta x=0 + \theta_1 \delta x=1}}$. λ is an arbitrary scale parameter that we use to demonstrate that even mildly different parameterization lead to dramatically different behavior. Below we plot the resulting track through the space of policies using the log ratio of probabilities of the actions for each of the possible states. We start from a policy that makes it somewhat more likely to choose action 0 in state 0 and action 1 in state 1. We then scale λ to 1, .5, and 2 and plot the log odds of the policy from state 1 (Figure 2).

The non-covariant behavior is quite clear in this graph. Further, we note that it is *very* difficult to achieve a good policy using this algorithm from this starting point as the *wrong* action becomes overwhelmingly likely to be chosen from state 0. If sampling were used to compute the gradient, we would nearly never get samples from state 1— which we need to improve the policy.

2.3 Path distribution manifolds

The control problem is essentially a coupled one of optimization and integration over the space of possible paths, Ξ . This motivates the idea that instead of considering the (arbitrary) distance in terms of parameterization of the policy, we may consider distance in terms of the change in distribution over paths resulting from the policy change. We may view the distribution over paths $p(\xi; \theta)$ as a parameterized manifold (nominally embedded in $R^{|\Xi|}$) of dimension $|\theta|$. This takes some work to visualize; as an example consider a space of three possible paths. All possible distributions over these three paths can be smoothly represented with two parameters. In Figure 3 (left) we see one visualization with the embedding

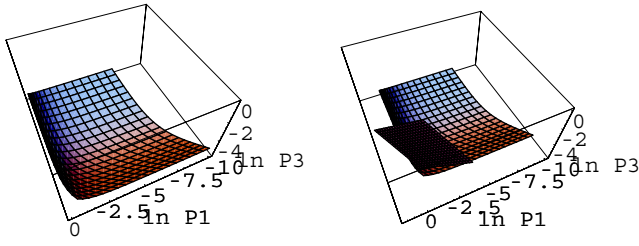


Figure 3: An example probability manifold over paths. Each axis represents the log probability of one of the three possible paths. The manifold is formed as we vary the parameters defining it throughout their domain. On the right we attach the tangent space at a particular point in parameter space.

$\log p(\xi; \theta) :$

With n parameters, assuming no redundancy, we generate an n dimensional manifold. In the case pictured, it is the set of all distributions on 3 paths, but in general, the dimensionality of the path probability manifold will be tremendously less than the number of possible paths. It is important to understand the manifold under consideration is that of the probability distribution over paths and not of paths themselves.

The study of parameterized manifolds like that pictured above is the domain of differential geometry. Here we are interested in establishing a *Riemannian* structure on the manifold of paths. By that we mean we wish to establish a metric on the tangent space (the local linear approximation to the manifold at a point ξ), so we can measure small parameter changes. We do this with the metric $(X||Y)_{p_\theta(\xi)}$ on the tangent space (which is spanned by the partials with respect to each parameter) as $\sum_{i,j} G^{ij}(\xi) X_i Y_j$, where G is a positive definite matrix. This is a very natural thing to do— instead of just the standard dot product, we have a dot product that allows us to represent rotations and scalings (exactly what a positive definite matrix can represent) and that can vary throughout the manifold. In Figure 3 (right) we depict the tangent space at a point in our parameterization as the local linear approximation at that point.

2.4 Steepest ascent on Riemannian manifold

There are two questions that then naturally arise— what is steepest descent on a function defined over a Riemannian space, and is there a Riemannian metric on the manifold of the paths that is in some sense natural. We quickly answer the first question, and pursue the second in the next two sections. A Lagrange multiplier argument (given schematically here) makes it easy to see the form of the steepest descent direction.

$$\min L(\theta + \delta\theta) = L(\theta) + \nabla L(\theta) \cdot \delta\theta \quad (1)$$

$$\text{subject to } (\theta||\theta) = \epsilon \quad (2)$$

Form the Lagrangian:

$$\begin{aligned} \mathcal{L}(\theta, \lambda) &= L(\theta) + \nabla L(\theta) \cdot \delta\theta + \\ &\quad \lambda(G^{ij}\delta\theta_i\delta\theta_j - \epsilon) \end{aligned} \quad (3)$$

and take derivatives with respect to each $\delta\theta$, then set to zero to solve for the optimal direction:

$$\nabla_{\delta\theta} \mathcal{L}(\delta\theta, \lambda) = \nabla L(\theta) + \lambda G^{ij}\delta\theta_j = 0$$

This implies that (since G is positive definite and hence invertible), giving us:

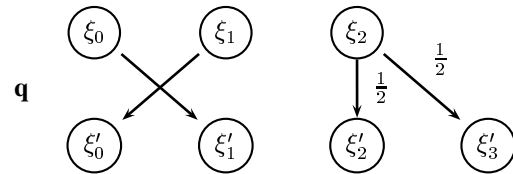
$$\delta\theta \propto G^{-1}\nabla L(\theta) \quad (4)$$

That is, the direction of steepest descent is simply the normal gradient times the inverse of the metric evaluated at the point of tangency. We call this the “natural gradient” for the Riemannian manifold. [Amari and Nagaoka, 2000]

3 Invariance and Chentsov’s Theorem

Some confusion seems to surround the choice of metric on probability manifolds. There is *no* unique answer for this metric, even under the supposition of parametric invariance as suggested by some authors. The issue is rather more subtle. Any metric that transforms under parameter change according to the Jacobian of the function connecting parameters meets this requirement. This type of parametric covariance is a minimum requirement for us. It is natural to suggest a metric that preserves essential probabilistic aspects of the manifold. Consider, for example, functions q (Markov mappings, congruent embeddings, or sufficient statistics, depending on your viewpoint) that carry one distribution over paths to another in a natural and recoverable way. For example, consider the mapping between the two distributions $P(\text{paths})$ and $P(\text{paths}')$ given by congruent embedding q depicted below:

Paths



Paths'

The mapping q above interchanges the role of two paths and splits one path into two (with probability $1/2$ for each split). In a probabilistically fundamental way, the manifolds $P(\text{path})$ and $P(\text{path}')$ (where we imagine $P(\text{path})$ is a smoothly parameterized set of distributions) are similar to each other. For each path' we can uniquely recover the original probability. In this way, a parameterized distribution over paths can be embedded into a different path space. (This equivalence can actually be phrased in a category theoretic way where the morphisms are these congruent embeddings.) [Chentsov, 1980] General congruent embeddings can

be thought of as simply generalizations of the example depicted above to allow arbitrary permutations and arbitrary probabilities for different paths stemming from a single path, as well as compositions of these two. In the control problem this could arise by simple permutation of the state variables or change of co-ordinates to ones with redundant information. It is natural to require that with our metric the congruent embedding is an isometry on the tangent space. (i.e. preserves the length of the vectors). That is, if we make a change of size ϵ (under the metric G_θ) to the distribution P_θ (paths) then carrying that change through q we should measure the same change ϵ using $G_{q(P_\theta)}$. Adding the requirement of invariance with respect to congruent embeddings leads to a unique (up to scale) metric on the manifold. [Chentsov, 1980] This metric is well-known in statistical inference as the Fisher-Rao metric and it can be written as the Fisher information matrix [DeGroot, 1970]:

$$G = \langle \partial_i \log p(\xi; \theta) \partial_j \log p(\xi; \theta) \rangle_{p(\xi; \theta)}$$

Another way to derive the metric is to think about “distances” on probability spaces. The KL-divergence (or relative entropy) between two distributions is a natural divergence on changes in a distribution. It is also manifestly invariant to re-parameterization. If we think about derivatives as small changes in parameters (differentials) then we discover that we also get a unique metric as the second-order Taylor expansion of the KL-divergence. This too agrees with the Fisher information (up to scale). We note that the direction of the KL-divergence is irrelevant as to second-order KL(p, q) = KL(q, p) [Amari and Nagaoka, 2000].

4 Fisher-Rao Metric on the Path-space Manifold

The issue now becomes how to derive the Fisher metric on the space of path distributions. It turns out in the case of processes with underlying Markovian state to be rather easy, and involves only computations we already make in the likelihood ratio approach standard in gradient-based reinforcement learning.

4.1 Derivation of finite-time path metric

The Fisher information metric involves computing $\langle \partial_i \log p(\xi; \theta) \partial_j \log p(\xi; \theta) \rangle_{p(\xi; \theta)}$. Fortunately, this is easy to do. The essential algorithm within gradient methods like REINFORCE and GPOMDP is a clever method of computing $\langle \partial_i \log p(\xi; \theta) \rangle_{p(\xi; \theta)}$, the expected score function. Thus, while the expected score is the gradient, the correlation of the score is the Fisher matrix. The following simple algorithm is unbiased and converges almost surely (under the same regularity conditions as in [Baxter *et al.*, 1999] as the number of sample paths m goes to infinity:

Algorithm 1 (Finite-horizon metric computation) For i in 1 to m :

- Sample a path using the policy $\pi(\theta)$: $\xi_i = (x_1, a_1, x_2, a_2, \dots, x_n)$
- For t in 1 to n :

– Compute $z_j = \frac{t-1}{t}z + \frac{1}{t}\partial_j \log \pi(a|x; \theta)$

- $G = \frac{i-1}{i}G + \frac{1}{i}z^T z$

Return G

We use the Markov property and the invariance of the transition probabilities (given the actions) in the algorithm above to compute $\partial_i \log p(\xi; \theta)$ simply. These details can be extracted from the proofs below, as well as other, potentially better ways to sample.

To compute the natural gradient, we simply invert this matrix and multiply it with the gradient computed using standard policy search methods.

4.2 Limiting metric for infinite horizon problems

A well-known result in statistics [DeGroot, 1970] gives a different form for the Fisher metric under appropriate regularity conditions. We quickly derive that result, and then show how this gives us a simple form for the path probability metric as $t \rightarrow \infty$.

$$\begin{aligned} G_\theta &= \langle \partial_i \log p(\xi; \theta) \partial_j \log p(\xi; \theta) \rangle_{p(\xi; \theta)} \\ &= \sum_{\Xi} \partial_i p(\xi; \theta) \partial_j (\log p(\xi; \theta)) \\ &= \sum_{\Xi} \partial_i (p(\xi; \theta) \partial_j \log p(\xi; \theta)) - \sum_{\Xi} p(\xi; \theta) \partial_i \partial_j \log p(\xi; \theta) \\ &= -\langle \partial_i \partial_j \log p(\xi; \theta) \rangle_{p(\xi; \theta)} + \partial_i \sum_{\Xi} \partial_j p(\xi; \theta) \\ &= -\langle \partial_i \partial_j \log p(\xi; \theta) \rangle_{p(\xi; \theta)} \end{aligned} \quad (5)$$

The third line follows from the second by integrating by parts and the fifth follows by observing that the total probability is constant.

Now we show that in the limit this leads to a simple metric for the infinite horizon case. To get a convergent metric, we must normalize the metric, in particular by the total length of the path denoted t . Since the metric is defined only up to scale, this is perfectly justified.

Theorem 1 (Infinite-Horizon Metric) For an ergodic Markov process the Fisher information matrix limits to the expected Fisher information of the policy for each state and control under stationary distribution of states and actions. *Proof:* We use G_θ^t to indicate the t -step finite-horizon metric.

$$\begin{aligned} \lim_{t \rightarrow \infty} \frac{1}{t} G_\theta^t &= \lim_{t \rightarrow \infty} -\frac{1}{t} \langle \partial_i \partial_j \log p(\xi; \theta) \rangle_{p(\xi; \theta)} \\ &= \lim_{t \rightarrow \infty} -\frac{1}{t} \sum_{\Xi} p(\xi; \theta) \partial_i \frac{\partial_j p(\xi; \theta)}{p(\xi; \theta)} \end{aligned} \quad (6)$$

For a Markov process we can write the likelihood ratio $\frac{\partial_i p(\xi; \theta)}{p(\xi; \theta)}$ as

$$\frac{\partial_i p(\xi; \theta)}{p(\xi; \theta)} = \sum_t \frac{\partial_i p(x_{t+1}|x_t; \theta)}{p(x_{t+1}|x_t; \theta)}$$

Using the chain rule we continue the derivation with $d^\pi(x)$ indicating the stationary distribution:

$$\begin{aligned}
\lim_{t \rightarrow \infty} \frac{1}{t} G_\theta &= \lim_{t \rightarrow \infty} -\frac{1}{t} \left\langle \sum_t \frac{\partial_i \partial_j p(x_{t+1}|x_t; \theta)}{p(x_{t+1}|x_t; \theta)} - \right. \\
&\quad \left. \frac{\partial_i p(x_{t+1}|x_t; \theta) \partial_j p(x_{t+1}|x_t; \theta)}{p(x_{t+1}|x_t; \theta) p(x_{t+1}|x_t; \theta)} \right\rangle_{p(\xi; \theta)} \\
&= - \sum_{x,a} d^\pi(x) \pi(a|x; \theta) \left(\frac{\partial_i \partial_j \pi(a_t|x_t; \theta)}{\pi(a_t|x_t; \theta)} \right) + \\
&\quad \frac{\partial_i \pi(a_t|x_t; \theta) \partial_j \pi(a_t|x_t; \theta)}{\pi(a_t|x_t; \theta) \pi(a_t|x_t; \theta)} \\
&= \sum_{x,a} d^\pi(x) \pi(a|x; \theta) \frac{\partial_i \pi(a_t|x_t; \theta) \partial_j \pi(a_t|x_t; \theta)}{\pi(a_t|x_t; \theta)^2} \\
&\quad - \sum_x d^\pi(x) \sum_a \partial_i \partial_j \pi(a_t|x_t; \theta) \\
&= \langle G_{x;\theta} \rangle_{d^\pi(x)}
\end{aligned}$$

where the second equality follows by noting that the likelihood ratio is independent of transition probability matrix given the action probability, and by application of the ergodic theorem. The last line follows (with $G_{x;\theta}$ the Fisher information for the policy at a given state) as the second term in line 3 vanishes since the total probability is constant with respect to θ . ■

It is also interesting to consider what happens in the start-state, discounted formulation of the problem. In this case we would like our metric, using the general definition over path distributions given above, to naturally weigh the start state more than it necessarily is in the infinite horizon average case. It is well-known that a discount factor γ is equivalent to an undiscounted problem where each trajectory terminates with probability $1 - \gamma$ at each step. We can use this fact to derive a metric more appropriate for the discounted formalism.

Theorem 2 (Start-State Metric) *For a discounted Markov process the Fisher information matrix equals the Fisher information of the policy for each state and control under the limiting distribution of states and actions. Proof: The proof is very similar to the infinite horizon case and so we simply sketch it:*

$$\begin{aligned}
G_\theta &= -\langle \partial_i \log p(\xi; \theta) \partial_j \log p(\xi; \theta) \rangle_{p(\xi; \theta)} \\
&= - \sum_{\Xi} p(\xi; \theta) \partial_i \frac{\partial_j p(\xi; \theta)}{p(\xi; \theta)} \\
&= - \sum_{\Xi} p(\xi; \theta) \partial_i \sum_t \frac{\partial_j \pi(a_t|x_t)}{\pi(a_t|x_t)} \\
&= \sum_{\Xi} p(\xi; \theta) \sum_t \frac{\partial_i \pi(a_t|x_t) \partial_j \pi(a_t|x_t)}{\pi(a_t|x_t)^2} - \\
&\quad \sum_{\Xi} p(\xi; \theta) \sum_t \frac{\partial_i \partial_j \pi(a_t|x_t)}{\pi(a_t|x_t)} \\
&= \left\langle \frac{\partial_i \partial_j \pi(a|x)}{\pi(a|x)} \right\rangle_{d^\pi(x) \pi(a|x_t)}
\end{aligned}$$

■

where $d^\pi(x) = \sum_t \gamma^t Pr^\pi(x_t = x)$ is the limiting distribution of states. Thus the infinite horizon (undiscounted) and start-state metrics give essentially the same metric, with only the effective weighting by states differing.

4.3 Metrics for Partially Observed Problems

For policies that map the observation space of a partially-observed markov decision process into distributions over actions, it is just as easy to derive appropriate metric using our approach. The tuple (x_t, a_t, y_t) is also a markov chain, and with only subtle changes to the arguments above we end up with the same metric except using the limiting distribution of observations instead of states.

5 Relation to Compatible Value Function Actor Critic

Kakade noticed a fascinating connection between the limiting metric given in Theorem 1 and the improvement direction computed by a class of actor-critic methods that use a special compatible function approximation technique. [Sutton *et al.*, 1999; Konda and Tsitsiklis, 2002] Following Kakade we let $\psi(s, a) = \partial_i \log p(a|x; \theta)$ and let the compatible function approximator be linear in ψ :

$$f(x, a; \omega) = \sum_i \omega_i \psi_i$$

This type of value function approximator was initially suggested as it may be used to compute the true gradient. In practice, it has not been clear what advantage this brings over the gradient estimation routines of [Baxter *et al.*, 1999]. However, “folk-wisdom” has it that performing an that infinitesimal policy iteration (moving in the direction of the best policy according to $f^\pi(x, a; \omega)$) using this approximation has very good properties and often significantly outperforms standard gradient methods. The natural gradient provides insight into this behavior. Let ω minimize the squared value-function error:

$$\epsilon(\omega; \theta) = \sum_{x,a} d^\pi(x) \pi(a|x) (f(x, a; \theta) - Q(x, a; \theta))^2$$

where $Q(x, a; \theta)$ is the exact advantage function. [Sutton *et al.*, 1999] It is easy to check that ([Kakade, 2002], Theorem 1) $\omega \propto G_\theta^{-1} \partial_i J(\theta)$. That is, the direction of maximum policy improvement is exactly the natural gradient direction. This can be seen by simply differentiating $\epsilon(\omega; \theta)$ to minimize it with respect to ω and noting the result of [Sutton *et al.*, 1999] that

$$\partial_i J(\theta) = \sum_{x,a} d^\pi(x) Q(x, a; \theta) \partial_i p(a|x; \theta)$$

6 Demonstration

As a consequence of the results of section 5 and [Kakade, 2002], experimental results already exist demonstrating the effectiveness of the natural gradient method. That is, all results using policy improvement with compatible function approximators are implicitly computing this result. As a demonstration, we computed analytically the natural gradient for the

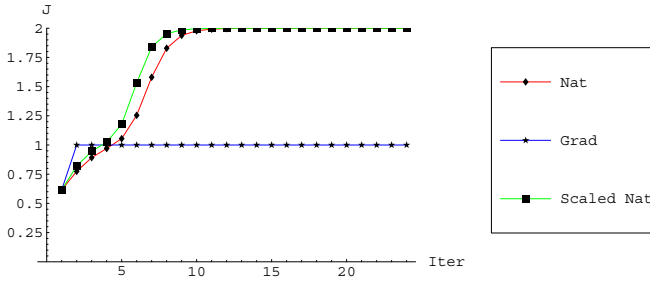


Figure 4: Performance comparison of covariant (nat and scaled) and non-covariant (grad) on the 2-state MDP. The covariant learning algorithm dramatically outperforms the standard gradient algorithm. The standard method achieves $J=2$ after more than 1000 iterations.

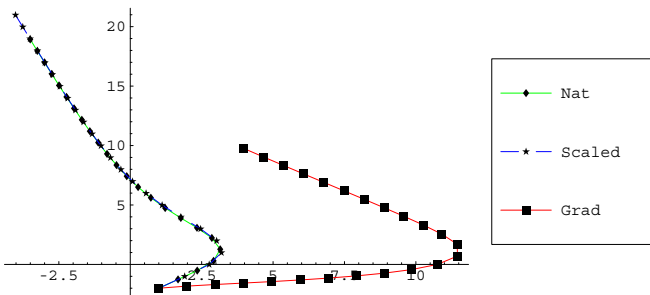


Figure 5: Path through policy space (showing log-odds of the actions with state 0 along the horizontal axis) of covariant (nat and scaled) and non-covariant (grad) algorithms on the 2-state MDP. Note that, as required, the path is independent of scaling for the covariant algorithm.

problem given in Section 2.2. This problem is interesting as it reveals some of the properties of the natural gradient method. First it is easily checked that for a complete Boltzmann policy in an MDP the natural gradient computes:

$$G^{-1}(\theta) \partial_i J(\theta) = \sum_{x,a} Q(x, a; \theta) \partial_i p(a|x; \theta)$$

This means that it is very similar to the standard gradient except that it removes the weighting of states by the stationary distribution. Rather, each state is treated equally. This leads to much more reasonable results in the problem discussed as the partial derivative component for state 1 does not shrink as the policy changes initially.

In Figure 4 we plot the performance of the natural gradient (using two different scalings) and the standard gradient methods in optimizing the policy for this problem.

It is interesting to note that in this graph the behavior of the natural gradient descent algorithm appears to be non-covariant. This is simply due to the step size heuristic not computing equivalent steps in the policy space. The direction however is constant as illustrated in Figure 5.

7 Conclusions

[Kakade, 2002] suggested that covariant behavior of gradient algorithms is an important desiderata in reinforcement learn-

ing. Unfortunately, working in the space of policies, it was difficult to generate such an algorithm. Here we proposed considering the induced path-distribution manifold and used notions from information geometry to propose a natural covariant algorithm. This leads to interesting insight and a practical algorithm. Fortunately, it agrees with the heuristic suggested by Kakade (despite the suggestion in the paper that the algorithm there was actually not covariant) in the infinite horizon case and extends to cover new problems. [Peters *et al.*, 2002] independently developed theory related to ours (in particular the theorems in Section 4.2) and presented results in the context of robot dynamic control.

Further work will provide more investigation into the experimental behavior of the algorithms presented. Future effort may also yield deeper insight into the relationship between the method presented here and value-function approximations.

References

- [Amari and Nagaoka, 2000] S. Amari and H. Nagaoka. *Methods of Information Geometry*. Oxford University Press, 2000.
- [Bagnell and Schneider, 2001] J. Bagnell and J. Schneider. Autonomous helicopter control by policy-search based reinforcement learning. In *Proceedings of the 2001 IEEE Int. Conference on Robotics and Automation*. IEEE, 2001.
- [Baxter *et al.*, 1999] J. Baxter, L. Weaver, and P. Bartlett. Direct-gradient-based reinforcement learning I: Gradient estimation algorithms. Technical report, Computer Sciences Lab, Australian National University, 1999.
- [Bertsekas and Tsitsiklis, 1996] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- [Chentsov, 1980] N.N. Chentsov. *Statistical Decision Rules and Optimal Inference*. American Mathematical Society, 1980.
- [DeGroot, 1970] M. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, 1970.
- [Kakade, 2002] S. Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems (NIPS-14)*, 2002.
- [Konda and Tsitsiklis, 2002] V. Konda and J. Tsitsiklis. Actor-critic algorithms. to appear in the *SIAM Journal on Control and Optimization*, 2002.
- [Peters *et al.*, 2002] Jan Peters, Sethu Vijaykumar, and Stefan Schaal. Policy gradient methods for robot control. Technical Report 00-737, USC, 2002.
- [Sutton *et al.*, 1999] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Neural Information Processing Systems 12*, 1999.

Point-based value iteration: An anytime algorithm for POMDPs

Joelle Pineau, Geoff Gordon and Sebastian Thrun

Carnegie Mellon University
Robotics Institute
5000 Forbes Avenue
Pittsburgh, PA 15213
{jpineau,ggordon,thrun}@cs.cmu.edu

Abstract

This paper introduces the *Point-Based Value Iteration* (PBVI) algorithm for POMDP planning. PBVI approximates an exact value iteration solution by selecting a small set of representative belief points and then tracking the value and its derivative for those points only. By using stochastic trajectories to choose belief points, and by maintaining only one value hyper-plane per point, PBVI successfully solves large problems: we present results on a robotic laser tag problem as well as three test domains from the literature.

1 Introduction

The value iteration algorithm for planning in partially observable Markov decision processes (POMDPs) was introduced in the 1970s [Sondik, 1971]. Since its introduction numerous authors have refined it [Cassandra *et al.*, 1997; Kaelbling *et al.*, 1998; Zhang and Zhang, 2001] so that it can solve harder problems. But, as the situation currently stands, POMDP value iteration algorithms are widely believed not to be able to scale to real-world-sized problems.

There are two distinct but interdependent reasons for the limited scalability of POMDP value iteration algorithms. The more widely-known reason is the so-called curse of dimensionality [Kaelbling *et al.*, 1998]: in a problem with n physical states, POMDP planners must reason about belief states in an $(n - 1)$ -dimensional continuous space. So, naive approaches like discretizing the belief space scale exponentially with the number of states.

The less-well-known reason for poor scaling behavior is what we will call the curse of history: POMDP value iteration is in many ways like breadth-first search in the space of belief states. Starting from the empty history, it grows a set of histories (each corresponding to a reachable belief) by simulating the POMDP. So, the number of distinct action-observation histories considered grows exponentially with the planning horizon. Various clever pruning strategies [Littman *et al.*, 1995; Cassandra *et al.*, 1997] have been proposed to whittle down the set of histories considered, but the pruning steps are usually expensive and seem to make a difference only in the constant factors rather than the order of growth.

The two curses, history and dimensionality, are related: the higher the dimension of a belief space, the more room it has

for distinct histories. But, they can act independently: planning complexity can grow exponentially with horizon even in problems with only a few states, and problems with a large number of physical states may still only have a small number of relevant histories. In most domains, the curse of history affects POMDP value iteration far more strongly than the curse of dimensionality [Kaelbling *et al.*, 1998; Zhou and Hansen, 2001]. That is, the number of distinct histories which the algorithm maintains is a far better predictor of running time than is the number of states. The main claim of this paper is that, if we can avoid the curse of history, there are many real-world POMDPs where the curse of dimensionality is *not* a problem.

Building on this insight, we present *Point-Based Value Iteration* (PBVI), a new approximate POMDP planning algorithm. PBVI selects a small set of representative belief points and iteratively applies value updates to those points. The point-based update is significantly more efficient than an exact update (quadratic vs. exponential), and because it updates both value and value gradient, it generalizes better to unexplored beliefs than interpolation-type grid-based approximations which only update the value [Lovejoy, 1991; Brafman, 1997; Hauskrecht, 2000; Zhou and Hansen, 2001; Bonet, 2002]). In addition, exploiting an insight from policy search methods and MDP exploration [Ng and Jordan, 2000; Thrun, 1992], PBVI uses explorative stochastic trajectories to select belief points, thus reducing the number of belief points necessary to find a good solution compared to earlier approaches. Finally, the theoretical analysis of PBVI included in this paper shows that it is guaranteed to have bounded error.

This paper presents empirical results demonstrating the successful performance of the algorithm on a large (870 states) robot domain called *Tag*, inspired by the game of lasertag. This is an order of magnitude larger than other problems commonly used to test scalable POMDP algorithms. In addition, we include results for three well-known POMDPs, where PBVI is able to match (in control quality, but with fewer belief points) the performance of earlier algorithms.

2 An overview of POMDPs

The POMDP framework is a generalized model for planning under uncertainty [Kaelbling *et al.*, 1998; Sondik, 1971]. A POMDP can be represented using the following n -tuple: $\{S, A, O, b_0, T, \Omega, R, \gamma\}$, where S is a (finite) set of discrete

states, A is a set of discrete actions, and O is a set of discrete observations providing incomplete and/or noisy state information. The POMDP model is parameterized by: $b_0(s)$, the initial belief distribution; $T(s, a, s') := Pr(s_{t+1} = s' | a_t = a, s_t = s)$, the distribution describing the probability of transitioning from state s to state s' when taking action a ; $\Omega(o, s, a) := Pr(o_{t+1} = o | a_t = a, s_{t+1} = s)$, the distribution describing the probability of observing o from state s after taking action a ; $R(s, a)$, the reward signal received when executing action a in state s ; and γ , the discount factor.

A key assumption of POMDPs is that the state is only partially observable. Therefore we rely on the concept of a belief state, denoted b , to represent a probability distribution over states. The belief is a sufficient statistic for a given history:

$$b_t := Pr(s_t | b_0, a_0, o_1, \dots, o_{t-1}, a_{t-1}, o_t) \quad (1)$$

and is updated at each time-step to incorporate the latest action, observation pair:

$$b_t(s') := \eta \Omega(o, s', a) \sum_{s \in S} T(s, a, s') b_{t-1}(s) \quad (2)$$

where η is the normalizing constant.

The goal of POMDP planning is to find a sequence of actions $\{a_0, \dots, a_t\}$ maximizing the expected sum of rewards $E[\sum_t \gamma^t R(s_t, a_t)]$. Given that the state is not necessarily fully observable, the goal is to maximize expected reward for each belief. The value function can be formulated as:

$$V(b) = \max_{a \in A} \left[R(b, a) + \gamma \sum_{b' \in B} T(b, a, b') V(b') \right] \quad (3)$$

When optimized exactly, this value function is always piecewise linear and convex in the belief [Sondik, 1971] (see Fig. 1, left side). After n consecutive iterations, the solution consists of a set of α -vectors: $V_n = \{\alpha_0, \alpha_1, \dots, \alpha_m\}$. Each α -vector represents an $|S|$ -dimensional hyper-plane, and defines the value function over a bounded region of the belief: $V_n(b) = \max_{\alpha \in V_n} \sum_{s \in S} \alpha(s) b(s)$. In addition, each α -vector is associated with an action, defining the best immediate policy assuming optimal behavior for the following $(n-1)$ steps (as defined respectively by the sets $\{V_{n-1}, \dots, V_0\}$).

The n -th horizon value function can be built from the previous solution V_{n-1} using the *Backup* operator, H . We use notation $V = HV'$ to denote an exact value backup:

$$V(b) = \max_{a \in A} \left[\sum_{s \in S} R(s, a) b(s) + \gamma \sum_{o \in O} \max_{a' \in V'} \sum_{s \in S} \sum_{s' \in S} T(s, a, s') \Omega(o, s', a) \alpha'(s') b(s) \right] \quad (4)$$

A number of algorithms have been proposed to implement this backup by directly manipulating α -vectors, using a combination of set projection and pruning operations [Sondik, 1971; Cassandra *et al.*, 1997; Zhang and Zhang, 2001]. We now describe the most straight-forward version of exact POMDP value iteration.

To implement the exact update $V = HV'$, we first generate intermediate sets $\Gamma^{a,*}$ and $\Gamma^{a,o}$, $\forall a \in A, \forall o \in O$ (Step 1):

$$\Gamma^{a,*} \leftarrow \alpha^{a,*}(s) = R(s, a) \quad (5)$$

$$\Gamma^{a,o} \leftarrow \alpha_i^{a,o}(s) = \gamma \sum_{s' \in S} T(s, a, s') \Omega(o, s', a) \alpha'_i(s'), \forall \alpha'_i \in V'$$

Next we create Γ^a ($\forall a \in A$), the cross-sum over observations, which includes one $\alpha^{a,o}$ from each $\Gamma^{a,o}$ (Step 2):

$$\Gamma^a = \Gamma^{a,*} \oplus \Gamma^{a,o_1} \oplus \Gamma^{a,o_2} \oplus \dots \quad (6)$$

Finally we take the union of Γ^a sets (Step 3):

$$V = \cup_{a \in A} \Gamma^a \quad (7)$$

In practice, many of the vectors in the final set V may be completely dominated by another vector ($\alpha_i \cdot b < \alpha_j \cdot b, \forall b$), or by a combination of other vectors. Those vectors can be pruned away without affecting the solution. Finding dominated vectors can be expensive (checking whether a single vector is dominated requires solving a linear program), but is usually worthwhile to avoid an explosion of the solution size.

To better understand the complexity of the exact update, let $|V'|$ be the number of α -vectors in the previous solution set. Step 1 creates $|A| |O| |V'|$ projections and Step 2 generates $|A| |V'|^{|O|}$ cross-sums. So, in the worst case, the new solution $|V|$ has size $|A| |V'|^{|O|}$ (time $|S|^2 |A| |V'|^{|O|}$). Given that this exponential growth occurs for every iteration, the importance of pruning away unnecessary vectors is clear. It also highlights the impetus for approximate solutions.

3 Point-based value iteration

It is a well understood fact that most POMDP problems, even given arbitrary action and observation sequences of infinite length, are unlikely to reach most of the points in the belief simplex. Thus it seems unnecessary to plan equally for all beliefs, as exact algorithms do, and preferable to concentrate planning on most probable beliefs.

The *point-based value iteration* (PBVI) algorithm solves a POMDP for a finite set of belief points $B = \{b_0, b_1, \dots, b_q\}$. It initializes a separate α -vector for each selected point, and repeatedly updates (via value backups) the value of that α -vector. As shown in Figure 1, by maintaining a full α -vector for each belief point, PBVI preserves the piecewise linearity and convexity of the value function, and defines a value function over the entire belief simplex. This is in contrast to grid-based approaches [Lovejoy, 1991; Brafman, 1997; Hauskrecht, 2000; Zhou and Hansen, 2001; Bonet, 2002], which update only the value at each belief grid point.

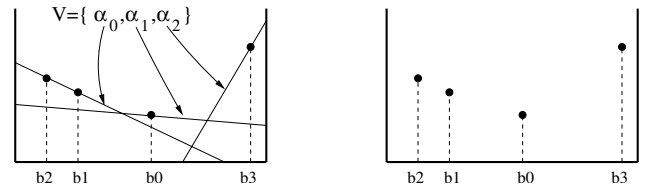


Figure 1: POMDP value function representation using PBVI (on the left) and a grid (on the right).

The complete PBVI algorithm is designed as an *anytime* algorithm, interleaving steps of value iteration and steps of belief set expansion. It starts with an initial set of belief points for which it applies a first series of backup operations. It then grows the set of belief points, and finds a new solution for the expanded set. By interleaving value backup iterations with

expansions of the belief set, PBVI offers a range of solutions, gradually trading off computation time and solution quality. We now describe how we can efficiently perform point-based value backups and how we select belief points.

3.1 Point-based value backup

To plan for a finite set of belief points B , we modify the backup operator (Eqn 4) such that only one α -vector per belief point is maintained. For a point-based update $V = \tilde{H}V'$, we start by creating projections (exactly as in Eqn 5) $\forall a \in A, \forall o \in O$ (Step 1):

$$\begin{aligned}\Gamma^{a,*} &\leftarrow \alpha^{a,*}(s) = R(s, a) \\ \Gamma^{a,o} &\leftarrow \alpha_i^{a,o}(s) = \gamma \sum_{s' \in S} T(s, a, s') \Omega(o, s', a) \alpha'_i(s'), \forall \alpha'_i \in V'\end{aligned}\quad (8)$$

Next, the cross-sum operation (Eqn 6) is much simplified by the fact that we are now operating over a finite set of points. We construct $\forall b \in B, \forall a \in A$ (Step 2):

$$\Gamma_b^a = \Gamma^{a,*} + \sum_{o \in O} \operatorname{argmax}_{\alpha \in \Gamma^{a,o}} (\alpha \cdot b) \quad (9)$$

Finally, we find the best action for each belief point (Step 3):

$$V \leftarrow \operatorname{argmax}_{\Gamma_b^a, \forall a \in A} (\Gamma_b^a \cdot b), \quad \forall b \in B \quad (10)$$

When performing point-based updates, the backup creates $|A| |O| |V'|$ projections as in exact VI. However the final solution V is limited to containing only $|B|$ components (in time $|S| |A| |V'| |O| |B|$). Thus a full point-based value update takes only polynomial time, and even more crucial, the size of the solution set V remains constant. As a result, the pruning of α vectors (and solving of linear programs), so crucial in exact POMDP algorithms, is now unnecessary. The only pruning step is to refrain from adding to V any vector already included, which arises when two nearby belief points support the same vector (e.g. b_1, b_2 in Fig. 1).

In problems with a finite horizon h , we run h value backups before expanding the set of belief points. In infinite-horizon problems, we select the horizon so that $(R_{max} - R_{min})\gamma^h < \epsilon$.

3.2 Belief point set expansion

As explained above, PBVI focuses its planning on relevant beliefs. More specifically, our error bound below suggests that PBVI performs best when its belief set is uniformly dense in the set of reachable beliefs. So, we initialize the set B to contain the initial belief b_0 and expand B by greedily choosing new reachable beliefs that improve the worst-case density as rapidly as possible.

For a given $b \in B$, PBVI stochastically simulates a single-step forward trajectory using each action to produce new beliefs $\{b_{a_0}, b_{a_1}, \dots\}$.¹ It then measures the L_1 distance from b_{a_i} to B , and throws away b_{a_i} if $b_{a_i} \in B$. Finally, it keeps only

¹To simulate an action a , first sample a state s from the distribution b . Then sample an observation according to $\Omega(s, a, o)$ for the given s and a . Finally compute b_a using a Bayesian update (Eqn 2).

the new belief b_{a_i} which is farthest away from any point already in B .² PBVI tries to generate one new belief from each previous belief; so, B at most doubles in size on each expansion.³ Since expansion phases are interleaved with value iteration, PBVI offers an *anytime* solution.

3.3 Convergence and error bounds

For any belief set B and horizon n , PBVI produces an estimate V_n^B . The error between V_n^B and the true value function V_n^* is bounded. The bound depends on how densely B samples the belief simplex Δ ; with denser sampling, V_n^B converges to V_n^* , the true value function.⁴ As $n \rightarrow \infty$, V_n^B does not necessarily converge; but, our error bound still holds. Cutting off the PBVI iteration at *any* sufficiently large horizon, we know that the difference between V_n^B and the optimal infinite-horizon V^* is not too large. (The overall error is bounded by $\|V_n^B - V_n^*\|_\infty + \|V_n^* - V^*\|_\infty$. The first term is bounded by our theorem below; the second is bounded by $\gamma^n \|V_0^* - V^*\|_\infty$.) The remainder of this section states and proves our error bound.

Define the density ϵ_B of a set of belief points B to be the maximum distance from any legal belief to B . More precisely, $\epsilon_B = \max_{b' \in \Delta} \min_{b \in B} \|b - b'\|_1$. Then, we can prove:

Lemma 1 *The error introduced by PBVI's pruning step is at most $\eta_{\text{prune}} = \frac{(R_{\max} - R_{\min})\epsilon_B}{1 - \gamma}$.*

Proof: Let $b' \in \Delta$ be the point where PBVI makes its worst pruning error, and $b \in B$ be the closest (1-norm) sampled belief to b' . Let α be the vector which is maximal at b , and α' be maximal at b' . By erroneously pruning α' , PBVI makes an error of at most $\alpha' \cdot b' - \alpha \cdot b'$. On the other hand, since α is maximal at b , then $\alpha' \cdot b \leq \alpha \cdot b$. So,

$$\begin{aligned}\eta_{\text{prune}} &\leq \alpha' \cdot b' - \alpha \cdot b' \\ &= \alpha' \cdot b' - \alpha \cdot b' + (\alpha' \cdot b - \alpha' \cdot b) && \text{add zero} \\ &\leq \alpha' \cdot b' - \alpha \cdot b' + \alpha \cdot b - \alpha' \cdot b && \alpha \text{ opt. at } b \\ &= (\alpha' - \alpha) \cdot (b' - b) && \text{collect terms} \\ &\leq \|\alpha' - \alpha\|_\infty \|b' - b\|_1 && \text{Hölder} \\ &\leq \|\alpha' - \alpha\|_\infty \epsilon_B && \text{def'n of } \epsilon_B \\ &\leq \frac{R_{\max} - R_{\min}}{1 - \gamma} \epsilon_B && \text{see text}\end{aligned}$$

The last inequality holds because each α -vector represents the reward achievable starting from some state and following some sequence of actions and observations. ■

Theorem 1 *For any belief set B and any horizon n , the error of the PBVI algorithm $\eta_n = \|V_n^B - V_n^*\|_\infty$ is bounded by*

$$\eta_n \leq \frac{(R_{\max} - R_{\min})\epsilon_B}{(1 - \gamma)^2}$$

²The actual choice of norm doesn't appear to matter in practice; some of our experiments below used Euclidean distance (instead of L_1) and the results appear identical.

³We experimented with other strategies such as adding a fixed number of new beliefs, but since value iteration is much more expensive than belief computation the above algorithm worked best. If desired, we can impose a maximum size on B based on time constraints or performance requirements.

⁴If not all beliefs are reachable, we don't need to sample all of Δ densely, but replace Δ by the set of reachable beliefs $\bar{\Delta}$ below. The error bounds and convergence results hold on $\bar{\Delta}$.

Proof:

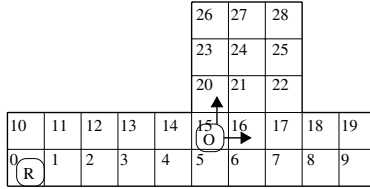
$$\begin{aligned}
\eta_n &= \|V_n^B - V_n^*\|_\infty && \text{def'n of } \eta_n \\
&= \|\tilde{H}V_{n-1}^B - HV_{n-1}^*\|_\infty && \text{def'n of } H, \tilde{H} \\
&\leq \|\tilde{H}V_{n-1}^B - HV_{n-1}^B\|_\infty + && \\
&\quad \|HV_{n-1}^B - HV_{n-1}^*\|_\infty && \text{triangle ineq.} \\
&\leq \eta_{\text{prune}} + \|\tilde{H}V_{n-1}^B - HV_{n-1}^B\|_\infty && \text{def'n of } \eta_{\text{prune}} \\
&\leq \eta_{\text{prune}} + \gamma \|V_{n-1}^B - V_{n-1}^*\|_\infty && \text{contraction} \\
&= \eta_{\text{prune}} + \gamma \eta_{n-1} && \text{def'n of } \eta_{n-1} \\
&\leq \frac{(R_{\max} - R_{\min})\epsilon_B}{1-\gamma} + \gamma \eta_{n-1} && \text{lemma 1} \\
&\leq \frac{(R_{\max} - R_{\min})\epsilon_B}{(1-\gamma)^2} && \text{series sum} \quad \blacksquare
\end{aligned}$$

4 Experimental results

The domain of *Tag* is based on the popular game of lasertag. The goal is to search for and tag a moving opponent [Rosenkrantz *et al.*, 2003]. Figure 2a shows the live robot as it moves in to capture an opponent. In our POMDP formulation, the opponent moves stochastically according to a fixed policy. The spatial configuration of the domain used for planning is illustrated in Figure 2b. This domain is an order of magnitude larger (870 states) than most other POMDP problems considered thus far in the literature [Cassandra, 1999], and is proposed as a new challenge for fast, scalable, POMDP algorithms. A single iteration of optimal value iteration on a problem of this size could produce over 10^{20} α -vectors before pruning.



a. Robots playing Tag



b. Tag configuration

Figure 2: Tag domain (870 states, 5 actions, 30 observations)

The state space is described by the cross-product of two features, $Robot = \{s_0, \dots, s_{29}\}$ and $Opponent = \{s_0, \dots, s_{29}, s_{tagged}\}$. Both agents start in independently-selected random positions, and the game finishes when $Opponent = s_{tagged}$. The robot can select from five actions: $\{North, South, East, West, Tag\}$. A reward of -1 is imposed for each motion action; the *Tag* action results in a $+10$ reward if $Robot = Opponent$, or -10 otherwise. Throughout the game, the Robot's position is fully observable, and the effect of a *Move* action has the predictable deterministic effect, e.g.:

$$Pr(Robot = s_{10} \mid Robot = s_0, North) = 1$$

The position of the opponent is completely unobservable unless both agents are in the same cell. At each step, the opponent (with omniscient knowledge) moves away from the robot with $Pr = 0.8$ and stays in place with $Pr = 0.2$, e.g.:

$$Pr(Opponent = s_{16} \mid Opponent = s_{15} \& Robot = s_0) = 0.4$$

$$Pr(Opponent = s_{20} \mid Opponent = s_{15} \& Robot = s_0) = 0.4$$

$$Pr(Opponent = s_{15} \mid Opponent = s_{15} \& Robot = s_0) = 0.2$$

Figure 3 shows the performance of PBVI on the Tag domain. Results are averaged over 10 runs of the algorithm, times 100 different (randomly chosen) start positions for each run. It shows the gradual improvement in performance as samples are added (each shown data point represents a new expansion of the belief set with value backups). In addition to PBVI, we also apply the QMDP approximation as a baseline comparison [Littman *et al.*, 1995]. The QMDP approximation is calculated by solving a POMDP as though it were fully observable: $Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V(s')$, and linearizing across Q -values to obtain the value at a belief: $V(b) = \max_{a \in A} \sum_{s \in S} b(s) Q(s, a)$. This approximation is quick to compute, and is remarkably effective in some domains. In the Tag domain, however, it lacks the representational power to compute a good policy.

5 Additional experiments

5.1 Comparison on well-known problems

To further analyze the performance of PBVI, we applied it to three well-known problems from the POMDP literature. We selected Maze33, Hallway and Hallway2 because they are commonly used to test scalable POMDP algorithms [Littman *et al.*, 1995; Brafman, 1997; Poon, 2001]. Figure 3 presents results for each domain. Replicating earlier experiments, results for Maze33 are averaged over 151 runs (reset after goal, terminate after 500 steps); results for Hallway and Hallway2 are averaged over 251 runs (terminate at goal, max 251 steps). In all cases, PBVI is able to find a good policy. Table 1 compares PBVI's performance with previously published results, comparing goal completion rates, sum of rewards, policy computation time, and number of required belief points. In all domains, PBVI achieves competitive performance, but with fewer samples.

5.2 Validation of the belief set expansion

To further investigate the validity of our approach for generating new belief states (Section 3.2), we compared our approach with three other techniques which might appear promising. In all cases, we assume that the initial belief b_0 (given as part of the model) is the sole point in the initial set, and consider four expansion methods:

1. Random (**RA**)
2. Stochastic Simulation with Random Action (**SSRA**)
3. Stochastic Simulation with Greedy Action (**SSGA**)
4. Stochastic Simulation with Explorative Action (**SSEA**)

The **RA** method consists of sampling a belief point from a uniform distribution over the entire belief simplex. **SSEA** is the standard PBVI expansion heuristic (Section 3.2). **SSRA** similarly uses single-step forward simulation, but rather than try all actions, it randomly selects one and automatically accepts the posterior belief unless it was already in B . Finally, **SSGA** uses the most recent value function solution to pick the greedy action at the given belief b , and performs a single-step simulation to get a new belief $b' \rightarrow B$.

We revisited the Hallway, Hallway2, and Tag problems from sections 4 and 5.1 to compare the performance of these

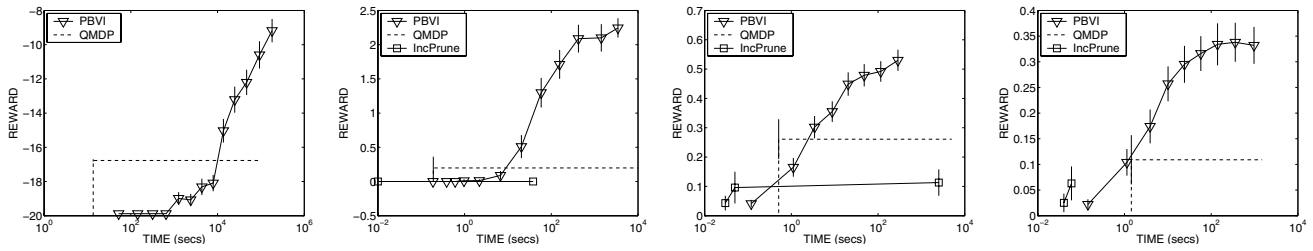


Figure 3: PBVI performance for four problems: Tag(left), Maze33(center-left), Hallway(center-right) and Hallway2(right)

Method	Goal%	Reward	Time(s)	$ B $
Maze33 / Tiger-Grid				
QMDP[*]	n.a.	0.198	0.19	n.a.
Grid [Brafman, 1997]	n.a.	0.94	n.v.	174
PBUA [Poon, 2001]	n.a.	2.30	12116	660
PBVI[*]	n.a.	2.25	3448	470
Hallway				
QMDP[*]	47	0.261	0.51	n.a.
QMDP [Littman <i>et al.</i> , 1995]	47.4	n.v.	n.v.	n.a.
PBUA [Poon, 2001]	100	0.53	450	300
PBVI[*]	96	0.53	288	86
Hallway2				
QMDP[*]	22	0.109	1.44	n.a.
QMDP [Littman <i>et al.</i> , 1995]	25.9	n.v.	n.v.	n.a.
Grid [Brafman, 1997]	98	n.v.	n.v.	337
PBUA [Poon, 2001]	100	0.35	27898	1840
PBVI[*]	98	0.34	360	95
Tag				
QMDP[*]	17	-16.769	13.55	n.a.
PBVI[*]	59	-9.180	180880	1334
n.a.=not applicable n.v.=not available				

Table 1: Results for POMDP domains. Those marked [*] were computed by us; other results were likely computed on different platforms, and therefore time comparisons may be approximate at best. All results assume a standard (not *lookahead*) controller.

four heuristics. For each problem we apply PBVI using each of the belief-point selection heuristics, and include the QMDP approximation as a baseline comparison. Figure 4 shows the computation time versus the reward performance for each domain.

The key result from Figure 4 is the rightmost panel, which shows performance on the largest, most complicated domain. In this domain our SSEA rule clearly performs best. In smaller domains (left two panels) the choice of heuristic matters less: all heuristics except random exploration (RA) perform equivalently well.

6 Related work

Significant work has been done in recent years to improve the tractability of POMDP solutions. A number of increasingly efficient exact value iteration algorithms have been proposed [Cassandra *et al.*, 1997; Kaelbling *et al.*, 1998; Zhang and Zhang, 2001]. They are successful in finding optimal solutions, however are generally limited to very small problems (a dozen states) since they plan optimally for all beliefs. PBVI avoids the exponential growth in plan size by

restricting value updates to a finite set of (reachable) beliefs.

There are several approximate value iteration algorithms which are related to PBVI. For example, there are many grid-based methods which iteratively update the values of discrete belief points. These methods differ in how they partition the belief space into a grid [Brafman, 1997; Zhou and Hansen, 2001].

More similar to PBVI are those approaches which update both the value and gradient at each grid point [Lovejoy, 1991; Hauskrecht, 2000; Poon, 2001]. While the actual point-based update is essentially the same between all of these, the overall algorithms differ in a few important aspects. Whereas Poon only accepts updates that increase the value at a grid point (requiring special initialization of the value function), and Hauskrecht always keeps earlier α -vectors (causing the set to grow too quickly), PBVI requires no such assumptions. A more important benefit of PBVI is the theoretical guarantees it provides: our guarantees are more widely applicable and provide stronger error bounds than those for other point-based updates.

In addition, PBVI is significantly smarter than previous algorithms about how it selects belief points. PBVI selects only reachable beliefs; other algorithms use random beliefs, or (like Poon’s and Lovejoy’s) require the inclusion of a large number of fixed beliefs such as the corners of the probability simplex. Moreover, PBVI selects belief points which improve its error bounds as quickly as possible. In practice, our experiments on the large domain of lasertag demonstrate that PBVI’s belief-selection rule handily outperforms several alternate methods. (Both Hauskrecht and Poon did consider using stochastic simulation to generate new points, but neither found simulation to be superior to random point placements. We attribute this result to the smaller size of their test domains. We believe that as more POMDP research moves to larger planning domains, newer and smarter belief selection rules will become more and more important.)

Gradient-based policy search methods have also been used to optimize POMDP solutions [Baxter and Bartlett, 2000; Kearns *et al.*, 1999; Ng and Jordan, 2000], successfully solving multi-dimensional, continuous-state problems. In our view, one of the strengths of these methods lies in the fact that they restrict optimization to reachable beliefs (as does PBVI). Unfortunately, policy search techniques can be hampered by low-gradient plateaus and poor local minima, and typically require the selection of a restricted policy class.

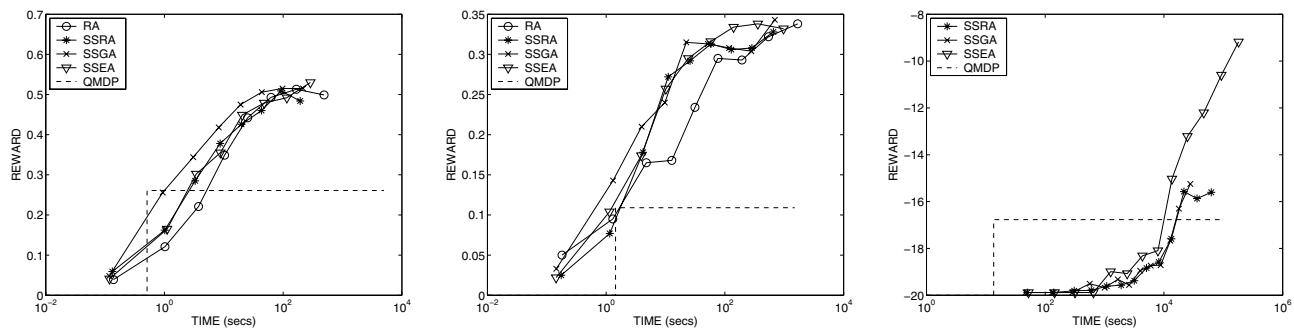


Figure 4: Belief expansion results for three problems: Hallway(left), Hallway2(center) and Tag(right)

7 Conclusion

This paper presents PBVI, a scalable anytime algorithm for approximately solving POMDPs. We applied PBVI to a robotic version of lasertag, where it successfully developed a policy for capturing a moving opponent. Other POMDP solvers had trouble computing useful policies for this domain. PBVI also compared favorably with other solvers on three well-known smaller test problems. We attribute PBVI's success to two features, both of which directly target the curse of history. First, by using a trajectory-based approach to select belief points, PBVI focuses planning on reachable beliefs. Second, because it uses a fixed set of belief points, it can perform fast value backups.

In experiments, PBVI beats back the curse of history far enough that we can solve POMDPs an order of magnitude larger than most previous algorithms. With this success, we can now identify the next hurdle for POMDP research: contrary to our expectation, it turns out to be the old-fashioned MDP problem of having too many distinct physical states. This problem hits us in the cost of updating the point-based value function vectors. (This cost is quadratic in the number of physical states.) While this problem is not necessarily easy to overcome, we believe that sparse matrix computations, together with other approaches from the existing literature [Poupart and Boutilier, 2003; Roy and Gordon, 2003], will allow us to scale PBVI to problems which are at least another order of magnitude larger. So, PBVI represents a considerable step towards making POMDPs usable for real-world problems.

References

- [Baxter and Bartlett, 2000] J. Baxter and P. L. Bartlett. Reinforcement learning on POMDPs via direct gradient ascent. In *ICML*, 2000.
- [Bonet, 2002] B. Bonet. An e-optimal grid-based algorithm for partially observable Markov decision processes. In *ICML*, 2002.
- [Brafman, 1997] R. I. Brafman. A heuristic variable grid solution method for POMDPs. In *AAAI*, 1997.
- [Cassandra et al., 1997] A. Cassandra, M. Littman, and N. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *UAI*, 1997.
- [Cassandra, 1999] A. Cassandra. Tony's POMDP page. <http://www.cs.brown.edu/research/ai/pomdp/code/index.html>, 1999.
- [Hauskrecht, 2000] M. Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.
- [Kaelbling et al., 1998] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.
- [Kearns et al., 1999] M. Kearns, Y. Mansour, and A. Y. Ng. Approximate planning in large POMDPs via reusable trajectories. *NIPS* 12, 1999.
- [Littman et al., 1995] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML*, 1995.
- [Lovejoy, 1991] W. S. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39(1):162–175, 1991.
- [Ng and Jordan, 2000] A. Y. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *UAI*, 2000.
- [Poon, 2001] K.-M. Poon. A fast heuristic algorithm for decision-theoretic planning. Master's thesis, The Hong-Kong University of Science and Technology, 2001.
- [Poupart and Boutilier, 2003] P. Poupart and C. Boutilier. Value-directed compression of POMDPs. In *NIPS* 15, 2003.
- [Rosencrantz et al., 2003] M. Rosencrantz, G. Gordon, and S. Thrun. Locating moving entities in dynamic indoor environments with teams of mobile robots. In *AAMAS*, 2003.
- [Roy and Gordon, 2003] N. Roy and G. Gordon. Exponential family PCA for belief compression in POMDPs. In *NIPS* 15, 2003.
- [Sondik, 1971] E. J. Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, 1971.
- [Thrun, 1992] S. Thrun. *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, chapter The Role of Exploration in Learning Control. Van Nostrand Reinhold, 1992.
- [Zhang and Zhang, 2001] N. L. Zhang and W. Zhang. Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 14:29–51, 2001.
- [Zhou and Hansen, 2001] R. Zhou and E. A. Hansen. An improved grid-based approximation algorithm for POMDPs. In *IJCAI*, 2001.

QUALITATIVE REASONING

A New Look at the Semantics and Optimization Methods of CP-Networks

Ronen I. Brafman

Dept. of Computer Science
Ben-Gurion University
Beer-Sheva 84105, Israel
brafman@cs.bgu.ac.il

Yannis Dimopoulos

Dept. of Computer Science
University of Cyprus
Nicosia 1678, Cyprus
yannis@cs.ucy.ac.cy

Abstract

Preference elicitation is a serious bottleneck in many decision support applications and agent specification tasks. CP-nets were designed to make the preference elicitation process simpler and more intuitive for lay users by graphically structuring a set of *Ceteris Paribus* (CP) preference statements – preference statements most people find natural and intuitive. In various contexts, CP-nets with an underlying cyclic structure emerge naturally. Often, they are inconsistent according to the current semantics, and the user is required to revise them. In this paper we show how optimization queries can be meaningfully answered in many “inconsistent” networks without troubling the user with requests for revisions. We also describe a method for focusing users’ revision process when revisions are *truly* needed. In the process, we provide a formal semantics that justifies our approach and we introduce new techniques for computing optimal outcomes.

1 Introduction

Ceteris Paribus (CP) preference statements are among the most natural and most intuitive preference statements that people make. Thus, it is not surprising that they have drawn the attention of many researchers in philosophy and AI (e.g., [Doyle and Wellman, 1994; Hanson, 1996]). CP statements indicate a preference for one value over another in the context of a fixed background. For example, the statement “I prefer an apple pie to a chocolate cake as a dessert, *ceteris paribus*” expresses the fact that given two identical contexts – i.e., meals – that differ only in their dessert, the one containing an apple pie is preferred to the one containing a chocolate cake. Finer distinctions can be made using conditional CP statements. For example: “I prefer red wine over white wine if the main course is beef.” In this case, the preference for red wine to white wine is restricted to comparisons between identical meals in which the main course is beef.

CP-nets [Boutilier *et al.*, 1999] are a graphical tool for representing and for structuring a set of CP statements. A CP-network consists of a graph describing the preferential dependency relationship between different domain variables. Each node is annotated by a conditional preference table (CPT) that describes how the user’s preference over the different values

of the variable associated with this node depends on the variables associated with the parents of this node.

Cyclic CP-nets emerge naturally when there is a set of interdependent variables, none of which is more important than the other. For example, [Domshlak *et al.*, 2001] note that such dependency can emerge naturally among web-page components in their web-personalization tool. Cyclic CP-networks raise some conceptual and computational problems to which we still do not have satisfactory answers. Even worse, according to the standard semantics of CP-nets, most cyclic CP-nets are inconsistent. For example, in [Domshlak and Brafman, 2002], it was shown that the preference ordering induced by any simple cycle (i.e., a cycle that does not contain smaller cycles) with more than two nodes is inconsistent. That is, there will be at least two outcomes, o_1 and o_2 , such that one can show that o_1 is strictly preferred to o_2 and o_2 is strictly preferred to o_1 . In addition, whereas for acyclic networks one can determine the most preferred outcome in linear time even in the face of evidential constraints (i.e., constraints that fix the value of certain variables), no corresponding algorithm is known for networks that contain cycles.

The fact that many cyclic networks are inconsistent raises a serious practical concern. When a user specifies an inconsistent cyclic network, we must ask him to revise his network. To do so, we need to provide information that will help him obtain a consistent network. In this paper we make two practical contributions aimed at improving this process. First, we show that various optimization queries can be answered naturally even when the network is “inconsistent” according to the standard semantics. In such cases, no additional burden is placed on the user. Second, when revision is required, we show how the notion of a partial model, studied in the area of logic programming, can be used to identify those aspects of a model that require revision.

While pursuing our more practical goals we make a number of technical and semantic contributions. First, we provide a more flexible semantics for CP-nets that is identical with the current semantics on those networks the latter considers consistent. This semantics justifies our approach for generating optimal outcomes given networks that are inconsistent under the standard semantics. Second, we show how to answer optimization queries using various approaches that include reduction to SAT, and reduction to logic programs. This extends current methods which are restricted to acyclic nets, and helps us define the notion of a partial outcome.

The paper is structured as follows: In Section 2 we provide the necessary background on *ceteris paribus* preference statements and CP-nets. In Sections 3 and 4 we take a closer look at the notion of consistency in CP-nets and the problem that cyclic networks introduce, we suggest a natural definition of a preferred model as one that cannot be improved, and explain how the problem of the existence of such models can be reduced into a CSP problem. In Section 5 we discuss an alternative translation into nonmonotonic logic programs and some of its implications, and in Section 6 we conclude. The full version of this paper is available from the authors' web site. It contains proofs, an optimization method based on cut-set conditioning, a discussion of constrained optimization and other inference tasks, and a comparison with logic programs with ordered disjunction [Brewka *et al.*, 2002].¹

2 CP-Nets

We start with a review of *ceteris paribus* preference statements and preferential independence, followed by the definition of CP-nets.

2.1 Ceteris Paribus Preference Statements

A *preference relation* is a total pre-order (a *ranking*) over a set of outcomes. Given two outcomes o, o' , we write $o \succeq o'$ if o is at least as preferred as o' and we write $o \succ o'$ if o is strictly more preferred than o' .

The types of outcomes we are concerned with consist of possible assignments to some set of variables. More formally, we assume some given set $\mathbf{V} = \{X_1, \dots, X_n\}$ of variables with corresponding domains $\mathcal{D}(X_1), \dots, \mathcal{D}(X_n)$. The set of possible outcomes is then $\mathcal{D}(X_1) \times \dots \times \mathcal{D}(X_n)$. For example, in the context of the problem of configuring a personal computer (PC), the variables may be *processor type*, *screen size*, *operating system* etc., where *screen size* has the domain $\{17\text{in}, 19\text{in}, 21\text{in}\}$, *operating system* has the domain $\{\text{LINUX}, \text{Windows98}, \text{WindowsXP}\}$, etc. Each assignment to the set of variables specifies an outcome – a particular PC configuration. Thus, a preference ordering over these outcomes specifies a ranking over possible PC configurations.

The number of possible outcomes is exponential in n , while the set of possible total orders on them is doubly exponential in n . Therefore, explicit specification and representation of a ranking are not realistic. We must find implicit means of describing this preference relation. The notion of conditional preferential independence plays a key role in such representations. Intuitively, $\mathbf{X}$ and $\mathbf{Y}$ are *conditionally preferentially independent* given $\mathbf{Z}$ (where $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ are disjoint subsets of $\mathbf{V}$) if for every fixed assignment to $\mathbf{Z}$, the ranking of $\mathbf{X}$ values is independent of the value of $\mathbf{Y}$. Formally, let $\mathbf{X}, \mathbf{Y}$ and $\mathbf{Z}$ be a partition of $\mathbf{V}$ and let $\mathbf{z} \in \mathcal{D}(\mathbf{Z})$. $\mathbf{X}$ and $\mathbf{Y}$ are *conditionally preferentially independent* given $\mathbf{z}$ iff, for all $\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}_1, \mathbf{y}_2$ we have that

$$\mathbf{x}_1 \mathbf{y}_1 \mathbf{z} \succeq \mathbf{x}_2 \mathbf{y}_1 \mathbf{z} \text{ iff } \mathbf{x}_1 \mathbf{y}_2 \mathbf{z} \succeq \mathbf{x}_2 \mathbf{y}_2 \mathbf{z}$$

$\mathbf{X}$ and $\mathbf{Y}$ are conditionally preferentially independent given $\mathbf{Z}$ if they are conditionally preferentially independent given

¹A note on notation. Due to the difficulty of using the logical negation operator, $\neg$, in drawing packages, we use two alternative notations: $\bar{a}$ instead of $\neg a$, and $a1, a2$ instead of $a, \neg a$.

any assignment $\mathbf{z} \in \mathcal{D}(\mathbf{Z})$. In our PC example, the user may assess *operating system* to be independent of other features given *processor type*. That is, it always prefers LINUX given an AMD processor and Windows98 given an Intel processor (e.g., because he believes Windows98 is optimized for the Intel processor, whereas LINUX is otherwise better). Conditional preferential independence is a standard notion in multi-attribute utility theory [Keeney and Raiffa, 1976].

2.2 CP-nets

CP-nets were introduced in [Boutilier *et al.*, 1999] as a tool for compactly and intuitively representing qualitative preference relations. This graphical model exploits conditional preferential independence in structuring a decision maker's preferences under *ceteris paribus* (all else being equal) semantics. CP-net is the first model based on the notions of purely qualitative preferential independence, and bears a surface similarity to Bayesian nets [Pearl, 1988]. However, the nature of the relationship between nodes within a CP-net is generally quite weak compared with the probabilistic relations in Bayes nets.

During preference elicitation, for each variable X in the variable set $\mathbf{V}$, the decision maker is asked to specify a set of *parent* variables $Pa(X)$ that can affect her preferences over the values of X . That is, given a particular value assignment to $Pa(X)$, the decision maker should be able to determine a preference order over the domain of X (denoted as $\mathcal{D}(X)$), all other things being equal.

The above information is used to create the graph of the CP-net in which each node X has $Pa(X)$ as its immediate predecessors. Given this structural information, the decision maker is asked to explicitly specify her preferences over the values of X for each instantiation of $Pa(X)$. This conditional preference over the values of X is captured by a *conditional preference table* (CPT) which is annotated with the node X in the CP-net. That is, for each assignment to $Pa(X)$, $CPT(X)$ specifies a total order (denoted $\succ$) over $\mathcal{D}(X)$, such that for any two values $x_i, x_j \in \mathcal{D}(X)$, either $x_i \succ x_j$, or $x_j \succ x_i$.

Formally, a *CP-net* $\mathcal{N}$ over variables $\mathbf{V} = \{X_1, \dots, X_n\}$ is a directed graph G over $X_1, \dots, X_n$ and a set of conditional preference tables $CPT(X_i)$ for each $X_i \in \mathbf{V}$. Each conditional preference table $CPT(X_i)$ associates a total order $\succ_j(\mathbf{u}_j)$ over the domain of X_i with each instantiation $\mathbf{u}_j$ of X_i 's parents $Pa(X_i) = \mathbf{U}$.

The semantics of CP-nets is defined as follows. A *model* for a CP-net is a total-order over the set of possible outcomes that satisfies every CPT within the CP-net. A total order satisfies $CPT(X)$ if for every pair of X -values x_i, x_j and every assignment c to $Pa(X)$ such that $x_i \succ x_j$ given c according to $CPT(X)$, the following holds: Let o, o' be two outcomes that differ only in their assignment to X and satisfy c , and suppose that o assigns x_i to X and o' assigns x_j to X , then, $o \succ o'$ holds in the model. We say that $o \succ o'$ is *valid* according to the CP-network if $o \succ o'$ holds in all models of the CP-net. A CP-net is said to be *consistent* if it has a model.

As an example, consider the network in Figure 1(a), in which all variables are boolean. This network has three variables, A, B , and C . The preferences over A and B are unconditional, whereas the preferences for C 's values depend on the values of A and B . In Figure 1(b), we see the preference order induced by this CP-network. For every pair of

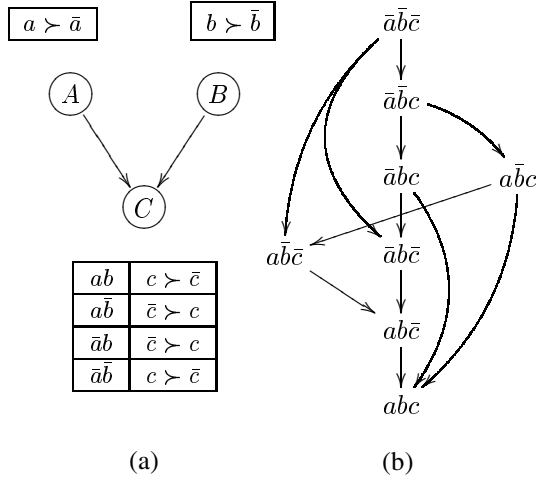


Figure 1: A CP-net and its corresponding preferential order.

outcomes o, o' , we have that $o \succ o'$ is valid iff there is a path from o' to o in this graph.

3 The Weak Preference Semantics

According to the standard semantics, the preferences expressed in the CPT of a CP-net are *strict* preferences. We suggest viewing them as *weak* preferences using pre-orders instead of strict orders. As we show, this leads to a semantics that is only slightly weaker than the standard semantics. In fact, it leads to identical orderings on CP-nets that are consistent according to the standard semantics. More formally, a *model* for a CP-network is a total *pre-order* over the set of outcomes that satisfies every CPT within the CP-net. A total pre-order satisfies $\text{CPT}(X)$ if for every pair of X -values x_i, x_j and every assignment c to $\text{Pa}(X)$ such that $x_i \succeq x_j$ given c according to $\text{CPT}(X)$ ² the following holds: Let o, o' be two outcomes that satisfy c and differ only in their assignment to X , such that o assigns x_i to X and o' assigns x_j to X . Then, $o \succeq o'$ holds in the ordering. We say that $o \succeq o'$ is *valid* according to the CP-network if $o \succeq o'$ holds in all models of the CP-net. Finally, we say that $o \succ o'$ is *valid* according to the CP-network if $o \succeq o'$ is valid but $o' \succeq o$ is not valid. Thus, if $o \succeq o'$ holds in all models and there is a model in which $o \succ o'$ holds, we say that $o \succ o'$ is valid. Had we required $o \succ o'$ to hold in all models for $o \succ o'$ to be valid, no strict preference would be valid. This is because the preference relation in which all outcomes are equally desirable is a model for every CP-network according to the above semantics. Thus, *all* CP-nets are “consistent” according to our semantics.

The above semantics can be related to a more syntactic notion of a proof of preference, or a flipping sequence [Boutilier *et al.*, 1999]. Let o be an outcome, and suppose that o assigns to X some value x_j that the user ranks lower than x_i in parental context c satisfied in o . Then, we can improve o by flipping the value of X from x_j to x_i . An *improving flipping sequence* $o_1, o_2, \dots, o_k$ is a sequence of outcomes such that

²In the standard semantics we have $x_i \succ x_j$ instead of $x_i \succeq x_j$.

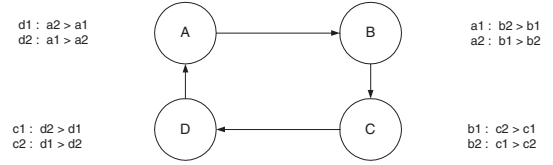


Figure 2: A Simple 4-Cycle CP-network.

o_i is obtained from o_{i-1} via a single improving flip. Naturally, these definitions are with respect to some CP-network, which we take to be fixed by the appropriate context.

Theorem 1 $o \succeq o'$ is valid iff there is an improving flipping sequence from o' to o .

An immediate consequence is the following:

Consequence 1 $o \succ o'$ is valid iff there is an improving flipping sequence from o' to o but no improving flipping sequence from o to o' .

Another interesting theorem relates the standard, stronger semantics, and our semantics.

Theorem 2 Let G be a CP-network that is consistent according to the standard semantics. Then, G satisfies $o \succ o'$ according to our semantics iff G satisfies $o \succ o'$ according to the standard semantics.

This theorem shows that our semantics is a direct extension of the standard semantics. To understand it better, we will soon take a look at networks that are inconsistent according to the standard semantics.

Finally, we define the notion of optimality in the context of a CP-network. There are two possible definitions: An outcome o is said to be *strongly optimal* iff there is no other outcome o' such that $o' \succeq o$ holds. An outcome o is said to be *weakly optimal* iff there is no other outcome o' such that $o' \succ o$ holds. Thus, in the first case, o is either strictly better than any other outcome or incomparable. In the second case, there may be other outcomes that are as preferred as o , but no outcome that is strictly preferred over o . In networks that are consistent according to the standard semantics, there is a unique best outcome that is strictly better than any other outcome. However, when we move beyond this class of networks, we can have more than one optimal outcome, and it can be either strongly or weakly optimal. The latter class is computationally more challenging to identify, and so we concentrate on strongly optimal outcomes.

4 Optimality in Cyclic Networks

The semantics of CP-nets allows for cycles. But only the consistency of acyclic CP-nets is guaranteed according to the standard semantics [Boutilier *et al.*, 1999]. Moreover, such networks have a unique optimal outcome. This remains true even if we introduce evidential constraints, i.e., constraints that fix the value of some variables.

Cyclic networks often induce an inconsistent (i.e., cyclic) preference order according to the standard semantics. For example, consider the network in Figure 2 which contains a cycle of size 4, called a simple cycle in [Domshlak and Brafman, 2002]. The web-personalization applications described in [Domshlak *et al.*, 2001] naturally gives rise to such cyclic structures. There, variables correspond to articles, ads, and other content element in an online newspaper. The value of each variable indicates whether it is currently displayed or not. The CP-net captures the preferences of the editor regarding the presentation of different elements on the user's current screen. These preferences together with user-generated content constraints lead to a personalized view that takes into account the user's interests and the editor's expertise on preferred combinations of news items. For example, in the CP-net in Figure 2, *A* could be a review of a new Toyota 4x4 vehicle, *B* a test-drive of a new BMW series 7 car, *C* a story about a recent Man. United match, and *D* a story about Man. City's recent success. Suppose that Man. United is sponsored by Toyota, and Man. City by BMW, and the editor would prefer not to display stories about competing teams and/or companies in the same screen. This is expressed in the CP-net in Figure 2 by stipulating that if *A* is present then *B* should not, if *A* is not present then *B* should be present, etc.

In [Domshlak and Brafman, 2002] it is shown that such a network is not consistent, i.e., there is no total order of the set of outcomes that satisfies all the preferences embodied in this network. It is easy to see why this is so when we examine Figure 3 which describes the relationships among different possible outcomes. The nodes in that figure correspond to outcomes and the edges correspond to legal improving (single) flips. For example, consider the outcome $a2b2c1d2$ in the lower left-hand side of Figure 3. Given that *A* is assigned $a2$, we can see from the *B*'s CPT that $b1$ is a more preferred value for *B*. Thus, there is an edge from $a2b2c1d2$ to $a2b1c1d2$.

We can see that Figure 3 contains cycles, making it impossible to totally order its elements consistently. However, it is also apparent that it contains two elements, $a1b2c1d2$ and $a2b1c2d1$ that, in some sense, can be viewed as optimal elements, as well as two elements, $a1b1c1d1$ and $a2b2c2d2$, that in some sense can be viewed as the worst elements.

Indeed, according to our semantics, this network induces three classes of outcomes described in Figure 4. The outcomes within each class are either equally preferred or incomparable, and there is a strict preference between outcomes belonging to different classes. An important consequence of this is that there are clear candidates for the set of optimal outcomes: the two outcomes in the top class.

We can see that our semantics is more lenient, allowing for specifications that, in some sense, are cyclic. If the cycle contains all outcomes, then all outcomes are equally preferred. This specification is not informative, and the user needs to be informed of this fact. However, the example above shows that the cycle may contain some, but not all of the outcomes. Whereas the standard semantics will dismiss this CP-network as inconsistent, our semantics is more tolerant, and can use this CP-net to determine an optimal outcome.

We now turn to the computation of an optimal outcome via a simple reduction to a CSP problem (or a SAT problem when the variables are binary). The variables in our reduction

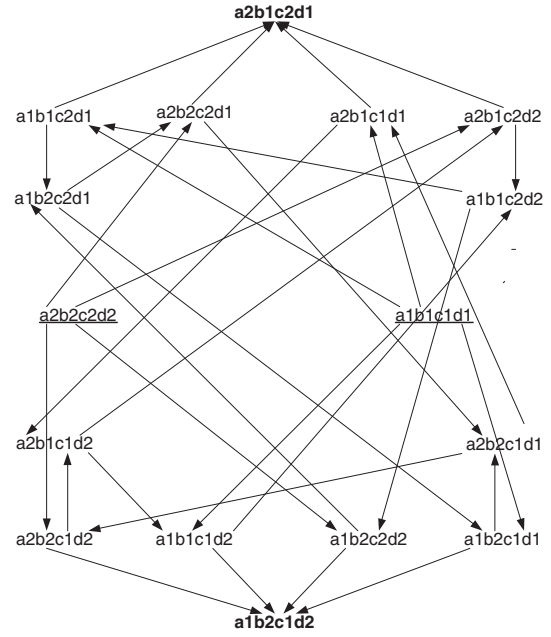


Figure 3: Outcome Space for the 4-Cycle Network.

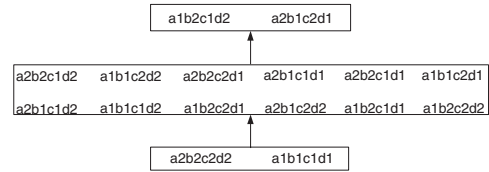


Figure 4: Ordered Outcome Classes for 4-Cycle Network

consist of the variables in the CP-network. For every entry in the CP Table of every variable v , we add the constraint $\phi \rightarrow v_1$, where ϕ denotes the context (i.e., the assignment to the parents of v) and v_1 is the preferred value of v .

As an example, for the CP-network in Figure 1 we obtain the following propositional formula.

$$a \wedge b \wedge (a \wedge b \rightarrow c) \wedge (a \wedge \bar{b} \rightarrow \bar{c}) \wedge (\bar{a} \wedge a \rightarrow \bar{c}) \wedge (\bar{a} \wedge \bar{b} \rightarrow c).$$

In the case of Figure 2, we get the formula

$$(d1 \rightarrow a2) \wedge (d2 \rightarrow a1) \wedge (a1 \rightarrow b2) \wedge (a2 \rightarrow b1) \wedge \\ (b1 \rightarrow c2) \wedge (b2 \rightarrow c1) \wedge (c1 \rightarrow d2) \wedge (c2 \rightarrow d2).$$

Lemma 1 *o is a strongly optimal outcome for a CP-net iff it satisfies the above CSP.*

The above algorithm does not work when we have weakly optimal outcomes, i.e., when we have a set of outcomes that are equally preferred, but cannot be strictly improved. Figures 5 and 6 show an example of a network giving rise to such a case and the relation over the outcome space it induces. From now on, our treatment centers on *strongly optimal* outcomes only, to which we simply refer as *optimal outcomes*.

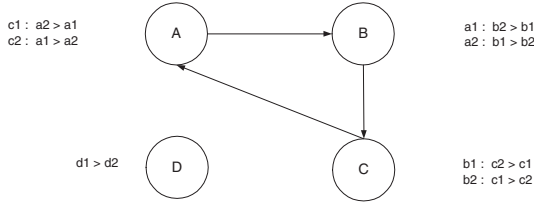


Figure 5: A Network Without a Strongly Optimal Outcome

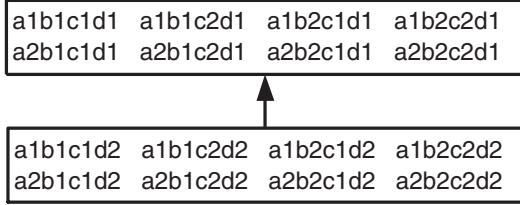


Figure 6: Outcome Classes for Network in Figure 5

Lemma 1 implies that the problem of finding a single optimal outcome is in NP. When the CP-network is acyclic, an optimal outcome can be found in linear time [Boutilier *et al.*, 1999]. Our result indicates a linear time solution for simple-cycle nets, too, if the variables are binary, because the resulting CSP is an instance of 2-SAT [Aspvall *et al.*, 1979].

5 Logic Programs and Partial Consistency

We start with some background on the semantics of logic programs, followed by a reduction of CP-net optimization to logic programs. Then, using the notion of a partial stable model of a logic program, we can define a corresponding notion for a CP-net that captures those variables for which we have some reasonable candidate for an optimal value.

5.1 The Semantics of Logic Programs

A propositional *normal logic program* P is a set of *normal rules*. A normal rule is a rule of the form $h \leftarrow a_1, a_2, \dots, a_n, \text{not } b_1, \text{not } b_2, \dots, \text{not } b_m$ where $h, a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m$ are propositional atoms. The set of atoms of P is denoted by $Atoms(P)$. Let S be a truth or value assignment on the atoms of $Atoms(P)$. We denote by S^+ the set of atoms of S that are assigned the value true, and S^- the set of atoms that are assigned the value false. The *reduct* P^S of a logic program P w.r.t. an assignment S is the logic program obtained from P after deleting (i) every rule of P that has a negated atom $\text{not } b_i$, with $b_i \in S^+$ (ii) every negated atom from the body of the remaining rules.

The resulting program does not contain negated atoms, and is called a *definite logic program*. Let $cl(P)$ denote the deductive closure of a definite logic program P , which coincides with its minimal model. A *stable model* [Gelfond and Lifschitz, 1988] of a logic program P is an assignment S , such that $S^+ = cl(P^S)$. A normal logic program may have none, one or several stable models. The problem of deciding whether a normal logic program has a stable model is

NP-complete, but many problems can be solved efficiently in practice by state-of-the art systems such as *smodels* [Surjanen and Niemela, 2001] and *DLV* [Dell'Armi *et al.*, 2001].

An alternative semantics for normal logic programs is the *partial stable model* semantics [Sacca and Zaniolo, 1990]. This is a three-valued semantics, where each atom may assume the values: true, false or undefined. Given a three-valued assignment S of $Atoms(P)$, we say that S is a *partial model* of P if for each $\neg A \in S^-$, every rule with head A contains at least one literal B in its body, such that $\neg B \in S$. The reduct P^S of P w.r.t. a partial model S is defined as the program that is obtained from P after deleting (i) every rule of P that has a negated atom *not* b_i , with $b_i \in S^+$ (ii) every rule where an undefined atom occurs (iii) every negated atom from the body of the remaining rules. A partial model S such that $S^+ = cl(P^S)$ is called a *founded model* of P . A maximal founded model is called *partial stable model* of P . It is not hard to see that every program has a partial stable model.

Example 1 Consider the program

$a \leftarrow \text{not } d \quad b \leftarrow \text{not } a \quad c \leftarrow \text{not } b \quad d \leftarrow \text{not } c$

This program has two stable models, namely $M_1 = \{a, c\}$ and $M_2 = \{b, d\}$. Consider now the program

$a \leftarrow \text{not } c \quad b \leftarrow \text{not } a \quad c \leftarrow \text{not } b$

$d \leftarrow \text{not } e \quad e \leftarrow \text{not } d$

This program has two partial stable models, namely $M_1 = \{d\}$ and $M_2 = \{e\}$, but no stable model.

The intuition behind a founded model is similar to that behind a stable model, except that we are willing to ignore certain variables and the rules that they participate in. A partial stable model is one where we try to ignore as few variables (w.r.t. set inclusion) as we can.

5.2 Translating CP-nets into Logic Programs

In this section we translate CP-nets into nonmonotonic logic programs, showing a one-to-one correspondence between the net's optimal outcomes and the stable models of the corresponding logic program.

A preference $p_1 \vee p_2 \vee \dots \vee p_k : q_1 > q_2 > \dots > q_n$, where each p_i is a conjunction of atoms, translates into a set of rules of the following form $q_1 \leftarrow p_j$ for $1 \leq j \leq k$. Additionally, for every variable X of the network with $\text{dom}(X) = \{v_1, v_2, \dots, v_n\}$ a set of rules of the form $v_i \leftarrow \text{not } v_{i_1}, \text{not } v_{i_2}, \dots, \text{not } v_{i_{n-1}}$, for $i_j \neq i$, is added to the program. We call this set *choice rules set*. Finally for each variable X , with $\text{dom}(X) = \{v_1, v_2, \dots, v_n\}$ we add the set of *uniqueness constraints* $\leftarrow v_i, v_j$, for every pair of values such that $i \neq j$.

Theorem 3 o is a strongly optimal outcome for a CP-net iff it is a stable model of its corresponding logic program.

Note that only the most preferred values of each preference statement participate in the translation of a CP-network into a logic program. Furthermore, the *not* operator appears only in the choice rules set, a feature that seems to indicate a rather weak link between CP-networks and nonmonotonic logic programs. However, in the long version of the paper we show that in the case of CP-networks with constraints, the *not* operator plays a more central role in the translation and the relation between CP-networks and logic programs is stronger than it may appear here.

Example 2 Consider the network:

$a_1 : b_1 > b_2 \quad a_2 : b_2 > b_1 \quad b_1 : a_1 > a_2 \quad b_2 : a_2 > a_1$

The corresponding logic program is

$b_1 \leftarrow a_1 \quad b_2 \leftarrow a_2$

$a_1 \leftarrow b_1 \quad a_2 \leftarrow b_2$

$a_1 \leftarrow \text{not } a_2 \quad a_2 \leftarrow \text{not } a_1 \quad b_1 \leftarrow \text{not } b_2$

$b_2 \leftarrow \text{not } b_1 \quad \leftarrow a_1, a_2 \quad \leftarrow b_1, b_2.$

The above program has exactly two stable models, namely

$M_1 = \{a_1, b_1\}$ and $M_2 = \{a_2, b_2\}$.

5.3 Partial Stable Models and Partial Outcomes

We showed a direct correspondence between the optimal outcomes of a CP-network and the stable models of its associated logic program. However, some logic programs have *no* stable model, much like some CP-nets have no strongly optimal outcome. We can interpret the non-existence of an optimal outcome as an indication that some of the preferences are not well-defined. There are two ways to handle such situations: we can isolate the ill-defined preferences and reason with the rest of the network, or we can try to revise the ill-defined preferences. In both case, we require the ability to identify the problematic parts of the network. Here we show how partial stable models can provide valuable assistance in this task.

Example 3 Consider the network defined as follows.

$a_1 : b_1 > b_2 \quad a_2 : b_2 > b_1 \quad b_1 : a_2 > a_1 \quad b_2 : a_1 > a_2$

$c_1 : d_1 > d_2 \quad c_2 : d_2 > d_1 \quad d_1 : c_1 > c_2 \quad d_2 : c_2 > c_1$

Its corresponding logic program is

$b_1 \leftarrow a_1 \quad b_2 \leftarrow a_2 \quad a_2 \leftarrow b_1 \quad a_1 \leftarrow b_2$

$c_1 \leftarrow d_1 \quad c_2 \leftarrow d_2 \quad d_1 \leftarrow c_1 \quad d_2 \leftarrow c_2$

together with the corresponding choice rules set and uniqueness constraints. This program has no stable model, but it has two partial stable models, $M_1 = \{c_1, d_1\}$ and $M_2 = \{c_2, d_2\}$. These partial stable models do not assign a value to variables a_1, a_2, b_1, b_2 because the corresponding subnetwork does not possess an optimal outcome.

We call the outcomes of a CP-network that correspond to the partial stable models of its associated logic program, *partial optimal outcomes*. Intuitively, a partial stable model corresponds to an optimal assignment to a coherent sub-net of the original CP-net that contains the same preferences as the CP-net. Thus, we attempt to remove certain nodes, until we get a coherent specification. Those nodes that were removed are the ones we should point to the user as problematic.

Existing knowledge on partial stable models (e.g., [Dimopoulos et al., 2002]) implies that a total partial optimal outcome (i.e., one that assigns a value to every variable) is an optimal outcome, and that deciding whether a logic program has a partial stable model other than the empty set is NP-complete. Discovering whether a variable appears in *some* partial optimal outcome is NP-complete, whereas deciding whether a variable appears in *all* partial optimal outcomes is Π_2^P -complete.

6 Conclusion

We showed that the current semantics of CP-network is too conservative in its interpretation of user specification. This is contrary to the aim of research on preference elicitation techniques, i.e., reducing the burden on users. Consequently, we

suggested a different semantics under which optimization can be meaningfully applied to CP-nets that are currently considered inconsistent. We showed how such an optimization process can be practically implemented, extending current optimization techniques to cyclic network. Moreover, we described a computational method for identifying sub-networks that are meaningful, providing the first tool that can point out to a user what part of her specification requires revision.

References

- [Aspvall et al., 1979] B. Aspvall, M. Plass, and R. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Proc. Letters*, 1979.
- [Boutilier et al., 1999] C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. Reasoning with conditional ceteris paribus preference statements. In *UAI'99*, 1999.
- [Brewka et al., 2002] G. Brewka, I. Niemelä, and T. Syrjänen. Implementing ordered disjunction using answer set solvers for normal programs. In *JELIA'02*, 2002.
- [Dell'Armi et al., 2001] T. Dell'Armi, W. Faber, G. Ielpa, C. Koch, N. Leone, S. Perri, and G. Pfeifer. System description: DLV. In *Proceedings of LPNMR-01*, 2001.
- [Dimopoulos et al., 2002] Y. Dimopoulos, B. Nebel, and F. Toni. On the computational complexity of assumption-based argumentation for default reasoning. *Artificial Intelligence*, 141, 2002.
- [Domshlak and Brafman, 2002] C. Domshlak and R. I. Brafman. CP-Nets – reasoning and consistency testing. In *Proc. KR'02*, 2002.
- [Domshlak et al., 2001] C. Domshlak, R. I. Brafman, and E. S. Shimony. Preference-based configuration of web page content. In *Proc. IJCAI'01*, 2001.
- [Doyle and Wellman, 1994] J. Doyle and M. Wellman. Representing preferences as *ceteris paribus* comparatives. In *AAAI Spring Sym. on Decision-Theoretic Planning*, 1994.
- [Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. ICSLP-88*, 1988.
- [Hanson, 1996] S. O. Hanson. What is a ceteris paribus preference. *J. of Philosophical Logic*, 25:307–332, 1996.
- [Keeney and Raiffa, 1976] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, New York, 1976.
- [Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [Sacca and Zaniolo, 1990] D. Sacca and C. Zaniolo. Stable models and non-determinism in logic programs with negation. In *Proc. PODS'90*, 1990.
- [Surjanen and Niemela, 2001] T. Surjanen and I. Niemela. The Smodels system. In *Proceedings of LPNMR-01*, 2001.

Categorizing classes of signals by means of fuzzy gradual rules

Sylvie Galichet, Didier Dubois, Henri Prade

Institut de Recherche en Informatique de Toulouse (IRIT)

118 route de Narbonne, 31062 Toulouse Cédex 4, France

{galichet, dubois, prade}@irit.fr

Abstract

This paper presents an approach to the approximate description of univariate real-valued functions in terms of precise or imprecise reference points and interpolation between these points. It is achieved by means of gradual rules which express that the closer the variable to the abscissa of a reference point, the closer the value of the function to the ordinate of this reference point. Gradual rules enable us to specify sophisticated gauges, under the form of connected areas, inside of which the function belonging to the class under consideration should remain. This provides a simple and efficient tool for categorizing signals. This tool can be further improved by making the gauge flexible by means of fuzzy gradual rules. This is illustrated on a benchmark example.

1 Introduction

Signal functions, such as time series, medical ECG's, are usually viewed as analytical mappings. Then, a precise representation is often used. Even when uncertainty is dealt with, it is supposed to pervade parameters of the analytical models, leading to probabilistic or interval-based processing. The main objective of this paper is to propose an alternative to this classical type of approaches by investigating the interest of a special kind of fuzzy « if .. then » rules, named gradual rules [Dubois and Prade, 1992; 1994], for developing imprecise representations. Actually, the proposed strategy relies on interpolative reasoning. By specifying the interpolative representation with gradual rules, there is no need to choose an analytic form for the interpolator and an imprecise model is directly obtained from the constraints expressed by the rules. The proposed rule-based approach is an alternative to works based on fuzzy polynomials [Lowen, 1990] or fuzzy spline interpolation, e.g. [Kawaguchi and Miyakoshi, 1998], which rely on fuzzy-valued functions and still depend on the analytical form of the interpolant.

What is supposed to be known, in a precise or in an imprecise way, is the behaviour of the function at some reference points, the problem being to interpolate between these points. Figure 1 illustrates our purpose of building an imprecise model in a case where the points on which interpolation is based are imprecise and modelled by rectangular areas.

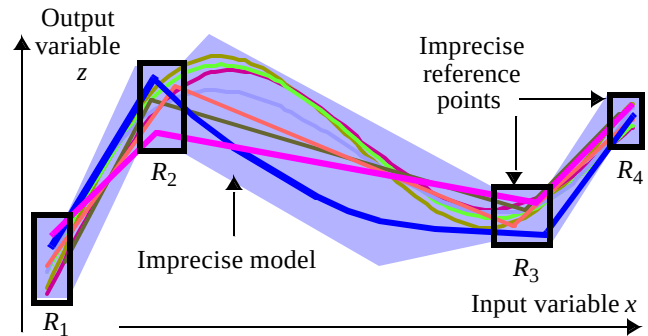


figure 1: Imprecise interpolation

So, we are no longer looking for a function, but for a relation linking the variable to the possible value of the function. Then, this relation is represented by its graph Γ defined on the Cartesian product $X \times Z$ (where X is the input domain, and Z the function range). A similar approach used in automated control, recently proposed in [Sala and Albertos, 2001], also considers the building of uncertain fuzzy models in the setting of the approximation of multi-valued functions called “ambiguous functions”.

The paper, after some brief background on gradual rules, discusses the design of the imprecise interpolative representation in terms of gradual rules, constrained by precise or imprecise reference points. The proposed approach is then applied to the classification of time series. Moreover, a more powerful representation framework, based on fuzzy sets of gradual rules (called fuzzy gradual rules for short) is proposed. It enables us to describe areas through which it is not completely possible that the function may go.

2 Interpolation and gradual rules

The idea of imprecise interpolation suggested above is based on constraints to be satisfied. Namely the results of the interpolation should agree with the reference points. These constraints define the graph Γ of a relation on $X \times Z$.

We first consider the case of precise interpolation points P_i with coordinates (x_i, z_i) , $i = 1, \dots, n$. Then the relation Γ should satisfy:

$$\begin{aligned} \Gamma(x_i, z_i) &= 1, \\ \forall z \neq z_i \in Z, \quad \Gamma(x_i, z) &= 0, \end{aligned}$$

for $i = 1, \dots, n$. Without any further constraint on the nature of the interpolation, we only have:

$$\forall x \neq x_i \in X, \forall z \in Z, \quad \Gamma(x, z) = 1.$$

Thus each interpolation point induces the constraint “If $x = x_i$ then $z = z_i$ ”, represented by the rule $(x = x_i) \rightarrow (z = z_i)$ where $\rightarrow$ is material implication. The relation Γ is thus obtained as the conjunction:

$$\Gamma(x, z) = \bigwedge_{i=1, \dots, n} (x = x_i) \rightarrow (z = z_i). \quad (1)$$

This relation is extremely imprecise since there is no constraint at all outside the reference points. Instead of using a precise type of interpolation function, one may use fuzzy rules in order to express constraints in the vicinity of the interpolation points. The idea is to use rules of the form “the closer x is to x_i , the closer z is to z_i ”. The extension of equation (1) to gradual rules provides the following expression for the graph Γ :

$$\Gamma(x, z) = \min_{i=1, \dots, n} \mu_{\text{close to } x_i}(x) \rightarrow \mu_{\text{close to } z_i}(z) \quad (2)$$

where $\rightarrow$ represents Rescher-Gaines implication ($a \rightarrow b = 1$ if $a \leq b$ and $a \rightarrow b = 0$ if $a > b$), and $\mu_{\text{close to } x_i}(x)$ is the degree of truth of the proposition “ x is close to x_i ”.

We have just to define what is meant by “close to”. Let A_i denote the fuzzy set of values close to x_i . It is natural to set $\mu_{A_i}(x) = 1$ if $x = x_i$ and to assume that the membership degree to A_i decreases on each side of x_i with the distance to x_i . However A_i will not be necessarily symmetrical. The simplest solution consists in choosing triangular fuzzy sets with a support denoted by $[x_i^-, x_i^+]$. In a similar way, the closeness to z_i will be modelled by a triangular fuzzy set B_i with modal value z_i and support $[z_i^-, z_i^+]$. Then the interpolation relation only depends on $4n$ parameters $x_i^-, x_i^+, z_i^-, z_i^+$ for n interpolation points. In order to simplify their tuning, we further assume that at most two rules are simultaneously fired at each point of the input domain, i.e. $x_i \leq x_{i+1}^-$ and $x_i^+ \leq x_{i+1}$, $i = 1, \dots, n-1$. For increasing reference points (i.e. $x_i < x_{i+1}$ and $z_i < z_{i+1}$, $i = 1, \dots, n-1$), it has been established in [Galichet et al., 2002] that the interpolation graph is made of connected 4-sided areas (as pictured in figure 2) when the following constraints hold between the parameters:

$$(z_i^+ - z_{i+1})(x_{i+1}^- - x_{i+1}) = (z_{i+1}^+ - z_{i+1})(x_i^+ - x_{i+1}) \quad (3)$$

$$(z_{i+1}^- - z_i)(x_i^+ - x_i) = (z_i^- - z_i)(x_{i+1}^- - x_i). \quad (4)$$

Similar relationships can be obtained when considering decreasing reference points. These constraints can be related to consistency requirements between gradual rules [Dubois et al., 1997], ensuring the non emptiness of the image of each input point via the relation Γ .

Figure 2 pictures the interpolation graph which is obtained with three interpolation points, and thus three gradual rules whose A_i 's and B_i 's also appear in figure 2. The partitioning of X is obtained by cutting the intervals $[x_i, x_{i+1}]$ into three equal parts, i.e. $x_{i+1}^- - x_i = x_i^+ - x_{i+1}^- = x_{i+1} - x_i^+$, as illustrated by the parameters which define the fuzzy sets A_i . The extreme values z_1^- and z_3^+ are also predefined. Lastly, the other parameters can be obtained by solving the system of equations derived from (3) and (4).

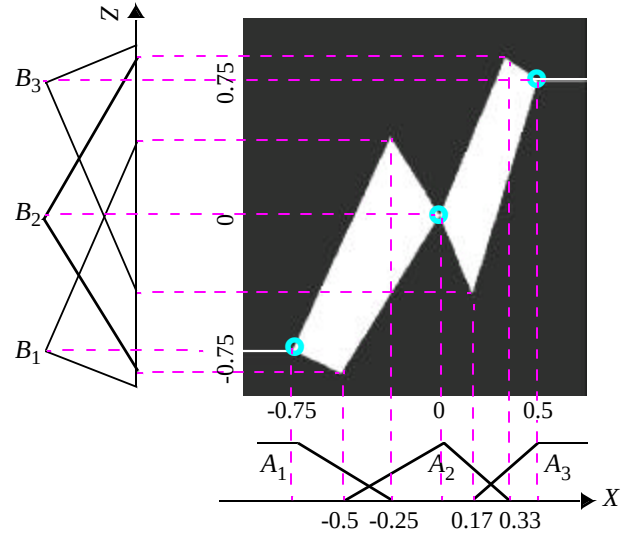


figure 2: Piecewise quadrangle-shaped graph

This approach straightforwardly extends to imprecise reference points provided that trapezoidal membership functions are used in place of triangular ones (see figure 3).

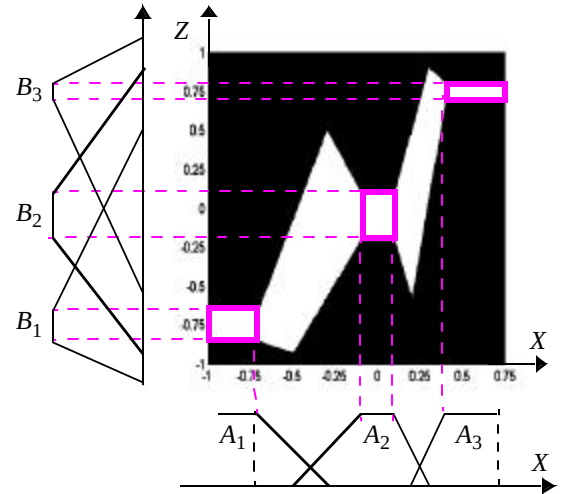


figure 3: Imprecise reference points

3 Classification of time series

Our purpose is now to illustrate how the imprecise interpolative representation framework can be used to classify time series, see, e.g. [Kadous, 2002]. Supervised classification is assumed, contrary to clustering techniques whose recent developments are often based on hidden Markov models [Biswas et al., 1998; Li, 2000]. The proposed experiment deals with the «Control-Chart» database, freely available from the UCI Data Archive [UCI KDD Archive]. It is a 6-class problem, with 100 examples of each class, a prototype of each class being presented in figure 4. A categorization of process trends, based on types of variation, is also adopted in qualitative reasoning and model-based diagnosis [Colomer *et al.*, 2002].

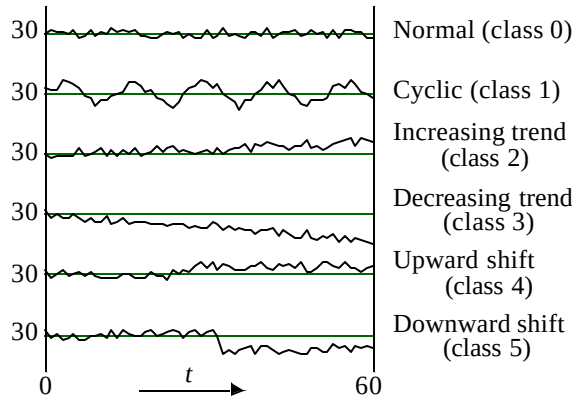


figure 4: One example of each class

Given an unlabeled time series, the aim of the classification is to decide to which class it belongs. The idea behind the proposed methodology consists in developing an imprecise model of each class. Then, the time series to classify will be assigned to the class whose model presents the maximal adequacy with the temporal data under consideration. The imprecise models are specified using gradual rules as advocated in the previous section.

In figure 5, ten examples of class 5 are plotted simultaneously. It clearly shows that the reference points are no longer precise. In this context, triangular membership functions are replaced by trapezoidal membership functions whose cores delimit the rectangular areas associated with the imprecise reference points. According to this slight modification, the graph plotted in figure 5 is obtained from two gradual rules, i.e. $A_1 \rightarrow B_1$ and $A_2 \rightarrow B_2$. It can be shown that the constraints on the graph shape expressed by equations (3) and (4) are still valid.

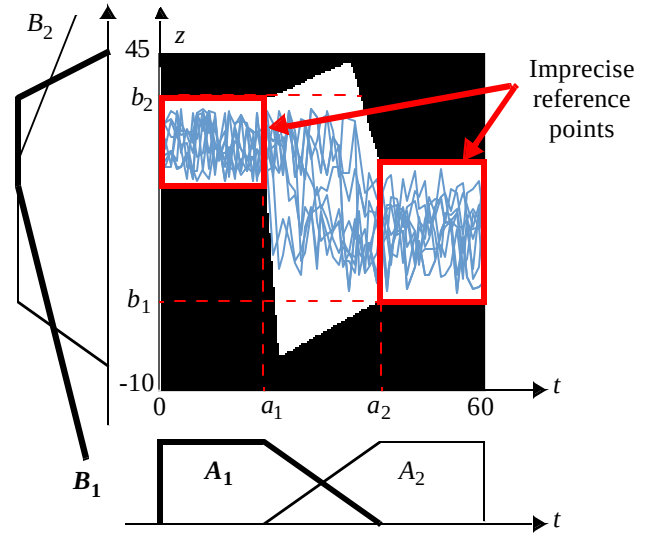


figure 5: Imprecise model of class 5

The model so built can be further improved by truncating the upper and lower parts of the quadrangle-shaped graph. An easy strategy to implement the graph cutting consists in adding a new rule that directly translates the interval-based constraint “If $t \in [a_1, a_2]$ then $z \in [b_1, b_2]$ ” where a_1 , a_2 , b_1 and b_2 are defined in figure 5. Such an approach results in the final graph given in figure 6.

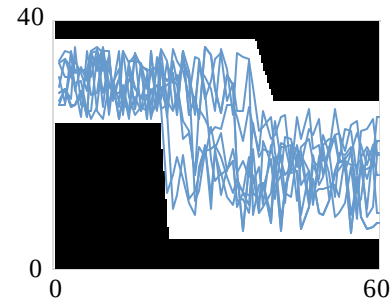


figure 6: Model of class 5 (truncated graph)

Figure 7 and 8 present the implemented models for two others classes. The first one associated with class 3 is based on reference points whose imprecision is only relative to the output. Using strong partitions with triangular input membership functions and trapezoidal output ones, imprecise linear interpolation is obtained by means of the two rules $A_1 \rightarrow B_1$ and $A_2 \rightarrow B_2$.

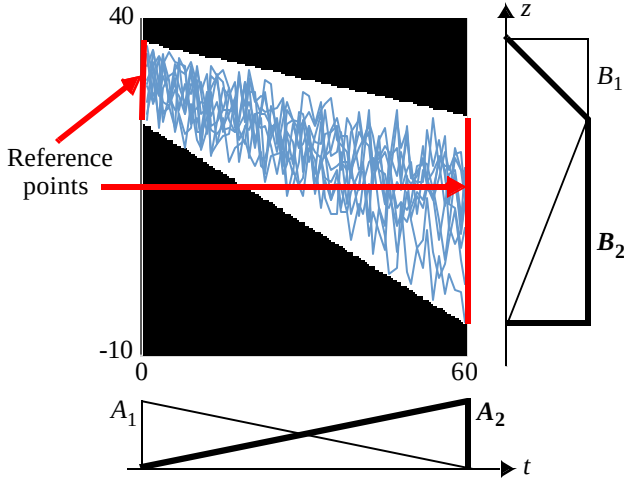


figure 7: Model for class 3 (decreasing trend)

Concerning cyclic time series (figure 8), the non-monotonic underlying behavior induces some difficulties in the modeling process. Actually, closeness on the right and on the left of the reference points must be handled in different ways. It means that two distinct fuzzy subsets are required for correctly dealing with each reference point. In this framework, the imprecise models of figure 8 is composed of 9 gradual rules.

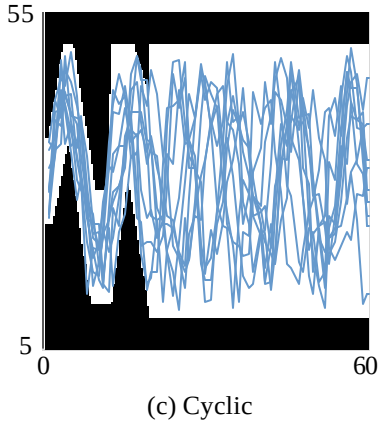


figure 8: Model for class 1 (cyclic)

Imprecise models are built for the six classes so that the graphs include all the points that compose the training time series (10 for each class). The classification of a time-series, given as a collection of points (t_i, z_i) , $t_i = i = 1, \dots, 60$, is then carried out according to its adequacy with the class models. The latter is determined from the number of points of the time series under consideration that belong to the graph of each model, that is:

$$N_j = \sum_{i=1, \dots, 60} \Gamma_j(t_i, z_i), j = 1, \dots, 6 \quad (5)$$

where Γ_j denotes the model graph of the j^{th} class. The final decision is then to assign the time series, supposed unlabeled, to the class that maximizes N_j , $j = 1, \dots, 6$. Applying this strategy

to classify the 600 available examples, perfect classification is obtained, i.e. the error rate is null for the training examples but also for the test time series. This result is better than the one obtained with other approaches of the same problem [Keogh and Kasetty, 2002; Nanopoulos et al., 2001]. It is however important to be cautious about this good performance. Indeed, the discrimination between some classes is not robust. This point is illustrated by figure 9 in the case of classes 3 and 5 which are difficult to differentiate. The adequacy between the 100 time series of class 3 and models of classes 3 and 5 is plotted. It can be stated that, for many time series, the difference between both obtained scores is small, which means that a slight modification of the models would probably result in different final decisions. Actually, an important intersection between both models induces a loss of the discrimination power of the adequacy index. In this framework, one may think of improving the robustness of the classification by refining the imprecise models. One possible strategy is then to introduce some membership degrees in the four sided areas while keeping their support unchanged. This can be made by using fuzzy sets of gradual rules as shown now.

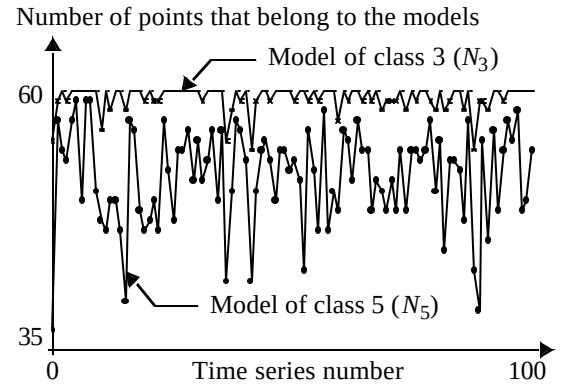


figure 9: Adequacy of class 3 examples with models of classes 3 and 5

4 Interpolative fuzzy graph

According to the previous sections, it is clear that given a set of rules, i.e. a set of reference points, a collection of crisp graphs can be obtained by varying the support parameters of the A_i 's and/or the B_i 's. Moreover, inclusion properties between the built graphs can be exhibited for controlled variations of the supports as expressed by the following statements.

P1: If $A_i \subseteq A_i^*$, $i=1, \dots, n$, then $\Gamma^* \subseteq \Gamma$, where Γ and Γ^* are the graphs associated respectively with rules $A_i \rightarrow B_i$ and $A_i^* \rightarrow B_i$. Indeed, $(x, z) \in \Gamma^*$ means that $\forall i, A_i^*(x) \leq B_i(z)$. According to the assumption that $\forall i, A_i \subseteq A_i^*$, it follows that $\forall i, A_i(x) \leq B_i(z)$ which results in $(x, z) \in \Gamma$.

P2: If $B_i^* \subseteq B_i$, $i=1, \dots, n$, then $\Gamma^* \subseteq \Gamma$, where Γ and Γ^* are now the graphs associated respectively with rules $A_i \rightarrow B_i$ and $A_i \rightarrow B_i^*$. Indeed, $(x, z) \in \Gamma^*$ if and

only if $\forall i, A_i(x) \leq B_i^*(z)$. Since $\forall i, B_i^* \subseteq B_i$, it follows that $\forall i, A_i(x) \leq B_i(z)$, i.e. $(x, z) \in \Gamma$.

The combination of P1 and P2 leads to:

P3: If $A_i \subseteq A_i^*$ and $B_i^* \subseteq B_i$, $i=1, \dots, n$, then $\Gamma^* \subseteq \Gamma$, where Γ and Γ^* are now the graphs respectively associated with rules $A_i \rightarrow B_i$ and $A_i^* \rightarrow B_i^*$.

Thus, according to the above inclusion properties, it is possible to design a family of nested graphs simply by building collections of nested fuzzy subsets on X and Z . Indeed, denote by $\{A_i^\lambda, \lambda \in [0,1]\}$ a family of fuzzy subsets on X such that $A_i^{\lambda'} \subseteq A_i^\lambda$ if $\lambda \geq \lambda'$, and $\{B_i^\lambda, \lambda \in [0,1]\}$ a family of fuzzy subsets on Z such that $B_i^\lambda \subseteq B_i^{\lambda'}$ if $\lambda \geq \lambda'$. The family of rules $A_i^\lambda \rightarrow B_i^\lambda$, $\lambda \in [0,1]$, is such that $\Gamma^\lambda \subseteq \Gamma^{\lambda'}$ if $\lambda \geq \lambda'$. Actually, such a construction of nested graphs simply expresses that implicative graphs increase in the sense of inclusion when underlying constraints become more permissive. More permissive rules are obtained either by restricting input conditions further, or by enlarging output fuzzy sets.

Using a convex linear combination enables the automatic construction of such a collection of nested fuzzy subsets ranging from the lower bound of the family to the upper one. Applying such a technique results in the following proposal:

$$A_i^\lambda = (1-\lambda)A_i^0 \oplus \lambda A_i^1, \lambda \in [0,1], i=1, \dots, n \quad (6)$$

where A_i^0 and A_i^1 , such that $A_i^0 \subseteq A_i^1$, are the lower and upper bounds of the family and $\oplus$ denotes the extended sum of fuzzy numbers. In the same way, nested output fuzzy subsets can be built according to:

$$B_i^\lambda = (1-\lambda)B_i^0 \oplus \lambda B_i^1, \lambda \in [0,1], i=1, \dots, n \quad (7)$$

where B_i^0 and B_i^1 , such that $B_i^1 \subseteq B_i^0$, are the upper and lower bounds of the family. It should be noted that the inclusion ordering of the B_i^λ using λ is the converse of the one of the A_i^λ , due to opposite behaviors with respect to graph inclusion.

Using the so-built fuzzy subset families (see figure 10) results in the following graph inclusions:

$$\Gamma^1 \subseteq \Gamma^\lambda \subseteq \Gamma^{\lambda'} \subseteq \Gamma^0, \lambda, \lambda' \in [0,1] \text{ and } \lambda \geq \lambda' \quad (8)$$

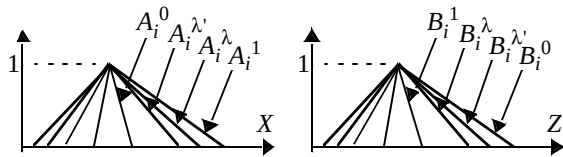


figure 10: Nested fuzzy subsets ($\lambda \geq \lambda'$)

One interesting point is that the 4-sided shape introduced in section 2 is shared by all nested graphs provided that the lower and upper graphs are themselves 4-sided areas. In other words, equalities (3) and (4) hold for any Γ^λ when they hold for Γ^0 and Γ^1 as expressed by property P4.

P4: If both (A_i^0, B_i^0) and (A_i^1, B_i^1) , $i=1, \dots, n$, satisfy equations (3) and (4), then (3) and (4) are still valid for any $(A_i^\lambda, B_i^\lambda)$, λ

$\in [0,1]$, $i=1, \dots, n$, when A_i^λ and B_i^λ are built according to (6) and (7).

An interpretation consists in viewing F as a fuzzy set of crisp graphs, that is as a level 2 fuzzy set, i.e. a fuzzy set of fuzzy sets. In this case, F is represented as:

$$F = \int_{\lambda \in [0,1]} \lambda / \Gamma^\lambda \quad (9)$$

according to Zadeh's notation where the integral sign stands for the union of the fuzzy singletons λ / Γ^λ . A single fuzzy gradual rule is pictured in figure 11.

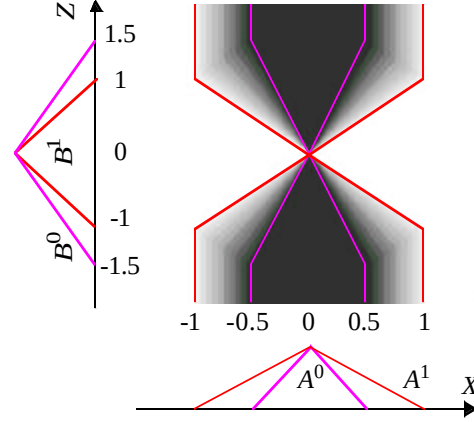


figure 11: A fuzzy gradual rule

Figure 12 plots the fuzzy graph obtained when the lower graph Γ^1 is precise and piecewise linear and the upper graph Γ^0 has the quadrangle-based shape of figure 2.

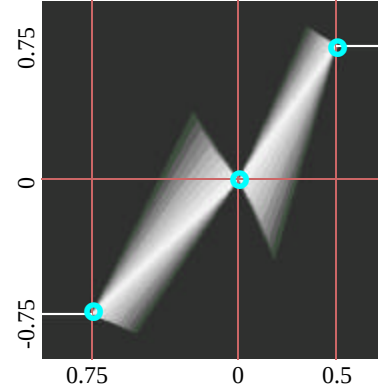


figure 12: A graph based on 3 fuzzy gradual rules

Using fuzzy gradual rules for dealing with the example of section 3 still results in perfect classification. Moreover, the robustness of the classification is improved as illustrated by figure 13, where histograms of the difference $N_3 - N_5$ are plotted for class 3 time series (see figure 9). It is clear that the number of examples for which the final decision is brittle, i.e. for the small values of $N_3 - N_5$ which are accounted for in the two first bars, decreases when fuzzy gradual rules are used. These results have been obtained by using the same form of indices N_j given by equation (5) for crisp and fuzzy graphs

except that $\Gamma_j(t_i, z_i) \in [0, 1]$ in the fuzzy case. Further improvements could probably be obtained by defining more sophisticated indices in the fuzzy case.

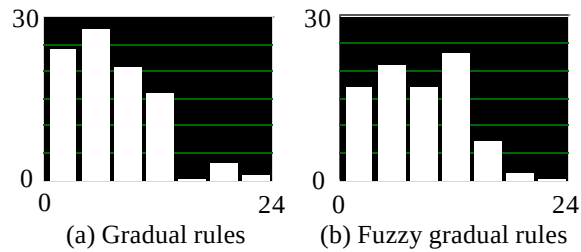


figure 13: Effect of fuzzy gradual rules on the discrimination

5 Conclusion

This paper has proposed a modelling framework which is faithful to the imprecision of available data. In the intervals between interpolation points where it is difficult to specify an analytical model, imprecision is captured by means of 4-sided areas. The application of the modelling methodology for classifying time series has exhibited interesting performance. Moreover, the discrimination power of the approach is improved by using fuzzy gradual rules as introduced in the previous section. Indeed, they enable us to distinguish between typical members of a class which remain in subareas with high membership degrees from borderline members which go through subareas with smaller membership degrees. Besides, one may probably take further advantage of the easy interface with the user, provided by the use of gradual rule, for specifying queries in data mining applications (see [Keogh et al., 2002] for an example of such possible use). Lastly, further research should deal with multidimensional spaces where the language of gradual rules may prove useful in the description of imprecise graphs.

References

- [Biswas et al., 1998] G. Biswas, J. Weinberg and D. Fisher. ITERATE: A conceptual clustering algorithm for data mining. *IEEE Trans. on Systems, Man and Cybernetics - Part C: Applications and reviews*, 28(2), 219–230, May 1998.
- [Colomer et al., 2002] J. Colomer, J. Melendez, F.I. Gamero. Pattern recognition based on episodes and DTW. Application to diagnosis of a level control system. *Proceedings of the 16th Int. Workshop on Qualitative Reasoning (QR'02)*, Barcelona, Spain, June 2002.
- [Dubois and Prade, 1992] D. Dubois and H. Prade. Gradual inference rules in approximate reasoning. *Information Sciences*, 61(1,2), 103–122, April 1992.
- [Dubois and Prade, 1994] D. Dubois and H. Prade. Basic issues on fuzzy rules and their application to fuzzy control. In *Fuzzy Logic and Fuzzy Control*, D. Driankov, P.W. Eklund, A.L. Ralescu Eds, Lecture Notes in Artificial Intelligence, 833, Springer-Verlag, 3–14, 1994.
- [Dubois et al., 1997] D. Dubois, H. Prade and L. Ughetto. Checking the Coherence and Redundancy of Fuzzy Knowledge Bases. *IEEE Trans. on Fuzzy Systems*, 5(3), 398–417, August. 1997.
- [Galichet et al., 2002] S. Galichet, D. Dubois and H. Prade. Imprecise specification of ill-known functions using gradual rules. *Proceedings of the 2nd Int. Workshop on Hybrid Methods for Adaptive Systems (EUNITE 2002)*, pages 512–520, Albufeira, Portugal, September 2002.
- [Kadous, 2002] M. W. Kadous. Temporal classification: extending the classification paradigm to multivariate time series. PhD thesis, The University of New South Wales, January 2002.
- [Keogh and Kasetty, 2002] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Proceedings of the 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 102–111, Edmonton, Canada, July 2002.
- [Kawaguchi and Miyakoshi, 1998] M.F. Kawaguchi and M. Miyakoshi. Fuzzy spline interpolation in sparse fuzzy rule bases. *Proceedings of the 5th International Conference on Soft Computing (IIZUKA'98)*, pages 664–667, Iizuka, Japan, October 1998.
- [Keogh et al., 2002] E. Keogh, H. Hochheiser and B. Shneiderman. An augmented visual query mechanism for finding patterns in time series data. *Proceedings of the 5th Int. Conf. on Flexible Query Answering Systems (FQAS 2002)*, pages 240–250, Copenhagen, Denmark, October 2002.
- [Li, 2000] C. Li. A bayesian approach to temporal data clustering using the hidden Markov model methodology. PhD thesis, Vanderbilt University, December 2000.
- [Lowen, 1990] R. Lowen. A fuzzy Lagrange interpolation theorem. *Fuzzy Sets and Systems*, 34, 33–38, 1990.
- [Nanopoulos et al., 2001] A. Nanopoulos, R. Alcock and Y. Manolopoulos. Feature-based classification of time-series data. *International Journal of Computer Research*, 10(3), 2001.
- [Sala and Albertos, 2001] A. Sala and P. Albertos. Inference error minimisation: fuzzy modelling of ambiguous functions. *Fuzzy Sets and Systems*, 121(1), 95–111, July 2001.
- [UCI KDD Archive] The UCI KDD Archive (<http://kdd.ics.uci.edu/>), Information and Computer Science, University of California, Irvine.

Gaussian Process Models of Spatial Aggregation Algorithms

Naren Ramakrishnan

Department of Computer Science
Virginia Tech, VA 24061, USA
naren@cs.vt.edu

Chris Bailey-Kellogg

Department of Computer Sciences
Purdue University, IN 47907, USA
cbk@cs.purdue.edu

Abstract

Multi-level spatial aggregates are important for data mining in a variety of scientific and engineering applications, from analysis of weather data (aggregating temperature and pressure data into ridges and fronts) to performance analysis of wireless systems (aggregating simulation results into configuration space regions exhibiting particular performance characteristics). In many of these applications, data collection is expensive and time consuming, so effort must be focused on gathering samples at locations that will be most important for the analysis. This requires that we be able to functionally model a data mining algorithm in order to assess the impact of potential samples on the mining of suitable spatial aggregates. This paper describes a novel Gaussian process approach to modeling multi-layer spatial aggregation algorithms, and demonstrates the ability of the resulting models to capture the essential underlying qualitative behaviors of the algorithms. By helping cast classical spatial aggregation algorithms in a rigorous quantitative framework, the Gaussian process models support diverse uses such as directed sampling, characterizing the sensitivity of a mining algorithm to particular parameters, and understanding how variations in input data fields percolate up through a spatial aggregation hierarchy.

1 Introduction

Many important tasks in data mining, scientific computing, and qualitative modeling involve the successive and systematic spatial aggregation and redescription of data into higher-level objects. For instance, consider the characterization of WCDMA (wideband code-division multiple access) wireless system configurations for a given indoor environment. In noisy channels, the performance goal is to quantitatively assess the relationship between the signal-to-noise ratio (SNR) and the bit error rate (BER) or bit error probability (BEP) of the realized configuration. To improve performance in office environments (characterized by doorways, walls, cubicles), a common trick used is to incorporate space-time transmit diversity (STTD). Instead of a single transmitter antenna,

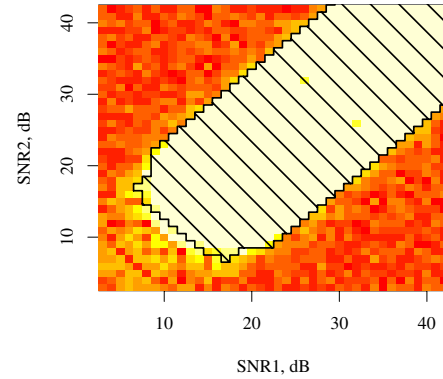


Figure 1: Mining configuration spaces in wireless system configurations. The shaded region denotes the largest portion of the configuration space where we can claim, with confidence at least 99%, that the average bit error probability (BEP) is acceptable for voice-based system usage. Each cell in the plot is the result of the spatial and temporal aggregation of hundreds of time-consuming wireless system simulations.

the base station uses two transmitter antennas separated by a small distance. If the signal from one of the antennas is weak, the signal from another is likely to be high, and the overall performance is expected to improve. In this application, it is important to assess how the power-imbalance between the two branches impacts the BEP of the simulated system, across a range of SNRs (see Fig. 1; [Verstak *et al.*, 2002]).

Characterizing the performance of WCDMA systems requires the identification of multi-level spatial aggregates in the high-dimensional configuration spaces of wireless systems. The lowest level (input) contains individual Monte Carlo simulation runs providing unbiased estimates of BEPs. This space is high-dimensional (e.g. ≥ 10), owing to the multitude of wireless system parameters (e.g. channel models, fading characteristics, coding configurations, and hardware controls). Wireless design engineers prefer to work in at most two or three dimensions (e.g. to study the effect of power imbalance on system performance) for ease of tunability and deployment. The next level of spatial aggregation thus contains *buckets* which aggregate data in terms of two dimensions, using various consistency constraints and design specifications. Finally, the third level aggregates buckets into *regions* of con-

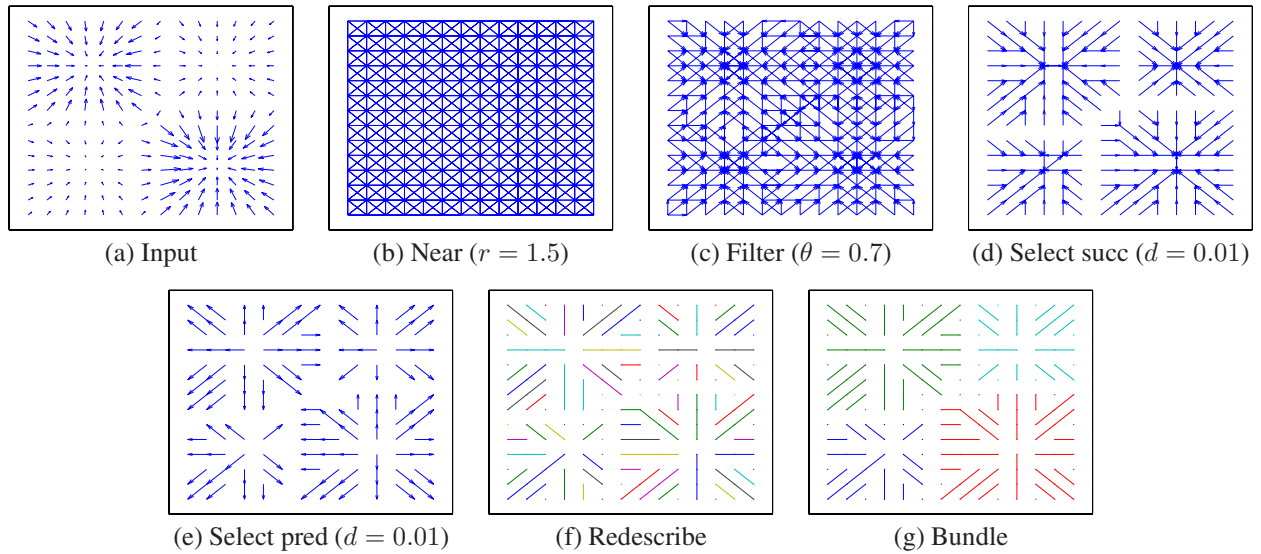


Figure 2: Key steps in vector field analysis.

strained shape; the shape of the regions illustrates the nature of joint influence of the two selected configuration parameters on performance. Specific region attributes, such as width, provide estimates for the thresholds of sensitivity of configurations to variations in parameter values.

The results of such mining are important for both qualitative and quantitative analysis. For instance, when the average SNRs of the two branches are equal, the BEP is minimal and the width of the mined region in Fig. 1 depicts the largest acceptable power imbalance (in this case, approximately 12dB). However, the width is not uniform and the region is narrower for smaller values of the SNRs. The qualitative result is that system designs situated in the lower left corner of the configuration space are more sensitive to power imbalance in the two branches.

Each input data point captures the results of a wireless system simulation which takes hours or even days (the simulations in Fig. 1 were conducted on a 200-node Beowulf cluster of workstations). Thus it is imperative that we focus data collection in only those regions that are most important to support our data mining objective, viz. to qualitatively assess the performance in configuration spaces. This requires that we model the functioning of the data mining algorithm, in order to optimize sample selection for utility of anticipated results. Modeling data mining algorithms in this manner is useful for closing the loop, characterizing the effects of the data mining algorithm's parameters, and improving our understanding of how variations in data fields percolate up through the layers. A particularly interesting application is to use such modeled structures to design information-theoretic measures for evaluating experimental designs [MacKay, 1992] and for active data selection [Cohn *et al.*, 1996; Denzler and Brown, 2002].

In order to address these goals, this paper develops a novel Gaussian process approach to modeling algorithms that mine spatial aggregates. We first overview the Spatial Aggregation mechanism for spatial data mining and the Gaussian process approach to Bayesian modeling. We then show how to inte-

grate the two approaches in order to achieve our goal of probabilistically modeling spatial data mining algorithms. We illustrate this ability within the context of identifying pockets underlying the gradient in a field — an application that captures many of the interesting characteristics of more complex studies like the wireless application.

2 Spatial Aggregation

The Spatial Aggregation Language (SAL) [Bailey-Kellogg *et al.*, 1996; Yip and Zhao, 1996], provides a set of operators and data types, parameterized by domain-specific knowledge, for uncovering and manipulating multi-layer geometric and topological structures in spatially distributed data. SAL applications construct increasingly abstract descriptions of the input data by utilizing knowledge of physical properties such as continuity and locality, expressed with the vocabulary of metrics, adjacency relations, and equivalence predicates. To understand the SAL approach (see Fig. 2), consider a SAL program for analyzing flows in a vector field (e.g. wind velocity or temperature gradient).

In the first level, the goal is to group input vectors (a) into paths so that each sample point has at most one predecessor and at most one successor. SAL breaks the process into two key steps, one capturing locality in the domain space (i.e. sample location), and the other capturing similarity in the feature space (i.e. vector direction). A neighborhood graph aggregates objects with a specified adjacency predicate expressing the notion of locality appropriate for a given domain. As shown (b), a sample point's neighbors include all other points within some specified radius r . Feature comparison then must consider only neighbors in this graph, thereby exploiting physical knowledge to gain computational efficiency while maintaining correctness. Here we break feature comparison into a sequence of predicates and graph operations. In particular, we first filter the graph (c), applying a predicate that keeps only those edges whose direction is similar enough

(within some angle tolerance θ) to the directions of the vectors at the endpoints. The remaining graph has some “junction” points where vector direction suggests multiple possible neighbors, and the most appropriate path extension from the point must be chosen. A similarity metric sums the distance between the junction and a neighbor, weighted by a constant d , and the difference in vector direction at the junction and the neighbor. The most similar neighbor for the junction is selected ((d) and (e), for successor and predecessor junctions, respectively).

The remaining graph edges are collected and redescribed as more abstract streamline curve objects (f), for the second level of analysis. Again, computation is localized so that only neighboring streamlines are compared. The neighborhood graph here (not shown) uses an adjacency predicate that declares streamlines neighbors if their constituent points were in the first level. It is then straightforward to identify convergent flows (g) with an equivalence predicate that tests when constituent points form a junction in the graph in (c). If desired, these flow bundles can be abstracted and analyzed at an even higher level.

SAL’s uniform spatial reasoning mechanism, instantiated with appropriate domain knowledge, has proved successful in applications ranging from decentralized control design [Bailey-Kellogg and Zhao, 1999; 2001], to weather data analysis [Huang and Zhao, 1999], to analysis of diffusion-reaction morphogenesis [Ordóñez and Zhao, 2000]. Recent work has focused on optimizing sample selection for applications where data collection is expensive, including identifying flows in multi-dimensional gradient fields [Bailey-Kellogg and Ramakrishnan, 2001] and analyzing matrix properties via perturbation sampling [Ramakrishnan and Bailey-Kellogg, 2002]. This paper provides the mathematical foundations necessary for the modeling of such SAL programs to support the meta-level reasoning tasks outlined in the introduction.

3 Gaussian Processes

Gaussian processes have become popular in the last few years, especially as a unifying framework for studying multivariate regression [Rasmussen, 1996], pattern classification [Williams and Barber, 1998], and hierarchical modeling [Menzefricke, 2000]. The underlying idea can be traced back to the geostatistics technique called *kriging* [Journel and Huijbregts, 1992], named after the South African miner Danie Krige. In kriging, the unknown function to be modeled (e.g., ozone concentration) over a (typically) 2D spatial field is expressed as the realization of a stochastic process. A prior is placed over the function space represented by this stochastic process, by suitably selecting a covariance function. Given measured function values at sample locations, kriging then proceeds to estimate the parameters of the covariance function (and any others pertaining to the random process). Using such values a prediction of the response variable can then be made for a new sample point, typically using MAP or ML inference. This basic approach is still popular in many tasks of spatial data analysis.

Even though parameters are estimated in this approach, it is important to note that kriging is fundamentally a memory-

based technique, since the estimated parameters only describe the underlying covariance function of a stochastic process. Thus, predictions of the response variable for new sample points are conditionally dependent on the measured values and *their* sample points; by unrolling the effect of the parameters of the random process, we can directly express this dependency.

Kriging is often motivated as a local modeling technique, capable of approximating or interpolating functions with multiple local extrema, and generalizes well to applications exhibiting anisotropies and trends. The stochastic prior is also viewed as a mathematically elegant mechanism to impart any available domain knowledge to the modeling technique. In 1989, Sacks et al. [Sacks et al., 1989] showed how kriging can actually be used to model processes with *deterministic* outcomes, especially in the context of computer experiments. The justification for modeling a deterministic code as a stochastic process is often that even though the response variable is deterministic, it may ‘resemble the sample path of a suitably chosen stochastic process’ [Sacks et al., 1989]. Alternatively, using a stochastic process prior can be viewed as a Bayesian approach to data analysis [Sivia, 1996], and this is the idea emphasized by most recent computer science research in Gaussian processes [Gibbs, 1997; Rasmussen, 1996]. The stochastic process can be suitably formulated to ensure that the model reproduces the same response value for repeated invocations of a given sample input (i.e., absence of random error). For instance, the Gaussian prior can be chosen so that the diagonal entries of the covariance matrix are 1, meaning that the model should interpolate the data points.

In the recent past, Gaussian processes have become popular in the statistical pattern recognition community [MacKay, 1997] and graphical models literature [Jordan (ed.), 1998]. Neal established the connection between Gaussian processes and neural networks with an infinite number of hidden units [Neal, 1996]. Such relationships allow us to take traditional learning techniques and re-express them as imposing a particular covariance structure on the joint distribution of inputs. For instance, we can take a trained neural network and mine the covariance structure implied by the weights (given mild assumptions such as a Gaussian prior over the weight space). Williams motivates the usefulness of such studies and describes common covariance functions [Williams, 1998].

Williams and Barber [Williams and Barber, 1998] describe how the Gaussian process framework can be extended to classification, where the modeled variable is categorical. Essentially, the idea is to (i) use a logistic function to conduct traditional Gaussian regression modeling, and (ii) adopt a *softmax* function to bin the logistic output into a given set of classes. This means that the logistic function uses a “latent variable” as input in its computation, since its values are not provided by the dataset.

4 Gaussian Processes for Spatial Aggregation

SAL programs construct multi-layer spatial aggregates based on specified local adjacency relations, similarity metrics, and consistency checks. We describe here how to capture the

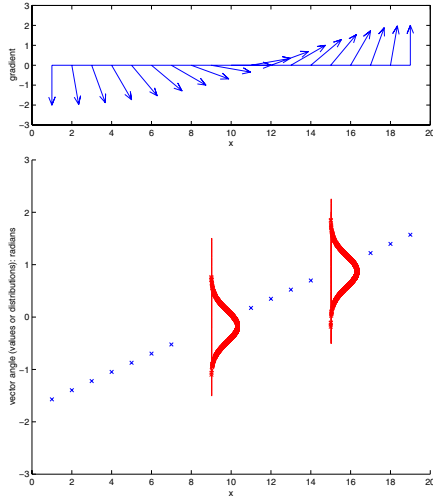


Figure 3: Modeling the reversal of gradients in a 1D field using Gaussian processes. (top) Original field. (bottom) Given measured values of gradient vector angles at specific data points (blue), the model posits that the conditional distribution of the angle at unseen data points is a Gaussian (shown in red).

qualitative behaviors of such aggregates using Gaussian processes. The essence of a Gaussian process is its covariance structure, so we focus on determining covariance structures in a SAL program. For example, in the two-layer SAL program of Sec. 2, the parameters (r, θ, d) impose a covariance structure by specifying the reach of the neighborhood graph, enforcing the similarity of angles in the vector field, and penalizing for the distance at decisions involving junctions.

4.1 Covariance Structure

We now describe how to model the covariance structure of a given SAL program. We give the mathematical framework for the case of mining a 1D field to determine if there is a reversal of gradient as we move along the spatial dimension, but essentially the same machinery applies to two and higher dimensional spaces. The basic problem is one of classifying 1D points to determine the qualitative structure of same-direction flows. Fig. 3 (top) depicts the given input field along the x dimension. As is shown, the field consists of unit vectors with different orientation. The Gaussian process approach is first to model an underlying regressed variable and then to use a logistic or softmax function to bin the output into classes. In our application, the regressed variable represents the gradient and can be simply summarized as the angle of unit vector orientation y in Fig. 3 (top). In other applications, the regressed variable could be an unobserved ‘latent’ variable. In either case, it is modeled as a function f of the input x .

First, assume f to be a Gaussian process on x , meaning that the conditional probability distribution of y given a value of x is a Gaussian. For instance, Fig. 3 (bottom) depicts measured values of y superposed with distributions of y at two unseen points. A covariance structure among the y values could, for example, capture the intuition that adjacent values of y should agree more than distant values. The goal of modeling is to

determine the extent and stringency of this neighborhood relation — one of the defining parameters of a SAL program. Specifically, we posit a process such as:

$$f(x) = \alpha + Z(x) \quad (1)$$

The idea then is to estimate a model f' of the same form as f , on the basis of a given set of k observations $\{(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)\}$. A typical choice for Z in f' is a random process with zero mean and covariance $\sigma^2 R$, where scalar σ^2 is the estimated variance and R is a matrix that captures the correlation between the inputs (i.e., the given locations). Notice that even though the input is one dimension, the size of R depends on the number of locations for which gradient measurements are available. The above model for f also includes the constant term α ; this can be estimated based on the k observations, or we can substitute more complex terms (e.g. linear), or even omit it altogether.

The functional form of R (including its parameterization) in effect defines the stochastic process and must be carefully chosen to reflect the underlying data’s fidelity or any domain-specific assumptions about local variation. The parameters of the process are then estimated using multidimensional optimization involving a suitable objective function. For instance, given the following form for R :

$$R(x_i, x_j) = e^{-\rho |x_i - x_j|^2} \quad (2)$$

the problem reduces to estimating ρ from the given data. Notice that this formulation for R implicitly enforces that the model exactly interpolate the given data points, since $R(x_i, x_i) = 1$. A common objective function for estimating ρ is to minimize the mean squared error (MSE), $E\{(f' - f)^2\}$ between f and f' . The ρ that minimizes MSE is given by the solution to the optimization problem:

$$\max_{\rho} \left(-\frac{k}{2} (\ln \sigma^2 + \ln |\mathbf{R}|) \right) \quad (3)$$

where $\mathbf{R}$ is the symmetric correlation matrix formed from R . For a new sample point x_{k+1} , a prediction for the regressed variable is given by:

$$f'(x_{k+1}) = \hat{\alpha} + \mathbf{r}^T(x_{k+1}) \mathbf{R}^{-1}(\mathbf{y} - \hat{\alpha} \mathbf{I}_k) \quad (4)$$

where $\mathbf{r}$ is the correlation vector between the response at x_{k+1} and all the other k points (derived from R), $\mathbf{I}_k$ is the identity vector of dimension k , and $\hat{\alpha}$ is the estimate of α given by:

$$\hat{\alpha} = (\mathbf{I}_k^T \mathbf{R}^{-1} \mathbf{I}_k)^{-1} \mathbf{I}_k^T \mathbf{R}^{-1} \mathbf{y} \quad (5)$$

The variance in the estimate is given by:

$$\hat{\sigma}^2 = \frac{(\mathbf{y} - \hat{\alpha} \mathbf{I}_k)^T \mathbf{R}^{-1} (\mathbf{y} - \hat{\alpha} \mathbf{I}_k)}{k} \quad (6)$$

In this case, the optimization is one-dimensional due to the presence of the single parameter ρ . With a different parameterization, we will employ multi-dimensional optimization over the entire set of hyperparameters. When dimensionality is large, the hyperparameters are estimated using MCMC methods. Once such a modeling is complete, as discussed in the previous section, we can relate a categorical class variable

to y using softmax functions. For instance, the reversal of the gradient in Fig. 3 can be captured by first using the Gaussian process model to make predictions of the gradient at untested points and then determining if (and where) a zero crossing occurs.

The above equations extend naturally to a 2D case such as that described in Sec. 2. The covariance prior has to be suitably parameterized and we also have the option of taking into account any interactions between the two dimensions (both linear and nonlinear).

4.2 Modeling Many Layers

When SAL programs consist of many layers, we need to develop a sequence of Gaussian process models, each with a suitable covariance function, which can then be superposed to yield a composite covariance function. Recall that while one could simply assess the covariance of the output field for sample values of the parameters and a given input field, the real purpose of a Gaussian process model is to express the covariance of the output *as a function of the characteristics of the input*. This is the key property that allows reasoning about closing the loop and selecting optimal samples. In addition, Gaussian process models help capture the randomness inherent in some of SAL’s computations, e.g. non-determinism in labeling, and variations due to how ties are broken for aggregation purposes. Refer again to Sec. 2 for an example of the types of operations that the covariance model must capture.

At the very bottom of the hierarchy is the input data field. For applications characterized by expensive data collection (as in the introduction), it can be advantageous to start with a *sparse* set of sample data. The Gaussian modeling approach to regression is ideal for creating *surrogate* representations of data fields from such a sparse dataset. That is, given a sparse set of samples, interpolate a dense field satisfying those values and incorporating any appropriate domain knowledge, as discussed above regarding kriging. Such surrogate functions can then be used as the starting points for qualitative analysis [Bailey-Kellogg and Ramakrishnan, 2001].

The operators in a SAL level deal with both locality (which object locations are close to which other ones, as encapsulated in a neighborhood graph) and similarity (which object features are close to which other ones, as encapsulated in metrics and predicates). For instance, in the example of Sec. 2, two points are assigned to the same pocket if they are spatially proximate and their flows converge. Here the Gaussian process is classification (or more generally, density estimation). A popular covariance structure for an n -dimensional input field captures locality:

$$R(x^{(k)}, x^{(l)}) = \zeta \prod_{i=1}^n e^{-\rho_i |x_i^{(k)} - x_i^{(l)}|^\eta} \quad (7)$$

where the expression relates the function values at positions $x^{(k)}$ and $x^{(l)}$. If $\eta \in [0, 2]$, then the covariance function will be positive definite, satisfying the normalization constraints of a *posteriori* inference.

To see how to capture similarity, consider when two sample locations are classified into the same trajectory in Sec. 2. In addition to being spatially proximate (as inferred by SAL’s

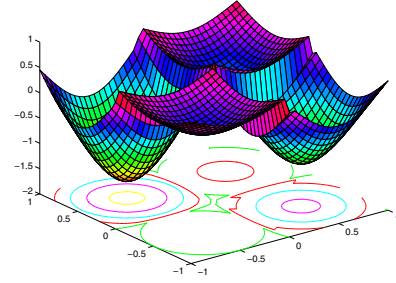


Figure 4: A 2D pocket function.

neighborhood calculations), the underlying vector fields must also be similar in direction. Expressing the covariance in terms of position alone can cause the resulting estimated hyperparameters to be misleading or difficult to interpret, as their effect is *confounded* with the underlying vector field. One solution is to artificially inflate the dimensionality, so that position and direction *together* describe the data. Besides increasing the dimensionality, this approach spells trouble for estimation using MCMC methods since significant portions of the sample space will remain unsampled and it would be difficult to assess their effects on the minimized functional. An alternative solution is to use the fact that the vector field is itself a surrogate and add a term to the covariance *outside the above structure*, capturing the contribution due to similarity in the vector field. We place a Gamma prior on this term with a shape parameter that ensures that its role is secondary to the covariance structure on position (directional similarity alone is not enough for high covariance at the output; the sample locations also must be spatially proximate). This is recognized in the statistics community as a hierarchical prior and described in detail in [Neal, 1997].

5 Experimental Results

In order to test our approach, we studied de Boor’s pocket function (see Fig. 4):

$$\begin{aligned} \alpha(\mathbf{X}) &= \cos \left(\sum_{i=1}^n 2^i \left(1 + \frac{x_i}{|x_i|} \right) \right) - 2 \\ \delta(\mathbf{X}) &= \|\mathbf{X} - 0.5\mathbf{I}\| \\ p(\mathbf{X}) &= \alpha(\mathbf{X})(1 - \delta^2(\mathbf{X})(3 - 2\delta(\mathbf{X}))) + 1 \end{aligned}$$

where $\mathbf{X}$ is the n -dimensional point $(x_1, x_2, \dots, x_n)$ at which the pocket function p is evaluated, $\mathbf{I}$ is the identity n -vector, and $\|\cdot\|$ is the L_2 norm. This function exploits the fact that the volume of a high dimensional cube is concentrated in its corners and p is designed so that it has a “dip” in each corner. It embodies many aspects of datasets like those encountered in the wireless simulation study, including multiple local extrema, non-systematic variation in the location of the pockets, and regional variation. The pocket function is also important as a benchmark for high-dimensional data exploration, where the goal is to identify the most interesting regions of the design space without necessarily conducting a (costly) global optimization over the entire design space. Data mining programs are hence required to identify the most promising regions using as few function evaluations as possible.

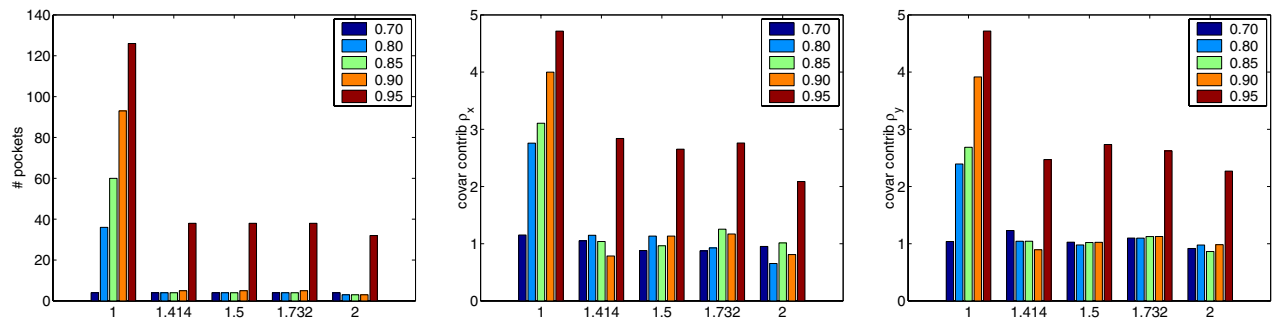


Figure 5: Modeling a SAL program to mine pockets in gradient fields. (a) Variation in number of pockets mined by the SAL program for various values of (r, θ) . (b) Covariance contribution in x dimension for various values of (r, θ) . (c) Covariance contribution in y dimension for various values of (r, θ) . In all charts, r varies by group and θ varies within group.

A SAL program to identify the number of pockets starts with some samples of the pocket function. The lowest level of modeling involves a kriging interpolation over a uniform grid (we chose size 13^n for testing). Then the approach of Sec. 2 is applied to the gradient vector field of this scalar field: the second level bundles points into curves, and the third aggregates these into flows. Each convergent flow represents one pocket. One could mine the covariance structures for each layer separately; we unfold these mappings here to obtain a single covariance structure summarizing all three layers. This is because the structure (esp. the contributions of each dimension) is easiest to interpret in terms of the original spatial field.

We conducted a parameter sweep over (r, θ, d) as:

$$\begin{aligned} r &\in \{1, \sqrt{2}, 1.5, \sqrt{3}, 2\} \\ \theta &\in \{0.7, 0.8, 0.85, 0.9, 0.95\} \\ d &\in \{0.01, 0.02, 0.03, 0.04, 0.05\} \end{aligned}$$

and used Neal’s Bayesian modeling software [Neal, 1997] to construct Gaussian process classifiers for the flow classes. Covariance contributions in the ρ terms (Eq. 7) from both the dimensions was estimated using hybrid Monte Carlo (aggressive schemes to evolve the system state by adding higher order terms). This procedure uses a leapfrog scheme to suppress random walk behavior by selective iteration between Gibbs sampling scans and latent value updates.

Our results indicated a strong positive correlation between the x and y covariance contributions, bringing out the symmetry in the underlying SAL computations. The number of pockets mined was constant across the values of d (other parameters fixed), and one of the goals of our study was to determine if this negligible effect of d is captured in the covariance structures. (The effect of d would actually be more pronounced in other spatial fields but not so much in the pocket function due to the inherent symmetry.) Parameters r and θ produced the most variation in the covariance contributions with $\theta = 0.95$ causing an abrupt jump in the number of mined pockets. This is due to the rather stringent limit imposed on vector similarity arising from the nonlinearity of the cosine metric. Fig. 5 summarizes the results for a 2D pocket function, where we have averaged the covariance contributions across all d s, for given r and θ .

As the number of pockets increases (Fig. 5(a)), the covariance contributions increase (Fig. 5(b,c)) approximately

quadratically. In other words, as the underlying latent function varies rapidly along the given dimensions, we cannot stray “too far” away from a given sample point when making predictions at test points. The reciprocal of the covariance scale term is often referred to the *characteristic length* of a dimension. This gives an estimate of “how far” a given dimension’s effect holds. When only four pockets are mined, the characteristic length is about 1, meaning pockets occupy a width of 1×1 (exactly one fourth of the total space). As more pockets are mined, the characteristic length drops to about 0.4. It is also interesting to note that the abrupt jump in the number of pockets for $\theta = 0.95$ is reflected by a similar increase in the covariance contributions for this value. Essentially, vector and edge directions have to be so similar that few long “runs” can be aggregated as streamlines. This brings out the capability of the Gaussian process approach to capture the essentials of a spatial aggregation algorithm.

6 Discussion

This paper has demonstrated a novel Gaussian process approach to modeling the qualitative behavior of SAL programs; in contrast to much of the literature where Gaussian processes are used for pattern classification and regression [Rasmussen, 1996; Gibbs, 1997], our work takes *existing* data mining algorithms and recasts them in terms of Gaussian priors. To the best of our knowledge, this is the first study to completely model a qualitative data mining algorithm in terms of a process framework, summarizing the transformation from data to higher-level aggregates. This is an important step in firmly establishing a probabilistic basis for spatial aggregation computations. The modeling undertaken here, while expensive, is justifiable for studies such as the wireless system characterization described in the introduction.

There are several immediate gains from the work presented here; due to space limitations we only mention them briefly. First, the Gaussian process model can characterize experimental design criteria such as entropy as a functional w.r.t. the input space, allowing us to use the mined covariance structure to focus sampling at the most informative points (e.g., see [Bailey-Kellogg and Ramakrishnan, 2001]). It is important to note, however, that the approach taken in [Bailey-Kellogg and Ramakrishnan, 2001] only addresses the lowest

levels of a hierarchy and is unable to reason about higher-level, more abstract processes of redescription and aggregation as is done here. Second, Gaussian process models allow us to study the effects of different SAL parameters for a given class of datasets, e.g. the inference above of the negligible role of d in the mining process. Finally, it allows us to take algorithms that function in differing ways (and using different sets of parameters) and places them on a common footing, namely the language of covariance structures. This means that we can reason about the applicability of different algorithms by studying the constraints they impose on spatial locality and field similarity.

Gaussian processes have recently been linked to kernel-based methods, as used in support vector machines [Cristianini and Shawe-Taylor, 2000]; we intend to explore this connection in future work. Kernel-based methods are attractive in their promise to overcome the curse of dimensionality by the use of nonlinear projections, a facet that is of critical importance for mining data from large parameter sweeps. As the need for data mining in computational science gains prominence, process models will be crucial to achieve effective utilization of data for mining purposes.

Acknowledgements

The authors thank Feng Zhao and Layne Watson for helpful comments. This work is supported by US NSF grants EIA-9974956, EIA-9984317, and EIA-0103660.

References

- [Bailey-Kellogg and Ramakrishnan, 2001] C. Bailey-Kellogg and N. Ramakrishnan. Ambiguity-Directed Sampling for Qualitative Analysis of Sparse Data from Spatially Distributed Physical Systems. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 43–50, 2001.
- [Bailey-Kellogg and Zhao, 1999] C. Bailey-Kellogg and F. Zhao. Influence-Based Model Decomposition. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, pages 402–409, 1999.
- [Bailey-Kellogg and Zhao, 2001] C. Bailey-Kellogg and F. Zhao. Influence-Based Model Decomposition for Reasoning about Spatially Distributed Physical Systems. *Artificial Intelligence*, Vol. 130(2):pages 125–166, 2001.
- [Bailey-Kellogg et al., 1996] C. Bailey-Kellogg, F. Zhao, and K. Yip. Spatial Aggregation: Language and Applications. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 517–522, 1996.
- [Cohn et al., 1996] D.A. Cohn, Z. Ghahramani, and M.I. Jordan. Active Learning with Statistical Models. *Journal of Artificial Intelligence Research*, Vol. 4:pages 129–145, 1996.
- [Cristianini and Shawe-Taylor, 2000] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.
- [Denzler and Brown, 2002] J. Denzler and C.M. Brown. Information Theoretic Sensor Data Selection for Active Object Recognition and State Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24(2):pages 145–157, Feb 2002.
- [Gibbs, 1997] M.N. Gibbs. *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, University of Cambridge, 1997.
- [Huang and Zhao, 1999] X. Huang and F. Zhao. Relation-Based Aggregation: Finding Objects in Large Spatial Datasets. In *Proceedings of the 3rd International Symposium on Intelligent Data Analysis*, 1999.
- [Jordan (ed.), 1998] M.I. Jordan (ed.). *Learning in Graphical Models*. MIT Press, 1998.
- [Journel and Huijbregts, 1992] A.G. Journel and C.J. Huijbregts. *Mining Geostatistics*. Academic Press, New York, 1992.
- [MacKay, 1992] D.J. MacKay. Information-Based Objective Functions for Active Data Selection. *Neural Computation*, Vol. 4(4):pages 590–604, 1992.
- [MacKay, 1997] D.J. MacKay. Gaussian Processes: A Replacement for Supervised Neural Networks? In *Lecture Notes of Tutorial at Neural Information Processing Systems (NIPS'97)*, 1997.
- [Menzefricke, 2000] U. Menzefricke. Hierarchical Modeling with Gaussian Processes. *Communications in Statistics*, Vol. 29(4):pages 1089–1108, 2000.
- [Neal, 1996] R.M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, NY, 1996. Lecture Notes in Statistics No. 118.
- [Neal, 1997] R.M. Neal. Monte Carlo Implementations of Gaussian Process Models for Bayesian Regression and Classification. Technical Report 9702, Department of Statistics, University of Toronto, Jan 1997.
- [Ordóñez and Zhao, 2000] I. Ordóñez and F. Zhao. STA: Spatio-Temporal Aggregation with Applications to Analysis of Diffusion-Reaction Phenomena. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00)*, pages 517–523, 2000.
- [Ramakrishnan and Bailey-Kellogg, 2002] N. Ramakrishnan and C. Bailey-Kellogg. Sampling Strategies for Mining in Data-Scarce Domains. *IEEE/AIP Computing in Science and Engineering*, Vol. 4(4):pages 31–43, July/Aug 2002.
- [Rasmussen, 1996] C.E. Rasmussen. *Evaluation of Gaussian Processes and other Methods for Non-Linear Regression*. PhD thesis, University of Toronto, 1996.
- [Sacks et al., 1989] J. Sacks, W.J. Welch, T.J. Mitchell, and H.P. Wynn. Design and Analysis of Computer Experiments. *Statistical Science*, Vol. 4(4):pages 409–435, 1989.
- [Sivia, 1996] D.S. Sivia. *Data Analysis: A Bayesian Tutorial*. Oxford University Press, 1996.
- [Verstak et al., 2002] A. Verstak, N. Ramakrishnan, K.K. Bae, W.H. Tranter, L.T. Watson, J. He, C.A. Shaffer, and T.S. Rappaport. Using Hierarchical Data Mining to Characterize Performance of Wireless System Configurations. Technical Report cs.CE/0208040, Computing Research Repository, Aug 2002.
- [Williams and Barber, 1998] C.K.I. Williams and D. Barber. Bayesian Classification with Gaussian Processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20(12):pages 1342–1351, Dec 1998.
- [Williams, 1998] C.K.I. Williams. Prediction with Gaussian Processes: From Linear Regression to Linear Prediction and Beyond. In M.I. Jordan, editor, *Learning in Graphical Models*, pages 599–621. MIT Press, Cambridge, MA, 1998.
- [Yip and Zhao, 1996] K.M. Yip and F. Zhao. Spatial Aggregation: Theory and Applications. *Journal of Artificial Intelligence Research*, Vol. 5:pages 1–26, 1996.

Qualitatively Faithful Quantitative Prediction

Dorian Šuc, Daniel Vladušič, Ivan Bratko

Faculty of Computer and Information Science, University of Ljubljana
Tržaška 25, 1000 Ljubljana, Slovenia
{dorian.suc, ivan.bratko, daniel.vladusic}@fri.uni-lj.si

Abstract

In this paper we describe a case study in which we applied an approach to qualitative machine learning to induce, from system's behaviour data, a qualitative model of a complex, industrially relevant mechanical system (a car wheel suspension system). The induced qualitative model enables nice causal interpretation of the relations in the modelled system. Moreover, we also show that the qualitative model can be used to guide the *quantitative* modelling process leading to numerical predictions that may be considerably more accurate than those obtained by state-of-the-art numerical modelling methods. This idea of combining qualitative and quantitative machine learning for system identification is in this paper carried out in two stages: (1) induction of qualitative constraints from system's behaviour data, and (2) induction of a numerical regression function that both respects the qualitative constraints and fits the training data numerically. We call this approach Q^2 learning, which stands for Qualitatively faithful Quantitative learning.

1 Introduction

It is generally accepted that qualitative models are easier to understand and reason about than quantitative models. Qualitative models thus provide a better basis for the explanation of phenomena in a modelled system than numerical models. In this paper we describe a case study in which we applied an approach to qualitative machine learning to induce, from system's behaviour data, a qualitative model of a complex, industrially relevant mechanical system (a car wheel suspension system). The induced qualitative model enables nice causal interpretation of the relations among the variables in the system. This is precisely as one would expect from a qualitative model. More surprisingly, however, we also show in this case study that the qualitative model can be used to guide the *quantitative* modelling process that may lead to numerical predictions that are considerably more accurate than those provided by state-of-the-art numerical modelling methods.

Thus the main message of this paper is that a combination of methods for qualitative and quantitative system identifica-

tion has good chances to attain significant improvements over numerical system identification techniques, including techniques of numerical machine learning methods, such as regression trees [Breiman *et al.*, 1984] and model trees [Quinlan, 1992]. The potential improvements are in two respects: first, the predictions are qualitatively consistent with the properties of the modelled system, and in addition they are also numerically more accurate.

This idea of combining qualitative and quantitative machine learning for system identification is in this paper carried out in two stages. First, induce qualitative constraints from system's behaviour data (training data) with program QUIN (overviewed in Section 3). Second, induce a numerical regression function that both respects the qualitative constraints and fits well the training data numerically (called Qualitative to Quantitative transformation or Q2Q for short, described in Section 4). We call this approach Q^2 learning, which stands for Qualitatively faithful Quantitative learning. To underline the importance of qualitative fidelity, we illustrate in Section 2 some problems that numerical learners typically have in respect of qualitative consistency. In Section 5 we present the case study in applying Q^2 to the chosen problem of modelling car suspension.

There are several approaches to learning qualitative models from numerical data that may support alternative approaches to Q^2 learning. These include the program QMN [Džeroski and Todorovski, 1995], QSI [Say and Kuru, 1996], SQUID [Kay *et al.*, 2000], and QOPH [Coghill *et al.*, 2002].

2 Qualitative difficulties of numerical learning

Consider a simple container of cylindrical shape and a drain at the bottom. If we fill the container with water, the water drains out. Water level monotonically decreases, until it reaches zero. Suppose we fill the container with water, and measure initial outflow $\Phi_0 = \Phi(t_0)$ and the time behaviour of water level $h(t)$. Since this is a rather simple behaviour, one would naturally expect that machine learning techniques should be able to fairly well predict time behaviour of water level if enough learning examples are given. Quite surprisingly, even in such simple cases, the usual numerical predictors can give strange and qualitatively unacceptable predictions. We illustrate the problems with a simple experiment, using well-known techniques for numerical prediction: model trees and locally weighted regression.

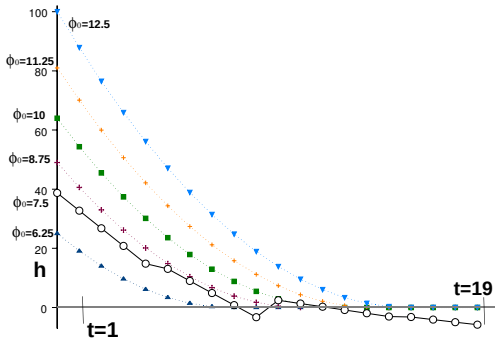


Figure 1: M5 predictions of water outflow: empty circles are M5 predictions of level $h(t)$ on a test set with $\Phi_0 = 7.5$. Other dots are the the learning examples. Note that M5 predicts that water level *increases* at time $t=10$.

In our experiment we used container outflow simulation data to evaluate how different numerical predictors learn the time behaviour of water level. The outflow from a container is $\Phi(t) = c\sqrt{h(t)}$, where c is a parameter depending on the area of the drain. For the simulation we used Euler integration $h(t + \Delta t) = h(t) - \Phi(t)\Delta t$, where $\Delta t = 0.1$ seconds and $c = 1.25$. We used six example sets, generated with different initial water levels and initial outflows $\Phi_0 = 5 + ic$, $i = 1, \dots, 6$. An example set has 20 examples $(t, h(t), \Phi_0)$ corresponding to 19 seconds of simulation.

We used the Weka [Witten and Frank, 2000] implementations of locally weighted regression [Atkeson *et al.*, 1997] (LWR, for short), and M5 regression and model trees [Quinlan, 1992] to learn the time behaviour of level given the initial outflow, i.e. $h(t) = f(t, \Phi_0)$. One example set was used as a test set and other five sets (100 examples) for learning. When the test set was the set with $\Phi_0 = 7.5$, M5 with the default parameters induced a model tree with 9 leaves. Figure 1 shows the learning examples and the M5 prediction of level $h(t)$ on a test set. Note that M5 predicts that water level *increases* at time $t=10$. The same happens if we change the pruning parameter. This is of course qualitatively unacceptable as water level can never increase. Also, there are no learning examples where water level increases.

One might think that this is an isolated weird case or M5 bug. But it is not. LWR makes a similar *qualitative error* on the test set with $\Phi_0 = 11.25$, when it predicts that water level increases at $t=5$. Of course, LWR predictions depend on its parameter k , i.e. the number of neighbors used, but often the appropriate k that doesn't give qualitative errors on one container, gives qualitative errors when learning from similar data but with different area of the drain. Often, the errors are even more obvious if we make predictions at the edges of the space covered by learning examples, i.e. using as test set $\Phi_0 = 6.25$ or $\Phi_0 = 12.5$. As one would expect, regression trees make similar qualitative errors.

We believe that other numerical predictors make similar *qualitative errors*, at least in more complex domains. This might be quite acceptable in applications where we just want to minimize *numerical* prediction errors. But often it is also important to respect qualitative relations that are either given

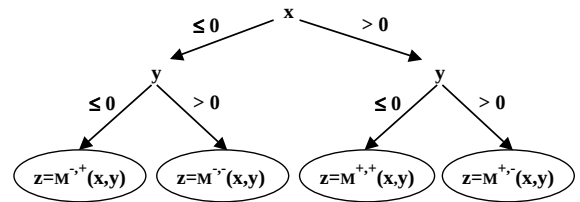


Figure 2: A qualitative tree induced from a set of examples for the function $Z = X^2 - Y^2$. The rightmost leaf, applying when attributes X and Y are positive, says that Z is strictly increasing in its dependence on X and strictly decreasing in its dependence on Y .

in advance or hidden in the data. Ignoring them results in clearly unrealistic predictions that a domain expert would not approve. The idea of this paper is to use such qualitative relations, either given or induced from data, not only to avoid qualitatively incorrect predictions, but also to improve the accuracy of numerical prediction.

3 Qualitative data mining with QUIN

QUIN (Qualitative Induction) is a learning program that looks for qualitative patterns in numerical data [Šuc, 2001; Šuc and Bratko, 2001; 2002]. Induction of the so-called qualitative trees is similar to induction of decision trees. The difference is that in decision trees the leaves are labelled with class values, whereas in qualitative trees the leaves are labelled with what we call qualitatively constrained functions.

Qualitatively constrained functions (QCFs for short) are a kind of monotonicity constraints that are widely used in the field of qualitative reasoning. A simple example of QCF is: $Y = M^+(X)$. This says that Y is a monotonically increasing function of X . In general, QCFs can have more than one argument. For example, $Z = M^{+,-}(X, Y)$ says that Z monotonically increases in X and decreases in Y . We say that Z is positively related to X and negatively related to Y . If both X and Y increase, then according to this constraint, Z may increase, decrease or stay unchanged. In such a case, a QCF cannot make an unambiguous prediction of the qualitative change in Z .

QUIN takes as input a set of numerical examples and looks for qualitative patterns in the data. More precisely, QUIN looks for regions in the data space where monotonicity constraints hold. Such a set of qualitative patterns are represented in terms of a qualitative tree. As in decision trees, the internal nodes in a qualitative tree specify conditions that split the attribute space into subspaces. In a qualitative tree, however, each leaf specifies a QCF that holds among the input data that fall into that leaf. For example, consider a set of data points (X, Y, Z) where $Z = X^2 - Y^2$ possibly with some noise added. When QUIN is asked to find in these data qualitative constraints on Z as a function of X and Y , QUIN generates the qualitative tree shown in Figure 2. This tree partitions the data space into four regions that correspond to the four leaves of the tree. A different QCF applies in each of the leaves. The tree describes how Z qualitatively depends on X and Y .

QUIN constructs a tree in the top-down greedy fashion,

similarly to decision tree induction algorithms. A more elaborate description of the QUIN algorithm and its evaluation on a set of artificial domains is given elsewhere [Šuc, 2001; Šuc and Bratko, 2001]. QUIN has been applied to qualitative reconstruction of human control strategies [Šuc, 2001], and to reverse engineer a complex industrial controller for gantry cranes [Šuc and Bratko, 2002].

4 Q2Q transformation

In this section we describe the qualitative-to-quantitative transformation (Q2Q for short). Given a set of numerical data and a qualitative tree, Q2Q attempts to find a regression function that fits the data well numerically, and also respects the qualitative constraints in the tree. We say that such a regression function is *qualitatively consistent*. In fact, Q2Q finds a qualitatively consistent regression function with good fit for each leaf in the tree separately. The overall regression function is then obtained by gluing together the regression functions for the leaves.

The Q2Q procedure is as follows. First, we partition the learning examples according to the leaves of the qualitative tree. These subsets are then used for learning qualitatively consistent functions of the corresponding leaves. Let us focus on learning a qualitatively consistent function for some particular leaf. Suppose we have a leaf with the qualitative constraint $y = M^+(x)$. We then have to find some monotonically increasing function that fits the data well. One straightforward solution, used in Q2Q, is to divide the range of variable x with a number of equidistant points (i.e. $\{x_1, x_2, \dots, x_n\}$) in which we learn from the given data the function values y . The result is a set of pairs $\{(x_1, y_1), \dots, (x_n, y_n)\}$ that defines a piece-wise linear function which can be easily checked for compliance with the given qualitative constraint. This procedure can be generalized to qualitative constraints of any dimension.

In our implementation the function values y_i were determined with a standard version of locally weighted regression (LWR) [Atkeson *et al.*, 1997], which used Gaussian weighting function. Therefore, the two parameters of the transformation were the number of equidistant points per dimension ($N \in \{3, 4, 5, 6\}$) and the kernel size of the Gaussian weighting function ($K \in \{0.3, 0.4, \dots, 0.8\}$). All possible combinations of these two parameters ($4 \times 6 = 24$) define the space of all candidate piece-wise linear functions for each leaf. These sets of candidate functions are exhaustively searched by Q2Q to find functions that satisfy given qualitative constraints. For each leaf, Q2Q selects among these qualitatively consistent piece-wise linear functions one that has best fit with the data that fall into this leaf. Quality of the fit is measured with root mean squared error, *RMSE* for short. It is theoretically possible that none of the candidate functions is qualitatively consistent with the QCF in the leaf. In such a case the Q2Q procedure would fail to find a qualitatively consistent regression function. Although in our experiments this never happened, we are working on an improved version of Q2Q that is guaranteed to find a qualitatively consistent regression function.

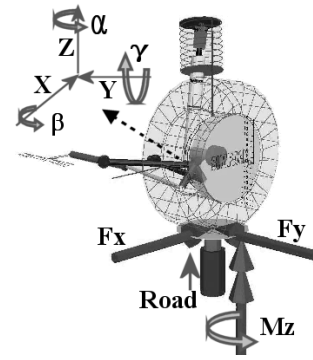


Figure 3: Intec wheel model: wheel position is given by x , y and z coordinates of the wheel center, and rotation angles about axes x , y and z , called enforced wheel-spin angle γ , camber β and toe angle α , respectively. These are affected by horizontal forces F_x and F_y , elevation of the road R and rotational moment M_z that act upon the tyre.

5 Intec case study

5.1 Intec wheel model

In this section we present an application of Q^2 learning to the modelling of car wheel suspension system. This is a complex mechanical system of industrial relevance. The model and simulation software used in this experiment were provided by a German car simulation company Intec. The main role of the application in this paper is to provide a controlled experiment to assess the potentials of Q^2 learning on a modelling problem of industrial complexity. However, although the target model was already known and developing such a model was not an issue of practical relevance, initially this case study was nevertheless motivated by a practical objective. Namely, the complexity of Intec's model is so high that it cannot be simulated on the present simulation platform in real time. Therefore the practical objective of the application of Q^2 learning was to speed up the wheel simulation. The goal thus was to obtain a simplified wheel model that would still be sufficiently accurate and at the same time significantly simpler than the original model to allow real-time simulation. Indeed, the simplified model obtained with Q^2 is computationally trivial compared with the original model.

The Intec wheel model (shown in Figure 3) is a multi-body model of a front wheel suspension built in compliance with the physical model assuming no car-body movement and no wheel-spin. In fact, the suspension system is modelled as if the car-body is fixed. Wheel position is given by x , y and z coordinates of the wheel center. Because the flexible joints in multi-body suspension system that links the wheel to the car-body allow several levels of displacements, rotation angles about axes x , y and z are also measured. These are called enforced wheel-spin angle γ , camber β and toe angle α , respectively.

The multi-body simulation software Simpack [Intec, 2002] was used to set up the model and to generate simulation traces. During simulation, a number of elements are acting upon the tyre: two horizontal forces F_x and F_y , vertical movement (measured as elevation of the road R) and rota-

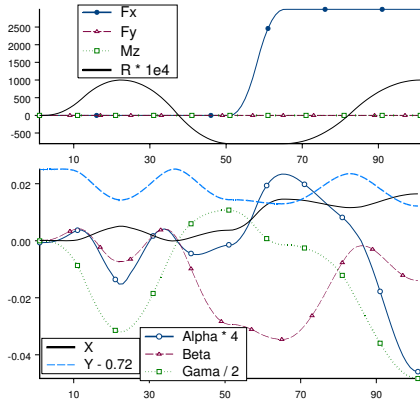


Figure 4: A typical simulation trace of the Intec wheel model: the input variables are on the upper graph, the output variables (except z that changes the same as road R) are on the lower graph. x -axis is time in steps $dt=0.7$ seconds. Note the complex behaviour of the output variables resulting from changes in F_x and road R .

tional moment M_z . For example, F_x is acting upon the tyre when braking, F_y when driving through corners (centripetal force) and rotational moment M_z when parking the car.

During the simulation, input and output variables are logged to a file called simulation trace. We used traces of wheel simulation with different trajectories of input variables. Each trace lasted for 70 seconds, and was sampled with $dt=0.7$ seconds. In this way a trace gives 100 examples, each example contains 10 values, corresponding to the values of four input and six output variables at a given time. Figure 4 shows a typical simulation trace. It should be noted that all these traces correspond to very slow changes of input variables, and as a result the traces are illustrative mainly of the kinematics of the mechanism, but not also of its dynamics.

5.2 Details of experiments

The experiments reported in this paper were done using a black-box approach. We did not use any knowledge of the model. The simulation traces were provided by our partners from Czech Technical University in the European project Clockwork.

In all the experiments we used 7 traces for learning with the same road profile as in the trace of Figure 4. In the first learning trace all other three input variables were zero. In the next three traces two of the other three input variables were zero and one other variable (F_x , F_y or M_z) was changing. Figure 4 shows one such trace. The remaining three traces were similar, but the trajectory of the changing variable was different, i.e. it first increased, stayed unchanged for 20 seconds, and then slowly decreased to zero. Each trace gives 100 examples, giving altogether 700 learning examples with 10 variables.

The task was to learn each of the six output variables as a function of input variables. In this way we have six learning problems, where an output variable is the class and the input variables are the attributes. For example, angle α was learned as $\alpha = f(R, F_x, F_y, M_z)$.

The prediction accuracy was tested on 7 test traces. All the

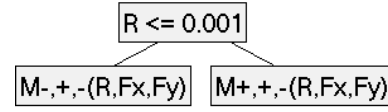


Figure 5: Induced qualitative tree for x wheel center position.

test traces have the same road profile as the traces used for learning, but different profiles of other three input variables, i.e. F_x , F_y and M_z . In the first trace all of the three input variables change similar as F_x in the trace in Figure 4. This trace was recommended as the most critical test trace by the domain expert, who considered it far more difficult (all 4 input variables change) than other 6 test traces where one or two input variables were always zero.

5.3 Inducing a qualitative wheel model with QUIN

As described above, QUIN was used to induce a qualitative tree for each of the six output variables, where the input variables were the attributes. All of the induced qualitative trees had over 99 % consistency on the learning set of examples. We say that a QCF is consistent with an example if the QCF's qualitative prediction of the dependent variable does not contradict the direction of change in the example. The level of consistency of a qualitative tree with the examples is the percentage of the examples with which the tree is consistent. Consistency of 99% indicates that the induced qualitative model fits the data nearly perfectly.

The simplest qualitative tree was induced for translation in the z -axis. This tree only has one leaf with QCF $z = M^+(R)$. This tree has a simple and obvious explanation. It says that z changes in the direction of the road change. If road increases then z increases, i.e. the wheel center moves upwards.

Qualitative trees for translations in x and y axes are a bit more complicated. Since they have similar explanations we will present just the qualitative tree for x translations, given in Figure 5. Note that x is measured in the opposite direction as usual, i.e. positive x means wheel center moving in the direction of car driving backwards. Both leaves of the tree have the same qualitative dependence on F_x and F_y , but differ in qualitative dependence on road R . The qualitative tree says that x is positively related to force F_x that acts in the direction of x . Obviously, wheel center position x changes (wheel moves backward or forward) in the direction of force in x direction. Second, x is negatively related to force F_y . This means that if we push the wheels together (we apply force in the y direction), the wheels will move forward (x position decreasing). This is not so obvious, but can be understood if we consider the usual mechanics of wheel suspension. The qualitative dependence on road R is a bit more complicated. The qualitative tree of Figure 5 says that x is negatively related to R when $R \leq 0.001$. Otherwise x is positively related to R . When R increases from its minimum to its maximum, x will first decrease and then increase, i.e. the wheel center will first move forward and then backward.

Rotations about axes x , y and z are measured by enforced wheel-spin γ , camber β and toe angle α , respectively. For enforced wheel-spin γ , QUIN induced a simple one-leaf tree that says $\gamma = M^{-,+,-}(R, F_x, F_y)$. Note that γ changes in the

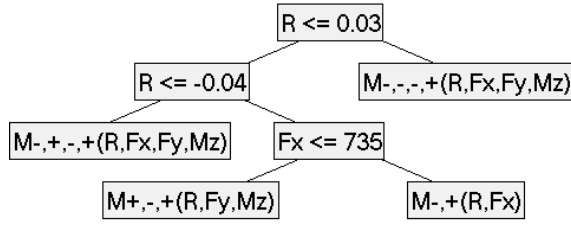


Figure 6: Induced qualitative tree for toe angle α .

direction of the tyre rotation when driving forward. Consider for example the dependence of γ on force F_x that is positive during braking. Since γ is negatively related to F_x , increasing F_x causes γ to decrease, i.e. during braking enforced wheel spin angle changes in the direction of the tyre rotation. For camber angle β QUIN induced a qualitative tree that is similar to qualitative trees for x and y translations.

The toe angle α , i.e. the rotation about z -axis is effected by all input variables and is the most complicated. The induced tree is given in Figure 6. We will omit explanation of this qualitative tree, since it requires understanding of the flexible nature of the multi-body suspension system that links the wheel to the car-body.

These qualitative trees give a good explanation of wheel suspension system behaviour. Moreover, they provide a qualitative model of wheel suspension system that enables qualitative simulation. In this way, they enable to predict all possible qualitative changes of output variables over an arbitrary time interval given qualitative changes of all or some input variables. This qualitative model also enables to improve numerical predictions.

5.4 Qualitative correctness of numerical predictors

In this section we illustrate why Q^2 learning may have an advantage over the usual numerical predictors. Figure 7 shows α predicted with M5 model tree, LWR and LWR with optimized parameters, on the most critical test trace, where all the input variables are changing simultaneously. The figure shows that both M5 and LWR sometimes make large errors. Moreover these errors are not only numerical, but also *qualitative*. Consider for example the LWR predictions at the beginning of the trace. Here the predicted α is decreasing, but the correct α is increasing. This error could be avoided by considering the induced qualitative tree for α given in Figure 6. Since at the beginning of the test trace road R is near zero and increasing, and all other input variables are zero, the second leftmost leaf of the qualitative tree applies. Its QCF $\alpha = M^+, -, -, +(R, F_y, M_z)$ requires increasing α since road R is increasing.

As can be observed in Figure 7, M5 and LWR often make qualitative errors. Q^2 predictions are qualitatively correct. The use of (induced) qualitative model enables Q^2 to better generalize in the areas sparsely covered by the training examples, resulting in better numerical accuracy.

5.5 Numerical accuracy of the induced models

Here we compare the numerical accuracy of the Weka implementations of LWR, M5 model trees and Q^2 learning. All the

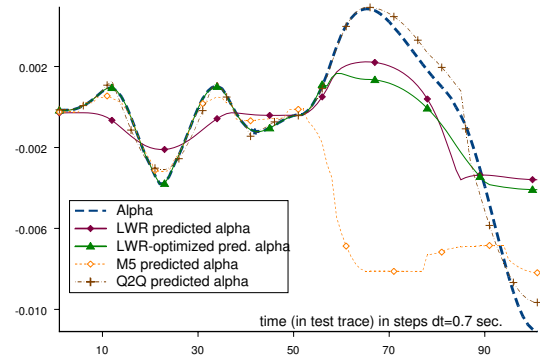


Figure 7: LWR, LWR with optimized parameters, M5 and Q^2 predictions of α on the most critical test trace. With each method, α at time step t_i (on x -axis) was predicted according to the values of input variables at time t_i in the test trace.

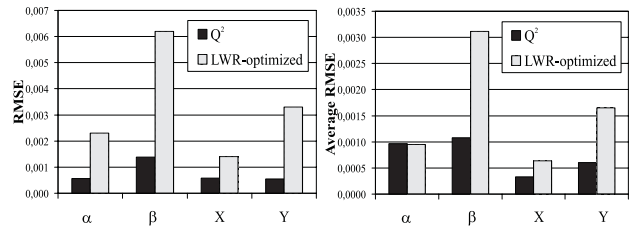


Figure 8: Comparing accuracy (measured with $RMSE$) of Q^2 and LWR with optimized parameters: the left graph compares $RMSE$ on the most difficult test trace and the right graph shows $RMSE$ on the remaining test traces.

methods learned from 7 learning traces (also used for learning of qualitative trees) and were tested against 7 test traces described in Section 5.2. The test results are divided in two groups. The first group consists of results on a single test trace. This trace was recommended as the most critical test trace by the domain expert, who considered it far more difficult (all 4 input variables are changing simultaneously) than the 6 test traces in the second group.

Learning of qualitatively consistent functions in the leaves was performed as described in Section 4. The best fitting regression functions were then taken and glued together into the overall induced numerical model. We compared the accuracy of our Q^2 method with LWR and M5. The parameters of LWR were optimized for each output variable (α , β , x and y) according to the $RMSE$ criterion, through internal cross-validation on the training set. When experimenting with M5, we noticed that it was grossly inferior both in terms of qualitative acceptability as well as numerical error. Attempts at optimizing M5's parameters did not help noticeably.

Figure 8 gives the prediction accuracy for variables α , β , x and y . The predictions of the remaining variables γ and z were not much affected by qualitative constraints. The results on the most difficult test trace (left graph in Figure 8) show that even our simple Q2Q method improves the numerical prediction on all the variables (compared to LWR). Results on the second test trace group are given on the right graph in Figure 8. As these traces were more similar to the learning traces, the improvement of Q^2 over LWR is smaller.

6 Discussion

In this paper we introduced a new approach to machine learning in numerical domains, which we call Q^2 learning (qualitatively faithful quantitative learning). This combines the induction of qualitative properties from numerical data and numerical regression that respects the induced qualitative properties. We showed by an experimental case study that Q^2 learning may lead to the following advantages compared to the usual numerical learning:

(1) Induced models tend to be *qualitatively* consistent with the data and therefore have better chances to correspond to the qualitative mechanisms in the domain of modeling. For example, if the amount of water in a container is decreasing, the level of water cannot be increasing. This is important with respect to the interpretation of induced models and explanation of phenomena in the domain based on these models.

(2) Qualitative consistency of induced models with learning data also affects the accuracy of the model's *numerical* predictions: numerical accuracy may be considerably improved. This is illustrated by the experimental results.

In respect of numerical prediction accuracy, in our case study Q^2 overall outperformed all competing numerical learners. Among these, locally weighted regression (LWR) with optimized parameters (through internal cross validation on the training set) performed best in terms of mean squared error. However its performance may sharply degrade under more difficult circumstances. Consider LWR-optimized performance on the the most difficult test set (Figure 7). It achieves excellent accuracy on the first part of this trace which is similar to data in the training sets. LWR-optimized accuracy there is actually better than that of Q^2 . However, problems begin for LWR in the second part of this trace where the input variables start to deviate considerably from the training data, and LWR's predictive error increases sharply. In this part of the trace, Q^2 manages to largely preserve qualitative consistency with the true results, and maintains the numerical accuracy at a comparable level as in the area densely populated by training examples.

LWR-optimized was the best among standard numerical learners, and therefore our presentation of experimental results largely concentrated on comparison between Q^2 and LWR. The performance of M5 was grossly inferior both in terms of qualitative acceptability as well as numerical error. Optimizing M5's parameters did not help noticeably.

It should be noted that qualitatively faithful regression as carried out by the Q2Q program is actually inferior to LWR as a regression method. Struggling to satisfy qualitative consistency, Q2Q is limited to piece-wise linear regression with a small number of linear segments. This numerical inferiority of Q2Q usually turns out to be more than compensated by preserving qualitative consistency.

In this paper, qualitative constraints for Q2Q were induced from training data with QUIN. Alternatively, such constraints can be defined directly by a domain expert. In such a case, the Q^2 learning can be viewed as an approach that enables the use of expert's qualitative knowledge in system identification.

Among the limitations of our realization of Q^2 , the primitive numerical regression method in Q2Q should be noted.

This method allows sharp changes in variable values (discontinuities in the variables' derivatives) at the borders between leaves of a qualitative tree. Future work should include a method for smoothing such discontinuities.

Acknowledgments

The work reported in this paper was partially supported by the European Fifth Framework project Clockwork and the Slovenian Ministry of Education, Science and Sport. We thank A. Eichberger of Intec, for providing the wheel suspension model and the Simpack simulation software for this study, and for helpful notes on the model. M. Valašek and P. Steinbauer from Czech Technical University also helped in the Intec case study.

References

- [Atkeson *et al.*, 1997] C.G. Atkeson, A.W. Moore, and S.A. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.
- [Breiman *et al.*, 1984] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, California, 1984.
- [Coghill *et al.*, 2002] G.M. Coghill, Garrett S.M., and R.D. King. Learning qualitative models in the presence of noise. In *Proceedings of the QR'02 Workshop on Qualitative Reasoning*, 2002. Sitges, Spain.
- [Džeroski and Todorovski, 1995] S. Džeroski and L. Todorovski. Discovering dynamics: from inductive logic programming to machine discovery. *J. Intell. Information Syst.*, 4:89–108, 1995.
- [Intec, 2002] INTEC GmbH. Simpack software. 2002. www.simpack.com.
- [Kay *et al.*, 2000] H. Kay, B. Rinner, and B. Kuipers. Semi-quantitative system identification. *Artificial Intelligence*, 119:103–140, 2000.
- [Quinlan, 1992] J.R. Quinlan. Learning with continuous classes. In *Proc. 5th Australian Joint Conf. on Artificial Intelligence*, pages 343–348, 1992. World Scientific.
- [Say and Kuru, 1996] A.C.C. Say and S. Kuru. Qualitative system identification: deriving structure from behavior. *Artificial Intelligence*, 83:75–141, 1996.
- [Šuc and Bratko, 2001] D. Šuc and I. Bratko. Induction of qualitative trees. In L. De Raedt and P. Flach, editors, *Proc. 12th European Conference on Machine Learning*, pages 442–453. Springer, 2001. Freiburg, Germany.
- [Šuc and Bratko, 2002] D. Šuc and I. Bratko. Qualitative reverse engineering. In *Proc. 19th Int. Conf. on Machine Learning*, pages 610–617. Morgan Kaufmann, 2002.
- [Šuc, 2001] D. Šuc. *Machine reconstruction of human control strategies*. PhD thesis, Faculty of Computer and Information Sc., University of Ljubljana, Slovenia, 2001.
- [Witten and Frank, 2000] I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, chapter 8, pages 265–320. Morgan Kaufmann, San Francisco, 2000.

REASONING ABOUT ACTIONS AND CHANGE

Compiling Control Knowledge into Preconditions for Planning in the Situation Calculus

Alfredo Gabaldon

Department of Computer Science

University of Toronto

alfredo@cs.toronto.edu

Abstract

A promising technique used in some planning systems to improve their performance is the use of domain dependent search control knowledge. We present a procedure for compiling search control knowledge, expressed declaratively in a logic, into the preconditions of the plan actions (operators). We do this within the framework of the situation calculus by introducing a transformation of non-Markovian action theories into classical Markovian situation calculus theories.

1 Introduction

One of the strategies used in planning to mitigate the complexity of the general problem is to employ some form of domain specific knowledge to assist the search for a plan. For instance, Hierarchical Task Network (HTN) planning systems [Sacerdoti, 1974; Wilkins, 1988; Erol *et al.*, 1996] use domain specific procedural knowledge in the form of task decomposition methods. The forward-chaining planners TLPlan [Bacchus and Kabanza, 2000], TALPlanner [Kvarnström and Doherty, 2000] and the SAT based planner SATPlan [Kautz and Selman, 1998] use domain knowledge in the form of declarative constraints expressed in a logical language. This strategy has been shown experimentally to yield remarkable improvements in performance. Both TLPlan and TALPlanner use control constraints expressed in the form of temporal logic formulas. These formulas are used to eliminate plan prefixes which will lead to a suboptimal plan or cannot be extended into a complete plan at all.

Most practical planning systems use STRIPS, ADL or extensions of these to describe actions and their effects. However, the first formal specification of the classical planning problem, due to [Green, 1969], was postulated in the language of the situation calculus [McCarthy, 1963]. The situation calculus has proven to be a very powerful formalism and has been employed in the formalization of many different aspects of dynamical systems (see e.g. [Reiter, 2001]). In this paper, we use the situation calculus as our formal framework and consider how to incorporate search control into action theories. Specifically, we show that control formulas with a certain syntactic form can be incorporated into

nonMarkovian action theories in the situation calculus, as recently introduced in [Gabaldon, 2002], in a similar way as [Lin and Reiter, 1994] treat qualification state constraints. In these nonMarkovian action theories, the effects and preconditions of actions are not assumed to depend solely on the current situation, but on any past situation. This nonMarkovian property of the theories allows an easier incorporation of dynamic constraints into database specifications, a more natural and concise axiomatization of nonMarkovian operations like *rollback* in transaction systems and other domains with nonMarkovian properties. In this paper, we are concerned with another problem where these theories are useful: incorporating search control knowledge by treating it as dynamic, qualification state constraints. Dynamic in the sense that they may refer to the current and any past situation, as opposed to static state constraints which refer only to the current state. Moreover, they are understood as qualification constraints because they pose further restrictions on the “executability” of actions. (We do not consider the use of these constraints for expressing ramifications, i.e. indirect effects of actions.)

Furthermore, we present a transformation from nonMarkovian action theories into Markovian ones. This transformation takes an action theory with nonMarkovian axioms and by applying regression steps and introducing additional fluents and their corresponding successor state axioms, produces a classical Markovian theory as introduced by [Reiter, 1991]. We then show how this transformation can be used for compiling search control knowledge into normal action preconditions. This nonMarkovian to Markovian transformation procedure is of independent interest. Toward the end we extend this approach for theories with explicit time and close with a discussion.

2 Formal Preliminaries

In this section we give an overview of the situation calculus and the main definitions necessary for action theories without the Markov assumption.

2.1 The Situation Calculus

The situation calculus is a first order logic language with three basic components: actions, situations, and fluents. Actions are responsible for all the changes in the world. Situations are sequences of actions which represent possible histories of the world. Fluents are properties of the world that

change from situation to situation as a result of the execution of actions. Formally, the language has three sorts: *action*, *situation* and *fluent*. In addition to variables of these sorts, the language includes functions such as $move(x, y)$ to represent actions, a constant S_0 and a function $do(a, s)$ for situations such as $do(move(x, y), S_0)$, and predicates for representing fluents such as $atLocation(x, l, do(a, s))$. The initial situation, or empty history, is denoted by the constant S_0 . Non-empty histories are built by means of the function do . For a complete description see [Pirri and Reiter, 1999; Reiter, 2001].

A situation calculus axiomatization of a domain, includes the following set of axioms:

1. For each action function $A(\vec{x})$, an *action precondition axiom* of the form: $Poss(A(x_1, \dots, x_n), s) \equiv \Pi_A(x_1, \dots, x_n, s)$ where s is the only term of sort situation in $\Pi_A(x_1, \dots, x_n, s)$.
2. For each fluent $F(\vec{x}, s)$, a *successor state axiom* of the form: $F(x_1, \dots, x_n, do(a, s)) \equiv \Phi_F(x_1, \dots, x_n, a, s)$ where s is the only term of sort situation in $\Phi_F(x_1, \dots, x_n, a, s)$.
3. Unique names axioms for actions. For instance: $move(x, y) \neq pickup(x)$.
4. Axioms describing the initial situation of the world: a finite set of sentences whose only situation term is the constant S_0 .

A set of these axioms, together with a set of domain independent foundational axioms Σ , is called a (Markovian) *basic action theory*.

2.2 A nonMarkovian Situation Calculus

For a basic action theory without the Markov assumption, we need some definitions. These are based on those in [Gabaldon, 2002].

We will use the following abbreviations in the definitions to follow:

$$(\exists s : \sigma' \sqsubset \sigma'' \sim \sigma)W \stackrel{\text{def}}{=} (\exists s)[\sigma' \sqsubset \sigma'' \wedge \sigma'' \sim \sigma \wedge W]$$

$$(\forall s : \sigma' \sqsubset \sigma'' \sim \sigma)W \stackrel{\text{def}}{=} (\forall s)[(\sigma' \sqsubset \sigma'' \wedge \sigma'' \sim \sigma) \supset W]$$

where $\sim$ stands for either $\sqsubset$ or $=$ and variable s appears in σ'' . If σ' is S_0 then we may write $(\exists s : \sigma'' \sim \sigma)W$ and $(\forall s : \sigma'' \sim \sigma)W$ instead.

Definition 1 (Bounded Formulas) For $n \geq 0$, let σ be a term $do([\alpha_1, \dots, \alpha_n], \lambda)$ rooted at λ . The formulas of $\mathcal{L}_{sitcalc}$ *bounded* by σ are the smallest set of formulas such that:

1. If W is an atom whose situation terms are all rooted at λ , then W is bounded by σ .
2. If W', W'' are formulas bounded by situation terms rooted at s and λ , respectively, then $(\exists s : \sigma' \sqsubset \sigma'' \sim \sigma)W$ and $(\forall s : \sigma' \sqsubset \sigma'' \sim \sigma)W$ are formulas bounded by σ , where σ'' is rooted at s and $W = (\neg)(W' \wedge W'')$.
3. If W_1, W_2 are formulas bounded by situation terms rooted at λ , then $\neg W_1$, $W_1 \wedge W_2$ and $(\exists v)W_1$, where v is of sort *action* or *object*, are formulas bounded by σ .

The set of formulas *strictly bounded* by σ is similarly defined by requiring in item (1) above that all situation terms of W be subterms of σ , in item (2) that W' be strictly bounded by a subterm of σ'' and W'' by a subterm of σ ; and in item (3) that W_1, W_2 be strictly bounded by subterms of σ .

Example 1 Past temporal logic connectives can be expressed in the situation calculus with strictly bounded formulas as follows:

1. $prev(\varphi, s) \stackrel{\text{def}}{=} (\exists a).(\exists s' : do(a, s') = s) \varphi(a, s')$.
2. $since(\varphi_1, \varphi_2, s) \stackrel{\text{def}}{=} (\exists s' : s' \sqsubset s). \varphi_2(s') \wedge (\forall s'' : s' \sqsubset s'' \sqsubseteq s) \varphi_1(s'')$.
3. $sometime(\varphi, s) \stackrel{\text{def}}{=} (\exists s' : s' \sqsubset s) \varphi(s')$.
4. $always(\varphi, s) \stackrel{\text{def}}{=} (\forall s' : s' \sqsubset s) \varphi(s')$.

NonMarkovian basic action theories differ from those which include the Markov assumption in that preconditions and effects of actions may depend on any past situation, not only on the current one.

Hence the rhs, $\Pi_A(x_1, \dots, x_n, s)$, of action precondition axioms in a nonMarkovian basic action theory are formulas bounded by situation term s which do not mention predicate $Poss$ and may refer to past situations. Similarly, the rhs, $\Phi_F(x_1, \dots, x_n, a, s)$ of successor state axioms are formulas strictly bounded by s .

2.3 Regression

For basic action theories with the Markov assumption, [Pirri and Reiter, 1999] define a provenly correct *regression* mechanism that takes a situation calculus sentence and, under certain restrictions on the form of this sentence, transforms it into an equivalent sentence whose only situation term is S_0 . This allows proving sentences without appealing to the foundational axioms Σ which include a second order axiom. This regression operator was generalized for nonMarkovian theories in [Gabaldon, 2002].

In a nutshell, the regression operator, denoted by $\mathcal{R}$, takes a sentence and recursively replaces each fluent atom $F(\vec{t}, do(\alpha, \sigma))$ by its definition according to its successor state axiom, i.e. by $\Phi_F(\vec{t}, \alpha, \sigma)$. Atoms $Poss(A(\vec{t}), do(\alpha, \sigma))$ are similarly replaced by their definitions given by the action precondition axioms. Regression recursively replaces these atoms until all the situation terms are reduced to S_0 . For lack of space we refer the reader to [Pirri and Reiter, 1999; Reiter, 2001; Gabaldon, 2002] for the formal details.

In the transformation operator we introduce later, we will use a one-step version of the regression operator: $\mathcal{R}^1[W[do(\alpha, \sigma)]]$ stands for the regression of a sentence $W[do(\alpha, \sigma)]$ bounded by $do(\alpha, \sigma)$ that results in a sentence bounded by σ .

3 Control Knowledge and the Qualification Problem

As mentioned earlier, the use of search control knowledge has proven to be a promising recourse for improving the performance of planning systems. Both TLPlan [Bacchus and

Kabanza, 2000] and TALPlanner [Kvarnström and Doherty, 2000] use declarative control knowledge in the form of temporal logic and have shown substantial computational improvement.

Since our goal is to compile search control into preconditions, we express control knowledge in terms of the past, not the future as it is done in TLPlan and TALPlanner (this is further discussed in the last section). However, declarative search control knowledge is typically expressed in a future temporal logic and we believe that obtaining preconditions from knowledge in that form is an important problem. In the last section we comment on how we are approaching this.

From the point of view of logical theories of action, such as the situation calculus axiomatizations we discuss in this paper, taking control knowledge into account is closely related to the classical *qualification problem* [McCarthy, 1977]. This is the problem of determining all the conditions that need to be satisfied for an action to be executable and control knowledge is effectively a set of additional constraints on the executability of actions, i.e. on what actions can be considered part of an executable plan.

Lin & Reiter [1994] have considered the qualification problem for situation calculus basic action theories with *state constraints*, which are formulas of the form $(\forall s)Q(s)$, uniform in s . Their solution is to add the qualifications to the precondition axiom of each action type:

$$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s) \wedge \Pi_Q,$$

where $\Pi_Q \equiv \mathcal{R}^1[Q(do(A(\vec{x}), s))]$. After adding the qualifications to the precondition axioms, one can discard the constraints if two conditions are satisfied: all the state constraints hold initially, i.e. $Q(S_0)$ holds for each Q ; and the domain closure assumption on actions is included in the theory.

Control formulas are similar to state constraints in that they pose additional qualifications on actions, but they are more general since they may quantify over past situations.

In the nonMarkovian situation calculus, successor state axioms and action precondition axioms can refer to past situations in addition to the current situation. This permits control knowledge to be incorporated into action precondition axioms in the same way Lin & Reiter do. Formally, if $C(s)$ is a formula bounded by s representing some piece of control knowledge, we take this into account by adding it as an additional precondition for actions:

$$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s) \wedge C(do(A(\vec{x}), s)). \quad (1)$$

Precondition axioms in nonMarkovian action theories allow situation terms rooted at s such as $do(A(\vec{x}), s)$ to appear in their rhs, so we can add $C(do(A(\vec{x}), s))$ directly without modification. By including the domain closure assumption on actions, we are then guaranteed that a situation is executable only if it satisfies the constraints.

Example 2 (Logistics domain constraint) Consider a constraint from a logistics domain, which involves vehicles delivering packages to locations, saying that a truck must remain in a location if there is a package at that location and it's never been in the truck (so it hasn't just been delivered and needs to be moved):

$$\begin{aligned} &(\forall p, t, loc). \\ &since(\neg inTruck(p, t), atObj(p, loc) \wedge atTruck(t, loc), s) \\ &\quad \supset \\ &since(atTruck(t, loc), atObj(p, loc) \wedge atTruck(t, loc), s) \end{aligned}$$

Suppose that the action precondition axiom for $driveTruck(t, loc)$, driving a truck t to a location loc is:

$$\begin{aligned} Poss(driveTruck(t, loc), s) \equiv \\ (\exists loc'). atLoc(t, loc', s) \wedge loc' \neq loc. \end{aligned}$$

We can add the control formula as a new conjunct in the rhs after simply replacing s with $s^* = do(driveTruck(t, loc), s)$:

$$\begin{aligned} Poss(driveTruck(t, loc), s) \equiv \\ (\exists loc'). atLoc(t, loc', s) \wedge loc' \neq loc \wedge (\forall p, t, loc). \\ since(\neg inTruck(p, t), atObj(p, loc) \wedge atTruck(t, loc), s^*) \\ \quad \supset \\ since(atTruck(t, loc), atObj(p, loc) \wedge atTruck(t, loc), s^*). \end{aligned}$$

In the next section, we introduce an operator which takes a basic action theory with such nonMarkovian axioms, and transforms it into one with Markovian ones.

4 Transforming a nonMarkovian Theory into Markovian

In order simplify the presentation, let us first make some simplifying assumptions on the form and the nesting of formulas on which we define the transformation.

Formally, in reference to item (2) of Definition 1, we will assume that formulas are of the forms:

$$(\exists s' : s^* \sqsubset s' \sqsubset s) (W_1[s'] \wedge (\forall s'' : s' \sqsubset s'' \sqsubset s) W_2[s'']) \quad (2)$$

$$(\forall s' : s^* \sqsubset s' \sqsubset s) (W_1[s'] \wedge (\exists s'' : s' \sqsubset s'' \sqsubset s) W_2[s'']) \quad (3)$$

where all (if any) free variables of W_2 are among the free variables of the full formula. Other combinations of quantifiers and logical connectives reduce to these two cases. The restriction on the free variables of W_2 does make the transformation simpler. We treat the general case in the full version of this paper. Next, we restrict the nesting of formulas in the following way. By definition, W_2 must itself be of the form $(\neg)(W' \wedge W'')$ where W' is bounded by s'' and W'' by s . We will restrict W_2 by requiring it to be of the form $(\neg)W'$, i.e. without the conjunct W'' .

Notice however that, even with this restrictions, arbitrary nesting of the past temporal logic abbreviations (Example 1) is still expressible.

The transformation performs a combination of regression and of replacement of formulas by new fluents:

Definition 2 Let φ be an $\mathcal{L}_{sitcalc}$ formula bounded by s . We define a formula $\mathcal{M}[\varphi]$ obtained from φ recursively as follows. By Definition 1, formula φ must be of one of the following forms:

1. An atom whose only situation term is s . Then $\mathcal{M}[\varphi] = \varphi$.
2. A formula of the form (2). Then, $\mathcal{M}[\varphi] = P_\varphi(\vec{x}, s)$ which is a new fluent with successor state axiom:

$$P_\varphi(\vec{x}, do(a, s)) \equiv \mathcal{R}^1[P_{W_2}(\vec{x}_2, do(a, s))] \wedge \{P_{W_1}(\vec{x}_1, s) \vee P_\varphi(\vec{x}, s)\}$$

where $\mathcal{M}[W_1] = P_{W_1}(\vec{x}_1, s)$ and $\mathcal{M}[W_2] = P_{W_2}(\vec{x}_2, s)$.

Initially, the new fluent is set to false, i.e. $(\forall \vec{x}) \neg P_\varphi(\vec{x}, S_0) \in \mathcal{D}_{S_0}$.

3. A formula of the form (3). Then, $\mathcal{M}[\varphi] = P_\varphi(\vec{x}, s)$ which is a new fluent with successor state axiom:

$$P_\varphi(\vec{x}, do(a, s)) \equiv P_{W_1}(\vec{x}_1, s) \wedge \{\mathcal{R}^1[P_{W_2}(\vec{x}_2, do(a, s))] \vee P_\varphi(\vec{x}, s)\}.$$

where $\mathcal{M}[W_1] = P_{W_1}(\vec{x}_1, s)$ and $\mathcal{M}[W_2] = P_{W_2}(\vec{x}_2, s)$ as before, and the new fluent is initially set to true, i.e. $(\forall \vec{x}) P_\varphi(\vec{x}, S_0) \in \mathcal{D}_{S_0}$.

Theorem 1 Let φ denote a formula of the form (2) or (3) and $\mathcal{D}_\varphi$ be the basic action theory obtained from $\mathcal{D}$ by adding the fluents and axioms resulting from applying $\mathcal{M}$ to φ . Then,

$$\mathcal{D}_\varphi \models (\forall)[\varphi \equiv P_\varphi(\vec{x}, s)].$$

Now, we can apply the transformation to a nonMarkovian action theory $\mathcal{D}$ whose action precondition axioms have as an additional condition a constraint $C(s)$, and obtain a Markovian theory $\mathcal{D}^\mathcal{M}$ that enforces the constraint. This produces a theory with precondition axioms of the form:

$$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s) \wedge \mathcal{M}[C(do(A(\vec{x}), s))].$$

Theorem 2 Let $C(s)$ be a control knowledge formula bounded by s and let $\mathcal{D}$ be a nonMarkovian basic action theory whose precondition axioms have the form (1) and includes domain closure axioms for actions. Then,

$$\mathcal{D}^\mathcal{M} \models (\forall a, s). Poss(a, s) \supset C(do(a, s)).$$

Example 3 Consider again the logistics domain control formula from Example 2. Let φ_1 and φ_2 stand for premise and consequence, respectively. Both formulas are of form (2). The premise φ_1 stands for:

$$(\exists s' : s' \sqsubseteq s) \{atObj(p, loc, s') \wedge atTruck(t, loc, s') \wedge (\forall s'' : s' \sqsubseteq s'' \sqsubseteq s) \neg inTruck(p, t, s'')\}$$

Applying operator $\mathcal{M}$ on φ_1 produces three new fluents $P_1(p, t, loc, s)$, $P_2(p, t, loc, s)$ and $P_{\varphi_1}(p, t, loc, s)$ with the following successor state axioms:

$$P_1(p, t, loc, do(a, s)) \equiv atObj(p, loc, s) \wedge atTruck(t, loc, s) \vee P_1(p, t, loc, s)$$

$$P_2(p, t, loc, do(a, s)) \equiv \mathcal{R}^1[\neg atTruck(t, loc, do(a, s))] \wedge \{P_2(t, loc, s) \vee (atObj(p, loc, s) \wedge atTruck(t, loc, s))\}$$

$$P_{\varphi_1}(p, t, loc, do(a, s)) \equiv \mathcal{R}^1[P_2(t, loc, do(a, s))] \wedge P_1(p, t, loc, s) \vee P_{\varphi_1}(p, t, loc, s)$$

The result of $\mathcal{M}$ on φ_2 is similar. Then the constraint becomes:

$$P_{\varphi_1}(p, t, loc, s) \supset P_{\varphi_2}(p, t, loc, s).$$

Finally, we add the constraint to action precondition axioms:

$$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s) \wedge \mathcal{R}^1[P_{\varphi_1}(p, t, loc, do(A(\vec{x}), s)) \supset P_{\varphi_2}(p, t, loc, do(A(\vec{x}), s))].$$

Example 4 (Closed form solution for a fluent) Consider a fluent $F(\vec{x}, s)$ and its closed form solution [Reiter, 2001]:

$$\begin{aligned} F(\vec{x}, s) \equiv & F(\vec{x}, S_0) \wedge \neg(\exists s_1 : s_1 \sqsubseteq s) \{(\exists a', s'). \\ & do(a', s') = s_1 \wedge \gamma_F^-(\vec{x}, a', s')\} \vee \\ & (\exists s_1 : s_1 \sqsubseteq s) \{(\exists a', s'). do(a', s') = s_1 \wedge \gamma_F^+(\vec{x}, a', s') \wedge \\ & (\forall s_2 : s_1 \sqsubseteq s_2 \sqsubseteq s). \\ & \neg(\exists a'', s''). do(a'', s'') = s_2 \wedge \gamma_F^-(\vec{x}, a'', s'')\} \end{aligned} \quad (4)$$

Clearly, the rhs of the above sentence is formed by $F(\vec{x}, S_0)$ and two formulas which are almost of the form (2). The difference is in the $\sqsubseteq$'s immediately after the existential quantifiers on situations s_1, s_2 , which would need to be strict $\sqsubset$ to fit form (2), and the subformulas of the form $(\exists a', s') do(a', s') = s$. The latter will be simplified away. Further, it is easy to modify the transformation on (2) to handle the $\sqsubseteq$'s. In general, all we need to do is replace $P_{W_1}(\vec{x}_1, s)$ with $\mathcal{R}^1[P_{W_1}(\vec{x}_1, do(a, s))]$. However, in this example the one step regression won't be necessary.

Applying the transformation to the subformula

$$\varphi_1 = (\exists s_1 : s_1 \sqsubseteq s) \{(\exists a', s'). do(a', s') = s_1 \wedge \gamma_F^-(\vec{x}, a', s')\}$$

results in a predicate $P_{\varphi_1}(\vec{x}, s)$ with successor state axiom (after simplifying) $P_{\varphi_1}(\vec{x}, do(a, s)) \equiv \gamma_F^-(\vec{x}, a, s) \vee P_{\varphi_1}(\vec{x}, s)$

The transformation of subformula $\varphi_2 =$

$$\begin{aligned} & (\exists s_1 : s_1 \sqsubseteq s) \{(\exists a', s'). do(a', s') = s_1 \wedge \gamma_F^+(\vec{x}, a', s') \wedge \\ & (\forall s_2 : s_1 \sqsubseteq s_2 \sqsubseteq s). \\ & \{ \neg(\exists a'', s''). do(a'', s'') = s_2 \wedge \gamma_F^-(\vec{x}, a'', s'') \} \end{aligned}$$

results in a predicate $P_{\varphi_2}(\vec{x}, s)$ with successor state axiom (after simplifying) $P_{\varphi_2}(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee \neg \gamma_F^-(\vec{x}, a, s) \wedge P_{\varphi_2}(\vec{x}, s)$

Noticeably, this successor state axiom for $P_{\varphi_2}(\vec{x}, s)$ is almost identical to the canonical form successor state axiom for $F(\vec{x}, s)$ which is $F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee F(\vec{x}, s) \wedge \neg \gamma_F^-(\vec{x}, a, s)$.

Finally, after applying the transformation to subformulas φ_1, φ_2 of the closed form formula (4), we obtain the following equivalent sentence:

$$F(\vec{x}, s) \equiv F(\vec{x}, S_0) \wedge \neg P_{\varphi_1}(\vec{x}, s) \vee P_{\varphi_2}(\vec{x}, s)$$

The transformation comes close to producing a canonical Markovian successor state axiom. It does not because the closed form solution formula uses information about the value of F in the initial situation.

5 Constraints with explicit time

In this section, we extend our approach to search control knowledge with explicit time. In the temporal situation calculus [Reiter, 2001], actions have an additional temporal argument denoting the time of occurrence, e.g. $drive(truck, loc, 3.2)$ for driving a *truck* to a location *loc* at time 3.2. There are also some additional axioms: $time(A(\vec{x}, t)) = t$ specifying the occurrence time for each action *A*, and $start(do(a, s)) = time(a)$ specifying the start time of situations.

We will use $[t_1, t_2] + t$ to denote $[t_1 + t, t_2 + t]$ and $t \in [t_1, t_2]$ to denote $t_1 \leq t \wedge t \leq t_2$. So $start(s) \in [t_1, t_2] + start(s')$ stands for $start(s') + t_1 \leq start(s) \wedge start(s) \leq start(s') + t_2$ which means that *s'* precedes *s* by at least t_1 and at most t_2 .

Analogous to forms (2) and (3), we have:

$$(\exists s' : s^* \sqsubset s' \sqsubset s) \quad \{W_1[s'] \wedge start(s) \in [t_1, t_2] + start(s') \wedge (\forall s'' : s' \sqsubset s'' \sqsubseteq s) W_2[s'']\} \quad (5)$$

$$(\forall s' : s^* \sqsubset s' \sqsubset s) \quad \{W_1[s'] \wedge start(s) \in [t_1, t_2] + start(s') \wedge (\exists s'' : s' \sqsubset s'' \sqsubseteq s) W_2[s'']\} \quad (6)$$

For a formula φ of the form (5), $\mathcal{M}$ yields a predicate $P_\varphi(\vec{x}, t, s)$ with a successor state axiom:

$$P_\varphi(\vec{x}, t, do(a, s)) \equiv \mathcal{R}^1[P_{W_2}(\vec{x}_2, do(a, s))] \wedge \{t = start(s) \wedge P_{W_1}(\vec{x}_1, s) \vee P_\varphi(\vec{x}, t, s)\} \quad (7)$$

and $\mathcal{M}[(5)] = P_\varphi(\vec{x}, t, s) \wedge start(s) \in [t_1, t_2] + t$. The initial database would include $(\forall t) \neg P_\varphi(\vec{x}, t, S_0)$.

For a formula of the form (6), the successor state axiom for $P_\varphi(\vec{x}, t, s)$ is:

$$P_\varphi(\vec{x}, t, do(a, s)) \equiv P_{W_1}(\vec{x}_1, s) \wedge \{\mathcal{R}^1[P_{W_2}(\vec{x}_2, do(a, s))] \wedge t = start(s) \vee P_\varphi(\vec{x}, t, s)\} \quad (8)$$

with $\mathcal{M}[(6)] = P_\varphi(\vec{x}, t, s) \wedge start(s) \in [t_1, t_2] + t$. The initial database would include $(\forall t) P_\varphi(\vec{x}, t, S_0)$.

The statement of the following theorem, which establishes the correctness of the transformation, requires adding the conjunct $t = start(s')$ to formulas (5) and (6):¹

$$(\exists s' : s^* \sqsubset s' \sqsubset s) \quad \{W_1[s'] \wedge start(s) \in [t_1, t_2] + start(s') \wedge t = start(s') \wedge (\forall s'' : s' \sqsubset s'' \sqsubseteq s) W_2[s'']\}$$

Similarly for formula (6).

Theorem 3 Let φ denote a formula of the form (5) or (6) and define $\mathcal{D}^\mathcal{M}$ as before. Then,

$$\mathcal{D}^\mathcal{M} \models (\forall)[\varphi \equiv \{P_\varphi(\vec{x}, t, s) \wedge start(s) \in [t_1, t_2] + t\}].$$

6 Discussion and conclusion

We have shown that incorporating control knowledge as additional preconditions in nonMarkovian action theories is trivial when this knowledge is in the form of bounded formulas (which include encodings of Past temporal logic modalities). We then introduced an operator for transforming non-Markovian basic action theories into Markovian ones. This

¹The variable *t* is a fresh new variable so there is no effect on the meaning of the formulas.

operator introduces into a theory the additional fluents and axioms needed for keeping track of the relevant past information. We then showed how this operator can be used for compiling TLPlan style search control knowledge into action preconditions in the situation calculus.

In the TLPlan system, control knowledge is expressed in terms of linear (future) temporal logic, that is, temporal modalities *next*, *always in the future*, *until*, and *sometime in the future* are used. Every time a new operator is added to the plan prefix being considered, a control formula is *progressed* through it. If the formula is progressed into *false*, the plan prefix is pruned since a plan with this prefix will violate the constraint. Although it is easy to write situation calculus formulas corresponding to future temporal logic, regression cannot be used on them. However, we argue that if instead of progression, as in TLPlan, we evaluate the control formulas against plan prefixes, it is reasonable to use a logic that refers to past situations instead of future situations. Indeed, all the reasoning used by a planning system to decide if a partial plan should be discarded or not must be done relative to the properties of the partial plan itself. Furthermore, some future temporal formulas are satisfied by all plan prefixes (e.g. “sometime in the future ϕ ” where ϕ is not unsatisfiable) and are not useful for search control. Thus it seems to us that restricting control formulas to refer exclusively to the past is appropriate.

Moreover, using past temporal logic is semantically cleaner for the following reason. The semantics of future temporal logic is defined in terms of infinite sequences of states. However, plans are finite sequences of actions and thus produce a finite sequence of states. In order to deal with this technical difficulty, TLPlan makes the assumption that the world never changes after the plan is completely executed and therefore the last state infinitely repeats. This assumption is reasonable under another assumption typically made in classical planning: that the agent executing the plan is the only agent that changes the world, so when this agent terminates executing its plan, the world remains unchanged. This assumption is unnecessary if using past temporal logic or bounded situation calculus formulas.

Nevertheless, it would be convenient to be able to handle control knowledge expressed in future temporal logic. We are currently working on developing a procedure that would allow us to take a future temporal logic control formula, such as this Briefcase domain one:

$$goal(at(x, loc)) \supset atBriefcase(loc) \text{ Until } \neg inBriefcase(x)$$

and produce a bounded situation calculus formula:

$$(\forall s_1 : S_0 \sqsubseteq s_1 \sqsubseteq s) \neg goal(at(x, loc), s_1) \vee (\forall a, s_2 : s_1 \sqsubseteq do(a, s_2) \sqsubseteq s) \{atBriefcase(loc, s_2) \vee (\exists s_3 : s_1 \sqsubseteq s_3 \sqsubseteq do(a, s_2)) \neg inBriefcase(x, s_3)\}$$

that a plan prefix must satisfy. In turn, we can then apply the procedure introduced in this paper to compile these formulas into the preconditions of actions.

Bacchus & Kabanza [1998] have also extended their system for planning with *temporally extended goals*, which are conditions on sequences of states and not only on the final state as in classical planning. They use a first-order extension of the Metric Interval Temporal Logic (MITL) [Alur et

al., 1991] to specify the temporally extended goals. Defining plan correctness with respect to MITL formulas as goals again requires one to make the assumption that the world remains unchanged after the plan is executed. In the case of temporally extended goals this is a further complication since testing if the current state satisfies the goal ϕ means checking whether ϕ is true if the world were to indefinitely remain as it currently is. (Past) MITL formulas can be encoded as formulas of the situation calculus with explicit time as discussed in the previous section. Hence our approach also shows a way to reduce planning with temporally extended goals into planning with classical goals.

Some preliminary experiments by Bacchus & Ady [1999] have shown that compiling search control knowledge into action preconditions can result in better performance compared to systems such as TLPlan which keep control knowledge as a separate set of formulas. A general method for compiling temporal formulas into preconditions in their framework has not yet been developed.

The work of [Rintanen, 2000] is also concerned with the use of control knowledge as additional preconditions of plan operators. He uses “auxiliary facts” which is similar to our introduction of new fluents in the transformation. Like Bacchus & Kabanza, Rintanen uses future linear temporal logic for representing control knowledge for planners with ADL-like operators. With the help of these auxiliary facts, he then compiles the control knowledge into the operators. The main difference between his work and ours is that his framework is propositional, while ours is in the more expressive first order situation calculus. Furthermore, he does not allow nesting of temporal operators—in our framework this would correspond to disallowing nesting of quantifiers on situations—which we do allow. Also related but with a different approach, is the work of Sierra [1998]. There, domain knowledge in the form of *action selection rules* is used for controlling a STRIPS planner. Analyzing the relationship of Sierra’s work with ours is among our future work.

References

- [Alur et al., 1991] Rajeev Alur, Tomas Feder, and Thomas Henzinger. The benefits of relaxing punctuality. In *Proceedings of the Thenth Annual ACM Symposium on Principles of Distributed Computing (PODC’91)*, pages 139–152, 1991.
- [Bacchus and Ady, 1999] Fahiem Bacchus and Michael Ady. Precondition control. 1999.
- [Bacchus and Kabanza, 1998] Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22:5–27, 1998.
- [Bacchus and Kabanza, 2000] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191, 2000.
- [Erol et al., 1996] Kutluhan Erol, James A. Hendler, and Dana S. Nau. Complexity results for hierarchical task-network planning. *Annals of Mathematics and Artificial Intelligence*, 18:69–93, 1996.
- [Gabaldon, 2002] Alfredo Gabaldon. Non-markovian control in the situation calculus. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI’02)*, pages 519–524, Edmonton, Canada, 2002.
- [Green, 1969] C.C. Green. Theorem proving by resolution as a basis for question-answering systems. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 183–205. American Elsevier, New York, 1969.
- [Kautz and Selman, 1998] Henry Kautz and Bart Selman. The role of domain-specific knowledge in the planning as satisfiability framework. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS’98)*, pages 181–189, 1998.
- [Kvarnström and Doherty, 2000] Jonas Kvarnström and Patrick Doherty. TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, 30:119–169, 2000.
- [Lin and Reiter, 1994] Fangzen Lin and Ray Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5):655–678, October 1994.
- [McCarthy, 1963] J. McCarthy. Situations, actions and causal laws. Technical report, Stanford University, 1963. Reprinted in *Semantic Information Processing* (M. Minsky ed.), MIT Press, Cambridge, Mass., 1968, pp. 410–417.
- [McCarthy, 1977] John McCarthy. Epistemological problems of artificial intelligence. In *Proceedings of the Fifth International Conference on Artificial Intelligence (IJCAI’77)*, pages 1038–1044, 1977.
- [Pirri and Reiter, 1999] Fiora Pirri and Ray Reiter. Some contributions to the metatheory of the Situation Calculus. *Journal of the ACM*, 46(3):325–364, 1999.
- [Reiter, 1991] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press, 1991.
- [Reiter, 2001] Ray Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. MIT Press, Cambridge, MA, 2001.
- [Rintanen, 2000] Jussi Rintanen. Incorporation of temporal logic control into plan operators. In W. Horn, editor, *Procs. of the 14th European Conference on Artificial Intelligence (ECAI’00)*, pages 526–530, Amsterdam, 2000. IOS Press.
- [Sacerdoti, 1974] E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.
- [Sierra, 1998] Josefina Sierra. Declarative formalization of strategies for action selection. In *Procs. of the 7th Intl. Workshop on Nonmonotonic Reasoning NMR’98*, 1998.
- [Wilkins, 1988] David E. Wilkins. *Practical Planning: Extending the classic AI planning paradigm*. Morgan Kaufmann, San Mateo, CA, 1988.

Action representation and partially observable planning using epistemic logic

Andreas Herzig

IRIT/UPS

F-31062 Toulouse Cedex

France

herzig@irit.fr

Jérôme Lang

IRIT/UPS

F-31062 Toulouse Cedex

France

lang@irit.fr

Pierre Marquis

CRIL/Université d'Artois

F-62307 Lens Cedex

France

marquis@cril.univ-artois.fr

Abstract

We propose a purely logical framework for planning in partially observable environments. Knowledge states are expressed in a suitable fragment of the epistemic logic **S5**. We show how to lift the effects of actions (both physical actions and sensing actions) from the state level to the epistemic level. We show how progression, regression and plan generation can be achieved in our framework.

1 Introduction

Planning under incomplete knowledge and partial observability is a tricky issue in AI, because of its computational (temporal and spatial) hardness. *Partially observable Markov decision processes* (POMDP) are the mainstream approach to partially observable planning. Nevertheless, the applicability of the POMDP approach is limited from the practical side as soon as the set of states has a strong combinatorial structure, rendering the number of states much too high for an explicit representation of actions, preferences, and policies. On the other hand, *logical approaches to planning* under incomplete knowledge allow for much more compact encodings of planning problems than POMDPs; most of them deal with an incomplete initial state and/or nondeterministic actions, but either they do not handle partial observability, or at least they do it in a very simple way, by assuming for instance that the set of variables is partitioned between (directly) observable and unobservable variables.

To fill the gap between POMDPs and logical approaches, an abstraction of the POMDP model (leaving aside probabilities and expected utility) can be considered. It should account at least for the following elements: a set $\mathcal{S}$ of states; a set $\mathcal{B}$ of belief states built from $\mathcal{S}$; a set $\mathcal{A}$ of actions, where each action is associated with a transition model between states and/or belief states; some preference structure (e.g., a simple goal or a utility function); and a set $\mathcal{O}$ of observations, together with some correlation function between states and observations.

While policies for a fully observable MDP map states to actions, the output of such an abstract POMDP is a policy mapping *belief states* to actions; indeed, a POMDP can be viewed as a fully observable MDP over the set of belief states

(this is a classical way of understanding POMDPs – and even to solve them).

In this paper, we present a rich *logical* framework that instantiates the abstraction above, views a partially observable process as a fully observable process over belief states, and allows for expressing actions and policies in a compact way. The framework has a fairly good level of generality (since it avoids for instance to commit to a particular action language, see Section 3.1) and is therefore modular enough to be easily adapted or extended.

The simplest and best-known way of distinguishing between truths and beliefs in logic consists in expressing the problem in an epistemic or doxastic logic. To make the exposition simpler, we assume that the agent has *accurate beliefs*, i.e., all she believes is true in the actual world. This means that we identify belief and knowledge (since knowledge is usually viewed as true belief); therefore our framework is based on the logic **S5** and instead of *belief* we use the term *knowledge* throughout the paper.¹ **S5** is computationally no more complex than classical logic: **S5** satisfiability is NP-complete [Ladner, 1977].

In Section 2 we define two notions of knowledge states: simple knowledge states (for on-line plan execution) and complex knowledge states (for off-line reasoning about the effects of a plan). In Section 3 we show how a knowledge state evolves when an action is performed. Then we show in Section 4 how to perform goal regression, and we show in Section 5 how it can be used so as to implement a sound and complete plan generator. Section 6 discusses related work.

2 Knowledge states

The language $\mathcal{L}_K$ of propositional logic **S5** is built up from a finite set of propositional variables VAR , the usual connectives, the logical constants $\top$ and $\perp$ and the epistemic modality **K**. **S5** formulas are denoted by capital Greek letters Φ , Ψ etc. An **S5** formula is *objective* (or modality-free) iff it does not contain any occurrence of **K** (i.e., it is a classical propositional formula). Objective formulas are denoted

¹Alternatively, we could have chosen to work with *beliefs*, using the doxastic logic KD45, which is very similar to **S5** except that beliefs may be wrong, that is, $\mathbf{K}\varphi \rightarrow \varphi$ is not an axiom. The technical issues developed in this paper would have been almost identical. Now, choosing another logic than **S5** or KD45 would induce a lot of complications, including an important complexity gap.

by small Greek letters φ, ψ etc. and the set of all objective formulas from $\mathcal{L}_K$ is denoted by $\mathcal{L}_{VAR}$.

A fundamental property of S5 is that nested modalities collapse, i.e., $\mathbf{K}\mathbf{K}\Phi$ is equivalent to $\mathbf{K}\Phi$ and $\mathbf{K}\neg\mathbf{K}\Phi$ to $\neg\mathbf{K}\Phi$; for this reason, we assume without loss of generality that the scope of each occurrence of modality $\mathbf{K}$ in formula Φ is an objective formula.

An *epistemic atom* is an S5 formula of the form $\mathbf{K}\varphi$, where φ is objective. An *epistemic formula* is a formula built up from epistemic atoms and usual connectives: $\mathbf{K}(a \vee b) \vee \neg\mathbf{K}(c \wedge \neg d)$ is an epistemic formula, while $a \wedge \mathbf{K}b$ is not. An epistemic formula is *positive* iff it does not contain any occurrence of $\mathbf{K}$ in the scope of a negation: $\mathbf{K}(a \vee b) \vee \neg\mathbf{K}(c \wedge \neg d)$ is not positive, while $\mathbf{K}(a \vee b) \vee \mathbf{K}(c \wedge \neg d)$ is.

$S = 2^{VAR}$ is the set of all interpretations of $\mathcal{L}_{VAR}$, also called *states*. States are denoted by s, s' etc. If φ is an objective formula, we let $Mod(\varphi) = \{s \in S \mid s \text{ is a model of } \varphi\}$.

A structure² for S5 is defined as a nonempty set of states $M \subseteq S$. Rather than “structure”, we call M a *knowledge state* (SKS). Intuitively, it represents all the states the agent considers possible. The satisfaction of an S5 formula by an SKS M for a state $s \in M$ is defined inductively by:

- for φ objective, $(M, s) \models \varphi$ iff $s \models \varphi$;
- for φ objective, $(M, s) \models \mathbf{K}\varphi$ iff $\forall s' \in M$ we have $(M, s') \models \varphi$;
- $(M, s) \models \Phi \wedge \Psi$ iff $(M, s) \models \Phi$ and $(M, s) \models \Psi$;
- $(M, s) \models \Phi \vee \Psi$ iff $(M, s) \models \Phi$ or $(M, s) \models \Psi$;
- $(M, s) \models \neg\Phi$ iff $(M, s) \not\models \Phi$.

Finally, an SKS M satisfies Φ , denoted by $M \models \Phi$, iff for all $s \in M$ we have $(M, s) \models \Phi$. An S5 formula is valid (resp. satisfiable) iff it is satisfied by all SKSs (resp. by at least one SKS). An S5 formula Ψ is a consequence of an S5 formula Φ , noted $\Phi \models \Psi$ iff every SKS that satisfies Φ also satisfies Ψ . Φ and Ψ are equivalent, noted $\Phi \equiv \Psi$ iff $\Phi \models \Psi$ and $\Psi \models \Phi$ holds. Note that $\mathbf{K}(\varphi \wedge \psi)$ is equivalent to $\mathbf{K}\varphi \wedge \mathbf{K}\psi$ but $\mathbf{K}(\varphi \vee \psi)$ is implied by but generally not equivalent to $\mathbf{K}\varphi \vee \mathbf{K}\psi$.

Importantly, an SKS M can be identified with the strongest epistemic atom $\mathbf{K}\varphi$ that is satisfied by M . Such a $\mathbf{K}\varphi$ is unique up to equivalence. Therefore, SKSs will be denoted syntactically, and without ambiguity, by epistemic atoms $\mathbf{K}\varphi$.

A *complex knowledge state* (CKS) is a positive epistemic formula,³ generated by epistemic atoms and the connectives

²This semantics is equivalent (and simpler for our purpose) to the usual semantics by means of Kripke models $\langle W, val, R \rangle$ where W is a set of worlds, val a valuation function and R an equivalence relation. See for instance [Fagin et al., 1995].

³We restrict the syntax of CKS to positive epistemic formulas because for almost all problems, ignorance can already be expressed by the fact that positive knowledge is not provable from the current CKS. This way of generating explicit ignorance from implicit is a kind of Epistemic Closed World Assumption, already at work in [Reiter, 2001] and reminiscent of autoepistemic logic; its principle can be roughly be stated as “if I cannot prove that I know φ then I don’t know φ ”. However, for the purpose of this paper, we do not need this completion because the set of valid plans from a CKS and the set of valid plans from its completion coincide.

$\wedge$ and $\vee$. For instance, $[\mathbf{K}(a \vee b) \wedge \mathbf{K}\neg c] \vee \mathbf{K}(b \leftrightarrow c)$ is a CKS but $\mathbf{K}(a \vee b) \rightarrow \mathbf{K}c$ is not, $\neg\mathbf{K}a$ neither.

Example 1 In a model-based diagnosis context, let $\Sigma = \bigwedge_{c \in COMP} \mathbf{K}(\neg ab(c) \leftrightarrow \varphi_c)$ the formula representing the behaviour of a set of components $COMP$. Suppose that the agent is performing a test plan so as to identify the faulty components. The formula characterizing the condition for stopping performing tests and starting repairing the faulty components is $\Phi = \bigwedge_{c \in COMP} (\mathbf{K}ab(c) \vee \mathbf{K}\neg ab(c))$. The latter formula is a CKS. It is equivalent to an exponentially large disjunction of SKS: $\Phi \equiv \bigvee \{\mathbf{K}\delta_I \mid I \subseteq COMP\}$ where $\delta_I = (\bigwedge_{c \in I} ab(c)) \wedge (\bigwedge_{c \in COMP \setminus I} \neg ab(c))$.

A CKS Φ is said to be in *epistemic disjunctive normal form* (EDNF) iff it has the form $\mathbf{K}\varphi_1 \vee \dots \vee \mathbf{K}\varphi_n$, where each φ_i is an objective formula. Clearly, every CKS has an equivalent EDNF.

3 Actions and progression

In general, actions have both physical (or *ontic*) and epistemic effects, i.e., they are meant to change both the state of the world and the agent’s knowledge state, but without loss of generality we assume (as usually in AI) that any “mixed” action can be decomposed into two actions, the first one having only ontic effects and the second one only epistemic effects. For instance, the action of tossing a coin is decomposed into a `blind-toss` action followed by a `see` action telling the agent whether the coin has landed on `heads` or on `tails`.

3.1 Ontic actions

Ontic actions are meant to have effects on the world outside the agent, especially *physical* effects such as moving a block, switching the light, moving etc. They are assumed to be described in a *propositional action language* (allowing or not for nondeterminism, for ramifications/causality). Any action language $\mathcal{L}_A$ can be chosen, provided that it is propositional and that it expresses the effects of an action α within a formula Σ_α involving atoms labelled by t and atoms labelled by $t + 1$ (the former for the state before the action is performed, the latter for the state after it is performed). Among candidate languages we find those of the $\mathcal{A}$ family [Gelfond and Lifschitz, 1993], “propositionalized” situation calculus [Lin, 2000] and causality languages [McCain and Turner, 1998].⁴

The description Σ_α of an action α allows for computing the successor state of a state s (or the set of successor states if α is nondeterministic). This set is represented by any objective formula $prog(s, \alpha)$ (whose models form the set of successor states). This definition extends to sets of initial states, or equivalently to propositional formulas, by $prog(\varphi, \alpha) = \bigvee_{s \in Mod(\varphi)} prog(s, \alpha)$.

Now, given the description Σ_α of action α and a CKS Φ , the subsequent CKS, denoted by $Prog(\Phi, \alpha)$, is defined as follows:

⁴The common core of these languages is the use of explicit or implicit *successor state axioms*. These languages coincide for deterministic, ramification-free actions and differ in the way they treat nondeterminism (using for instance exogenous variables or Release statements), ramifications, etc.

Definition 1 (progression for ontic actions) Let α be an ontic action and let $\Phi = \mathbf{K}\varphi_1 \vee \dots \vee \mathbf{K}\varphi_n$ be a CKS in EDNF.

$$\text{Prog}(\Phi, \alpha) = \mathbf{K}\text{prog}(\varphi_1, \alpha) \vee \dots \vee \mathbf{K}\text{prog}(\varphi_n, \alpha).$$

The problem with the latter definition is that the CKS has to be put in EDNF first. This does not induce any loss of expressivity but the transformation may be exponentially large so that we may want to compute the successor CKS directly from a CKS expressed in any form, such as in Example 1.

We now give a more elegant way of computing progression via an extension of *variable forgetting* to S5 formulas. We recall first from [Lin and Reiter, 1994] the inductive definition of *forget*(X, φ) where φ is objective and $X \subseteq \text{VAR}$:

- (0) $\text{forget}(\emptyset, \varphi) = \varphi$;
- (1) $\text{forget}(\{x\}, \varphi) = \varphi_{x \leftarrow \perp} \vee \varphi_{x \leftarrow \top}$;
- (2) $\text{forget}(\{X \cup \{x\}, \varphi) = \text{forget}(\{x\}, \text{forget}(X, \varphi))$.⁵

Definition 2 (forgetting) Let Φ be an S5 formula and $X \subseteq \text{VAR}$. *Forget*(X, Φ) is the strongest S5 formula (a) entailed by Φ and (b) not mentioning any variable of X .

Forget($\cdot, \cdot$) is well-defined, because the set of formulas satisfying (a) and (b) is closed by conjunction. Thus *Forget*(X, Φ) is unique up to equivalence in S5.

Proposition 1

- If Φ is a CKS, then *Forget*(X, Φ) is a CKS.
- $\text{Forget}(X, \mathbf{K}\varphi) \equiv \mathbf{K}\text{forget}(X, \varphi)$.
- $\text{Forget}(X, \Phi_1 \vee \Phi_2) \equiv \text{Forget}(X, \Phi_1) \vee \text{Forget}(X, \Phi_2)$.
- If Φ_1 and Φ_2 do not share any variable then $\text{Forget}(X, \Phi_1 \wedge \Phi_2) \equiv \text{Forget}(X, \Phi_1) \wedge \text{Forget}(X, \Phi_2)$.

The interest of defining this forgetting operator is made clear by the following result. From every symbol $x \in \text{VAR}$ we make two copies x_t, x_{t+1} , where t (resp. $t+1$) represents the time point *before* (resp. *after*) the action is performed; then, for any formula $\varphi \in \mathcal{L}_{\text{VAR}}$, φ_t (resp. φ_{t+1}) is the formula from φ obtained by replacing each variable x by x_t (resp. x_{t+1}).

Proposition 2 Let α be an ontic action and let Φ be a CKS. We have:

$$\text{Prog}(\Phi, \alpha)_{t+1} \equiv \text{Forget}(\text{VAR}_t, \Phi_t \wedge \mathbf{K}\Sigma_\alpha)$$

where $\text{VAR}_t = \{x_t \mid x \in \text{VAR}\}$.

3.2 Epistemic actions

An epistemic action β is described wlog by the finite set of its possible outcomes $\{o_1, \dots, o_p\} \subseteq \mathcal{O}$, where each o_i is an objective formula describing a possible observation, and $o_1 \vee \dots \vee o_p$ is a tautology.⁶ Accordingly, we have $\mathcal{O} = \mathcal{L}_{\text{VAR}}$ in our framework. The effects of β are motivated by the following principles: *non-intrusiveness* (for any objective formula φ , if φ holds before β is performed, then it still holds after), *no-forgetting* (if the agent knows an objective formula

⁵In (1), $\varphi_{x \leftarrow \top}$ (resp. $\varphi_{x \leftarrow \perp}$) represents the formula obtained by replacing in φ each occurrence of variable x by $\top$ (resp. $\perp$).

⁶Or, more generally, $\Sigma \models o_1 \vee \dots \vee o_p$, where Σ is the given set of static domain rules.

φ before performing β , then she still knows it after) and *reliability* (if the agent observes o_j after performing β , then o_j holds, so that she knows o_j after observing it). These three properties imply that the effects of an epistemic action associated with the outcome set $\{o_1, \dots, o_p\}$ can be represented by the following progression operator:

Definition 3 (progression for epistemic actions) Let β be an epistemic action associated with the set of outcomes $\{o_1, \dots, o_p\}$, and let Φ be a CKS. We have:

$$\text{Prog}(\Phi, \beta) = \Phi \wedge (\mathbf{K}o_1 \vee \dots \vee \mathbf{K}o_p).$$

It follows that for any epistemic actions β, β_1, β_2 and a CKS Φ , progression is indifferent to the order of epistemic actions: $\text{Prog}(\text{Prog}(\Phi, \beta_1), \beta_2) \equiv \text{Prog}(\text{Prog}(\Phi, \beta_2), \beta_1)$; it is idempotent: $\text{Prog}(\text{Prog}(\Phi, \beta), \beta) \equiv \text{Prog}(\Phi, \beta)$; and it preserves positive knowledge: $\text{Prog}(\Phi, \beta) \models \Phi$.

Epistemic actions can be assumed fully executable without loss of generality (adding new symbols if necessary).⁷

A simple and well-known class of epistemic actions is the class of *binary tests*, also called *truth tests*: if φ is an objective formula, the action $\text{test}(\varphi)$ is defined by the outcome set $\{\varphi, \neg\varphi\}$. Therefore we have $\text{Prog}(\Phi, \text{test}(\varphi)) = \Phi \wedge (\mathbf{K}\varphi \vee \mathbf{K}\neg\varphi)$. In the rest of the paper, we focus on truth tests, which will make things simpler to expose.⁸

Note that a straightforward (but important) consequence of Definitions 1 and 3 is that the set of all CKS is closed under progression, i.e., if Φ is a CKS and α is an (ontic or epistemic) action, then $\text{Prog}(\Phi, \alpha)$ is a CKS.

4 Regression

The problem is stated as follows: given a CKS Φ (representing a goal knowledge state) and an (ontic or epistemic) action α , characterize the weakest CKS denoted by $\text{Reg}(\Phi, \alpha)$, in which performing α leads to a CKS satisfying Φ .

Definition 4 (regression) Let α be an (ontic or epistemic) action and let Φ be a CKS. The regression of Φ by α , is the unique CKS (up to logical equivalence), denoted by $\text{Reg}(\Phi, \alpha)$, such that (a) $\text{Prog}(\text{Reg}(\Phi, \alpha), \alpha) \models \Phi$ and (b) for any Φ' such that $\text{Prog}(\Phi', \alpha) \models \Phi$ we have $\Phi' \models \text{Reg}(\Phi, \alpha)$.

This definition is well-founded: indeed, if $\text{Prog}(\Phi_1, \alpha) \models \Phi$ then for any $\Phi_2 \models \Phi_1$ we have $\text{Prog}(\Phi_2, \alpha) \models \Phi$ (this is easily verified both for ontic actions and for epistemic actions). Therefore the set $\{\Psi \mid \text{Prog}(\Psi, \alpha) \models \Phi\}$ is closed w.r.t. disjunction, which entails that there exists a weakest Ψ such that $\text{Prog}(\Psi, \alpha) \models \Phi$, equivalent to $\bigvee \{\Xi \mid \text{Prog}(\Xi, \alpha) \models \Phi\}$.

⁷Note that actions that may fail to be informative can only be expressed here by the occurrence of a tautology in their outcomes. In this case, unfortunately, we get that $\text{Prog}(\Phi, \beta) \equiv \Phi$, because $\mathbf{K}o_1 \vee \dots \vee \mathbf{K}o_p \equiv \top$ as soon as $o_i \equiv \top$ for some i .

⁸This does not induce any loss of generality, because any epistemic action can be rewritten equivalently into a logarithmic sequence of binary tests, together with the addition of some domain constraints.

4.1 Regression for ontic actions

For any objective formula φ and any ontic action α , let $reg(\varphi, \alpha)$ be the standard (non-epistemic) regression of φ by α , i.e., any objective formula whose set of models consists of all states $s \in \mathcal{S}$ s.t. $Mod(prog(s, \alpha)) \subseteq Mod(\varphi)$ [Reiter, 1993].

Definition 5 (sufficient success conditions) A consistent conjunction of literals γ is a sufficient success condition for α and φ if and only if for any pair of states $(s, s') \in \mathcal{S} \times \mathcal{S}$ such that s is a model of γ and s' is a model of $prog(s, \alpha)$, s' is a model of φ . γ is a minimal sufficient success condition for α and φ if and only if it is a sufficient success condition for α and φ and no proper subconjunction of γ is. We denote by $\Gamma(\alpha, \varphi)$ the disjunction of all minimal sufficient success conditions for α and φ .

When actions are described using an action language satisfying the properties stated in Section 3.1, such conditions can be computed abductively. Clearly, γ is a sufficient success condition for α and φ if and only if $\gamma_t \wedge \Sigma_\alpha$ is satisfiable, and every model of $\gamma_t \wedge \Sigma_\alpha$ is a model of φ_{t+1} .

Proposition 3 Let Σ_α be the description of α in a propositional action language $\mathcal{L}_A$. $reg(\varphi, \alpha)$ is equivalent to

$$\Gamma(\alpha, \varphi) = \bigvee \{ \gamma \mid \gamma_t \in PI_{\Sigma_\alpha}^{VAR_t}(\varphi_{t+1}) \}.$$

In this proposition, $PI_{\Sigma_\alpha}^{VAR_t}((\varphi)_{t+1})$ denotes the set of prime implicants of the formula $\Sigma_\alpha \rightarrow (\varphi)_{t+1}$ that are consistent with Σ_α and built up from variables of VAR_t , only.

The latter abductive characterization of ontic actions is independent of the action language chosen – and it now allows for characterizing the regression of complex epistemic states by an ontic action.

Proposition 4 Let α be an ontic action, Φ a CKS and let $\mathbf{K}\varphi_1 \vee \dots \vee \mathbf{K}\varphi_n$ be an EDNF formula equivalent to Φ . Then

$$Reg(\Phi, \alpha) \equiv \mathbf{K}reg(\varphi_1, \alpha) \vee \dots \vee \mathbf{K}reg(\varphi_n, \alpha).$$

4.2 Regression for epistemic actions

Proposition 5 Let β be an epistemic action whose outcome set is $\{o_1, \dots, o_p\}$ and let Φ be a CKS. Let $\mathbf{K}\varphi_1 \vee \dots \vee \mathbf{K}\varphi_n$ be any EDNF formula equivalent to Φ . Then

$$Reg(\Phi, \beta) \equiv \bigvee \{ \mathbf{K}((o_1 \rightarrow \varphi_{f(1)}) \wedge \dots \wedge (o_p \rightarrow \varphi_{f(p)})) \mid f \in \{1..n\}^{\{1..p\}} \}.$$

In particular, if $\beta = test(\psi)$ is a binary test then

$$Reg(\Phi, \beta) \equiv \bigvee \{ \mathbf{K}((\psi \rightarrow \varphi_i) \wedge (\neg\psi \rightarrow \neg\varphi_j)) \mid i, j \in 1..n \}.$$

Sketch of proof. We give it in the latter case only (the proof for the general case is similar). We abbreviate $Reg(\Phi, \beta)$ by Δ . We first establish (the proof is omitted) that Δ is necessarily a CKS, i.e., $\Delta = \mathbf{K}\delta_1 \vee \dots \vee \mathbf{K}\delta_q$. Now,

$$\begin{aligned} & \models \Delta \wedge (\mathbf{K}\psi \vee \mathbf{K}\neg\psi) \rightarrow \Phi \\ \text{iff } \forall k \in 1..q \quad & \left\{ \begin{array}{l} \models \mathbf{K}(\delta_k \wedge \psi) \rightarrow \mathbf{K}\varphi_1 \vee \dots \vee \mathbf{K}\varphi_n \\ \models \mathbf{K}(\delta_k \wedge \neg\psi) \rightarrow \mathbf{K}\varphi_1 \vee \dots \vee \mathbf{K}\varphi_n \end{array} \right. \quad (*) \end{aligned}$$

At that stage, we make use of the following lemma: for any objective formulas A, B and C , $\mathbf{K}A \rightarrow (\mathbf{K}B \vee \mathbf{K}C)$ is valid (in S5) iff $A \rightarrow B$ is valid or $A \rightarrow C$ is valid. (*) is then

equivalent to $\forall k \in 1..q, \exists i, j \in 1..n$ s.t. $\delta_k \wedge \psi \rightarrow \varphi_i$ and $\delta_k \wedge \neg\psi \rightarrow \varphi_j$ are classically valid, which is equivalent to: $\forall k \in 1..q, \exists i, j \in 1..n$ s.t. $\delta_k \rightarrow ((\psi \rightarrow \varphi_i) \wedge (\neg\psi \rightarrow \varphi_j))$ is valid. The rest of the proof is easy. ■

Example 2 Consider the epistemic actions $\alpha = test(u \wedge v)$ and $\beta = test(u \leftrightarrow v)$, and the ontic action γ that switches the value of u , thus described by $(u_{t+1} \leftrightarrow \neg u_t) \wedge (v_{t+1} \leftrightarrow v_t)$. Let $\Phi = \mathbf{K}v \vee \mathbf{K}\neg v$. Applying Proposition 5 we get $Reg(\Phi, \alpha) \equiv \mathbf{K}v \vee \mathbf{K}\neg v \vee \mathbf{K}(v \rightarrow u) \vee \mathbf{K}(v \wedge \neg u) \equiv \mathbf{K}v \vee \mathbf{K}(v \rightarrow u)$; $Reg(\Phi, \beta) \equiv \mathbf{K}v \vee \mathbf{K}\neg v \vee \mathbf{K}u \vee \mathbf{K}\neg u$; $Reg(Reg(\Phi, \alpha), \beta) \equiv \mathbf{K}(u \rightarrow v) \vee \mathbf{K}(v \rightarrow u)$. Applying Proposition 3 we get $Reg(\Phi, \gamma) \equiv \Phi$, $Reg(Reg(\Phi, \alpha), \gamma) \equiv \mathbf{K}v \vee \mathbf{K}(v \rightarrow \neg u)$; $Reg(Reg(\Phi, \beta), \gamma) \equiv Reg(\Phi, \beta)$; $Reg(Reg(Reg(\Phi, \alpha), \beta), \gamma) \equiv \mathbf{K}(u \vee v) \vee \mathbf{K}(\neg u \vee \neg v)$.

5 Plan generation

Definition 6 (planning problems) A planning problem $\mathcal{P}$ w.r.t. a propositional action language $\mathcal{L}_A$ consists of an SKS $\Phi = \mathbf{K}\varphi_{init}$ describing the initial knowledge state of the agent, a finite set of actions $\mathcal{A} = \mathcal{A}_O$ (ontic) $\cup \mathcal{A}_E$ (epistemic) and a CKS Γ describing the goal. Effects of ontic actions are described in $\mathcal{L}_A$.

The reason why Γ is a CKS is that it is not sufficient to reach the goal, it must also be the case that the goal is known to be reached. Γ may be purely epistemic goal such as $\mathbf{K}v \vee \mathbf{K}\neg v$, i.e., an agent may have the ultimate goal to know whether φ holds or not.

Plans are defined inductively as follows:

- the empty plan λ is a plan;
- any action (ontic or epistemic) is a plan;
- if π and π' are plans then $\pi; \pi'$ is a plan;
- if π and π' are plans and Φ a CKS, then $\text{if } \Phi \text{ then } \pi \text{ else } \pi'$ is a plan.

Therefore, a plan can be seen as a program without loops, whose branching conditions are *epistemic formulas*: the agent can decide whether she *knows* that a given objective formula is true (whereas she is not always able to decide whether a given objective formula is true in the actual world).

While CKS are relevant for off-line planning, i.e., for reasoning about the possible effects of a plan, they are no longer relevant for representing knowledge during plan execution, since at each time step the agent is in exactly one knowledge state.

Definition 7 (progression of a CKS by a plan)

The progression of a CKS Φ in EDNF by a plan is defined inductively by

1. $Prog(\Phi, \lambda) = \Phi$;
2. $Prog(\Phi, (\pi; \pi')) = Prog(Prog(\Phi, \pi), \pi')$;
3. if $\Phi = \mathbf{K}\varphi$, then $Prog(\Phi, \text{if } \Psi \text{ then } \pi \text{ else } \pi') = \begin{cases} Prog(\Phi, \pi) & \text{if } \Phi \models \Psi \\ Prog(\Phi, \pi') & \text{if } \Phi \not\models \Psi \end{cases}$;
4. if $\Phi = \mathbf{K}\varphi_1 \vee \dots \vee \mathbf{K}\varphi_n$, then $Prog(\Phi, \text{if } \Psi \text{ then } \pi \text{ else } \pi') = \bigvee_{i=1..n} Prog(\mathbf{K}\varphi_i, \text{if } \Psi \text{ then } \pi \text{ else } \pi')$.

Definition 8 (valid plans) A plan π is a valid plan for the planning problem $\mathcal{P}$ if and only if $\text{Prog}(\Phi, \pi) \models \Gamma$.

Example 2 (cont'd) Initially, the agent does not know the values of u and v (her initial knowledge state is $\mathbf{K}\top$), and her goal is to reach a belief state where she knows the value of v : $\Gamma = \mathbf{K}v \vee \mathbf{K}\neg v$.

Let $\pi = \beta; \text{if } \mathbf{K}(u \leftrightarrow v) \text{ then } \alpha \text{ else } (\gamma; \alpha)$ be a plan. We have:

$$\begin{aligned} & \text{Prog}(\mathbf{K}\top, \pi) \\ &= \text{Prog}(\text{Prog}(\mathbf{K}\top, \beta), \text{if } \mathbf{K}(u \leftrightarrow v) \text{ then } \alpha \text{ else } (\gamma; \alpha)) \\ &= \text{Prog}(\mathbf{K}(u \leftrightarrow v), \alpha) \vee \text{Prog}(\mathbf{K}(u \leftrightarrow \neg v), (\gamma; \alpha)) \\ &= \mathbf{K}(u \wedge v) \vee \mathbf{K}(\neg u \wedge \neg v) \end{aligned}$$

Therefore, π is valid.

A valid plan can be computed by the following backward algorithm based on goal regression which is reminiscent of dynamic programming. The current goal Φ_C , expressed in EDNF, is initialized as Γ . Then we nondeterministically pick an action α and compute $\text{Reg}(\Phi_C, \alpha)$. The current goal is then updated by $\Phi_C := \Phi_C \vee \text{Reg}(\Phi_C, \alpha)$. The process is iterated until $\mathbf{K}\varphi_{\text{init}} \models \Phi_C$ or it is not possible to improve Φ_C anymore. Since there is a finite number of possible belief states, the algorithms stop and returns a valid plan, if such a plan exists. An ordered list L is constantly updated, initialized by $\langle \langle \mathbf{K}G_1, \lambda \rangle, \dots, \langle \mathbf{K}G_m, \lambda \rangle \rangle$ where $\Gamma \equiv \mathbf{K}G_1 \vee \dots \vee \mathbf{K}G_m$; each time a new disjunct (i.e., not subsumed by any previous disjunct of Φ_C) $\mathbf{K}\psi$ is added to Φ_C after regressing by action α , the pair $\langle \mathbf{K}\psi, \alpha \rangle$ is added to L .

There are two slightly different possible outputs: (1) either the output is just L , i.e., an *ordered knowledge-based program* (or decision list): at execution time, when observations are made, the new knowledge state is computed, then we look for the leftmost $\langle \mathbf{K}\psi, \alpha \rangle$ in L such that $\mathbf{K}\psi$ is true in the current knowledge state and α is performed; (2) or the output is a *ready-to-use conditional plan* computed by “simulating” possible executions from $\mathbf{K}\varphi_{\text{init}}$ to Γ using L .

Example 2 (cont'd) Regressing by α , γ and then β , the algorithm finds a valid knowledge-based program for Γ :

$L = \langle \langle \mathbf{K}v, \lambda \rangle, \langle \mathbf{K}\neg v, \lambda \rangle, \langle \mathbf{K}(v \rightarrow u), \alpha \rangle, \langle \mathbf{K}(v \rightarrow \neg u), \gamma \rangle, \langle \mathbf{K}\top, \beta \rangle \rangle$. An associated conditional plan is $\pi = \beta; \text{if } \mathbf{K}(u \leftrightarrow v) \text{ then } \alpha \text{ else } (\gamma; \alpha)$.

6 Related work

Knowledge-based programs In the planning community, the idea of using explicit knowledge preconditions for actions and plans comes back to [Moore, 1985; Morgenstern, 1987]. Developed in a different perspective (agent design), *knowledge-based programs* [Fagin et al., 1995; Brafman et al., 1998; Herzig et al., 2000; Reiter, 2001] are high-level protocols that describe the actions an agent should perform as a function of her knowledge. Thus, in a knowledge-based program, branching conditions are epistemically interpretable, and plans explicitly involve deduction tasks during on-line execution (just like in our framework). Actually, the output of our plan generation process is a *knowledge-based program*. Therefore, our work can be seen as an upstream task that generates a valid knowledge-based program from a compact specification of action effects and goals.

Action languages A number of works have extended action languages so as to handle explicit knowledge and partial observability, especially [Lobo et al., 1997; de Giacomo and Rosati, 1999; Baral and Son, 2001]. Knowledge is represented in all cases by an explicit or implicit epistemic modality (plus a “minimal knowledge” semantics in [de Giacomo and Rosati, 1999]). The line of work most related to ours is of [Baral and Son, 2001]; indeed, not only they represent epistemic actions with an epistemic modality but they also allow for conditional plans with epistemic branching conditions. Our work can be seen as an extension of theirs:⁹ (i) our formalism is general enough to accept any propositional action language (including those handling causal rules) for representing the effects of ontic actions; (ii) our syntax is less restricted, since we allow for any and-or combination of SKS (i.e., CKSs) while they consider SKS only; as argued in Section 3, this makes the representation more compact, when reasoning at planning time about the future consequences of actions; (iii) our progression and regression operators have significant computational characterizations (e.g., ontic regression has an abductive characterization); lastly, we have a sound and complete algorithm for plan generation.

Planning under partial observability There is a number of recent approaches for logic-based plan generation under partial observability.

[Bonet and Geffner, 1998] give a high-level language for describing action effects on both the world and the agent’s beliefs. Their language is a decidable fragment of first-order logic. By describing ontic actions with successor state axioms, they allow for handling the frame problem and ramification problems. After a problem has been represented in their language, its description is automatically translated into a POMDP model and solved by using POMDP algorithms, so that there is no need to define progression and regression directly in the logic, nor to have an explicit knowledge modality: this is the main difference with our approach, where the compact logical representation is kept and propagated throughout the process.

The next three approaches solve the plan generation problem directly in a high-level language but, on the other hand, they all make important restrictions that lead to a loss of expressivity. These restrictions imply that none of these approaches makes use of action languages, while ours can benefit from the huge amount of work in this area and accordingly, can handle the frame problem as well as ramification and causality in the best possible way while maintaining computational complexity at a reasonable level.

[Bacchus and Petrick, 1998; Petrick and Bacchus, 2002], like us, use an epistemic modality. Apart from the representation of ontic actions (less expressive than ours due to the abovementioned point¹⁰), they restrict the syntax of epis-

⁹Actually, only of the first part of [Baral and Son, 2001], since the second half of their paper gives a detailed study of sound and efficient approximations of their formalism. We plan to integrate similar approximations in our framework.

¹⁰On the other hand, they use a fragment of first-order logic which allows for expressing some actions (such as value tests) elegantly, and they motivate their expressivity restrictions by efficiency considerations, so that their approach is a good trade-off between effi-

temic formulas (for instance, simple disjunctive beliefs such as $\mathbf{K}(a \vee b)$ cannot be expressed) and consequently, as they notice, their algorithm sometimes fails to discover a valid plan.

The approaches [Bertoli *et al.*, 2001; Rintanen, 2002] do not make use of an epistemic modality, and therefore cannot explicitly express disjunctions of belief states (i.e., CKSs) or complex knowledge-based programs. The representation of belief states in both approaches uses BDDs, which allow for a compact representation but not as space efficient as DAG-based propositional formulas. While the algorithm in [Bertoli *et al.*, 2001] uses progression (based on model checking), [Rintanen, 2002] has a regression operator, and, interestingly enough, his combination operator $\otimes$ between belief states (which aims at computing, given two belief states B_1 and B_2 , the maximal belief states in which, after observing the values of observable variables, leads to know that the true state is in B_1 or to know that the true state is in B_2) can be reformulated using our epistemic regression (Section 4.2)¹¹ and thus epistemic logic helps understanding how and why this operator works.¹²

Situation calculus [Scherl and Levesque, 1993] represent sensing actions in the situation calculus by means of an explicit accessibility relation between situations (which means that knowledge is treated as an ordinary fluent) which corresponds exactly to the semantics of our epistemic modality (once situations have been identified with states). Our approach expresses the problem at the formula level and enables thus a more concise representation and can benefit from existing complexity and automated deduction results for S5. Levesque [Levesque, 1996] builds on the above framework towards a general theory of planning with sensing, handling complex plans involving, like ours, nondeterminism, observations and branching (and also loops).

Acknowledgements

The third author has been partly supported by the IUT de Lens, the Université d'Artois, the Région Nord / Pas-de-Calais under the TACT-TIC project, and by the European Community FEDER Program.

References

- [Bacchus and Petrick, 1998] F. Bacchus and R. Petrick. Modelling an agent's incomplete knowledge during planning and execution. In *KR-98*, pages 432–443, 1998.
- [Baral and Son, 2001] C. Baral and T. Son. Formalizing sensing actions – a transition function based approach. *Artificial Intelligence*, 125(1–2):19–91, 2001.

ciency and expressivity.

¹¹Indeed, introducing the action *observe* performed systematically after any ontic action (Rintanen does not consider real epistemic actions but assumes instead *automatic observability*: there is a set of variables whose value is observed after *any* action) and giving the truth value of each observable variable o_i , $B_1 \otimes B_2$ can be identified with $Reg(observe, \mathbf{K}for(B_1) \vee \mathbf{K}for(B_2))$ where $for(B)$ is a propositional formula such that $Mod(for(B)) = B$. More precisely, if $B \in B_1 \otimes B_2$ then $\mathbf{K}for(B)$ is one of the disjuncts of $Reg(observe, \mathbf{K}for(B_1) \vee \mathbf{K}for(B_2))$ after minimization.

¹²Note also that all approaches discussed in this section pay attention to efficiency (which may explain the restrictions made) and all of them have been implemented and report experimental results.

- [Bertoli *et al.*, 2001] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI-01*, pages 473–478, 2001.
- [Bonet and Geffner, 1998] B. Bonet and H. Geffner. High-level planning and control with incomplete information using POMDPs. In *AAAI Fall Symposium on POMDPs*, 1998.
- [Brafman *et al.*, 1998] R. Brafman, J. Halpern, and Y. Shoham. On the knowledge requirements of tasks. *Artificial Intelligence*, 98, 1998.
- [de Giacomo and Rosati, 1999] G. de Giacomo and R. Rosati. Minimal knowledge approach to reasoning about actions and sensing. *Electronic Transactions on Artificial Intelligence*, 3 (section C):1–18, 1999.
- [Fagin *et al.*, 1995] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.
- [Gelfond and Lifschitz, 1993] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.
- [Herzig *et al.*, 2000] A. Herzig, J. Lang, D. Longin, and Th. Polacsek. A logic for planning under partial observability. In *AAAI-00*, pages 768–773, 2000.
- [Ladner, 1977] R.E. Ladner. The computational complexity of provability in systems of modal propositional logics. *SIAM Journal of Computing*, 6:467–480, 1977.
- [Levesque, 1996] H. Levesque. What is planning in the presence of sensing? In *AAAI-96*, pages 1139–1146, 1996.
- [Lin and Reiter, 1994] F. Lin and R. Reiter. Forget it! In *AAAI Fall Symposium on Relevance*, pages 154–159, New Orleans, 1994.
- [Lin, 2000] F. Lin. From causal theories to successor state axioms and STRIPS-like systems. In *AAAI-00*, pages 786–791, 2000.
- [Lobo *et al.*, 1997] J. Lobo, G. Mendez, and S. R. Taylor. Adding knowledge to the action description language A. In *AAAI-97*, pages 454–459, 1997.
- [McCain and Turner, 1998] N. McCain and H. Turner. Satisfiability planning with causal theories. In *KR-98*, pages 212–223, 1998.
- [Moore, 1985] R.C. Moore. *A formal theory of knowledge and action*, chapter Formal Theories of the Commonsense World. Ablex, 1985.
- [Morgenstern, 1987] L. Morgenstern. Knowledge preconditions for actions and plans. In *IJCAI-87*, pages 867–874, 1987.
- [Petrick and Bacchus, 2002] R. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *AIPS-02*, pages 212–221, 2002.
- [Reiter, 1993] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1993.
- [Reiter, 2001] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.
- [Rintanen, 2002] J. Rintanen. Backward plan construction for planning with partial observability. In *AIPS-02*, pages 173–182, 2002.
- [Scherl and Levesque, 1993] R. B. Scherl and H. J. Levesque. The frame problem and knowledge-producing actions. In *AAAI-93*, pages 698–695, 1993.

Causal Theories of Action: A Computational Core

Jérôme Lang

IRIT/UPS
F-31062 Toulouse Cedex, France
lang@irit.fr

Fangzhen Lin

Hong Kong Univ of Sci and Tech
Dept of Computer Science
Clear Water Bay, KLN, Hong Kong
flin@cs.ust.hk

Pierre Marquis

CRIL/Université d'Artois
F-62307 Lens Cedex, France
marquis@cril.univ-artois.fr

Abstract

We propose a framework for simple causal theories of action, and study the computational complexity in it of various reasoning tasks such as determinism, progression and regression under various assumptions. As it turned out, even the simplest one among them, one-step temporal projection with complete initial state, is intractable. We also briefly consider an extension of the framework to allow truly indeterministic actions, and find that this extension does not increase the complexity of any of the tasks considered here.

1 Introduction

While there are abundance of formalisms for writing action theories that incorporate explicit causal rules not only between actions and fluents but also between fluents (e.g. [Baral, 1995; Lin, 1995; Thielscher, 1995; Gustafsson and Doherty, 1996; McCain and Turner, 1997; Zhang and Foo, 2001; Kakas *et al.*, 2001]), and there are some implementations of causal action theories [Lin, 2000; McCain and Turner, 1998; Kakas *et al.*, 2001], there have been few formal studies of complexities of various reasoning tasks in these causal action theories. In this paper, we investigate this issue. We first study various reasoning tasks such as computing the precondition of an action, checking if an action theory specifies a deterministic action, performing temporal projection, and computing regression in a very simple causal action framework that is primarily aimed at representing deterministic actions. Basically, an action theory for action α in this framework is just a finite set of action effect rules of the form

if $l_1 \wedge \dots \wedge l_n$ is true, then action α causes l to be true

and causal rules of the form

$l_1 \wedge \dots \wedge l_n$ causes l to be true

where l 's are literals.

Surprisingly, even in this simple framework, one-step temporal projection problem with complete initial state in the following form: given a set of literals that completely determines the initial state, and an action α , determine if a given fluent will hold in the state resulted from doing α in the initial state, is intractable. This compares with STRIPS and ADL,

which are like this simple framework except that they do not allow causal rules, where problems like temporal projection and regression are easy.

As simple as this framework may be, it nonetheless captures the computational core of causal action theories. Indeed, we show that if one extends it to allow arbitrary formulas in both the right and left hand sides of the effect and causal rules: (1) this extends the expressiveness of the language, but (2) it does not affect the complexities of any of the reasoning tasks considered here.

The rest of this paper is organized as follows. After some formal preliminaries (Section 2), we present a framework for simple action theories in Section 3. Complexity results are presented in Section 4. Before presenting some related work and concluding, we briefly show in Section 5 how our simple action theories can be generalized, without questioning the complexity results (in the general case). For space reasons, proofs are only sketched or omitted.

2 Formal preliminaries

Let $PROP_{PS}$ be a propositional language defined inductively from a finite set PS of propositional symbols (atoms), the boolean constants $\top$ (true) and $\perp$ (false), and the connectives $\neg$, $\vee$, $\wedge$, $\Rightarrow$ and $\Leftrightarrow$. L_{PS} is the set of all literals generated from PS . For each formula ϕ , $Var(\phi)$ denotes the set of atoms occurring in ϕ .

We shall also consider two isomorphic copies of $PROP_{PS}$: $PROP_{PS_t}$ and $PROP_{PS_{t+1}}$. Each formula ϕ_t (resp. ϕ_{t+1}) from $PROP_{PS_t}$ (resp. $PROP_{PS_{t+1}}$) is obtained by substituting in a uniform way in the formula $\phi \in PROP_{PS}$ every atom $p \in PS$ by the atom p_t (resp. p_{t+1}). Propositions indexed by t are used to express conditions about current situation, and those indexed by $t+1$ about the situation resulted from doing action α in the current situation. In the following, by an *initial state formula* we mean a formula from $PROP_{PS_t}$, and by a *successor state formula* we mean a formula from $PROP_{PS_{t+1}}$. Finally, we shall consider the language $PROP_{PS_t \cup PS_{t+1}}$ and take advantage of it to characterize the effects of action α .

A truth assignment over PS is called a *complete state*. A truth assignment over PS_t is called a *complete initial state*, while a truth assignment over PS_{t+1} is called a *complete successor state*. In order to avoid heavy notations, we shall identify each complete state s with the conjunction of literals s is the unique model of it.

3 Simple action theories

We shall first consider *simple causal action theories*.

Definition 1 (simple action theory) A simple action theory for action α is an ordered pair $T_\alpha = \langle \text{Eff}(\alpha), \text{Causal} \rangle$ where

- $\text{Eff}(\alpha)$ is a finite set of effect rules, of the following form:

$$l_1 \wedge \dots \wedge l_n \xrightarrow{\alpha} l,$$

where l 's are literals from L_{PS} , and l can also be $\perp$. The left side is understood to be a tautology when $n = 0$. When $l = \perp$, the effect rule is really an action precondition axiom about α . Notice that for our purpose in this paper, we assume that α is the only action.

- Causal is a finite set of causal rules of the following form:

$$l_1 \wedge \dots \wedge l_n \longrightarrow l,$$

where l 's are literals from L_{PS} . Again, the left side is understood to be a tautology when $n = 0$.

Such simple action theories are sufficient to encode actions with conditional effects. The ramification problem is addressed through causal rules, which represent static laws and express how fluents are related. While primarily targeted at representing deterministic actions, they can sometimes give rise to indeterministic effects when there are cyclic causal rules such as $p \longrightarrow p$.

Causal theories of this form are special cases of causal theories in situation calculus [Lin, 1995], domain descriptions in action languages [Giunchiglia and Lifschitz, 1998], and other more general formalisms (e.g. [McCain and Turner, 1997; Zhang and Foo, 2001]). While these formalisms do not always agree on their semantics in the general case, most of them coincide on this special class.

Definition 2 (completion) The completion $cl(T_\alpha)$ of a (simple) action theory T_α is the formula defined by

- $T(\alpha) = \bigwedge \{pre_t \Rightarrow post_{t+1} \mid \text{Eff}(\alpha) \text{ contains } pre \xrightarrow{\alpha} post\}$;
- $T(\text{Causal}) = \bigwedge \{left_t \Rightarrow right_t, left_{t+1} \Rightarrow right_{t+1} \mid \text{Causal} \text{ contains } left \longrightarrow right\}$;
- for each literal $l \in L_{PS}$, let
 $\Phi^\alpha(l) = \bigvee \{pre_t \mid \text{Eff}(\alpha) \text{ contains } pre \xrightarrow{\alpha} l\}$,
 $\Phi^C(l) = \bigvee \{left_{t+1} \mid \text{Causal} \text{ contains } left \longrightarrow l\}$,
 $\Phi(l) = \Phi^\alpha(l) \vee \Phi^C(l)$,
 $frame(l) = (l_t \wedge \neg \Phi(l)) \Rightarrow l_{t+1}$, where $\hat{l}$, the complement of l is defined as follows: for any atom p , $\hat{p} = \neg p$ and $\neg \hat{p} = p$;
- $cl(T_\alpha) = T(\alpha) \wedge T(\text{Causal}) \wedge \bigwedge_{l \in L_{PS}} frame(l)$.

We assume that T_α mentions each atom of PS (otherwise some atoms of PS would be useless); under this assumption, the size of $cl(T_\alpha)$ is polynomial in the size of T_α .

Given a simple action theory T_α , a truth assignment over $PROP_{PS_t \cup PS_{t+1}}$ is said to be a *model* of T_α if and only if it satisfies $cl(T_\alpha)$. Each model of T_α is formed by the union (s_t, s'_{t+1}) of a complete initial state s_t and a complete successor state s'_{t+1} .

As one can see, the construction of $cl(T_\alpha)$ from T_α is similar to the way a causal theory is transformed to classical logic in [Lin, 1995] and to literal completions [McCain and Turner, 1997; Lifschitz, 1997]. In fact, all these approaches yield logically equivalent theories.

4 Complexity issues

4.1 Executability and determinism

Several properties are of interest when dealing with action theories. First of all, there is the question of whether an action is executable in a given initial situation.

Definition 3 (executability) Let T_α be an action theory for action α and let φ be a formula from $PROP_{PS}$. $\langle T_\alpha, \varphi \rangle$ is a positive instance of EXECUTABILITY (the executability problem) if and only if for any complete state s that satisfies φ , there is a state s' such that $(s, s') \models cl(T_\alpha)$.

Whenever $\langle T_\alpha, \varphi \rangle$ is a positive instance of EXECUTABILITY, α is said to be *executable under* φ ; α is said to be *fully executable* if and only if it is executable under $\top$.

Proposition 1

1. In the general case, EXECUTABILITY is Π_2^P -complete.
2. Under the restriction where φ is a complete state, EXECUTABILITY is NP-complete.

Proof.

1. Membership is easy. Hardness comes from the following polynomial reduction from the Σ_2^P -complete problem $QBF_{2,\exists}$ to the problem of checking that α is not executable under $\top$. Let $Q = \langle \{a_1, \dots, a_n\}, \{b_1, \dots, b_p\}, \beta \rangle$ be an instance of $QBF_{2,\exists}$.¹ Wlog, we assume that β is a DNF formula $D_1 \vee \dots \vee D_m$. We now define $M(Q) = T_\alpha = \langle \emptyset, \text{Causal} \rangle$ where $\text{Causal} = \{D_i \longrightarrow new, D_i \longrightarrow \neg new \mid i = 1 \dots m\} \cup \{b_i \longrightarrow b_i, \neg b_i \longrightarrow \neg b_i \mid i = 1 \dots p\}$ and new is a new atom, not occurring in β . Q is a positive instance of $QBF_{2,\exists}$ if and only if α (as given by $M(Q)$) is not executable under $\top$.
2. Comes directly from the fact that, when φ is complete, then α is executable under it iff the propositional theory $\{\varphi\} \cup cl(T_\alpha)$ is consistent. ■

Another important property is *determinism*. Intuitively, an action is deterministic if there is at most one successor state corresponding to any initial state.

Definition 4 (determinism) An action theory T_α for action α is a positive instance of DETERMINISM (the determinism problem) if and only if α is deterministic, i.e., for all $s_t \in 2^{PS_t}$ and for all $s'_{t+1}, s''_{t+1} \in 2^{PS_{t+1}}$ such that $(s_t, s'_{t+1}) \models cl(T_\alpha)$ and $(s_t, s''_{t+1}) \models cl(T_\alpha)$, then $s'_{t+1} = s''_{t+1}$.

Proposition 2 DETERMINISM is coNP-complete.

¹Let us recall that Q is a positive instance of $QBF_{2,\exists}$ if and only if it is valid, i.e., there exists an assignment $\vec{a}$ of variables $a_1, \dots, a_n$ such that for all assignments $\vec{b}$ of variables $b_1, \dots, b_p$ we have $(\vec{a}, \vec{b}) \models \beta$ (where β is a formula from $PROP_{PS}$ s.t. $Var(\beta) = \{a_1, \dots, a_n, b_1, \dots, b_p\}$). $QBF_{2,\exists}$ is still Σ_2^P -complete when β is under DNF.

Proof: Membership is easy. Hardness comes from the following polynomial reduction from DNF-VALIDITY. Let $\xi = D_1 \vee \dots \vee D_m$ be any formula from $PROP_{PS}$ under DNF and let $M(\xi)$ be the instance of DETERMINISM defined by $T_\alpha = \langle Eff(\alpha), Causal \rangle$ with $Eff(\alpha) = \{D_1 \xrightarrow{\alpha} new, \dots, D_m \xrightarrow{\alpha} new\}$ and $Causal = \{new \rightarrow new, \neg new \rightarrow \neg new\}$, where new is a new atom not mentioned in ξ . ξ is valid if and only if $M(\xi)$ is deterministic. ■

4.2 Progression

There are two forms of the *progression* problem, also referred to as (one-step) *temporal projection*. The simpler one, in the form of a query about the effects of an action given some information about the current situation, can be described as a triple $\langle T_\alpha, \phi, \psi \rangle$ where

- T_α is an action theory;
- ϕ is a formula from $PROP_{PS}$ representing the knowledge state before the action is performed; we assume that the action α is executable under ϕ ;
- ψ is a formula from $PROP_{PS}$ we are interested in the truth value of which after the action has been performed.

Definition 5 (progression) Let T_α be an action theory and ϕ a formula from $PROP_{PS}$ such that α is executable under ϕ . Let ψ be a formula from $PROP_{PS}$. $\langle T_\alpha, \phi, \psi \rangle$ is a positive instance of PROGRESSION (the progression problem) if and only if ψ_{t+1} holds in any possible complete successor state by action α of any complete initial state s_t satisfying ϕ_t ; equivalently:

$$\phi_t \wedge cl(T_\alpha) \models \psi_{t+1}.$$

According to this definition, progression is really a two-step process: first make sure that the action is executable under the given condition about the initial state, and then compute the changes that the action will have under the given initial condition. We have shown that the first step is Π_2^P -hard in the worst case. The following proposition shows the complexity of the second step under various assumptions.

Proposition 3 We have identified the following complexity results (“coNP-c” stands for coNP-complete).

Complexity of PROGRESSION	with causal rules	no causal rules
any ϕ , any ψ	coNP-c	coNP-c
ϕ complete state	coNP-c	P
ψ literal	coNP-c	coNP-c
ϕ complete state + ψ literal	coNP-c	P

Proof:

1. Membership to coNP comes easily from the fact that $\langle T_\alpha, \phi, \psi \rangle$ is a positive instance of PROGRESSION if and only if $\phi_t \wedge cl(T_\alpha) \models \psi_{t+1}$ holds.
2. In the case where ϕ_t is a complete initial state and there are no causal rules, $cl(T_\alpha) = \{(l_i)_{t+1} \Leftrightarrow \chi_{i,t} \mid l_i \in L_{PS} \cup \{\perp\}\}$, where for each i , $Var(\chi_{i,t}) \subseteq PS_t$. When ϕ_t is a complete initial state, each $\chi_{i,t}$ can be evaluated in linear time to a truth value. This truth value is given to the corresponding $(l_i)_{t+1}$. Since α is executable under ϕ , this will not lead to a contradiction, and a complete successor state s_{t+1} is characterized by the values given to the literals $(l_i)_{t+1}$. It remains to check in linear time whether $s_{t+1} \models \psi_{t+1}$ to determine whether the instance is positive or not.

3. What remains to be done is showing coNP-hardness in these two cases:

- (a) there are no causal rules and ψ is a literal;
- (b) ϕ is a complete state and ψ is a literal;

Case 1: no causal rules, ψ is a literal.

The proof comes from the following polynomial reduction from DNF-VALIDITY. Let $\xi = D_1 \vee \dots \vee D_m$ be any DNF formula and let $M(\xi)$ be the instance of PROGRESSION defined by

- $T_\alpha = \langle Eff(\alpha), Causal \rangle$ where $Eff(\alpha) = \{D_1 \xrightarrow{\alpha} new, \dots, D_m \xrightarrow{\alpha} new\}$ and $Causal = \emptyset$;
- $\phi = \top$; $\psi = new$, where new is a new atom not mentioned in ξ .

Clearly α is executable under $\top$, and ξ is valid iff $M(\xi)$ is a positive instance of PROGRESSION.

Case 2: ϕ is a complete state, ψ is a literal

The proof comes from the following polynomial reduction from DNF-VALIDITY. Let $\xi = D_1 \vee \dots \vee D_m$ be any DNF formula, and let $Var(\xi) = \{x_1, \dots, x_p\}$ be the set of variables appearing in ξ . We then define $M(\xi) = \langle T_\alpha, \phi, \psi \rangle$ where $T_\alpha = \langle Eff(\alpha), Causal \rangle$ is given by $Eff(\alpha) = \{\top \xrightarrow{\alpha} new'\}$ and

$$Causal = \{ D_i \wedge new' \rightarrow new \mid i = 1 \dots m \} \cup \{ x_i \rightarrow x_i, \neg x_i \rightarrow \neg x_i \mid i = 1 \dots p \}$$

and new, new' are new atoms (not appearing in ξ). Let $\phi = s$, s being any complete state satisfying $\neg new \wedge \neg new'$, and $\psi = new$. It can be seen that α is executable in (under) s , and ξ is valid iff $M(\xi)$ is a positive instance of PROGRESSION. ■

Our results about the complexity of executability and progression, taken together, are strongly related to a result in [Turner, 2002] (namely Theorem 8(i)) which says that one-stage conformant planning (without concurrency) is Π_2^P -complete. Indeed, checking that α is a valid plan in this context amounts to checking that α is executable in all possible initial states and that its progression satisfies the goal. By considering executability and progression separately, we see more clearly that Π_2^P -hardness in Turner’s result is solely due to the hardness of executability.

A second, perhaps a more difficult way of seeing the progression problem, is the following: given some information about the current situation, compute all possible successor states. Formally, this looks like consequence finding: given a formula $\phi \in PROP_{PS}$, compute the strongest successor state formula $\Psi \in PROP_{PS}$ such that $\phi_t \wedge cl(T_\alpha) \models \Psi_{t+1}$. Model-theoretically, this corresponds to finding all complete successor states s'_{t+1} such that there is a complete initial state s_t that satisfies ϕ_t for which $s_t \wedge cl(T_\alpha) \wedge s'_{t+1}$ is consistent. As it turns out, this formula Ψ_{t+1} is the strongest necessary condition of ϕ_t under $cl(T_\alpha)$ on $PROP_{PS_{t+1}}$ [Lin, 2001]:

Definition 6 ($Pro(T_\alpha, \phi)$) Given an action theory T_α and a formula ϕ from $PROP_{PS}$ such that α is executable under ϕ , the progression formula $Pro(T_\alpha, \phi)$ is the formula Ψ from $PROP_{PS}$ (unique up to logical equivalence) such that $\langle T_\alpha, \phi, \psi \rangle$ is a positive instance of PROGRESSION if and only if $\Psi \models \psi$ holds.

$Pro(T_\alpha, \phi)$ can be characterized equivalently by:

Proposition 4

1. Let s be any complete state. We have $s \models \text{Pro}(T_\alpha, \phi)$ iff $\exists s'_t \in 2^{PS_t}, ((s'_t, s_{t+1}) \models \text{cl}(T_\alpha) \text{ and } s'_t \models \phi_t)$, i.e., there exists a complete initial state s'_t such that $s'_t \models \phi_t$ and $(s'_t, s_{t+1}) \models \text{cl}(T_\alpha)$ (or equivalently, $\phi_t \wedge \text{cl}(T_\alpha) \wedge s_{t+1}$ is consistent).
2. $\text{Pro}(T_\alpha, \phi)_{t+1} \equiv \exists PS_t. (\phi_t \wedge \text{cl}(T_\alpha))$.

4.3 Regression

There are (at least) two possible definitions for *regression*, each of which corresponds to a given need: *deductive regression* (also referred to as *temporal explanation* or *weak preimage*) and *abductive regression* (also referred to as *strong preimage*).

Definition 7 (deductive regression) Let T_α be an action theory and let ϕ, ψ be two formulas from PROP_{PS} . $\langle T_\alpha, \phi, \psi \rangle$ is a positive instance of DEDUCTIVE REGRESSION (the deductive regression problem) if and only if

$$\psi_{t+1} \wedge \text{cl}(T_\alpha) \models \phi_t.$$

Similar to progression formulas, we can define deductive regression formulas:

Definition 8 ($\text{Reg}_D(T_\alpha, \psi)$) Given an action theory T_α and a formula ψ from PROP_{PS} , the deductive regression formula $\text{Reg}_D(T_\alpha, \psi)$ is the formula Φ from PROP_{PS} (unique up to logical equivalence) such that $\langle T_\alpha, \phi, \psi \rangle$ is a positive instance of DEDUCTIVE REGRESSION if and only if $\Phi \models \psi$.

We immediately get that for any two formulas ϕ, ψ from PROP_{PS} , $\langle T_\alpha, \phi, \psi \rangle$ is a positive instance of DEDUCTIVE REGRESSION if and only if $\langle T_\alpha, \neg\phi, \neg\psi \rangle$ is a positive instance of PROGRESSION. We also have:

Proposition 5

1. Let s be any complete state. We have $s \models \text{Reg}_D(T_\alpha, \psi)$ if and only if $\exists s'_{t+1} \in 2^{PS_{t+1}}, ((s_t, s'_{t+1}) \models \text{cl}(T_\alpha) \text{ and } s'_{t+1} \models \psi_{t+1})$, i.e., there exists a complete successor state s'_{t+1} such that $s'_{t+1} \models \psi_{t+1}$ and $(s_t, s'_{t+1}) \models \text{cl}(T_\alpha)$ (or equivalently, $\psi_{t+1} \wedge \text{cl}(T_\alpha) \wedge s_t$ is consistent).
2. $\text{Reg}_D(T_\alpha, \psi)_t \equiv \exists PS_{t+1}. (\psi_{t+1} \wedge \text{cl}(T_\alpha))$.

This characterization helps understanding deductive regression: it intuitively means that we are interested in finding the set of states which could be **possible** states before the action, knowing that ψ holds after the action. This type of regression is useful for *postdiction*, i.e., reasoning about the past state of the system.

Since deductive regression is expressed as a deduction problem, its complexity is easy to find out.

Proposition 6 DEDUCTIVE REGRESSION is coNP-complete.

Definition 9 (abductive regression) Let T_α be an action theory and let ϕ, ψ be two formulas from PROP_{PS} . $\langle T_\alpha, \phi, \psi \rangle$ is a positive instance of ABDUCTIVE REGRESSION (the abductive regression problem) if and only if for any complete state s , $s_t \wedge \text{cl}(T_\alpha) \models \psi_{t+1}$ implies $s \models \phi$.

The corresponding search problem is defined by:

Definition 10 Given an action theory T_α and a formula ψ from PROP_{PS} , the abductive regression formula $\text{Reg}_A(T_\alpha, \psi)$ is the formula Φ (unique up to logical equivalence) such that $\langle T_\alpha, \phi, \psi \rangle$ is a positive instance of ABDUCTIVE REGRESSION if and only if $\phi \models \Phi$.

In other words, Φ_t is the weakest sufficient condition of ψ_{t+1} under $\text{cl}(T_\alpha)$ on PROP_{PS_t} [Lin, 2001].

Proposition 7

1. Let s be any complete state. $s \models \text{Reg}_A(T_\alpha, \psi)$ if and only if $\forall s'_{t+1} \in 2^{PS_{t+1}}, \text{ if } (s_t, s'_{t+1}) \models \text{cl}(T_\alpha) \text{ then } s'_{t+1} \models \psi_{t+1}$ (or, equivalently, $s_t \wedge \text{cl}(T_\alpha) \models \psi_{t+1}$).
2. $\text{Reg}_A(T_\alpha, \psi)_t \equiv \forall PS_{t+1}. (\text{cl}(T_\alpha) \Rightarrow \psi_{t+1})$.

Thus, abductive regression amounts to finding the set of all complete initial states s_t for which all possible complete successor states s'_{t+1} w.r.t. T_α satisfy ψ . This means that we are interested in finding the set of states in which performing the action leads to states **necessarily** satisfying ψ - provided that the action can be performed in them. This type of regression is useful for *planning*, i.e., reasoning about the minimal conditions under which an action succeeds in reaching the goals. Thus, $\langle T_\alpha, \phi, \psi \rangle$ is a positive instance of ABDUCTIVE REGRESSION if and only if ϕ implies the minimal conditions under which the action leads to the goal.

This shows that the qualifications “deductive” and “abductive” are only related to the way initial states can be inferred, and not to the confidence we have in them to be satisfied. Indeed, as far as reasoning is concerned, deductive conclusions can be taken for sure since deduction is truth-preserving while abductive conclusions cannot be taken for sure in general since abduction is only falsity-preserving. Contrariwise to such a reasoning situation, initial states s_t obtained through deductive regression are only possible ones given ψ and α : it could be the case that performing action α in s leads to a successor state in which ψ does not hold. Contrastingly, initial states s_t obtained through abductive regression lead to successor states where ψ necessarily holds.

The following proposition makes precise the links between both forms of regression:

Proposition 8

1. $\text{Reg}_A(T_\alpha, \psi) \models \text{Reg}_D(T_\alpha, \psi)$ if and only if α is fully executable.
2. $\text{Reg}_A(T_\alpha, \psi) \equiv \text{Reg}_D(T_\alpha, \psi)$ if and only if α is fully executable and deterministic.

Proof: We give the proof of the first point:

$$\begin{aligned} & \text{Reg}_A(T_\alpha, \psi) \models \text{Reg}_D(T_\alpha, \psi) \\ \text{iff } & \forall PS_{t+1}. (\neg \text{cl}(T_\alpha)) \text{ is inconsistent} \\ \text{iff } & \exists PS_t. (\forall PS_{t+1}. (\neg \text{cl}(T_\alpha))) \text{ is inconsistent} \\ \text{iff } & \forall PS_t. (\exists PS_{t+1}. \text{cl}(T_\alpha)) \text{ is valid} \\ \text{iff } & \alpha \text{ is fully executable.} \end{aligned}$$

Abductive regression is computationally more expensive than deductive regression:

Proposition 9 ABDUCTIVE REGRESSION is Π_2^P -complete.

Proof: Membership is easy. Hardness comes from the following polynomial reduction from the Σ_2^P -complete problem QBF_{2,3} to ABDUCTIVE REGRESSION. Let $Q = \langle \{a_1, \dots, a_n\}, \{b_1, \dots, b_p\}, \beta \rangle$ be an instance of QBF_{2,3}. Wlog, we assume that β is a DNF formula $D_1 \vee \dots \vee D_m$. We now define $M(Q) = \langle T_\alpha, \phi, \psi \rangle$ where

- T_α is concerned by the fluents built over $\{a_1, \dots, a_n, b_1, \dots, b_p, new\}$ (where *new* is a new atom, not appearing in β) and the following set of static causal rules:

$$\begin{aligned} & \{ D_i \longrightarrow new \mid i = 1 \dots m \} \\ & \bigcup \{ b_i \longrightarrow b_i \mid i = 1 \dots p \} \\ & \bigcup \{ \neg b_i \longrightarrow \neg b_i \mid i = 1 \dots p \} \end{aligned}$$

(and no effect rule);

- $\phi = new$;
- $\psi = new$.

Q is a positive instance of $QBF_{2,\exists}$ iff $M(Q)$ is a positive instance of ABDUCTIVE REGRESSION. ■

5 Generalized action theories

Here we want to extend causal and effect rules so as to allow for any possible consequents, *including disjunctions*, while keeping the framework simple enough.

Definition 11 (generalized action theory) A generalized action theory T_α for action α is an ordered pair $\langle Eff(\alpha), Causal \rangle$ where

- $Eff(\alpha)$ is a finite set of effect rules of the form $pre \xrightarrow{\alpha} post[L]$ where *pre*, *post* are formulas from $PROP_{PS}$ and $L \subseteq L_{PS}$.
- $Causal$ is a finite set of causal rules of the form $left \longrightarrow right[L]$ where *left* and *right* are formulas from $PROP_{PS}$ and $L \subseteq L_{PS}$.

The completion consists now in writing down that every literal l persists if and only if there is no effect (action or causal) whose precondition is verified and on which l depends. More generally, rather than writing “there is no effect (action or causal) whose precondition is verified and on which l depends”, we specify for each action rule the fluents L that are influenced and for which there should not be any frame axiom if the action rule is enabled.

Definition 12 (completion) The completion $cl(T_\alpha)$ of generalized action theory T_α is the formula defined by

- $T(\alpha) = \bigwedge \{ pre_t \Rightarrow post_{t+1} \mid Eff(\alpha) \text{ contains } pre \xrightarrow{\alpha} post[L] \}$;
- $T(Causal) = \bigwedge \{ left_t \Rightarrow right_t, left_{t+1} \Rightarrow right_{t+1} \mid Causal \text{ contains } left \longrightarrow right[L] \}$;
- for each literal $l \in L_{PS}$, let

$$\Phi^\alpha(l) = \bigvee \{ pre_t \mid Eff(\alpha) \text{ contains } pre \xrightarrow{\alpha} post[L] \text{ and } l \in L \},$$

$$\Phi^C(l) = \bigvee \{ left_{t+1} \mid Causal \text{ contains } left \longrightarrow right[L] \text{ and } l \in L \},$$

$$\Phi(l) = \Phi^\alpha(l) \vee \Phi^C(l),$$

$$frame(l) = (l_t \wedge \neg \Phi(l)) \Rightarrow l_{t+1};$$
- $cl(T_\alpha) = T(\alpha) \wedge T(Causal) \wedge \bigwedge_{l \in L_{PS}} frame(l)$.

As for simple action theories, we assume that each symbol of PS occurs in T_α so that the size of $cl(T_\alpha)$ is polynomial in the size of T_α .

In the previous definition, $\Phi^\alpha(l)$ gathers the preconditions of effect rules whose postcondition influences positively l (and similarly for $\Phi^C(l)$); the completion $frame(l)$ means

that if l is initially true and there is no active action rule nor any causal rule whose consequent part influences negatively l , then l persists after α is performed.

By default, the set L associated with an action rule (resp. a causal rule) is the set of literals $Lit(post)$ (resp. $Lit(right)$) positively mentioned in the negation normal form of *post* (resp. of *right*) – an alternative, more refined possibility being the set of literals $DepLit(post)$ (resp. $DepLit(right)$) on which *post* (resp. *right*) semantically depends [Lang and Marquis, 1998]. It is easy to show that the completion of T_α as defined above (Definition 12) is equivalent to the completion given at Definition 2 whenever T_α is simple (in that case, the default choice for L in each rule is considered, i.e., $L = \{l\}$ whenever the consequent part of the rule is l).

For instance, so to say that flipping a coin release *head*, we simply write $holdingCoin \xrightarrow{flip} true[head, \neg head]$, i.e., if one is holding the coin initially, then after the action of flipping it, we can neither infer *head* nor $\neg head$ using inertia.

Due to space limitation, we cannot provide the full details here. In a nutshell, the main reasons why generalized action theories are interesting are twofold. On the one hand, we can prove that generalized action theories can be used to represent any nondeterministic action (associating a nonempty set of successor states to any initial state), while such a completeness property is not satisfied by simple action theories. On the other hand, the complexity of EXECUTABILITY, DETERMINISM, PROGRESSION, DEDUCTIVE/ABDUCTIVE REGRESSION, from generalized action theories coincide with the corresponding complexity result for simple action theories in the general case. This shows that the gain in expressiveness offered by generalized action theories is not balanced by a complexity increase for any of these reasoning tasks. Furthermore, this suggests that the simple action theories considered here constitute the computational core of causal action theories.

6 Other related work

[Liberatore, 1997] investigates the complexity of reasoning about action in the language A [Gelfond and Lifschitz, 1993], therefore he considers only deterministic actions without static causal rules. He shows the $coNP$ -completeness of the progression problem in language A , to be related to the right-upmost square in our Proposition 3. [Eiter *et al.*, 2001] study the computational of many planning problems (including the progression problem as a particular case) using an action description language based on answer set semantics.

[Drakengren and Bjärelund, 1997; 1999] investigate the complexity of checking the consistency of a scenario description in a temporal logic in the style of [Sandewall, 1994] which shares some similarities with causal theories of action, although both the syntax and the semantics are different. No static causal rules are considered (but on the other hand, the language allows for explicit time and concurrency). Checking the consistency of a scenario description is NP -complete and falls in P under specific syntactical restrictions.

Reasoning about action has strong connections with *belief update*, and therefore, the complexity study of belief update operators is relevant to our concern. The update operators whose complexity is studied in [Eiter and Gottlob, 1992;

Herzig and Rifi, 1999; Liberatore, 2000] do not consider static causal rules, but allow for expressing any effects, especially disjunctions. The complexity of checking whether $\psi \in KB \diamond \varphi$ – where $\diamond$ is the update operator, mapping a logical theory and a formula to a logical theory – corresponds to the progression problem².

7 Conclusions and future work

The main contribution of this paper is the identification of complexity results corresponding to many reasoning tasks considered when dealing with causal action theories. It remains to be checked to what extent these results are changed when considering additional capabilities such as concurrency (as in [Giunchiglia and Lifschitz, 1998]) or resources.

Because of the way the progression problem is stated, it was expected that for arbitrary description φ of the initial situation and arbitrary query ψ about the successor situation, the problem is **coNP**-complete. But when we first started studying the problem, we were really expecting that when φ is a complete state, the problem would be easier; we were surprised that even in this case it turned out to be intractable. In retrospect, what happens is that even though both action effect rules and causal rules have very restricted form, the complete action theory $cl(T_\alpha)$ can be complex, and may not always be deterministic. An interesting question is then if we already know that the action theory is deterministic, would progression with complete initial state still be intractable? We do not know the answer at that stage.

Acknowledgements

The authors would like to thank the anonymous referees for helpful comments. The second author's work was supported in part by HK RGC CERG HKUST6182/01E. The third author has been partly supported by the IUT de Lens, the Université d'Artois, the Région Nord / Pas-de-Calais under the TACT-TIC project, and by the European Community FEDER Program.

References

- [Baral, 1995] C. Baral. Reasoning about actions: nondeterministic effects, constraints, and qualification. In *IJCAI-95*, pages 2017–2023, 1995.
- [Drakengren and Bjärelund, 1997] T. Drakengren and M. Bjärelund. Reasoning about action in polynomial time. In *IJCAI-97*, pages 1447–1453, 1997.
- [Drakengren and Bjärelund, 1999] T. Drakengren and M. Bjärelund. Expressive reasoning about action in nondeterministic polynomial time. In *IJCAI-99*, pages 161–171, 1999.
- [Eiter and Gottlob, 1992] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence*, 57:227–270, 1992.
- [Eiter *et al.*, 2001] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A logic programming approach to knowledge-state planning: Semantics and complexity. Technical report, Tech. University Wien, 2001.
- [Gelfond and Lifschitz, 1993] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, pages 301–322, 1993.
- [Giunchiglia and Lifschitz, 1998] E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: Preliminary report. In *AAAI-98*, pp. 623–630, 1998.
- [Gustafsson and Doherty, 1996] J. Gustafsson and P. Doherty. Embracing occlusion in specifying the indirect effects of actions. In *KR-96*, pages 87–98, 1996.
- [Herzig and Rifi, 1999] A. Herzig and O. Rifi. Propositional belief update and minimal change. *Artificial Intelligence*, 115:107–138, 1999.
- [Kakas *et al.*, 2001] A. C. Kakas, R. Miller, and F. Toni. E-RES: reasoning about actions, events and observations. In *LPNMR-01*, pages 254–266, 2001.
- [Lang and Marquis, 1998] J. Lang and P. Marquis. Complexity results for independence and definability in propositional logic. In *KR-98*, pages 356–367, 1998.
- [Liberatore, 1997] P. Liberatore. The complexity of the language A. *Electronic Transactions on Artificial Intelligence*, 1:13–38, 1997.
- [Liberatore, 2000] P. Liberatore. The complexity of belief update. *Artificial Intelligence*, 119:141–190, 2000.
- [Lifschitz, 1997] V. Lifschitz. On the logic of causal explanation. *Artificial Intelligence*, 96:451–465, 1997.
- [Lin, 1995] F. Lin. Embracing causality in specifying the indirect effects of actions. In *IJCAI-95*, pages 1985–1993, 1995.
- [Lin, 2000] F. Lin. From causal theories to successor state axioms and STRIPS-like systems. In *AAAI-00*, pages 786–791, 2000.
- [Lin, 2001] F. Lin. On the strongest necessary and weakest sufficient conditions. *Artificial Intelligence*, 128:143–159, 2001.
- [McCain and Turner, 1997] N. McCain and H. Turner. Causal theories of action and change. In *AAAI-97*, pages 460–465, 1997.
- [McCain and Turner, 1998] N. McCain and H. Turner. Satisfiability planning with causal theories. In *KR-98*, pages 212–223, 1998.
- [Sandewall, 1994] E. Sandewall. *Features and Fluents*. Oxford University Press, 1994.
- [Thielscher, 1995] M. Thielscher. Computing ramification by post-processing. In *IJCAI-95*, pages 1994–2000, 1995.
- [Turner, 2002] H. Turner. Polynomial-length planning spans the polynomial hierarchy. In *JELIA-02*, pages 111–124, 2002.
- [Zhang and Foo, 2001] D. Zhang and N. Foo. EPDL: a logic for causal reasoning. In *IJCAI-01*, pages 131–138, 2001.

²Note that the fact that the complexity of update lies most often at the second level of the polynomial hierarchy (therefore more complex than progression in action theories) is not due to the presence of disjunctions but is due mainly to the minimization-based semantics of the update operators considered.

Describing Additive Fluents in Action Language $\mathcal{C}+$

Joohyung Lee and Vladimir Lifschitz

Department of Computer Sciences,
University of Texas, Austin, TX 78712
{appsmurf, vl}@cs.utexas.edu

Abstract

An additive fluent is a fluent with numerical values such that the effect of several concurrently executed actions on it can be computed by adding the effects of the individual actions. We propose a method for describing effects of actions on additive fluents in the declarative language $\mathcal{C}+$. An implementation of this language, called the Causal Calculator, can be used for the automation of examples of common-sense reasoning involving additive fluents.

1 Introduction

Action languages [Gelfond and Lifschitz, 1998] serve to describe effects of actions on the states of the world. For instance, the expression

$$Walk(x, l) \text{ causes } Location(x) = l \quad (1)$$

is a proposition, or “causal law,” of action language $\mathcal{C}+$ [Giunchiglia *et al.*, 2003] that describes an effect of action $Walk(x, l)$: this action causes the location of x to become l . The semantics of action languages is defined in terms of “transition systems”—directed graphs whose vertices correspond to the states of the world, and whose edges correspond to the execution of actions. The execution of a sequence of actions can be represented by a path in such a graph.

Some action languages, including $\mathcal{C}+$, allow us to talk about the effects of the concurrent execution of actions. Causal law (1) is understood in $\mathcal{C}+$ to imply that $Location(x) = l$ holds after any event that involves the execution of $Walk(x, l)$, even if other actions are executed concurrently. To distinguish the events involving the concurrent execution of actions a_1 and a_2 from the events that involve a_1 but not a_2 , we can write

$$\begin{aligned} a_1 \wedge a_2 &\text{ causes } \dots, \\ a_1 \wedge \neg a_2 &\text{ causes } \dots \end{aligned}$$

In this paper we investigate the possibility of using $\mathcal{C}+$ to represent the effects of actions on fluents of a special kind, called “additive.” An additive fluent is a fluent with numerical values such that the effect of several concurrently executed actions on it can be computed by adding the effects

of the individual actions. For example, the gross receipts of a store are represented by an additive fluent: when several customers pay to different cashiers simultaneously, the gross receipts will increase by the sum of the “contributions” of the individual customers. The voltage of a battery is an additive fluent: the increase in voltage obtained by adding several cells to a battery can be computed by addition. In mechanics, the velocity of a particle is an additive fluent, because the net effect of several forces on this fluent over a time interval equals the sum of the effects of the individual forces. Additive fluents are ubiquitous; this may be the reason why adding numbers is such a useful operation.

Unfortunately, the **causes** construct of $\mathcal{C}+$ and similar languages is not directly applicable to describing the effects of actions on additive fluents. Consider, for instance, the effect of the action $Buy(x, n)$ (customer x buys n books) on the number of books available at the bookstore. The causal law

$$Buy(x, n) \text{ causes } InStore = k - n \text{ if } InStore = k \quad (2)$$

is applicable in the case when no customer other than x is buying books at the same time: $k - n$ books are available after the event if there were k books in the store before the event. But (2) is not acceptable if we are interested in the concurrent execution of such actions.

We introduce here a syntactic construct, **increments**, that allows us to describe the effects of actions on additive fluents. Semantically this construct is treated as “syntactic sugar” on top of $\mathcal{C}+$: the propositions involving that construct are viewed as abbreviations for causal laws of $\mathcal{C}+$. The interpretation of **increments** described below has been used to extend the implementation of $\mathcal{C}+$, called the Causal Calculator (CCALC), to cover additive fluents.

After a review of transition systems and of the syntax of $\mathcal{C}+$ in the next two sections, we describe the syntax of increment laws (Section 4), define their semantics by showing how to treat them as abbreviations (Section 5), and illustrate the use of additive fluents in the language of CCALC by formalizing an example that involves buying and selling (Section 6). Two other examples of commonsense reasoning related to additive fluents are discussed in Sections 7 and 8. A proposition stated in Section 9 confirms that additive fluents behave as one would expect on the basis of the informal discussion of additivity above, and thus provide a justification for the approach to formalizing additive fluents proposed in this paper.

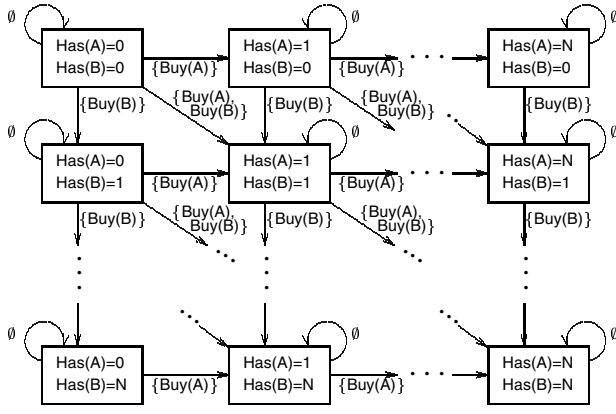


Figure 1: A transition system.

2 Transition Systems

Consider a set of symbols, called *constants*, along with a non-empty finite set of symbols $Dom(c)$ assigned to each constant c . We call $Dom(c)$ the *domain* of c .

The constants are divided into two groups—*fluent constants* and *action constants*. Intuitively, a fluent constant represents a fluent, and the elements of its domain are the possible values of that fluent. A state of the world is characterized by a function that maps each fluent constant to an element of its domain. In [Giunchiglia *et al.*, 2003, Section 4.2], fluent constants are further subdivided into *simple* and *statically determined*; in the examples below, only fluent constants of the first kind will be used.

A function that maps each action constant to an element of its domain characterizes an event occurring over a time interval between two successive states. In many examples of the use of $\mathcal{C}+$ the domain of every action constant is the set of truth values $\{f, t\}$; constants with this domain are called *Boolean*. Intuitively, a Boolean action constant represents an action; the value of such a constant is t if the action is one of those that are executed as part of the event.

A *transition system* is a directed graph whose vertices are functions that map every fluent constant to an element of its domain, and whose edges are labeled by functions that map every action constant to an element of its domain. The vertices of a transition system are called its *states*. The functions labeling the edges of a transition system are called *events*. As an example, consider a transition system representing the effect of buying a book on the number of books that the person owns (Figure 1). It uses two fluent constants— $Has(A)$ (the number of books that Alice has) and $Has(B)$ (the number of books that Bob has)—with the domain $\{0, \dots, N\}$, where N is a fixed nonnegative integer, and two Boolean action constants— $Buy(A)$ (Alice buys a book) and $Buy(B)$ (Bob buys a book). Every state is represented by two equations showing the values of the fluent constants. Every event is represented by the set of action constants that are mapped to t . The loops are labeled by the trivial event $\emptyset$ (no actions are executed). The horizontal edges are labeled by the event in which Alice buys a book and Bob doesn't; along each of the vertical edges, Bob buys a book and Alice doesn't. The

diagonal edges correspond to Alice and Bob buying books concurrently.

3 Syntax of $\mathcal{C}+$

The review of the syntax of $\mathcal{C}+$ in this section follows [Giunchiglia *et al.*, 2003, Section 4.2]. An *atom* is an expression of the form $c = v$, where c is a constant and $v \in Dom(c)$. For instance, the equalities in Figure 1 are atoms. A *formula* is a propositional combination of atoms. By a *fluent formula* we mean a formula such that all constants occurring in it are fluent constants. An *action formula* is a formula that contains at least one action constant and no fluent constants.

A *static law* is an expression of the form

$$\text{caused } F \text{ if } G \quad (3)$$

where F and G are fluent formulas. An *action dynamic law* is an expression of the form (3) in which F is an action formula and G is a formula. A *fluent dynamic law* is an expression of the form

$$\text{caused } F \text{ if } G \text{ after } H \quad (4)$$

where F and G are fluent formulas and H is a formula, provided that F does not contain statically determined constants. A *causal law* is a static law, or an action dynamic law, or a fluent dynamic law.

Here are two examples. The expression

$$Buy(x) \text{ causes } Has(x) = k + 1 \text{ if } Has(x) = k \quad (5)$$

where $x \in \{A, B\}$ and $k \in \{0, \dots, N - 1\}$, is an abbreviation for the fluent dynamic law

caused $Has(x) = k + 1$ **if** $\top$ **after** $Buy(x) = t \wedge Has(x) = k$ ($\top$ is the 0-place connective “true”). The expression

$$\text{nonexecutable } Buy(x) \text{ if } Has(x) = N \quad (6)$$

is an abbreviation for the fluent dynamic law

$$\text{caused } \perp \text{ if } \top \text{ after } Buy(x) = t \wedge Has(x) = N$$

($\perp$ is the 0-place connective “false”).

Causal laws (5) and (6) describe an effect of action $Buy(x)$ and a restriction on its executability. To get a complete description of Figure 1 in language $\mathcal{C}+$, we need two more postulates. First, we should specify that the edge labels of the transition system may assign truth values to the action constants $Buy(x)$ arbitrarily. This is expressed by

$$\text{exogenous } Buy(x), \quad (7)$$

which stands for the pair of action dynamic laws

$$\begin{aligned} \text{caused } Buy(x) = f & \text{ if } Buy(x) = f, \\ \text{caused } Buy(x) = t & \text{ if } Buy(x) = t. \end{aligned}$$

Action $Buy(x)$ is exogenous in the sense that the action description does not determine whether that action is executed. If it is not executed then there is a cause for this; if it is executed then there is a cause for that too. Second, we need to say that the fluent constant $Has(x)$ satisfies the “common-sense law of inertia”: in the absence of evidence to the contrary, its value after an event is assumed to be the same as its value before the event. This is expressed by

$$\text{inertial } Has(x), \quad (8)$$

which stands for $N + 1$ fluent dynamic laws

caused $Has(x) = k$ **if** $Has(x) = k$ **after** $Has(x) = k$

($k \in \{0, \dots, N\}$). If the value of $Has(x)$ after an event is the same as the value before the event then there is a cause for this. Intuitively, inertia is the cause.

An *action description* is a set of causal laws. The semantics of $\mathcal{C}+$, described in [Giunchiglia *et al.*, 2003, Section 4.4], specifies the transition system represented by any given action description. For instance, action description (5)–(8) represents the transition system shown in Figure 1.

This action description does not say explicitly that the trivial event $\emptyset$ has no effect on the values of $Has(A)$ and $Has(B)$, or that event $\{Buy(A)\}$ does not affect the value of $Has(B)$. Nevertheless, every edge of the transition system labeled $\emptyset$ is a loop, and every edge labeled $\{Buy(A)\}$ is horizontal, because of the postulates (8) that express, under the semantics of $\mathcal{C}+$, the persistence property of $Has(x)$.

Similarly, causal laws (5)–(8) do not say anything about the concurrent execution of actions $Buy(A)$ and $Buy(B)$. But the edges labeled $\{Buy(A), Buy(B)\}$ in Figure 1 are directed diagonally, in accordance with our commonsense expectations. This fact illustrates the convenience of the approach to concurrency incorporated in the semantics of $\mathcal{C}+$.

However, as discussed in the introduction, this built-in mechanism is not directly applicable to the effects of actions on additive fluents. We are now ready to turn to the main subject of this paper—extending $\mathcal{C}+$ with the additional notation that resolves this difficulty.

4 Increment Laws

In our proposed extension of $\mathcal{C}+$, some of the simple fluent constants can be designated as *additive*. The domain of every additive fluent constant is assumed to be a finite set of numbers. We understand “numbers” as (symbols for) elements of any set with an associative and commutative binary operation $+$ that has a neutral element 0.¹ Effects of actions on additive fluents are described in extended $\mathcal{C}+$ by causal laws of a new kind—“increment laws.” Accordingly, we modify the definition of a causal law reproduced in Section 3 in two ways. First, in causal laws (3) and (4) formula F is not allowed to contain additive fluent constants. Second, we extend the class of causal laws by including *increment laws*—expressions of the form

$$a \text{ increments } c \text{ by } n \text{ if } G \quad (9)$$

where

- a is a Boolean action constant,
- c is an additive fluent constant,
- n is a number, and
- G is a formula that contains no Boolean action constants.

¹The additive group of integers is the main example we are interested in, and this is the case that has been implemented. The max operation on an ordered set with the smallest element is another interesting case.

Notation: x ranges over $\{A, B\}$.

Action constants:

$Buy(x)$

Domains:

Boolean

Additive fluent constant:

$InStore$

Domain:

$\{0, \dots, N\}$

Causal laws:

$Buy(x)$ **increments** $InStore$ **by** -1
exogenous $Buy(x)$

Figure 2: An action description in extended $\mathcal{C}+$.

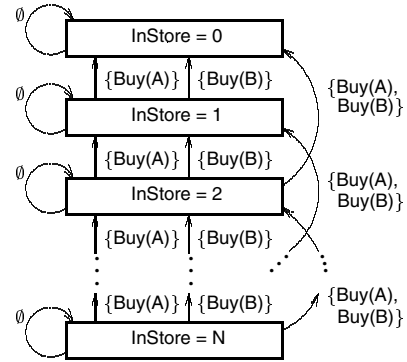


Figure 3: The transition system described by Figure 2.

We will drop ‘if G ’ in (9) if G is $\top$. In extended $\mathcal{C}+$, an *action description* is a set of causal laws that contains finitely many increment laws.

In the next section we define the semantics of extended $\mathcal{C}+$ by describing a translation that eliminates increment laws in favor of additional action constants.

As an example, consider the effects of actions $Buy(A)$, $Buy(B)$ on the number of books available in the bookstore where Alice and Bob are buying books. A description of these effects in extended $\mathcal{C}+$ is shown in Figure 2 (as before, N is a fixed nonnegative integer). The transition system represented by the translation of Figure 2 into the non-extended language $\mathcal{C}+$ is depicted in Figure 3 (with the auxiliary action constants dropped from the edge labels). The causal laws in Figure 2 do not say explicitly that the trivial event $\emptyset$ has no effect on the value of $InStore$, or that the concurrent execution of actions $Buy(A)$ and $Buy(B)$ decrements the value of this fluent by 2. Nevertheless, every edge of the corresponding transition system labeled $\emptyset$ is a loop, and every edge labeled $\{Buy(A), Buy(B)\}$ goes up 2 levels, in accordance with our commonsense expectations. This happens because Figure 2 classifies $InStore$ as an additive fluent constant.

The causal laws in this action description do not say explicitly that actions $Buy(x)$ are not executable when $InStore = 0$, and that actions $Buy(A)$, $Buy(B)$ cannot be executed concurrently when $InStore = 1$. This is taken care of by our semantics of increment laws, in view of the fact that the domain of $InStore$ does not contain negative numbers.

5 Translating Increment Laws

Let D be an action description in extended $\mathcal{C}+$. In connection with the increment laws (9) in D , the following terminology will be used: about the Boolean action constant a , the additive fluent constant c and the number n in (9) we will say that a is a c -contributing constant, and that n is a contribution of a to c .

The auxiliary action constants introduced in the translation are expressions of the form $Contr(a, c)$, where c is an additive fluent constant, and a is a c -contributing action constant. The domain of $Contr(a, c)$ consists of all contributions of a to c and number 0.

To translate the increment laws from D , we

- (i) replace each increment law (9) in D with the action dynamic law

$$\text{caused } Contr(a, c) = n \text{ if } a = t \wedge G, \quad (10)$$

- (ii) for every auxiliary constant $Contr(a, c)$, add the action dynamic law

$$\text{caused } Contr(a, c) = 0 \text{ if } Contr(a, c) = 0, \quad (11)$$

- (iii) add the fluent dynamic laws

$$\text{caused } c = v + \sum_a v_a \text{ if } \top \text{ after } c = v \wedge \bigwedge_a Contr(a, c) = v_a \quad (12)$$

for every additive fluent constant c , every $v \in Dom(c)$, and every function $a \mapsto v_a$ that maps each c -contributing constant a to an element of the domain of $Contr(a, c)$ so that $v + \sum_a v_a$ is in the domain of c .

The sum and the multiple conjunction in (12) range over all c -contributing constants a .

Causal law (10) interprets increment law (9) as the assertion that executing a (possibly along with other actions) causes constant $Contr(a, c)$ to get the value n , under some conditions characterized by formula G . Causal laws (11) say that the value of this constant is 0 by default, that is to say, when another value is not required by any increment law. Causal laws (12) say that the value of an additive fluent constant after an event can be computed as the sum of the value of this constant prior to the event and the contributions of all actions to this constant.

The result of translating increment laws from Figure 2 is shown in Figure 4. In this case, the translation described above introduces two auxiliary action constants: $Contr(Buy(A), InStore)$ and $Contr(Buy(B), InStore)$. The domain of each of them has 2 elements: the contribution -1 of $Buy(x)$ to $InStore$ and number 0.

The edges of the transition system described by Figure 4, and the corresponding events, can be computed using the methods presented in [Giunchiglia *et al.*, 2003, Sections 4.2, 2.6]. (This calculation involves turning the action description into a definite causal theory and computing the models of the completion of this theory.) Every event assigns values to each action constant, including the auxiliary constants $Contr(Buy(x), InStore)$. For instance, the label

$$\{Buy(A), Buy(B)\}$$

Notation: x ranges over $\{A, B\}$.

Action constants:

$Buy(x)$
 $Contr(Buy(x), InStore)$

Domains:

Boolean
 $\{-1, 0\}$

Additive fluent constant:

$InStore$

Domain:

$\{0, \dots, N\}$

Causal laws:

caused $Contr(Buy(x), InStore) = -1$ **if** $Buy(x) = t$
caused $Contr(Buy(x), InStore) = 0$ **if** $Contr(Buy(x), InStore) = 0$
caused $InStore = v + v_1 + v_2$ **if** $\top$ **after** $InStore = v \wedge$
 $Contr(Buy(A), InStore) = v_1 \wedge Contr(Buy(B), InStore) = v_2$
for all $v \in \{0, \dots, N\}$ and $v_1, v_2 \in \{-1, 0\}$
such that $v + v_1 + v_2 \geq 0$
exogenous $Buy(x)$

Figure 4: The result of translating increment laws from Figure 2.

in Figure 3 represent the following event E :

$$E(Buy(A)) = t, E(Contr(Buy(A), InStore)) = -1, \\ E(Buy(B)) = t, E(Contr(Buy(B), InStore)) = -1.$$

The Causal Calculator, which implements $\mathcal{C}+$ with increment laws, can be downloaded from

<http://www.cs.utexas.edu/users/tag/ccalc/>.

6 Reasoning about Money

As an application of these ideas to automated commonsense reasoning, consider the following example:

I have \$6 in my pocket. A newspaper costs \$1, and a magazine costs \$3. Can I buy 2 newspapers and a magazine? A newspaper and 2 magazines?

These questions are about the executability of some concurrently executed actions, and the answers are determined by the effect of these actions on an additive fluent—the amount of money that I have.

Figure 5 describes the relevant properties of buying and selling in the input language of CCALC. There are objects of four sorts in this domain: agents, resources, items (to be purchased) and (nonnegative) integers; items are a subset of resources. The buyer and the seller are agents; money is a resource; $0, \dots, \text{maxInt}$ are integers. The price of an item is an integer. The number of units of a resource that an agent has is an integer-valued additive fluent. Buying is an exogenous action. The four causal laws that follow these declarations are self-explanatory; *decrements* is an abbreviation defined in terms of *increments*. The last causal law expresses that the number of units that are being purchased is uniquely defined.

Figure 6 expresses the first of the two questions stated at the beginning of this section. The line `maxstep : 1` tells CCALC that the query is about paths of length 1 in the transition system. The question is whether the transition system contains an edge that begins in a state in which the buyer

```

:- sorts
  agent; resource >> item; nnInteger.

:- variables
  Ag :: agent;    Res :: resource;
  It :: item;     M,N :: nnInteger.

:- objects
  buyer,seller   :: agent;
  money           :: resource;
  0..maxInt       :: nnInteger.

:- constants
  price(item)     :: nnInteger;
  has(agent,resource)
    :: additiveFluent(nnInteger);
  buy(item,nnInteger) :: exogenousAction.

buy(It,N) increments has(buyer,It) by N.
buy(It,N) decrements has(seller,It) by N.
buy(It,N) increments has(seller,money)
  by M*N if price(It)=M
      where M*N <= maxInt.
buy(It,N) decrements has(buyer,money)
  by M*N if price(It)=M
      where M*N <= maxInt.
nonexecutable buy(It,M) & buy(It,N)
  where M\=N.

```

Figure 5: Buying and selling

```

:- objects
  newspaper,magazine :: item.

price(newspaper)=1. price(magazine)=3.

:- query
maxstep :: 1;
0: has(buyer,money)=6,
  buy(newspaper,2), buy(magazine,1).

```

Figure 6: Do I have enough cash?

has \$6, and whose label includes buying 2 newspapers and 1 magazine. CCALC responds to this query by finding such an edge.² Its reply to a similar question about 1 newspaper and 2 magazines is negative.

7 Reasoning about Motion

Some additive fluents mentioned in the introduction—for instance, the velocity of a particle—are real-valued, rather than integer-valued. CCALC cannot deal with real numbers yet, and its input language does not allow us to express properties of such fluents.

But let’s imagine a movable object that is immune to this complication—the spacecraft Integer. Far away from stars

²This example involves the concurrent execution of two actions, but in general the CCALC implementation of additive fluents does not impose any specific restriction on the number of actions that can be executed concurrently.

and planets, the Integer is not affected by any external forces. As its proud name suggests, the mass of the spacecraft is an integer. For every integer t , the coordinates and all three components of the Integer’s velocity vector at time t are integers; the forces applied to the spacecraft by its jet engines over the interval $(t, t + 1)$, for any integer t , are constant vectors whose components are integers as well. If the crew of the Integer attempts to violate any of these conditions, the jets fail to operate!

Our formalization of the motion of the Integer uses the fluents $Pos(a)$, where a ranges over $\{X, Y, Z\}$, to represent the current position of the Integer along the a axis. The additive fluents $Vel(a)$ are the components of its velocity. According to Newton’s Second Law, the acceleration created by firing a jet can be computed by dividing the force by the mass of the spacecraft. This relationship can be expressed without mentioning the acceleration explicitly—in terms of the change in the velocity over a unit time interval:

Fire(j) increments Vel(a) by $n/Mass$ if Force(j, a) = n .

To test our representation, we instructed CCALC to answer the following question:

The mass of the Integer is 1. The Integer has two jets, and the force that can be applied by each jet along each axis is at most 2. The current position of the Integer is $(-1, 0, 1)$, and its current velocity is $(0, 1, 1)$. How can it get to $(0, 3, 1)$ within 1 time unit?

One of the solutions found by CCALC is to apply the forces $(2, 2, 0)$ and $(0, 2, -2)$.

8 Missionaries and Cannibals with Two Boats

In the Missionaries and Cannibals Problem (MCP), three missionaries and three cannibals come to a river and find a boat that can hold two people. If the cannibals ever outnumber the missionaries on either bank, the missionaries will be eaten. How shall they cross? The shortest solution involves 11 steps.

Lifschitz [2000] showed how to express this puzzle and some of its elaborations due to McCarthy [1999] in the language of CCALC. Some simple elaborations of MCP in the spirit of this work require that the number of members of a group (missionaries or cannibals) at a given location be treated as an additive fluent. This is the case, for instance, when several boats are available and are allowed to operate concurrently.

Using the ideas of this paper, we formalized the modification of MCP in which the travelers find two boats: a small boat that holds one, and a bigger boat that holds two. Using our formalization, CCALC has determined that the modified problem can be solved in 7 steps.

9 Properties of Additive Constants

By examining Figure 3 in isolation from its symbolic description in Figure 2 we can see that the constant *InStore* exhibits some features typical for additive fluent constants.

Consider, for instance, the edges that start at the vertex $InStore = 2$ and are labeled by the events $\{Buy(A)\}$ and

$\{Buy(B)\}$. Each of them leads to the vertex $InStore = 1$, so that each of these two events, when it occurs in the state $InStore = 2$, increments the value of $InStore$ by -1 . In accordance with the intuitive idea of an additive fluent, we can expect that the “union” of these events, when it occurs in the same state, will increment the value of $InStore$ by $(-1) + (-1)$. And this is true, because the edge in Figure 3 that starts at the vertex $InStore = 2$ and is labeled $\{Buy(A), Buy(B)\}$ leads to the vertex $InStore = 0$.

Proposition 1 below generalizes this observation to a class of action descriptions in the language $\mathcal{C}+$ extended as described in Sections 4, 5. By D we denote any action description in this language.

About events $e_0, e_1, \dots, e_n$ ($n \geq 0$) in the transition system represented by D we say that e_0 is a *disjoint union* of $e_1, \dots, e_n$ if, for every Boolean action constant a ,

- if $e_0(a) = t$ then there exists a unique $i > 0$ such that $e_i(a) = t$; for this i , $e_0(a') = e_i(a')$ for every non-Boolean action constant a' ;
- if $e_0(a) = f$ then, for all $i > 0$, $e_i(a) = f$.

In the rest of this section we assume that the set of numbers is a commutative group.

Proposition 1 *Let c be an additive fluent constant, let $s, s_0, \dots, s_n$ ($n \geq 0$) be states, and let $e_0, \dots, e_n$ be events such that e_0 is a disjoint union of $e_1, \dots, e_n$. If, for all $i \in \{0, \dots, n\}$, the transition system represented by D contains an edge that leads from s to s_i and is labeled e_i then*

$$s_0(c) - s(c) = \sum_{i=1}^n (s_i(c) - s(c)).$$

The special case corresponding to $n = 0$ tells us that additive fluent constants are not affected by “trivial” events. In this sense, they are similar to the fluent constants for which inertia is postulated:

Corollary 1 *Let e be an event such that for every Boolean action constant a , $e(a) = f$. If the transition system represented by D contains an edge that leads from a state s to a state s' and is labeled e then, for any additive fluent constant c , $s'(c) = s(c)$.*

The special case corresponding to $n = 1$ implies that the effects of any set of actions on an additive fluent is deterministic:

Corollary 2 *If the transition system represented by D contains an edge that leads from a state s to a state s_0 and is labeled e , and an edge that leads from s to a state s_1 and is also labeled e , then, for any additive fluent constant c , $s_0(c) = s_1(c)$.*

10 Discussion

In this paper we showed how an implemented, declarative language for describing actions can be used to talk about the effects of actions on additive fluents. This was accomplished by extending the syntax of the action language $\mathcal{C}+$ from [Giunchiglia *et al.*, 2003] by increment laws and by showing how to treat these laws as abbreviations.

It is interesting to note that this treatment of additive fluents would have been impossible if, instead of $\mathcal{C}+$, we used its predecessor $\mathcal{C}$ from [Giunchiglia and Lifschitz, 1998]. Non-Boolean, non-exogenous action constants such as $Contr(a, c)$, and action dynamic laws such as (10) and (11) are among the features of $\mathcal{C}+$ that were not available in $\mathcal{C}$.

In literature on planning, fluents with numerical values are often referred to as “resources” [Koehler, 1998]. The concurrent execution of the actions that involve resources is usually limited to the “serializable” case, when all ways of sequencing the concurrent actions are well-defined and equivalent. This condition is not satisfied, however, for many uses of additive fluents, including the space travel example (Section 7) and the modified MCP (Section 8). Consider, for instance, the state with two cannibals and three missionaries on the first bank, and imagine that the missionaries are using both boats to cross simultaneously. The concurrent execution of the actions cannot be replaced by the larger boat crossing first and the smaller boat crossing after that—in the state between the two actions, the missionaries on the first bank would be outnumbered by the cannibals. This example shows that crossing the river in the modified MCP is not serializable.

Acknowledgements

We are grateful to Yuliya Babovich, Jonathan Campbell, Esra Erdem, Selim Erdoğan, Paolo Ferraris and Wanwan Ren for useful discussions related to the subject of this paper. This work was partially supported by the National Science Foundation under Grant IIS-9732744 and by the Texas Higher Education Coordinating Board under Grant 003658-0322-2001.

References

- [Gelfond and Lifschitz, 1998] Michael Gelfond and Vladimir Lifschitz. Action languages.³ *Electronic Transactions on AI*, 3:195–210, 1998.
- [Giunchiglia and Lifschitz, 1998] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, pages 623–630. AAAI Press, 1998.
- [Giunchiglia *et al.*, 2003] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories.⁴ *Artificial Intelligence*, 2003. To appear.
- [Koehler, 1998] Jana Koehler. Planning under resource constraints. In *Proc. ECAI-98*, pages 489–493, 1998.
- [Lifschitz, 2000] Vladimir Lifschitz. Missionaries and cannibals in the Causal Calculator. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conf.*, pages 85–96, 2000.
- [McCarthy, 1999] John McCarthy. Elaboration tolerance.⁵ In progress, 1999.

³<http://www.ep.liu.se/ea/cis/1998/016/> .

⁴<http://www.cs.utexas.edu/users/vl/papers/nmct.ps> .

⁵<http://www-formal.stanford.edu/jmc/elaboration.html> .

The Concurrent, Continuous FLUX

Yves Martin

Technische Universität Dresden
D-01062 Dresden, Germany
ym1@inf.tu-dresden.de

Abstract

FLUX belongs to the high-level programming languages for cognitive agents that have been developed in recent years. Based on the established, general action representation formalism of the Fluent Calculus, FLUX allows to implement complex strategies in a concise and modular fashion. In this paper, we extend the FLUX language to reason about domains involving continuous change and where actions occur concurrently. Using constraint logic programming, we show that this reasoning is performed in an efficient way.

1 Introduction

One of the most challenging and promising goals of Artificial Intelligence research is the design of autonomous agents, including robots, that solve complex tasks in a dynamic world. To reach autonomy in partially known, constantly changing environments requires the high-level cognitive capabilities of reasoning and planning. Using a mental model of the state of their environment allows for the agents to calculate the outcome of different action sequences in advance and then choose the best plan to execute for a specific goal in mind.

Formal theories of reasoning about actions and change have the expressive power to provide such high-level capabilities. The Fluent Calculus [Thielscher, 1999], as one of the established action representation formalisms, uses the concept of state update axioms to solve the representational and inferential aspect of the classical Frame Problem. Based on this formal theory, the high-level programming method FLUX has been developed in recent years [Thielscher, 2002]. Using the paradigm of constraint logic programming, the powerful FLUX kernel provides general reasoning facilities, so that the agent programmer can focus on the high-level design of intelligent agents.

Autonomous agents in real-world environments have to take into account that the execution of actions takes different amounts of time. Some actions can be modeled as discrete changes, others involve continuous change and should rather be seen as the initiation or termination of complex processes. Such processes may contain parameters whose values change continuously and which are formalized as functions

over time. A car moving on a road with a constant velocity v , for instance, can be represented by a 'process fluent' $Movement(x_0, t_0, v)$, where the parameter x_0 denotes the location of the car at the time t_0 when the motion was initiated. The fluent $Movement$ itself, although it describes the particular continuous change $x = x_0 + v * (t - t_0)$ for the location x of the car at the time t , will stay unchanged until some other action will affect it. Continuous change is then modeled by fluents describing arbitrarily complex, continuous processes. These fluents remain stable in between the occurrence of two consecutive actions, and yet they internally represent continuous change.

In a world full of ongoing processes, however, an agent executing a plan is not the only source of change. Also the laws of physics frequently imply an evolution of the environment, like for example, the action of a falling ball bouncing when it reaches the floor. The fundamental property of such so-called *natural* actions [Reiter, 1996] is that they must occur at their predicted times, provided no earlier actions (natural or deliberative) prevent them from occurring. Because such actions may occur simultaneously, concurrency must be accommodated.

In this paper, we present a FLUX system which allows for the design of intelligent agents that reason and plan in domains involving continuous change and where actions occur concurrently. Using the paradigm of constraint logic programming, our extension to FLUX integrates both kinds of actions, deliberative and natural, into one method for the planning and execution of actions. Reasoning in terms of time intervals, our method allows for the efficient generation of plans in concurrent, continuous environments. Our work is based on the theoretical approach presented in [Herrmann and Thielscher, 1996; Thielscher, 2001a]. Other existing agent programming methods like the robot control language developed in [Shanahan and Witkowski, 2000] or the GOLOG programming language [Levesque *et al.*, 1997], do not handle, in domains involving concurrency and continuous change, the concept of a natural action at all or only have separate accounts for natural actions on the one hand [Reiter, 2001] and for deliberative actions on the other hand [Grosskreutz, 2002].

In the next section, we introduce our example scenario and shortly recapitulate the fundamentals of the FLUX system. We then give briefly the theoretic solution in the Fluent Cal-

culus. Thereafter, we present our extension of FLUX to concurrent, continuous domains and discuss the employed constrained handling techniques. In the last section we conclude and show some possible future work. All programs are available on our web site: <http://www.fluxagent.org/>

2 FLUX

The example agent program in this paper is set in a *waterway scenario*. The agent has to steer a barge through a system of canals. The water levels in the canals are dependent on tides. At some places of the canals there are locks which a ship can only pass on a high tide. Due to the geographical facts the tide levels differ for different locks. The canals themselves are too small to turn around but sometimes there are intersections where the agent steering the barge can choose one direction (see also Fig. 1).

To develop an agent for this scenario, we use the high-level programming method FLUX which is grounded in the action theory of the Fluent Calculus. The Fluent Calculus is a many-sorted predicate logic language with four standard sorts: FLUENT, STATE, ACTION, and SIT(for situations) [Thielscher, 1999]. States are composed of fluents (as atomic states) using the standard function $\circ : \text{STATE} \times \text{STATE} \mapsto \text{STATE}$ and constant $\emptyset : \text{STATE}$ (denoting the empty state). The program for our agent, for example, uses these two fluents: $At(x, y, t)$, representing that the barge is at cell (x, y) at time t^1 , and $Tide(l, w, t)$, denoting that at time t the water level w at lock l is at high/low tide. Similarly as in the Situation Calculus [Reiter, 2001], the constant S_0 denotes the initial situation and $Do(a, t, s)$ the situation after having performed action a in situation s at time t . The state of a situation s is denoted by the standard function $State(s)$. For example, the initial state in the waterway scenario of Fig. 1 may be axiomatized as²

$$\begin{aligned} (\exists z) (State(S_0) = & At(1, 2, 0.0) \circ Tide(Lock1, High, 0.0) \circ \\ & Tide(Lock2, High, 0.0) \circ Tide(Lock3, High, 0.0) \circ z \wedge \\ & (\forall u, v, w) \neg Holds(At(u, v, w), z) \wedge (\forall u, v, w) \\ & \neg Holds(Tide(u, v, w), z)) \end{aligned}$$

The reader may notice that an incomplete state with additional negative information has been specified, i.e., the sub-STATE z may contain many more fluents, but no more At or $Tide$ fluents. The foundational axioms of the Fluent Calculus ensure that the composition function “ $\circ$ ” exhibits the properties of the union function for sets (with $\emptyset$ as the empty set), so that a state is identified with all the fluents that hold. On this basis, the macros $Holds(f, z)$ and $Holds(f, s)$ are defined as follows:

$$\begin{aligned} Holds(f, z) & \stackrel{\text{def}}{=} (\exists z') z = f \circ z' \\ Holds(f, s) & \stackrel{\text{def}}{=} Holds(f, State(s)) \end{aligned}$$

¹In anticipation of the integration of continuous change, the argument t denotes the time when a fluent becomes true or an action takes place.

²Predicate and function symbols, including constants, start with a capital letter whereas variables are in lower case. Free variables in formulas are assumed universally quantified. Variables of sorts FLUENT, STATE, ACTION, and SIT shall be denoted by letters f, z, a , and s , respectively. The function $\circ$ is written in infix notation.

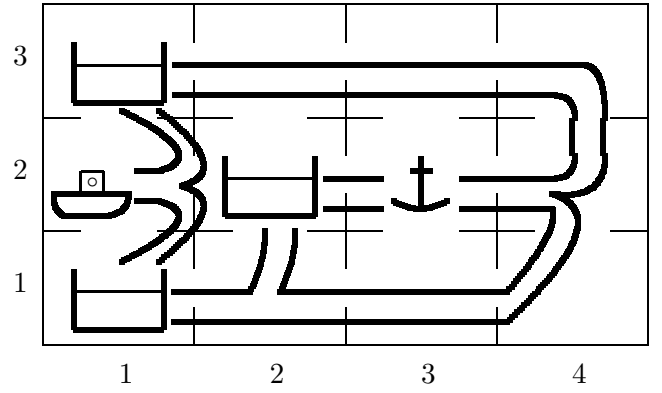


Figure 1: An example waterway scenario where the goal for the agent is to steer the barge from cell (1, 2) to the harbor at cell (3, 2). There are three locks at cells (1, 3), (1, 1), (2, 2).

In our waterway scenario there are two elementary actions: $GoByShip(d)$, the deliberative action of the agent to steer the barge to the next cell which lies in the direction d , and $TurnTide(l, w)$, the natural action indicating the turn to the tide w at lock l . The fundamental Frame Problem is solved in the Fluent Calculus by a so-called state update axiom for each action, which describes the effects of the action in terms of the difference between the states before and after the execution of it. For example, the action $TurnTide(l, w)$ can be specified as

$$\begin{aligned} Poss(TurnTide(l, w), t, s) \supset & \\ (\exists w_0, t_0) (Holds(Tide(l, w_0, t_0), s) \wedge & \\ w = High \wedge State(Do(TurnTide(l, w), t, s)) = & \\ (State(s) - Tide(l, w_0, t_0)) + Tide(l, High, t) \vee & \\ w = Low \wedge State(Do(TurnTide(l, w), t, s)) = & \\ (State(s) - Tide(l, w_0, t_0)) + Tide(l, Low, t)) & \end{aligned}$$

where “ $-$ ” and “ $+$ ” are macros for fluent removal and addition; and the macro $Poss(a, t, s) \stackrel{\text{def}}{=} Poss(a, t, State(s))$ denoting in the Fluent Calculus that at time t action a is possible in state $State(s)$.

To reflect the incomplete knowledge of an agent about its environment, incomplete states are encoded in FLUX as open lists, that is, lists with a variable tail, of fluents [Thielscher, 2002]. These lists are accompanied by constraints for negated state knowledge as well as for variable range restrictions. The constraints are of the form $NotHolds(f, z)$, indicating that fluent f does not hold in state z ; and $NotHoldsAll(f, z)$, indicating that no instance of f holds in z . In order to process these constraints, so-called declarative Constraint Handling Rules [Frühwirth, 1998] have been defined and proved correct under the foundational axioms of the Fluent Calculus (for details see [Thielscher, 2002]).

For example, the initial state depicted in Fig. 1 may be specified by this clause,

```
init(Z0) :-
  Z0=[at(1,2,0.0),tide(lock1,high,0.0),tide(
    lock2,high,0.0),tide(lock3,high,0.0)|Z],
  not_holds_all(at(_,_),Z),
  not_holds_all(tide(_,_),Z).
```

which also reflects the negative information that no *At* fluent occurs in sub-state z (the location of our agent is unique), and that there are no more *Tide* fluents other than specified in state z_0 .

The predicate $Poss(a, t, z)$ realizes the precondition axioms for actions, that is, it defines under which conditions an action is possible at time t in state z . There is one such predicate clause for each action. Conditioning in FLUX is based on the foundational predicates $Knows(f, z)$, $KnowsNot(f, z)$, and $KnowsVal(\vec{x}, f, z)$, representing that the agent knows that fluent f holds (respectively, does not hold) in state z , and that there exist ground instances of the variables in $\vec{x}$ such that fluent f is known to be true in state z . Take, for example, the precondition axiom for the action $TurnTide(l, w)$ which is implemented as follows,

```
poss(turntide(L,W),T,Z) :-
  (W=high, knows_val([L,TR],tide(L,low,TR),Z),
   duration(L,low,D);
   W=low, knows_val([L,TR],tide(L,high,TR),Z),
   duration(L,high,D)),
  {T := TR + D}.
```

where the auxiliary predicate $Duration(l, w, d)$ denotes the duration d of the corresponding tide w at lock l . The execution time t of the action will be constrained, using the syntax of the constraint handling library CLP(R) of the Eclipse-Prolog system, to be the sum of the starting time of the old tide plus the duration of the old tide.

As in the Fluent Calculus, the effects of actions are encoded as state update axioms. For this purpose, the FLUX kernel provides a definition of the auxiliary predicate $Update(z_1, \vartheta^+, \vartheta^-, z_2)$. Its intuitive meaning is that state z_2 is the result of positive and negative effects ϑ^+ and ϑ^- , respectively, wrt. state z_1 . In other words, the predicate encodes the state equation $z_2 = (z_1 - \vartheta^-) + \vartheta^+$. On this basis, the agent programmer can easily implement the update axioms by clauses which define the predicate $StateUpdate(z_1, a, t, z_2)$, as for example in the following encoding for the action $TurnTide(l, w)$:

```
state_update(Z1,turntide(L,W),T,Z2) :-
  holds(tide(L,W0,T0),Z1),
  W=high, update(Z1,[tide(L,high,T)],
                 [tide(L,W0,T0)],Z2);
  W=low, update(Z1,[tide(L,low,T)],
                [tide(L,W0,T0)],Z2).
```

3 The concurrent, continuous Fluent Calculus

The Fluent Calculus for concurrent actions is based on the additional pre-defined sort CONCURRENT, of which ACTION is a sub-sort [Thielscher, 2001a]. Single actions which are performed simultaneously are composed to terms of sort CONCURRENT by a new binary function. The latter is denoted by “.” and written in infix notation. This function shares with the function combining fluents to states the properties of associativity, commutativity, idempotency and existence of a unit element. The constant “ ϵ ” (read: *no-op*) of sort CONCURRENT acts as the unit element wrt. function “.”. Similar to the *Holds* macro the abbreviation $In(c_1, c)$ is used to denote that concurrent action c_1 is included in concurrent

action c^3 :

$$In(c_1, c) \stackrel{\text{def}}{=} (\exists c') c = c_1 \cdot c'$$

State update axioms for concurrent actions are recursive. They specify the effect of an action relative to the effect of arbitrary other, concurrent actions:

$$Poss(a \cdot c, t, s) \wedge \Delta(c, t, s) \supset \\ State(Do(a \cdot c, t, s)) \circ \vartheta^- = State(Do(c, t, s)) \circ \vartheta^+$$

I.e., ϑ^- and ϑ^+ are the additional negative and positive, respectively, effects which occur if action a is performed besides c . Here, a can be a single action or a compound action which produces synergic effects, that is, effects which no single action would have if performed alone. Using recursive state update axioms, the effect of, say, two simultaneous but independent actions can be inferred by first inferring the effect of one of them and, then, inferring the effect of the other action on the result of the first inference. The recursions stops with the base case of the empty action, which is defined as:

$$State(Do(\epsilon, t, s)) = State(s)$$

Two or more actions may interfere when executed concurrently, which is why the condition Δ in the above state update axiom may restrict the applicability of the implication in view of concurrent action c .

Integrating continuous change in the Fluent Calculus requires the introduction of process fluents which can represent arbitrarily complex, continuous processes. Because such processes may be modeled by equations of motions, continuous time must be represented. To this end, the new sort REAL is added, which is to be interpreted as the real numbers [Thielscher, 2001a]. The sort is accompanied by the usual arithmetic operations along with their standard interpretation. The continuous Fluent Calculus includes the pre-defined fluent $StartTime(t)$, where t is of sort REAL, determining the time $Start(z)$ at which a state arises, provided that $StartTime$ is unique:

$$(\exists t) Holds(StartTime(t), z) \supset \\ (\forall t) (Holds(StartTime(t), z) \supset Start(z) = t)$$

As already indicated throughout the paper, a parameter t of sort REAL is also used to denote the time at which a fluent arises as in $Tide(l, w, t)$ or to represent the occurrence of an action as in $Do(a, t, s)$. A standard requirement for the possibility to perform a concurrent action c at time t in state z may then be expressed as follows:⁴

$$Poss(c, t, z) \equiv c \neq \epsilon \wedge (\forall a) (In(a, c) \supset Poss(a, t, z))$$

The Fluent Calculus for continuous change includes the distinction between deliberative and natural actions [Thielscher, 2001a]. The latter are not subject to the free will of a planning agent. Rather they happen automatically under specific circumstances. In our example domain, the

³Variables of the new sort CONCURRENT are denoted by the letter c .

⁴In our example domain we have no actions that are in mutual conflict. Therefore, we do not need to specify additional constraints in this precondition axiom.

action of the turn of the tide is a natural one. The standard predicate *Natural(a)* [adopted from [Reiter, 1996]] declares the action *a* to be natural. To facilitate the formalization of the automatic evolution of natural actions, the continuous Fluent Calculus introduces two macros. The expression *ExpectedNatActions(c, t, z)* shall indicate that in state *z* actions *c* are all the natural actions that are expected to happen at time *t*:

$$\text{ExpectedNatActions}(c, t, z) \stackrel{\text{def}}{=} c \neq \epsilon \wedge (\forall a)(\text{In}(a, c) \equiv \text{Natural}(a) \wedge \text{Poss}(a, t, z))$$

Given the above notion, the macro *NextNatActions(c, t, z)* stands for the concurrent action *c* being all natural actions that happen in state *z* at time *t* with *t* being the earliest possible time point at which natural actions are expected:

$$\text{NextNatActions}(c, t, z) \stackrel{\text{def}}{=} \text{ExpectedNatActions}(c, t, z) \wedge \neg(\exists t', c')(\text{ExpectedNatActions}(c', t', z) \wedge t' < t)$$

The Fluent Calculus for continuous change uses the notion of a situation tree with trajectories [Thielscher, 2001a] where a trajectory is associated with a situation and denotes the further evolution of the state determined by the natural actions that are expected to happen. We do not follow this approach in this paper, as the original motivation for employing trajectories has been domains with uncertainty about the occurrence of natural actions, and we do not consider such domains here. The incorporation of domains with uncertain natural actions is left for further work. Instead, we include the natural actions in the situation terms, as with deliberative actions.

4 Integrating Concurrency and Continuous Change into FLUX

Similar to the binary function “o”, which denotes the composition of states from single fluents and is represented in FLUX by a list of fluents, we represent the binary function “.” as a list of actions in FLUX. In this way, we introduce concurrency into FLUX, i.e., all the actions in the list are performed concurrently. On this basis, the unit element of the function “.”, the constant “ε”, is encoded as the empty list [].

Given the notion of a list of concurrent actions, the state update axioms for the actions are defined recursively in FLUX. The predicate *Res(z₁, s₁, li, t, z₂, Do(li, t, s₁))* specifies the effect of performing at time *t* the list of concurrent actions *li* in state *z₁* and situation *s₁*, and leading to the new state *z₂* and the new situation *Do(li, t, s₁)* after the execution of the concurrent actions. It represents one plan step and is encoded as follows

```
res(Z1, S1, [], T, Z2, S1) :-
  holds(starttime(T0), Z1),
  update(Z1, [starttime(T)], [starttime(T0)], Z2).
```

```
res(Z1, S1, [A|L], T, Z2, do([A|L], T, S1)) :-
  update([A|L], [], [A], L1),
  state_update(Z1, A, T, Z3),
  res(Z3, _, L1, T, Z2, _).
```

where the clause for the base case of the recursion extends the effect of the constant “ε”, which by itself has none, to update the pre-defined fluent *StartTime(t)* needed for FLUX

with continuous change to the new value *t*. Similar to the encoding of the macros for fluent removal and addition, the predicate *Update* is used in the clause for the recursive case to implement the elimination of one single action from a term of arbitrary other, concurrent actions.

Consider, for example, the state update axiom of the single action *GoByShip(d)* which is implemented as:

```
state_update(Z1, gobyship(D), T, Z2) :-
  knows_val([X, Y], at(X, Y, T0), Z1),
  holds(starttime(ST), Z1), {T >= ST + 1.0},
  adjacent(X, Y, D, X1, Y1),
  update(Z1, [at(X1, Y1, T)], [at(X, Y, T0)], Z2).
```

That is, the location of the barge together with the agent will be updated from the old position (*x, y*) to the new cell (*x₁, y₁*), where the auxiliary predicate *Adjacent(x, y, d, x₁, y₁)* computes the adjacent cell (*x₁, y₁*) lying in direction *d* of cell (*x, y*). The travel from one cell to another is assumed to take one hour in our example scenario. Therefore, although the execution time *t* of this action is not fixed by the clause in any way, the effects of the action manifest at least one hour later as the formation of the old state. Now take, e.g., the FLUX query

```
?-init(Z0), res(Z0, s0, [gobyship(3),
  turntide(lock3, low)], 2.0, Z1, S1).
```

together with the above definition of the state update axiom and the definitions for the predicate *Init(z)* (with the additional inclusion of fluent *starttime(0.0)*) and the state update axiom of action *TurnTide* given in Section 2. Together with an appropriate encoding of the fact that a high tide lasts two hours at the third lock, *duration(lock3, high, 2.0)*, and the knowledge that going south is represented by direction number 3, our extension of FLUX can infer the effects of this concurrent action by first inferring the effects of action *gobyship(3)* and on the result of this inference infer the effects of action *turntide(lock3, low)*. FLUX yields the correct substitution:

```
Z1=[at(1, 1, 2.0), tide(lock1, high, 0.0),
  tide(lock2, high, 0.0), tide(lock3, low, 2.0),
  starttime(2.0)|Z]
S1=do([gobyship(3),
  turntide(lock3, low)], 2.0, s0)
```

Continuous time is, as already shown above and in Section 2, easily integrated into FLUX. The Eclipse-Prolog system Version 5.4, which we use, includes the constraint handling library CLP(R). This library allows for solving linear constraints with real numbers. Its syntax requires for constraints to be included in braces.

The precondition axioms and state update axioms for natural actions are encoded in our extension of FLUX in the same fashion as for deliberative actions (see Section 2). Additionally, we include an implementation for the predicate *Natural(a)* which is as follows for the example domain:

```
natural(A) :- A=turntide(L, W),
  (L=lock1; L=lock2; L=lock3).
```

Given this predicate, we model the macro *ExpectedNatActions(c, t, z)* by the built-in second order predicate *SetOf* as follows:

```
setof(A, (natural(A), poss(A, T, Z)), C)
```

On this basis, the macro *NextNatActions(c, t, z)* is defined in FLUX as:

```

NextNatTime(T,Z) :-
  natural(A), poss(A,T,Z),
  not (natural(A1), poss(A1,T1,Z), T1<T), !.

```

```

NextNatActions(C,T,Z) :- NextNatTime(T,Z),
  setof(A, (natural(A), poss(A,T,Z)), C).

```

Having defined natural actions in the same way as deliberative ones leads to the question: How to combine deliberative and natural actions into one common approach for planning? Natural actions must occur at their predicted times. The times for the execution of deliberative actions are not fixed in advance. How to determine these execution times? The examination of every possible time would lead to a combinatorial explosion and is, in general, not possible for time of sort *REAL*. One general solution, which reduces the search space to a minimum and still yields answers for all possible domains, is to use qualitative instead of quantitative information. We only consider periods of time in our approach. There we discriminate three time intervals for a deliberative action wrt. the next expected natural action(s): Firstly, the deliberative action can be postponed to the next plan step and the (possible set of) natural action(s) is executed. Secondly, the deliberative action is performed before all natural actions which are expected next. Finally, the deliberative action and the natural actions are joined together to a new concurrent action. The predicate *Exec*($z_1, s_1, z_2, s_2, depth$) encodes the recursive planner for a plan with *depth* steps integrating deliberative and natural actions into one method. The computed plan leads from state z_1 and situation s_1 to the new state z_2 and the new situation s_2 . The predicate is implemented as

```

exec(Z1,S1,Z1,S1,0).

```

```

exec(Z1,S1,Z2,S2,Depth) :-
  Depth>0, NextNatActions(C,T,Z),
  ({T1=T}, C1=C;
   action(A), poss(A,TA,Z1),
   holds(starttime(T0),Z1), {TA>=T0},
   ({TA<T, T1=TA}, C1=[A];
    {TA=T, T1=T}, append([A],C,C1))),
  res(Z1,S1,C1,T1,Z3,S3), Depth1 is Depth-1,
  exec(Z3,S3,Z2,S2,Depth1).

```

where the predicate *Action*(a) defines the action a to be deliberative and the auxiliary built-in predicate *Append* appends two lists. For our example domain, the predicate *Action* is encoded by the fact *action*(gobyship(_)). To ensure that the time never goes backward, the execution time TA of the deliberative action is constrained in an appropriate way.

Reasoning with time constraints instead of real time renders planning efficient. Using this plan method we are left with only three choices regarding the execution time of a deliberative action. Furthermore, the order of these choices, which represents a kind of heuristic, can be adjusted to the concrete domain at hand. Only after a plan, where the constraint solution lies in the appropriate time intervals, has been computed, a concrete time for the execution of the actions is fixed and the actions are executed.

To complete our planning method, we include the following definition of the predicate *Goal*(z) denoting the goal state z ,

```

goal(Z) :- knows(at(3,2,_),Z).

```

where the goal in our example scenario is that the barge is situated at the harbor in cell (3,2) (see also Fig. 1).

Given the predicate *Goal*, we define a recursive predicate *Ida*(z_0, s_0, z, s, n) representing the sequence of actions s which leads from the initial state z_0 in the initial situation s_0 to the goal state z in n steps. This predicate implements the iterative deepening algorithm, which is optimal and complete [Russell and Norvig, 1995]. It is encoded as follows:

```

ida(Z0,S0,Z,S,N) :-
  exec(Z0,S0,Z,S,N), goal(Z);
  N1 is N+1, findplan(Z0,S0,Z,S,N1).

```

The precondition axiom for the action *GoByShip* is specified in the following way:

```

poss(gobyship(D),T,Z) :-
  knows_val([X,Y],at(X,Y,_),Z),
  directions(X,Y,DL), member(D,DL),
  adjacent(X,Y,D,X1,Y1),
  (lockplace(L,X1,Y1),
   knows_val([L,W,T0],tide(L,W,T0),Z),
   (W=high; W=low,
    duration(L,low,TD), {T>=T0+TD}));
  not (lockplace(L,X1,Y1)).

```

That is, after having determined the current location, the auxiliary predicate *Directions*(x, y, dl) delivers a list of possible directions for the cell (x, y) and the standard predicate *Member* selects one direction d . Afterwards, with the help of the auxiliary predicate *LockPlace*(l, x, y), denoting the occurrence of a lock l at cell (x, y) , the adjacent cell (x_1, y_1) is searched for a lock. If there is none, the action is possible without further constraints for the execution time t . In the other case, the water level at the lock must be high or the action has to be executed after the disappearance of the low tide. Specifying the preconditions for the actions *GoByShip* and *TurnTide* as given above and in Section 2, respectively, fulfills the general condition for the possibility to perform a concurrent action as given in Section 3.

Consider now, for example, all specified FLUX clauses together with suitably specified facts for the example domain and the following query:

```

?-init(Z0), ida(Z0,s0,Z,S,1).

```

Our extended FLUX system then generates a plan with four steps and yields the following substitutions and linear constraints:

```

Z = [start(TA_2),at(3,2,TA_2),tide(lock3,
    high,4.0),tide(lock1,low,4.0),
    tide(lock2,high,0.0)|_]

```

```

S = do([gobyship(2)], do([gobyship(1),
    turntide(lock1,low),turntide(lock3,high)],
    do([gobyship(2),turntide(lock3,low)],
    do([gobyship(3)],s0,TA_1),2.0),4.0),TA_2)

```

```

Linear constraints: TA_1>=1.0, TA_1<2.0
                  TA_2>=5.0, TA_2<6.0

```

The above sequence of actions S constitutes a solution to our planning problem given in Fig. 1. It is not yet completely specified. Rather, the execution times of some deliberative actions are given as time intervals. The reader may also notice that some deliberative actions are planned simultaneously together with natural actions. Finally, we can apply the built-in predicate *Minimize*(t), which tries to find a minimal solution for a constraint variable t , to the above linear constraints and get the following:

TA_1 = 1.0

TA_2 = 5.0

5 Discussion

We have presented an extension to FLUX for domains involving continuous change and where actions occur concurrently. Our method is based on the theoretic solution in the Fluent Calculus [Thielscher, 2001a].

Our extension allows for the generation of plans including both, deliberative and natural actions. If necessary, the system generates and executes concurrent actions, i.e., where two or more single actions are performed simultaneously. In order to plan efficiently, our FLUX program computes with time intervals instead of single time points using the paradigm of constraint logic programming. We have illustrated how this method can be successfully applied to example domains like the waterway scenario. Additionally, our approach can easily be applied, with only minor modifications, to more complex domains involving, for example, compound concurrent actions which produce synergic effects.

Other high-level programming languages for reasoning about action and change, like GOLOG or the robot control language, have not yet an approach to integrate both, deliberative and natural actions in a common system to generate plans. The robot control language [Shanahan and Witkowski, 2000] does not have the notion of a natural action. The systems based on GOLOG either accommodate only natural actions [Reiter, 2001] or handle only deliberative actions [Grosskreutz and Lakemeyer, 2000] in domains involving concurrency and continuous change.

The extension to domains involving uncertainty about the occurrence of a natural action has not been tackled in this paper. An approach to this problem could be the use of conditional planning. Conditional plans based on a generalized concept of plan skeletons as search heuristics have been incorporated into FLUX [Thielscher, 2001b]. To use conditional plans as a method to accommodate such domains is an important aspect of future work.

Acknowledgments

I want to thank my supervisor Michael Thielscher, Olaf Perner and all members of the FLUX Group at Technische Universität Dresden for many fruitful discussions about this work. Three anonymous reviewers provided helpful comments on this article for which I am grateful.

References

- [Frühwirth, 1998] Thom Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1–3):95–138, 1998.
- [Grosskreutz and Lakemeyer, 2000] Henrik Grosskreutz and Gerhard Lakemeyer. cc-Golog: Towards more realistic logic-based robot control. In H. Kautz and B. Porter, editors, *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 476–482, Austin, TX, July 2000.
- [Grosskreutz, 2002] Henrik Grosskreutz. *Towards more realistic logic-based robot controllers in the Golog framework*. PhD thesis, RWTH Aachen, Germany, February 2002.
- [Herrmann and Thielscher, 1996] Christoph S. Herrmann and Michael Thielscher. Reasoning about continuous processes. In B. Clancey and D. Weld, editors, *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 639–644, Portland, OR, August 1996. MIT Press.
- [Levesque et al., 1997] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–83, 1997.
- [Reiter, 1996] Ray Reiter. Natural actions, concurrency and continuous time in the situation calculus. In L. C. Aiello, J. Doyle, and S. Shapiro, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 2–13, Cambridge, MA, November 1996. Morgan Kaufmann.
- [Reiter, 2001] Raymond Reiter. *Logic in Action*. MIT Press, 2001.
- [Russell and Norvig, 1995] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.
- [Shanahan and Witkowski, 2000] Murray Shanahan and Mark Witkowski. High-level robot control through logic. In C. Castelfranchi and Y. Lespérance, editors, *Proceedings of the International Workshop on Agent Theories Architectures and Languages (ATAL)*, volume 1986 of LNCS, pages 104–121, Boston, MA, July 2000. Springer.
- [Thielscher, 1999] Michael Thielscher. From Situation Calculus to Fluent Calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111(1–2):277–299, 1999.
- [Thielscher, 2001a] Michael Thielscher. The concurrent, continuous Fluent Calculus. *Studia Logica*, 67(1), 2001.
- [Thielscher, 2001b] Michael Thielscher. Inferring implicit state knowledge and plans with sensing actions. In F. Baader, G. Brewka, and T. Eiter, editors, *Proceedings of the German Annual Conference on Artificial Intelligence (KI)*, volume 2174 of LNAI, pages 366–380, Vienna, Austria, September 2001. Springer.
- [Thielscher, 2002] Michael Thielscher. Programming of reasoning and planning agents with FLUX. In D. Fensel, D. McGuinness, and M.-A. Williams, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, Toulouse, France, April 2002. Morgan Kaufmann.

Reasoning about the Interaction of Knowledge, Time and Concurrent Actions in the Situation Calculus

Richard B. Scherl

Computer Science Department
Monmouth University
West Long Branch, New Jersey
07764-1898 U.S.A.
rscherl@monmouth.edu

Abstract

A formal framework for specifying and developing agents/robots must handle not only knowledge and sensing actions, but also time and concurrency. Researchers have extended the situation calculus to handle knowledge and sensing actions. Other researchers have addressed the issue of adding time and concurrent actions. Here both of these features are combined into a unified logical theory of knowledge, sensing, time, and concurrency. The result preserves the solution to the frame problem of previous work, maintains the distinction between indexical and objective knowledge of time, and is capable of representing the various ways in which concurrency interacts with time and knowledge. Furthermore, a method based on regression is developed for solving the projection problem for theories specified in this version of the situation calculus.

1 Introduction

The aim of this paper is to develop a unified approach for axiomatizing the interaction of knowledge, sensing, time, and concurrency. Actions have preconditions which may include knowledge preconditions. Sensing actions alter knowledge. The knowledge produced depends upon the relative time at which sensing actions occur and also whether or not other sorts of actions occur concurrently. All of this interacts with the agent's evolving knowledge of time.

Consider a robot gathering knowledge about its environment. It moves about while concurrently panning the camera and at various points senses the environment for the presence of objects with various characteristics. The knowledge obtained through sensing (positions of objects of various sizes, shapes and colors) depends upon the position of the robot at a particular point in time, the angle of the camera, etc. For many purposes, the results of sensing actions that occur at the same time are important. Not only do we have the need for two distinct concurrent sensing actions in binocular stereopsis, but also in the simultaneous use of other features such as texture gradients and shading to achieve knowledge of depth relationships. Here it is relative, not absolute time, that is important.

Furthermore, specification of an agent's ability to achieve a goal in general involves requiring that the agent know what to do to arrive at a goal state [Moore, 1980; Lespérance *et al.*, 2000]. As the ability to achieve particular goals will often involve the ability to perform concurrent actions, the integration of knowledge and concurrency is an important step in fully formalizing these aspects of ability.

We develop our framework within the situation calculus—a first-order formalism for representing and reasoning about the effects of actions. The language is based upon the dialect of the situation calculus used in the *Cognitive Robotics Group* at the University of Toronto. Certainly, another formalism could have been used. Within A.I. numerous varying formalisms are available for representing and reasoning about action (to name a few, [Shanahan, 1995; Baral *et al.*, 1997]). Knowledge has been incorporated into a number of these (for example, [Lobo *et al.*, 2001; Thielscher, 2000]). Outside of A.I. proper, there is also [Fagin *et al.*, 1995] on the interaction of knowledge and time.

But by working in the situation calculus, we are able to extend previous work on reasoning (by regression and theorem proving) to cover the language developed here as well. Furthermore, our work is suitable to be incorporated into the robot programming languages GOLOG and CONGOLOG. It can then be used to specify agents that must reason about the interactions of time, knowledge, and concurrent actions (including sensing actions).

The situation calculus [McCarthy and Hayes, 1969] is a language for modeling dynamically changing worlds. Changes to the world are the results of *actions*, while possible world histories consisting of sequences of actions are represented by *situations*. The situation calculus can be used for agent planning via goal regression. Reiter [Reiter, 1991] proposed a simple solution to the frame problem, an approach to axiomatization within the situation calculus. Although this approach to the frame problem requires certain simplifying assumptions, it has proven to be useful and is the foundation for both goal regression and the programming language GOLOG. Goal regression was extended by Scherl and Levesque [Scherl and Levesque, 2003] to apply to an agent who can sense properties of the external world (e.g., read numbers on a piece of paper or determine the shape of an object).

This paper¹ combines and extends the work of Scherl and Levesque [Scherl and Levesque, 2003] incorporating the model of *concurrency* and *time* presented by Reiter [Reiter, 2001]. At the same time, Reiter’s simple solution to the frame problem is preserved. The real difficulty is to perform the synthesis in such a way that the important distinction between indexical and objective time [Lespérance and Levesque, 1995] is preserved.

If the agent currently knows the absolute time, then he knows the absolute time after executing an action. But if he doesn’t know the absolute time, then he only knows that he began executing the action some number of time units ago, unless of course he reads a clock. While maintaining these properties, the results presented here allow the representation of the various ways in which actions (including sensing actions and possibly other concurrent actions) interact with time and knowledge. The method of regression is also extended to work with this augmented language.

Section 2 gives a quick introduction to the situation calculus and Section 3 does the same for the foundational axioms. The representation of knowledge and sensing actions are covered in Section 4. Concurrency is integrated into the framework in Section 5. Section 6 covers some additional constructs of the language and illustrates their representational power. A number of properties of the formulation are discussed in Section 7. Regression is covered in Section 8. Finally, Section 9 is the conclusion.

2 Situation Calculus and the Frame Problem

Space does not permit a full exposition of the background material on the situation calculus. The framework developed in [Reiter, 2001] is followed and full details may be found there or in [Scherl, 2003]. We assume that the frame problem has been handled by utilizing successor state axioms.

3 Foundational Axioms

Following [Lin and Reiter, 1994; Reiter, 2001] the foundational axioms for the situation calculus are utilized. These axioms provide us with a definition of $s \preceq s'$, which says that there is a sequence of zero or more executable actions that move from situation s to situation s' . Again, space does not permit full development of this material here. Full details may be found in [Reiter, 2001; Scherl, 2003].

4 An Epistemic Fluent

Scherl and Levesque [Scherl and Levesque, 2003] adapt the standard possible-world model of knowledge to the situation calculus, as first done by Moore [Moore, 1980]. Informally, one can think of there being a binary accessibility relation over situations, where a situation s' is understood as being accessible from a situation s if as far as the agent knows in situation s , he might be in situation s' . So something is known in s if it is true in every s' accessible from s , and conversely something is not known if it is false in some accessible situation.

¹An earlier version of some of this work has appeared in [Zimmerbaum and Scherl, 2000].

To treat knowledge as a fluent, they introduce a binary relation $K(s', s)$, read as “ s' is accessible from s ” and treat it the same way we would any other fluent. In other words, from the point of view of the situation calculus, the last argument to K is the official situation argument (expressing what is known in situation s), and the first argument is just an auxiliary like the y in $BROKEN(y, s)$.²

The notation $\mathbf{Knows}(P(\mathbf{now}), s)$ (read as P is known in situation s) can then be introduced as an abbreviation for a formula that uses K . For example

$$\mathbf{Knows}(BROKEN(y, \mathbf{now}), s) \stackrel{\text{def}}{=} \forall s' K(s', s) \rightarrow BROKEN(y, s').$$

The special indexical **now** is instantiated with a situation variable upon expansion.

The approach also handles actions that make known the denotation of a term. For this case, one needs the notation $\mathbf{Kref}(T(\mathbf{now}), s)$ defined as follows:

$$\mathbf{Kref}(T(\mathbf{now}), s) \stackrel{\text{def}}{=} \exists x \mathbf{Knows}(T(\mathbf{now}) = x, s) \text{ where } x \text{ does not appear in } T.$$

In general, there may be many knowledge-producing actions, as well as many ordinary actions. To characterize all of these, we have following the presentation in [Scherl and Levesque, 2003], a function SR (for sensing result), and for each action α , a sensing-result axiom of the form:

$$SR(\alpha(\vec{x}), s) = r \equiv \phi_\alpha(\vec{x}, r, s) \quad (1)$$

This result is “YES” if “ Q ” is true and “NO” otherwise. The symbols are given in quotes to indicate that they are not fluents. The sensing result function for $SENSE_Q$ is axiomatized as follows:

$$SR(SENSE_Q, s) = r \equiv (r = \text{“YES”} \wedge Q(s)) \vee (r = \text{“NO”} \wedge \neg Q(s)) \quad (2)$$

For ordinary actions, the SR result is always the same, with the specific result not being significant. For example, we could have:

$$SR(PICKUP(x), s) = r \equiv r = \text{“OK”} \quad (3)$$

In the case of a $READ_\tau$ action that makes the denotation of the term τ known, we would have:

$$SR(READ_\tau, s) = r \equiv r = \tau(s) \quad (4)$$

Therefore, τ has the same denotation in all worlds s'' such that $K(s'', DO(READ_\tau, s))$, and so $\mathbf{Kref}(\tau, DO(READ_\tau, s))$ is true.

The form of the successor state axiom for K without concurrency is as follows:

$$K(s'', (DO(a, s))) \equiv (\exists s') s'' = DO(a, s') \wedge K(s', s) \wedge POSS(a, s') \wedge SR(a, s) = SR(a, s') \quad (5)$$

The relation K at a particular situation $DO(a, s)$ is completely determined by the relation at s and the action a .

²Note that using this convention means that the arguments to K are reversed from their normal modal logic use.

5 Concurrency

As originally defined in the situation calculus [McCarthy and Hayes, 1969], actions had to occur sequentially, with one action completed before another could begin. Furthermore, there was no facility to deal with the continuous passage of time. This contrasted with other formalisms such as the event calculus which could naturally handle concurrent actions and continuous time.

5.1 Concurrency with Knowledge

The work of Pinto [Pinto, 1994] and Reiter [Reiter, 2001] proposed an approach to dealing with concurrency, natural actions and continuous time while still maintaining the solution to the frame problem. Reiter [Reiter, 2001] defined a new sort *concurrent*, sets of simple actions. Variables $a, a', \dots$ represent the sort *actions* and $c, c', \dots$ represent the sort *concurrent*. In Reiter's notation, the time of an action's occurrence is the value of that action's temporal argument. Thus an action has the form $A(\vec{x}, t)$ and for each action an axiom of the form $\text{TIME}(A(\vec{x}, t)) = t$ is required to indicate the time of the action.

Concurrent actions are sets of ordinary actions that are taken to represent instantaneous acts. An action with duration is represented by two instantaneous actions—a start action and an end action. Additionally, the foundational axioms are modified to rule out the possibility of prior actions having later times.

So if we want to represent a *PRESSING* action with duration (as in pressing a button that keeps a light on), the approach is to define two actions; *STARTPRESS* and *ENDPRESS*. We also must introduce a fluent *PRESSING*. The needed successor state axiom is as follows:

$$\begin{aligned} \text{PRESSING}(\text{DO}(a, s)) \equiv \\ a = \text{STARTPRESS} \vee \\ (\text{PRESSING}(s) \wedge a \neq \text{ENDPRESS}) \end{aligned} \quad (6)$$

This approach to representing actions with duration is something that we make use of here.

But, the use of a temporal argument for actions is problematic in the presence of knowledge. Given our successor state axiom for *K*, it would require the agent to know the time after any action, even if it was unknown in the previous situation. To avoid this, we can not represent time as an argument to the instantaneous actions.

Instead, we represent the instantaneous actions and associated times as a tuple of the form $\langle a, t \rangle$ with functions *ACTION* and *TIME* defined, returning the first and second elements of the tuple:

$$\text{ACTION}(\langle a, t \rangle) = a \quad (7)$$

$$\text{TIME}(\langle a, t \rangle) = t \quad (8)$$

These pairs, represented by variables $p, p', \dots$ are elements of the sort *action-time pairs*. Concurrent Actions are now a set of such tuples. The sort *action* contains actions without a temporal argument.

We also have the following

$$\text{START}(\text{DO}(p, s)) = \text{TIME}(p) \quad (9)$$

which is needed to relate the time of the action/time pair to the start of a situation. There may also be an axiom giving the start time of the initial situation S_0 . We also define a variant notation:

$$\text{TIME}(\text{DO}(p, s)) = \text{TIME}(p) \quad (10)$$

We adopt, without significant change, Reiter's requirement that concurrent actions be coherent, that is there is at least one action-time pair p in the collection, and the time of all pairs in the collection is the same:

$$\begin{aligned} \text{COHERENT}(c) \equiv \\ (\exists p) p \in c \wedge (\exists t)(\forall p')[p' \in c \rightarrow \text{TIME}(p') = t]. \end{aligned} \quad (11)$$

A set of action-time pairs are coherent if each of them have the same time component.

The definition of *time* can readily be extended to sets of concurrent actions and this allows us to define the function *start* of a situation resulting from the execution of a concurrent action.

$$\begin{aligned} \text{COHERENT}(c) \rightarrow \\ [\text{TIME}(c) = t \equiv \exists p(p \in c \wedge \text{TIME}(p) = t)] \\ \wedge \text{START}(\text{do}(c, s)) = \text{TIME}(c)]. \end{aligned} \quad (12)$$

The predicate $\text{POSS}(c, s)$ means that it is possible to execute concurrent action c in situation s .

$$\begin{aligned} \text{POSS}(\text{ACTION}(p), s) \wedge \text{TIME}(p) > \text{TIME}(s) \\ \rightarrow \text{POSS}(p, s), \end{aligned} \quad (13)$$

$$\text{POSS}(p, s) \rightarrow \text{POSS}(\{p\}, s), \quad (14)$$

$$\text{POSS}(c, s) \rightarrow \text{COHERENT}(c) \wedge (\forall p)[p \in c \rightarrow \text{POSS}(p, s)]. \quad (15)$$

If it is possible to execute a concurrent actions, then it is coherent and each of the simple actions is possible.

We implicitly assume an additional sort ranging over time points which can be integers, rationals or reals; depending on how one wants to model time. The standard Arabic numerals are used to represent time points. Additionally, the symbols for addition and subtraction are interpreted as the usual operations on numbers (integers, reals etc.)

5.2 Precondition Interactions

We need to be able to conclude when a particular concurrent action c is possible in a situation c in order for the machinery being developed in the rest of this paper to work. Unfortunately, Sentence 15 does not suffice. The conditional can not be changed to a biconditional because of the precondition interaction problem [Pinto, 1994; 1998].

This issue needs to be handled by the axiomatizer of the domain. For example, the axiomatizer might provide the following axiom:

$$\begin{aligned} \text{POSS}(\{p_1, p_2\}, s) \equiv \\ \text{POSS}(\text{ACTION}(a_1), s) \wedge \text{POSS}(\text{ACTION}(a_2), s) \wedge \\ \neg \text{PRECINT}(a_1, a_2) \wedge \text{COHERENT}(\{p_1, p_2\}) \end{aligned} \quad (16)$$

As discussed in [Pinto, 1994; 1998], the axiomatization of *PRECINT* is domain dependent and can be done at increasing levels of detail.

For the purposes of this paper, the point here is that whatever solution is used for the precondition interaction problem in the ordinary situation calculus carries over to the case of knowledge and sensing.

5.3 Successor State Axiom for K with Concurrency

The Successor State Axiom for K using concurrency can be stated in several alternative ways depending on what conditions one wishes to apply regarding the agent's knowledge of time. We continue to require that the relation K at a particular situation $\text{DO}(c, s)$ is completely determined by the relation at s and the set of concurrent actions c .

The following successor state axiom models an agent who knows how much time is passing³. This is an agent who has an accurate clock.

$$\begin{aligned}
K(s'', (\text{DO}(c, s))) \equiv & \\
& (\exists s', c') s'' = \text{DO}(c', s') \wedge K(s', s) \wedge \text{POSS}(c', s') \\
& \wedge (\forall p') p' \in c' (\exists p) p \in c \\
& \wedge \text{ACTION}(p') = \text{ACTION}(p) \\
& \wedge (\forall p) p \in c (\exists p') p' \in c' \\
& \wedge \text{ACTION}(p) = \text{ACTION}(p') \\
& \wedge \text{START}(s'') = \text{START}(s') + (\text{TIME}(c) - \text{START}(s)) \\
& \wedge (\forall p) p \in c \rightarrow \\
& \quad \text{SR}(\text{ACTION}(p), s) = \text{SR}(\text{ACTION}(p), s')
\end{aligned} \tag{17}$$

After executing a set of concurrent actions c in situation s , as far as the agent knows, it could be in a situation s'' iff s'' is the result of performing c' in some previously accessible situation s' , provided that c is possible in s' and that s' is identical to s in terms of what is being sensed. Furthermore, it is required that the concurrent action c' being performed in all situations accessible from s be identical to c in terms of the individual actions that make up the set.

Note that it is not required that the TIME of the actions in all the accessible situations be identical. If this were the case, it would force the agent to know the objective time after executing any action. Rather, it is only required that the difference between the *start* of the current situation and the *start* of the previous situation be the same in all accessible situations (including the actual situation). This requirement does ensure that the agent knows how much time is passing, but the objective time is only known if it was known before the action has occurred.

5.4 Concurrency and Sensing

One can readily imagine cases of sensing actions where the desired result of sensing (knowing whether or not some proposition holds) depends upon some other action occurring concurrently. For example, the light needs to be turned on while the camera is clicked. If the light is off, then sensing produces no knowledge.

Consider representing, the requirement that the light switch be pressed while SENSE_P occurs for the knowledge of whether or not P holds to result from the execution of SENSE_P . We need to define a predicate SCOND .

$$\text{SCOND}(\text{SENSE}_P, s) \equiv \text{PRESSING}(s) \tag{18}$$

³Other possibilities are considered in [Scherl, 2003].

Now the successor state axiom for K needs to be modified to include SCOND as well.

$$\begin{aligned}
K(s'', (\text{DO}(c, s))) \equiv & \\
& (\exists s', c') s'' = \text{DO}(c', s') \wedge K(s', s) \wedge \text{POSS}(c', s') \\
& \wedge (\forall p') p' \in c' (\exists p) p \in c \\
& \wedge \text{ACTION}(p') = \text{ACTION}(p) \\
& \wedge (\forall p) p \in c (\exists p') p' \in c' \\
& \wedge \text{ACTION}(p) = \text{ACTION}(p') \\
& \wedge \text{START}(s'') = \text{START}(s') + (\text{TIME}(c) - \text{START}(s)) \\
& \wedge (\forall p) p \in c \wedge \\
& \quad (\text{SCOND}(\text{ACTION}(p), s) \wedge \text{SCOND}(\text{ACTION}(p), s') \rightarrow \\
& \quad \text{SR}(\text{ACTION}(p), s) = \text{SR}(\text{ACTION}(p), s'))
\end{aligned} \tag{19}$$

For every action an appropriate SCON axiom needs to be written. For most actions, the action is simply:

$$\text{SCOND}(a, s) \equiv \text{T} \tag{20}$$

6 The Language and Examples

6.1 Further Constructs

We need some way to refer to the current time without specifying the value of the current time. To achieve this we use the special indexical term **now**. Upon expansion the term is replaced with the appropriate situation term. So, $\text{START}(\text{now})$ can be used to refer to the current time. Here we illustrate by example. The agent's knowing the objective time is expressed as $\exists t \text{ Know}(\text{start}(\text{now}) = t, s)$. This expands into

$$\exists t \forall s' (K(s', s) \rightarrow \text{START}(s') = t).$$

We augment our language with a number of additional expressions. These are based on ideas developed by Lesperance and Levesque [Lesperance and Levesque, 1995] and require the use of the notion of precedence of situations as defined earlier. Note that we distinguish between the $<$ relation on integers used to represent time points and the $\prec$ relations on situations as defined in the foundational axioms for the situation calculus.

The macro **Happened** is introduced to allow one to talk about an action occurring at a particular time point.

$$\begin{aligned}
\text{Happened}(t, \text{Act}, s) \stackrel{\text{def}}{=} & (\exists c, s', s'') s'' = \text{DO}(c, s') \wedge \exists p \in c \\
& \wedge \text{ACTION}(p) = \text{Act} \wedge \text{TIME}(p) = t \wedge \\
& s'' \preceq s \wedge s' \prec s''
\end{aligned} \tag{21}$$

It specifies that an action occurred prior to s and it was time t at some point during the action's duration.

The macro **Wasat** is introduced to allow one to assert that a fluent was true at a particular point in time.

$$\begin{aligned}
\text{Wasat}(t, P(\text{then}), s) \stackrel{\text{def}}{=} & (\exists s') P(\text{then})\{s'/\text{then}\} \wedge \\
& t = \text{START}(s') \vee t > \text{START}(s') \\
& \wedge \neg(\exists s'') s' \prec s'' \prec s \wedge \text{START}(s'') \leq t
\end{aligned} \tag{22}$$

It specifies that P held at s' and t was the time of s' or s' preceded t and no other situation after s' preceded t . Here we introduce another special indexical **then** which is needed to ensure that the correct situation is substituted into the situation argument of the predicate which is the middle argument to **Wasat**.

7 Properties of the Formulation

First of all, we show that the distinction between indexical and objective time is preserved.

Proposition 1 (Persistence of Ignorance of Objective Time)

For all situations s , if $\neg \exists t \text{ Know}(\text{START}(\text{now}) = t, s)$ then in $\text{do}(c, s)$ where c is any concurrent action it is also the case that $\neg \exists t \text{ Know}(\text{START}(\text{now}) = t, s)$ unless there is some $a \in c$ with an SR axiom equivalent to the following:
 $\text{SR}(a, s) = r \equiv r = \text{TIME}(s)$

Proposition 2 (Persistence of Knowledge of Objective Time)

For all situations s , if $\exists t \text{ Know}(\text{start}(\text{now}) = t, s)$ then in $\text{do}(c, s)$ where c is any concurrent action it is also the case that $\exists t \text{ Know}(\text{start}(\text{now}) = t, s)$

Even if agents do not know the objective time, they do know how much time has passed since the last occurrence of a particular action or the last time at which a particular event was true.

Proposition 3 (Knowledge of Indexical Time 1)

For every, Act such that Act occurs in s ,
 $\exists t \text{ Knows}(\exists t' \text{ Happened}(t', \text{Act}, \text{now}) \wedge (\text{TIME}(\text{now}) - t' = t, s))$

Proposition 4 (Knowledge of Indexical Time 2) For every P such that there is some $s' \prec s$ such that $P(s')$ holds,
 $\exists t \text{ Knows}(\exists t' \text{ Wasat}(t', P(\text{then}), \text{now}) \wedge (\text{TIME}(\text{now}) - t' = t, s))$

Additionally, the crucial results of [Scherl and Levesque, 2003] carry over to the case considered here. These include

Proposition 5 (Default Persistence of Ignorance) For an action α and a situation s , if $\neg \text{Knows}(P, s)$ holds and the axiomatization entails

$$\forall s P(s) \equiv P(\text{DO}(\alpha, s))$$

and

$$\forall y \neg \text{Knows}((\text{POSS}(\alpha) \wedge \text{SR}(\alpha) = y) \rightarrow P, s)$$

then

$$\neg \text{Knows}(P, \text{DO}(\alpha, s))$$

holds as well.

Proposition 6 (Knowledge Incorporation) For a knowledge-producing action α , a fluent or the negation of a fluent F , a fluent or the negation of a fluent P , and a situation s , if the axiomatization entails

$$\exists y \text{ Knows}(F \equiv \text{SR}(\alpha) = y, s)$$

and also

$$F(s), \quad \text{POSS}(\alpha, s),$$

and

$$\text{Knows}(F \rightarrow P, s)$$

hold, then

$$\text{Knows}(P, \text{DO}(\alpha, s))$$

holds as well.

Proposition 7 (Memory) For all fluents P and situations s , if $\text{Knows}(P, s)$ holds then $\text{Knows}(P, \text{DO}(\alpha, s))$ holds as long as the axiomatization entails

$$\forall s P(s) \equiv P(\text{DO}(\alpha, s))$$

These results ensure that actions only affect knowledge in the appropriate way.

8 Reasoning

A regression operator $\mathcal{R}$ is defined relative to a set of successor state axioms Θ . Space limitations here only allow a sketch of the regression operators. Full details may be found in [Scherl, 2003].

The operators satisfy the following regression theorem:

Theorem 1 For any ground situation term s_{gr} and formula G :

$$\mathcal{F} \models G(s_{gr}) \text{ iff } \mathcal{F} - \mathcal{F}_{ss} \models \mathcal{R}_{\Theta}^*[G(s_{gr})]$$

Here $\mathcal{F}$ is the initial axiomatization of the domain and $\mathcal{F}_{ss}$ is the set of successor state axioms.

8.1 Regression Operators

The regression operator $\mathcal{R}$ is defined relative to a set of successor state axioms Θ . The first four parts of the definition of the regression operator $\mathcal{R}_{\Theta}$ concern ordinary (i.e. not knowledge-producing) actions [Reiter, 2001]. They are exactly the same as those in [Reiter, 2001]. Additionally, it is necessary to correctly regress the equality predicate as discussed in [Scherl and Levesque, 2003; Reiter, 2001].

Additional steps are needed to extend the regression operator to knowledge-producing actions. Two definitions are needed for the specification to follow. When φ is an arbitrary sentence and s a situation term, then $\varphi[s]$ is the sentence that results from instantiating every occurrence of now in φ with s' . The reverse operation φ^{-1} is the result of instantiating every occurrence of s' in φ with now .

Step **v** covers the case of regressing the **Knows** operator through a non-knowledge-producing action. Step **vi** covers the case of regressing the **Knows** operator through a knowledge producing action. In the definitions below, s' is a new situation variable.

- v.** Whenever c does not contain a knowledge-producing action,
 $\mathcal{R}_{\Theta}[\text{Knows}(W, \text{DO}(c, s))] = \text{Knows}(\text{POSS}(c) \rightarrow \mathcal{R}_{\Theta}[W[\text{DO}(\text{trans}(c), s')]]^{-1}, s).$
- vi.** Whenever c does contain a knowledge-producing action,
 $\mathcal{R}_{\Theta}[\text{Knows}(W, \text{DO}(c, s))] = \exists y \text{ SR}(\text{SENSE}_i, s) = y \wedge \text{Knows}((\text{POSS}(c) \wedge \text{SR}(\text{SENSE}_i) = y) \rightarrow \mathcal{R}_{\Theta}[W[\text{DO}(\text{trans}(c))]^{-1}, s)$

The special function **trans** replaces c with a c' that has the identical action in each of the action-time pairs. But it replaces the time portion with a relative time. This is an expression of the form $(3 = \text{TIME}(\text{now}))$. So, if c occurs in a situation term of the form $\text{DO}(c, s)$ and $\text{TIME}(c)$ is 7 and $\text{START}(s)$ is 4, then the 7 in the time part of every action-time pair in c would be replaced by $(3 = \text{TIME}(\text{now}))$. These are properly handled by the operators for regressing equality predicates.

The regression operators for **Happened** and **Wasat** follow:

viii.

$$\begin{aligned} \mathcal{R}[\text{Happened}(t, \text{Act}, \text{DO}(c, s))] = & (\text{Happened}(t, \text{Act}, s) \wedge t \leq \text{START}(s)) \\ & \vee \exists p \in c (\text{ACTION}(p) = \text{Act} \wedge t = \text{START}(\text{DO}(c, s))) \end{aligned}$$

ix.

$$\begin{aligned} \mathcal{R}[\text{Wasat}(t, W, \text{DO}(c, s))] = \\ (\text{Wasat}(t, W, s) \wedge t < \text{START}(s)) \\ \vee (W(s) \wedge t \geq \text{START}(s) \wedge t < \text{START}(\text{DO}(c, s))) \end{aligned}$$

The end result of regression is a sentence (possibly much larger) in a language without actions. The language is a modal language with both temporal and epistemic operators.

9 Conclusions

The results reported in this paper can be combined with Concurrent, Temporal Golog or RGolog[Reiter, 2001] or CONGOLOG[Giacomo *et al.*, 1997] to specify and control an agent (such as the robot discussed in Section 1) that concurrently moves about its environment and performs various actions including sensing actions. Future work will address the issue of developing a good theorem proving method for the language resulting from regression. This method needs to combine modal theorem proving with a method for solving integer constraints.

Acknowledgments

Numerous helpful discussions have been held with Yves Lespérance on many of the topics covered in this paper. Useful suggestions made by Patrick Doherty, Daniele Nardi, and a number of anonymous reviewers of earlier versions of this paper have been incorporated. Additional thanks are due to Steve Zimmerbaum with whom the initial stages of this work were carried out. This research was partially supported by NSF grants SES-9819116 and CISE-9818309.

References

- [Baral *et al.*, 1997] Chitta Baral, Michael Gelfond, and Alessandro Provetti. Representing actions: Laws, observations and hypotheses. *Journal of Logic Programming*, 31(1-3):201-243, 1997.
- [Fagin *et al.*, 1995] R. Fagin, J.Y. Halpern, Y.O. Moses, and M.Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, Mass, 1995.
- [Giacomo *et al.*, 1997] G. De Giacomo, Y. Lespérance, and H. J. Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogeneous actions in the situation calculus. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997.
- [Lespérance and Levesque, 1995] Yves Lespérance and Hector Levesque. Indexical knowledge and robot action—a logical account. *Artificial Intelligence*, 73(1-2):69-115, February 1995.
- [Lespérance *et al.*, 2000] Yves Lespérance, Hector J. Levesque, Fangzhen Lin, and Richard B. Scherl. Ability and knowing how in the situation calculus. *Studia Logica*, 66(1), 2000.
- [Lin and Reiter, 1994] Fangzhen Lin and Raymond Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5):655-678, 1994.
- [Lobo *et al.*, 2001] Jorge Lobo, Gisela Mendez, and Stuart Taylor. Knowledge and the action description language *A*. *Journal of Logic Programming*, 1(2):129-184, 2001.
- [McCarthy and Hayes, 1969] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463-502. Edinburgh University Press, Edinburgh, UK, 1969.
- [Moore, 1980] R.C. Moore. Reasoning about knowledge and action. Technical Note 191, SRI International, October 1980.
- [Pinto, 1994] J.A. Pinto. *Temporal Reasoning in the Situation Calculus*. PhD thesis, Department of Computer Science, University of Toronto, Toronto, Ontario, 1994. Available as technical report KRR-TR-94-1.
- [Pinto, 1998] Javier A. Pinto. Concurrent actions and interacting effects. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR-98)*, pages 292-303. Morgan Kaufmann Publishing, 1998.
- [Reiter, 1991] Raymond 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.
- [Reiter, 2001] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, Cambridge, Massachusetts, 2001.
- [Scherl and Levesque, 2003] Richard Scherl and Hector Levesque. Knowledge, action, and the frame problem. *Artificial Intelligence*, 144:1-39, 2003.
- [Scherl, 2003] Richard Scherl. Axiomatizing the interaction of knowledge, time, and concurrency with the situation calculus. journal version in progress, 2003.
- [Shanahan, 1995] Murray Shanahan. A circumscriptive calculus of events. *Artificial Intelligence*, 77(2):249-284, September 1995.
- [Thielscher, 2000] Michael Thielscher. Representing the knowledge of a robot. In A. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 109-120. Morgan Kaufmann, 2000.
- [Zimmerbaum and Scherl, 2000] Stephen Zimmerbaum and Richard Scherl. Knowledge, time, and concurrency in the situation calculus. In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents VII: Proceedings of the 2000 Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, LNAI, Berlin, 2000. Springer-Verlag.

RESOURCE-BOUNDED REASONING

Definition and Complexity of Some Basic Metareasoning Problems*

Vincent Conitzer and Tuomas Sandholm

Carnegie Mellon University
Computer Science Department

5000 Forbes Avenue
Pittsburgh, PA 15213, USA
{conitzer,sandholm}@cs.cmu.edu

Abstract

In most real-world settings, due to limited time or other resources, an agent cannot perform all potentially useful deliberation and information gathering actions. This leads to the metareasoning problem of selecting such actions. Decision-theoretic methods for metareasoning have been studied in AI, but there are few theoretical results on the complexity of metareasoning. We derive hardness results for three settings which most real metareasoning systems would have to encompass as special cases. In the first, the agent has to decide how to allocate its deliberation time across anytime algorithms running on different problem instances. We show this to be $\mathcal{NP}$ -complete. In the second, the agent has to (dynamically) allocate its deliberation or information gathering resources across multiple actions that it has to choose among. We show this to be $\mathcal{NP}$ -hard even when evaluating each individual action is extremely simple. In the third, the agent has to (dynamically) choose a limited number of deliberation or information gathering actions to disambiguate the state of the world. We show that this is $\mathcal{NP}$ -hard under a natural restriction, and $\mathcal{PSPACE}$ -hard in general.

1 Introduction

In most real-world settings, due to limited time, an agent cannot perform all potentially useful deliberation actions. As a result it will generally be unable to act rationally in the world. This phenomenon, known as bounded rationality, has been a long-standing research topic (e.g., [3, 17]). Most of that research has been *descriptive*: the goal has been to characterize how agents—in particular, humans—deal with this constraint. Another strand of bounded rationality research has the *normative (prescriptive)* goal of characterizing how agents *should* deal with this constraint. This is particularly important when building artificial agents.

Characterizing how an agent should deal with bounded rationality entails determining how the agent should deliberate.

*The material in this paper is based upon work supported by the National Science Foundation under CAREER Award IRI-9703122, Grant IIS-9800994, ITR IIS-0081246, and ITR IIS-0121678.

Because limited time (or other resources) prevent the agent from performing all potentially useful deliberation (or information gathering) actions, it has to select among such actions. Reasoning about which deliberation actions to take is called *metareasoning*. Decision theory [7, 10] provides a normative basis for metareasoning under uncertainty, and decision-theoretic deliberation control has been widely studied in AI (e.g., [2, 4–6, 8, 9, 12–15, 18–20]).

However, the approach of using metareasoning to control reasoning is impractical if the metareasoning problem itself is prohibitively complex. While this issue is widely acknowledged (e.g., [8, 12–14]), there are few theoretical results on the complexity of metareasoning.

We derive hardness results for three central metareasoning problems. In the first (Section 2), the agent has to decide how to allocate its deliberation time across anytime algorithms running on different problem instances. We show this to be $\mathcal{NP}$ -complete. In the second metareasoning problem (Section 3), the agent has to (dynamically) allocate its deliberation or information gathering resources across multiple actions that it has to choose among. We show this to be $\mathcal{NP}$ -hard even when evaluating each individual action is extremely simple. In the third metareasoning problem (Section 4), the agent has to (dynamically) choose a limited number of deliberation or information gathering actions to disambiguate the state of the world. We show that this is $\mathcal{NP}$ -hard under a natural restriction, and $\mathcal{PSPACE}$ -hard in general.

These results have general applicability in that most metareasoning systems must somehow deal with one or more of these problems (in addition to dealing with other issues). We also believe that these results give a good basic overview of the space of high-complexity issues in metareasoning.

2 Allocating anytime algorithm time across problems

In this section we study the setting where an agent has to allocate its deliberation time across different problems—each of which the agent can solve using an anytime algorithm. We show that this is hard even if the agent can perfectly predict the performance of the anytime algorithms.

2.1 Motivating example

Consider a newspaper company that has, by midnight, received the next day's orders from newspaper stands in the 3

cities where the newspaper is read. The company owns a fleet of delivery trucks in each of the cities. Each fleet needs its vehicle routing solution by 5am. The company has a default routing solution for each fleet, but can save costs by improving (tailoring to the day's particular orders) the routing solution of any individual fleet using an anytime algorithm. In this setting, the “solution quality” that the anytime algorithm provides on a fleet’s problem instance is the amount of savings compared to the default routing solution.

We assume that the company can perfectly predict the savings made on a given fleet’s problem instance as a function of deliberation time spent on it (we will prove hardness of metareasoning even in this deterministic variant). Such functions are called (*deterministic*) *performance profiles* [2, 6, 8, 9, 20]. Each fleet’s problem instance has its own performance profile.¹ Suppose the performance profiles are as shown in Fig. 1.

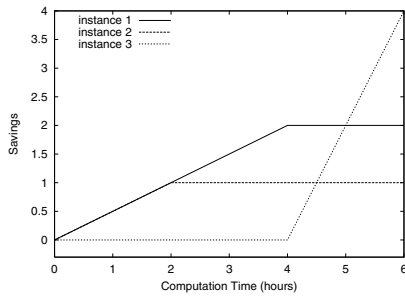


Figure 1: Performance profiles for the routing problems.

Then the maximum savings we can obtain with 5 hours of deliberation time is 2.5, for instance by spending 3 hours on instance 1 and 2 on instance 2. On the other hand, if we had until 6am to deliberate (6 hours), we could obtain a savings of 4 by spending 6 hours on instance 3.

2.2 Definitions and results

We now define the metareasoning problem of allocating deliberation across problems according to performance profiles.

Definition 1 (PERFORMANCE-PROFILES) We are given a list of performance profiles $(f_1, f_2, \dots, f_m)$ (where each f_i is a nondecreasing function of deliberation time, mapping to nonnegative real numbers), a number of deliberation steps N , and a target value K . We are asked whether we can distribute the deliberation steps across the

¹Because the anytime algorithm’s performance differs across instances, each instance has its own performance profile (in the setting of deterministic performance profiles). In reality, an anytime algorithm’s performance on an instance cannot be predicted perfectly. Rather, usually statistical performance profiles are kept that aggregate across instances. In that light one might question the assumption that different instances have different performance profiles. However, sophisticated deliberation control systems can condition the performance prediction on features of the instance—and this is necessary if the deliberation control is to be fully normative. (Research has already been conducted on conditioning performance profiles on instance features [8, 9, 15] or results of deliberation on the instance so far [4, 8, 9, 15, 18–20].)

problem instances to get a total performance of at least K ; that is, whether there exists a vector $(N_1, N_2, \dots, N_m)$ with $\sum_{1 \leq i \leq m} N_i \leq N$ and $\sum_{1 \leq i \leq m} f_i(N_i) \geq K$.

A reasonable approach to representing the performance profiles is to use piecewise linear performance profiles. They can model any performance profile arbitrarily closely, and have been used in the resource-bounded reasoning literature to characterize the performance of anytime algorithms (e.g. [2]). We now show that the metareasoning problem is $\mathcal{NP}$ -complete even under this restriction. We will reduce from the KNAPSACK problem.²

Definition 2 (KNAPSACK) We are given a set S of m pairs of positive integers (c_i, v_i) , a constraint $C > 0$ and a target value $V > 0$. We are asked whether there exists a set $I \subseteq S$ such that $\sum_{j \in I} C_j \leq C$ and $\sum_{j \in I} v_j \geq V$.

Theorem 1 PERFORMANCE-PROFILES is $\mathcal{NP}$ -complete even if each performance profile is continuous and piecewise linear.³

Proof: The problem is in $\mathcal{NP}$ because we can nondeterministically generate the N_i in polynomial time (since we do not need to bother trying numbers greater than N), and given the N_i , we can verify if the target value is reached in polynomial time. To show $\mathcal{NP}$ -hardness, we reduce an arbitrary KNAPSACK instance to the following PERFORMANCE-PROFILES instance. Let there be m performance profiles, given by $f_i(n) = 0$ for $n \leq c_i$, $f_i(n) = n - c_i$ for $c_i < n \leq c_i + v_i$, and $f_i(n) = v_i$ for $n > c_i + v_i$. Let $N = C + V$ and let $K = V$. We claim the two problem instances are equivalent. First suppose there is a solution to the KNAPSACK instance, that is, a set $I \subseteq S$ such that $\sum_{j \in I} C_j \leq C$ and

$\sum_{j \in I} v_j \geq V$. Then set the N_i as follows: $N_i = 0$ for $i \notin I$, and $N_i = c_i + \sum_{j \in I} \frac{V}{v_j} v_i$ for $i \in I$. Then, $\sum_{1 \leq i \leq m} N_i = \sum_{i \in I} N_i = \sum_{i \in I} (c_i + \sum_{j \in I} \frac{V}{v_j} v_i) = \sum_{i \in I} c_i + \sum_{j \in I} \frac{V}{v_j} \sum_{i \in I} v_i = \sum_{i \in I} c_i + V \leq C + V = N$ since $\sum_{j \in I} C_j \leq C$. Also, since $\sum_{j \in I} v_j \geq V$, it follows that for every $i \in I$, we have $c_i \leq N_i \leq c_i + v_i$, and hence $\sum_{1 \leq i \leq m} f_i(N_i) = \sum_{i \in I} f_i(N_i) = \sum_{i \in I} (\sum_{j \in I} \frac{V}{v_j} v_i) =$

²This only demonstrates *weak* NP-completeness, as KNAPSACK is weakly NP-complete; thus, perhaps pseudopolynomial time algorithms exist.

³If one additionally assumes that each performance profile is concave, then the metareasoning problem is solvable in polynomial time [2]. While returns to deliberation indeed tend to be diminishing, usually this is not the case throughout the performance profile. Algorithms often have a setup phase in the beginning during which there is no improvement. Also, iterative improvement algorithms can switch to using different local search operators once progress has ceased using one operator (for example, once 2-swap has reached a local optimum in TSP, one can switch to 3-swap and obtain gains from deliberation again) [16].

$\sum_{j \in I} \frac{V}{v_j} \sum_{i \in I} v_i = V = K$. So we have found a solution to the PERFORMANCE-PROFILES instance. On the other hand, suppose there is a solution to the PERFORMANCE-PROFILES instance, that is, a vector $(N_1, N_2, \dots, N_m)$ with $\sum_{1 \leq i \leq m} N_i \leq N$ and $\sum_{1 \leq i \leq m} f_i(N_i) \geq K$. Since spending more than $c_i + v_i$ deliberation steps on profile i is useless, we may assume that $N_i \leq c_i + v_i$ for all i . We now claim that $I = \{i : N_i \geq c_i\}$ is a solution to the KNAPSACK instance. First, using the fact that $f_j(N_j) = 0$ for all $j \notin I$, we have $\sum_{i \in I} v_i \geq \sum_{i \in I} f_i(N_i) = \sum_{1 \leq i \leq m} f_i(N_i) \geq K = V$. Also, $\sum_{i \in I} c_i = \sum_{i \in I} N_i - \sum_{i \in I} (N_i - c_i) = \sum_{i \in I} N_i - \sum_{i \in I} f_i(N_i) \leq \sum_{1 \leq i \leq m} N_i - \sum_{1 \leq i \leq m} f_i(N_i) \leq N - K = C + V - V = C$. So we have found a solution to the KNAPSACK instance. ■

The PERFORMANCE-PROFILES problem occurs naturally as a subproblem within many metareasoning problems, and thus its complexity leads to significant difficulties for metareasoning. This is the case even under the (unrealistic) assumption of perfect predictability of the efficacy of deliberation. On the other hand, in the remaining two metareasoning problems that we analyze, the complexity stems from uncertainty about the results that deliberation will provide.

3 Dynamically allocating evaluation effort across options (actions)

In this section we study the setting where an agent is faced with multiple options (actions) from which it eventually has to choose one. The agent can use deliberation (or information gathering) to evaluate each action. Given limited time, it has to decide which ones to evaluate. We show that this is hard even in very restricted cases.

3.1 Motivating example

Consider an autonomous robot looking for precious metals. It can choose between three sites for digging (it can dig at most one site). At site A it may find gold; at site B , silver; at site C , copper. If the robot chooses not to dig anywhere, it gets utility 1 (for saving digging costs). If the robot chooses to dig somewhere, the utility of finding nothing is 0; finding gold, 5; finding silver, 3; finding copper, 2. The prior probability of there being gold at site A is $\frac{1}{8}$, that of finding silver at site B is $\frac{1}{2}$, and that of finding copper at site C is $\frac{1}{2}$.

In general, the robot could perform deliberation or information gathering actions to evaluate the alternative (digging) actions. The metareasoning problem would be the same for both, so for simplicity of exposition, we will focus on information gathering only. Specifically, the robot can perform tests to better evaluate the likelihood of there being a precious metal at each site, but it has only limited time for such tests. The tests are the following: (1) Test for gold at A . If there is gold, the test will be positive with probability $\frac{14}{15}$; if there is no gold, the test will be positive with probability $\frac{1}{15}$. This test takes 2 units of time. (2) Test for silver at B . If there is silver, the test will be positive with probability 1; if there

is no silver, the test will be positive with probability 0. This test takes 3 units of time. (3) Test for copper at C . If there is copper, the test will be positive with probability 1; if there is no copper, the test will be positive with probability 0. This test takes 2 units of time.

Given the probabilities of the tests turning out positive under various circumstances, one can use Bayes' rule to compute the expected utility of each digging option given any (lack of) test result. For instance, letting a be the event that there is gold at A , and t_A be the event that the test at A is positive, we observe that $P(t_A) = P(t_A|a)P(a) + P(t_A|-a)P(-a) = \frac{14}{15} \frac{1}{8} + \frac{1}{15} \frac{7}{8} = \frac{7}{40}$. Then, the expected utility of digging at A given that the test at A was positive is $5 P(a|t_A)$, where $P(a|t_A) = \frac{P(t_A|a)P(a)}{P(t_A)} = \frac{\frac{14}{15} \frac{1}{8}}{\frac{7}{40}} = \frac{2}{3}$, so the expected utility is $\frac{10}{3}$. Doing a similar analysis everywhere, we can represent the problem by trees shown in Fig. 2. In these trees, be-

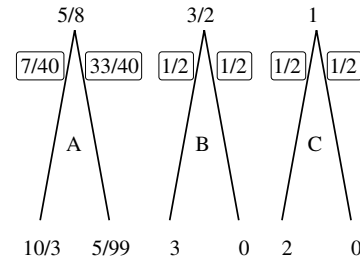


Figure 2: Tree representation of the action evaluation instance.

ing at the root represents not having done a test yet, whereas being at a left (right) leaf represents the test having turned out positive (negative); the value at each node is the expected value of digging at this site given the information corresponding to that node. The values on the edges are the probabilities of the test turning out positive or negative. We can subsequently use these trees for analyzing how we should gather information. For instance, if we have 5 units of time, the optimal information gathering policy is to test at B first; if the result is positive, test at A ; otherwise test at C . (We omit the proof because of space constraint.)

3.2 Definitions

In the example, there were four *actions* that we could evaluate: digging for a precious metal at one of three locations, or not digging at all. Given the results of all the tests that we might undertake on a given action, executing it has some expected value. If, on the other hand, we do not (yet) know all the results of these tests, we can still associate an expected value with the action by taking an additional expectation over the outcomes of the tests. In what follows, we will drop the word “expected” in its former meaning (that is, when talking about the expected value given the outcomes of all the tests), because the probabilistic process regarding this expectation has no relevance to how the agent should choose to test. Hence, all expectations are over the outcomes of the tests.

While we have presented this as a model for information gathering planning, we can use this as a model for planning

(computational) deliberation over multiple actions as well. In this case, we regard the tests as computational steps that the agent can take toward evaluating an action.⁴

To proceed, we need a formal model of how evaluation effort (information gathering or deliberation) invested on a given action changes the agent's beliefs about that action. For this model, we generalize the above example to the case where we can take multiple evaluation steps on a certain action (although we will later show hardness even when we can take at most one evaluation step per action).

Definition 3 An action evaluation tree is a tree with

- A root r , representing the start of the evaluation;
- For each nonleaf node w , a cost k_w for investing another step of evaluation effort at this point;
- For each edge e between parent node p and child node c , a probability $p_e = p_{(p,c)}$ of transitioning from p to c upon taking a step of evaluation effort at p ;
- For each leaf node l , a value u_l .

According to this definition, at each point in the evaluation of a single action, the agent's only choice is *whether* to invest further evaluation effort, but not *how* to continue the evaluation. This is a reasonable model when the agent does evaluation through deliberation and has one algorithm at its disposal. However, in general the agent may have different information gathering actions to choose from at a given point in the evaluation, or may be able to choose from among several deliberation actions (e.g., via search control [1, 14]). In Section 4, we will discuss how being able to choose between tests may introduce drastic complexity even when evaluating a single thing. In this section, however, our focus is on the complexities introduced by having to choose between different actions on which to invest evaluation effort next.

The agent can determine its expected value of an action, given its evaluation so far, using the subtree of the action evaluation tree that is rooted at the node where evaluation has brought us so far. This value can be determined in that subtree by propagating upwards from the leafs: for parent p with a set of children C , we have $u_p = \sum_{c \in C} (p_{(p,c)} u_c)$.

We now present the metareasoning problem. In general, the agent could use an *online* evaluation control policy where the choices of how to invest future evaluation effort can depend on evaluation results obtained so far. However, to avoid trivial complexity issues introduced by the fact that such a *contingency* strategy for evaluation can be exponential in size, we merely ask what action the agent should invest its first evaluation step on.

Definition 4 (ACTION-EVALUATION) We are given l action evaluation trees, indexed 1 through l , corresponding to l different actions. (The transition processes of the trees are independent.) Additionally, we are given an integer N . We are asked whether, among the online evaluation control policies that spend at most N units of effort, there exists one that

⁴For this to be a useful model, it is necessary that updating beliefs about the value of an action (after taking a deliberation step) is computationally easy relative to the evaluation problem itself.

takes its first evaluation step on action 1, and gives maximal expected utility among online evaluation control policies that spend at most N units of effort. (If at the end of the deliberation process, we are at node w_i for tree i , then our utility is $\max_{1 \leq i \leq m} u_{w_i}$, because we will choose the action with the highest expected value.)

3.3 Results

We now show that even a severely restricted version of this problem is $\mathcal{NP}$ -hard.⁵

Theorem 2 ACTION-EVALUATION is $\mathcal{NP}$ -hard, even when all trees have depth either 0 or 1, branching factor 2, and all leaf values are -1, 0, or 1.

Proof: If action evaluation tree j has depth 1 and branching factor 2, we represent it by $(p_1^j, p_2^j, u_1^j, u_2^j, k^j)$ where p_i^j is the transition probability to leaf i , u_i^j is the value at leaf i , and k^j is the cost of taking the (only) step of evaluation. We reduce an arbitrary KNAPSACK instance to the following ACTION-EVALUATION instance. Let $l = m + 3$, let $\delta = \frac{1}{16m(\sum_{1 \leq i \leq m} v_i)^2}$, and let $\epsilon = 2\delta \sum_{1 \leq i \leq m} v_i = \frac{1}{8m \sum_{1 \leq i \leq m} v_i}$.

We set tree 1 $(p_1^1, p_2^1, u_1^1, u_2^1, k^1) = (\epsilon, 1 - \epsilon, 1, -1, 1)$, and tree 2 $(p_1^2, p_2^2, u_1^2, u_2^2, k^1) = ((1 - \epsilon)^m(\epsilon + \delta V), 1 - (1 - \epsilon)^m(\epsilon + \delta V), 1, -1, C + 1)$ ⁶. Tree 3 has depth 0 and a value of 0. Finally, for each pair (c_i, v_i) in the KNAPSACK instance, there is an evaluation tree $(p_1^{i+3}, p_2^{i+3}, u_1^{i+3}, u_2^{i+3}, k^{i+3}) = (\delta v_i, 1 - \delta v_i, 1, -1, c_i)$. We set $N = C + 1$. We claim the instances are equivalent. First we make some observations about the constructed ACTION-EVALUATION instance. First, once we determine the value of a action to be 1, choosing this action is certainly optimal regardless of the rest of the deliberation process. Second, if at the end of the deliberation process we have not discovered the value of any action to be 1, then for any of the trees of depth 1, either we have discovered the corresponding action's value to be -1, or we have done no deliberation on it at all. In the latter case, the expected value of the action is always below 0 (δ is carefully set to achieve this). Hence, we will pick action 3 for value 0. It follows that an optimal deliberation policy is one that maximizes the probability of discovering that a action has value 1. Now, consider the *test set* of a policy, which is the set of actions that the policy would evaluate if no action turned out to have value 1. Then, the probability of discovering that a action has value 1 is simply equal to the probability that at least one of the actions in this set has value 1. So, in this case, the quality of a policy is determined by its test set. Now we observe that any optimal action is either the one that only evaluates action 2 (and then runs out of deliberation time), or one that has action 1 in its

⁵ACTION-EVALUATION is trivial for $l = 1$: the answer is "yes" if it is possible to take a step of evaluation. The same is true if there is no uncertainty with regard to the value of any action; in that case any evaluation is irrelevant.

⁶Note that using m in the exponent does not make the reduction exponential in size, because the length of the binary representation of numbers with m in the exponent is linear in m .

test set. (For consider any other policy; since evaluating action 1 has minimal cost, and gives strictly higher probability of discovering a action with value 1 than evaluating on any other action besides 2, simply replacing any other action in the test set with action 1 is possible and improves the policy.) Now suppose there is a solution to the KNAPSACK instance, that is, a set $I \subseteq S$ such that $\sum_{i \in I} C_i \leq C$ and

$\sum_{i \in I} v_i \geq V$. Then we can construct a policy which has as

test set $J = \{1\} \cup \{j : j - 3 \in I\}$. (Evaluating all these actions costs at most $C + 1$ deliberation units.) The probability of at least one of these actions having value 1 is at least the probability that exactly one of them has value 1, which is $\sum_{j \in J} p_1^j \prod_{k \in J, k \neq j} p_2^k \geq \sum_{j \in J} p_1^j (1 - \epsilon)^m = (1 - \epsilon)^m (\epsilon + \sum_{i \in I} \delta v_i) \geq (1 - \epsilon)^m (\epsilon + \delta V) = p_1^2$. Using our previous observation we can conclude that there is an optimal action that has action 1 in its test set, and since the order in which we evaluate actions in the test set does not matter, there is an optimal policy which evaluates action 1 first. On the other hand, suppose there is no solution to the KNAPSACK instance. Consider a policy which has 1 in its test set, that is, the test set can be expressed as $J = \{1\} \cup \{j : j - 3 \in I\}$ for some set I . Then we must have $\sum_{i \in I} C_i \leq C$, and since there is no solution to

the KNAPSACK instance, it follows that $\sum_{i \in I} v_i \leq V - 1$. But

the probability that at least one of the actions in the test set has value 1 is at most $\sum_{j \in J} p_1^j = \epsilon + \sum_{i \in I} \delta v_i \leq \epsilon + \delta(V - 1) = \epsilon + \delta V - \delta$. On the other hand, $p_1^2 = (1 - \epsilon)^m (\epsilon + \delta V) \geq (1 - m\epsilon)(\epsilon + \delta V) \geq \epsilon + \delta V - 2m\epsilon^2$. If we now observe that $2m\epsilon^2 = \frac{1}{32m(\sum_{1 \leq i \leq m} v_i)^2} < \frac{1}{16m(\sum_{1 \leq i \leq m} v_i)^2} = \delta$, it follows

that the policy of just evaluating action 2 is strictly better. So, there is no optimal policy which evaluates action 1 first. ■

We have no proof that the general problem is in $\mathcal{NP}$. It is an interesting open question whether stronger hardness results can be obtained for it. For instance, perhaps the general problem is $\mathcal{PSPACE}$ -complete.

4 Dynamically choosing how to disambiguate state

We now move to the setting where the agent has only one thing to evaluate, but can choose the order of deliberation (or information gathering) actions for doing so. In other words, the agent has to decide how to disambiguate its state. We show that this is hard. (We consider this to be the most significant result in the paper.)

4.1 Motivating example

Consider an autonomous robot that has discovered it is on the edge of the floor; there is a gap in front of it. It knows this gap can only be one of three things: a staircase (S), a hole (H), or a canyon (C) (assume a uniform prior distribution over these). The robot would like to continue its exploration beyond the gap. There are three courses of physical action available to the robot: attempt a descent down a staircase, attempt to jump

over a hole, or simply walk away. If the gap turns out to be a staircase and the robot descends down it, this gives utility 2. If it turns out to be a hole and the robot jumps over it, this gives utility 1 (discovering new floors is more interesting). If the robot walks away, this gives utility 0 no matter what the gap was. Unfortunately, attempting to jump over a staircase or canyon, or trying to descend into a hole or canyon, has the disastrous consequence of destroying the robot (utility $-\infty$). It follows that if the agent cannot determine with certainty what the gap is, it should walk away.

In order to determine the nature of the gap, the robot can conduct various tests (or *queries*). The tests can determine the answers to the following questions: (1) Am I inside a building? A *yes* answer is consistent only with S ; a *no* answer is consistent with S, H, C . (2) If I drop a small item into the gap, do I hear it hit the ground? A *yes* answer is consistent with S, H ; a *no* answer is consistent with H, C . (3) Can I walk around the gap? A *yes* answer is consistent with S, H ; a *no* answer is consistent with S, H, C .

Assume that if multiple answers to a query are consistent with the true state of the gap, the distributions over such answers are uniform and independent. Note that after a few queries, the set of states consistent with all the answers is the intersection of the sets consistent with the individual answers; once this set has been reduced to one element, the robot knows the state of the gap.

Suppose the agent only has time to run one test. Then, to maximize expected utility, the robot should run test 1, because the other tests give it no chance of learning the state of the gap for certain. Now suppose that the agent has time for two tests. Then the optimal test policy is as follows: run test 2 first; if the answer is *yes*, run test 1 second; otherwise, run test 3 second. (If the true state is S , this is discovered with probability $\frac{1}{2}$; if it is H , this is discovered with probability $\frac{1}{4}$, so total expected utility is $\frac{5}{12}$. Starting with test 1 or test 3 can only give expected utility $\frac{1}{3}$.)

4.2 Definitions

We now define the metareasoning problem of how the agent should dynamically choose queries to ask (deliberation or information gathering actions to take) so as to disambiguate the state of the world. While the illustrative example above was for information gathering actions, the same model applies to deliberation actions for state disambiguation (such as image processing, auditory scene analysis, sensor fusing, etc.).

Definition 5 (STATE-DISAMBIGUATION) *We are given*

- A set $\Theta = \{\theta_1, \theta_2, \dots, \theta_r\}$ of possible world states;⁷
- A probability function p over Θ ;

⁷If there are two situations that are equivalent from the agent's point of view (the agent's optimal course of action is the same and the utility is the same), then we consider those situations to be one state. Note that two such situations may lead to different answers to the queries. For example, one situation may be that the gap is an indoor staircase, and another situation may be that the gap is an outdoor staircase. These situations are considered to be the same state, but will give different answers to the query "Am I inside?"

- A utility function $u : \Theta \rightarrow \mathbb{R}^{\geq 0}$ where $u(\theta_i)$ gives the utility of knowing for certain that the world is in state θ_i at the end of the metareasoning process; (not knowing the state of the world for certain always gives utility 0);
- A query set Q , where each $q \in Q$ is a list of subsets of Θ . Each such subset corresponds to an answer to the query, and indicates the states that are consistent with that answer. We require that for each state, at least one of the answers is consistent with it: that is, for any $q = (a_1, a_2, \dots, a_m)$, we have $\bigcup_{1 \leq j \leq m} a_j = \Theta$. When a query is asked, the answer is chosen (uniformly) randomly by nature from the answers to that query that are consistent with the world's true state (these drawings are independent);
- An integer N ; A target value G .

We are asked whether there exists a policy for asking at most N queries that gives expected utility at least G . (Letting $\pi(\theta_t)$ be the probability of identifying the state when it is θ_t , the expected utility is given by $\sum_{1 \leq t \leq r} p(\theta_t) \pi(\theta_t) u(\theta_t)$.)⁸

4.3 Results

Before presenting our $\mathcal{PSPACE}$ -hardness result, we will first present a relatively straightforward $\mathcal{NP}$ -hardness result for the case where for each query, only one answer is consistent with the state of the world. This situation occurs when the states are so specific as to provide enough information to answer every query. Our reduction is from SET-COVER.

Definition 6 (SET-COVER) We are given a set S , a collection of subsets $\mathcal{T} = \{T_i \subseteq S\}$, and a positive integer M . We are asked whether any M of these subsets cover S , that is, whether there is a subcollection $\mathcal{U} \subseteq \mathcal{T}$ such that $|\mathcal{U}| = M$ and $\bigcup_{T_i \in \mathcal{U}} T_i = S$.

Theorem 3 *STATE-DISAMBIGUATION* is $\mathcal{NP}$ -hard, even when for each state-query pair there is only one consistent answer.

Proof: We reduce an arbitrary SET-COVER instance to the following STATE-DISAMBIGUATION instance. Let $\Theta = S \cup \{b\}$. Let p be uniform. Let $u(b) = 1$ and for any $s \in S$, let $u(s) = 0$. Let $Q = \{(\Theta - T_i, T_i) : T_i \in \mathcal{T}\}$. Let $M = N$ and let $G = \frac{1}{|S|+1}$. We claim the instances are equivalent.

⁸There are several natural generalizations of this metareasoning problem, each of which is at least as hard as the basic variant. One allows for positive utilities even if there remains some uncertainty about the state at the end of the disambiguation process. In this more general case, the utility function would have subsets of Θ as its domain (or perhaps even probability distributions over such subsets). In general, specifying such utility functions would require space exponential in the number of states, so some restriction of the utility function is likely to be necessary; nevertheless, there are utility functions specifiable in polynomial space that are more general than the one given here. Another generalization is to allow for different distributions for the query answers given. One could also attribute different execution costs to different queries. Finally, it is possible to drop the assumption that queries completely rule out certain states, and rather take a probabilistic approach.

First suppose there is a solution to the SET-COVER instance, that is, a subcollection $\mathcal{U} \subseteq \mathcal{T}$ such that $|\mathcal{U}| = M$ and $\bigcup_{T_i \in \mathcal{U}} T_i = S$. Then our policy for the STATE-DISAMBIGUATION instance is simply to ask the queries corresponding to the elements of $\mathcal{U}$, in whichever order and unconditionally on the answers of the query. If the true state is in S , we will get utility 0 regardless. If the true state is b , each query will eliminate the elements of the corresponding T_i from consideration. Since $\mathcal{U}$ is a set cover, it follows that after all the queries have been asked, all elements of S have been eliminated, and we know that the true state of the world is b , to get utility 1. So the expected utility is $\frac{1}{|S|+1}$, so there is a solution to the STATE-DISAMBIGUATION instance.

On the other hand, suppose there is a solution to the STATE-DISAMBIGUATION instance, that is, a policy for asking at most N queries that gives expected utility at least G . Because given the true state of the world, there is only one answer consistent with it for each query, it follows that the queries that will be asked, and the answers given, follow deterministically from the true state of the world. Since we cannot derive any utility from cases where the true state of the world is not b , it follows that when it is b , we must be able to conclude that this is so in order to get positive expected utility. Consider the queries that the policy will ask in this latter case. Each of these queries will eliminate precisely the corresponding T_i . Since at the end of the deliberation, all the elements of S must have been eliminated, it follows that these T_i in fact cover S . Hence, if we let $\mathcal{U}$ be the collection of these T_i , this is a solution to the SET-COVER instance. ■

We are now ready to present our $\mathcal{PSPACE}$ -hardness result. The reduction is from stochastic satisfiability, which is $\mathcal{PSPACE}$ -complete [11].

Definition 7 (STOCHASTIC-SAT (SSAT)) We are given a Boolean formula in conjunctive normal form (with a set of clauses C over variables $x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n$). We play the following game with nature: we pick a value for x_1 , subsequently nature (randomly) picks a value for y_1 , whereupon we pick a value for x_2 , after which nature picks a value for y_2 , etc., until all variables have a value. We are asked whether there is a policy (contingency plan) for playing this game such that the probability of the formula being eventually satisfied is at least $\frac{1}{2}$.

Now we can present our $\mathcal{PSPACE}$ -hardness result.

Theorem 4 *STATE-DISAMBIGUATION* is $\mathcal{PSPACE}$ -hard.

Proof: Let $\Theta = C \cup \{b\} \cup V$ where V consists of the elements of an upper triangular matrix, that is, $V = \{v_{11}, v_{12}, \dots, v_{1n}, v_{22}, v_{23}, \dots, v_{2n}, v_{33}, \dots, v_{nn}\}$. p is uniform over this set. u is defined as follows: $u(c) = 0$ for all $c \in C$, $u(b) = 1$, and $u(v_{ij}) = H = 2 \prod_{q \in Q} N_{ans}(q)$ for

all $v_{ij} \in V$, where $N_{ans}(q)$ is the number of possible answers to q . The queries are as follows. For every $v_{ij} \in V$, there is a query $q_{ij} = (\{v_{ij}\}, \Theta - \{v_{ij}\})$. Additionally, for each variable x_i there are the following two queries: letting $V_i = \{v_{ij} : j \geq i\}$ (that is, row i in the matrix), and letting $C_z = \{c \in C : z \in c\}$, we have

- $q_{x_i} = (V_i, C, \Theta - V_i - C_{x_i} - C_{y_i}, \Theta - V_i - C_{x_i} - C_{-y_i})$;

- $q_{-x_i} = (V_i, C, \Theta - V_i - C_{-x_i} - C_{y_i}, \Theta - V_i - C_{-x_i} - C_{-y_i})$.

We have n steps of deliberation. Finally, the goal is $G = \frac{2|V|H+1}{2|\Theta|}$. First suppose there is a solution to the SSAT instance, that is, there exists a contingency plan for setting the x_i such that the probability that the formula is eventually satisfied is at least $\frac{1}{2}$. Now, if we ask query q_{x_i} (q_{-x_i}), we say this corresponds to us selecting x_i ($-x_i$); if the answer to query q_{z_i} is $\Theta - V_i - C_{z_i} - C_{y_i}$ ($\Theta - V_i - C_{z_i} - C_{y_i}$), we say this corresponds to nature selecting y_i ($-y_i$). Then, consider the following contingency plan for asking queries:

- Start by asking the query corresponding to how the first variable is set in the SSAT instance (that is, q_{x_1} if x_1 is set to *true*, q_{-x_1} if x_1 is set to *false*);
- So long as all the queries and answers correspond to variables being selected, we follow the SSAT contingency plan; that is, whenever we have to ask a query, we ask the query that corresponds to the variable that would be selected in the SSAT contingency plan if variables so far had been selected in a manner corresponding to the queries and answers we have seen;
- If, on the other hand, we get V_i as an answer, we proceed to ask $v_{ii}, v_{i(i+1)}, \dots, v_{i(n-1)}$ in that order;
- Finally, if we get C as an answer, we simply stop.

We make two observations about this policy. First, if the true state of the world is one of the v_{ij} , we will certainly discover this. (Upon asking query i , which is q_{x_i} or $-q_{x_i}$, we will receive answer V_i and switch to q_{ik} queries; then if $j < n$, query $j + 1$ will be q_{ij} , we will receive answer $\{v_{ij}\}$, and know the state; whereas if $j = n$, we will eliminate all the other elements of V_{ij} with queries $i + 1$ through n , and know the state.) Second, if the true state is b , for any i ($1 \leq i \leq n$), query i will be either q_{x_i} or $-q_{x_i}$. This will certainly eliminate all the v_{ij} , so we will know the state at the end if and only if we also manage to eliminate all the clauses. But now notice that each query-answer pair eliminates exactly the same clauses as the corresponding variable selections satisfy. It follows that we will know the state in the end if and only if these corresponding variable selections satisfy all the literals. But the process by which the queries and answers are selected is exactly the same as in the SSAT instance with the solution policy. It follows we discover the true state with probability at least $\frac{1}{2}$. Hence, our total expected utility is at least $\frac{|V|}{|\Theta|}H + \frac{1}{|\Theta|}\frac{1}{2} = G$. So there is a policy that achieves the goal.

Now suppose there is a policy that achieves the goal. We first claim that such a policy will always discover the true state if it is one of the v_{ij} . For if a policy does not manage this, then there is some v_{ij} such that for some combination of answers consistent with v_{ij} , the policy will not discover the state. Suppose this is indeed the true state. Since each consistent answer to query q occurs with probability at least $\frac{1}{N_{ans}(q)}$, it follows that the unfavorable combination of answers occurs with probability at least $\prod_{q \in Q} \frac{1}{N_{ans}(q)}$. It follows that even if we discover the true state in every other scenario,

our expected utility is at most $(\frac{|V|}{|\Theta|} - \frac{1}{|\Theta|} \prod_{q \in Q} \frac{1}{N_{ans}(q)})H + \frac{1}{|\Theta|} = G + \frac{1}{|\Theta|}(-2 + \frac{1}{2}) < G$. Now, it is straightforward to show that this implies that so long as no answer has been one of the V_i or C , query i ($1 \leq i < n$) is either q_{x_i} or $-q_{x_i}$. Query n may still be q_{nn} under these conditions, but since queries q_{x_n} and q_{-x_n} are both more informative than q_{nn} , we may assume that the policy that achieves the target value asks one of the former two in this case as well. It follows that the part of this policy that handles the cases where no answers have been either one of the V_i or C corresponds exactly to a valid SSAT policy, according to the correspondence between queries/answers and variable selections outlined earlier in the proof. But now we observe, as before, that if the true state is b , the probability that we discover this with the STATE-DISAMBIGUATION policy is precisely the probability that this SSAT policy satisfies all the clauses. This probability must be at least $\frac{1}{2}$ in order for the STATE-DISAMBIGUATION policy to reach the target expected utility value. So there is a solution to the SSAT instance. ■

The following theorem allows us to make any hardness result on STATE-DISAMBIGUATION go through even when restricting ourselves to a uniform prior over states, or to a constant utility function over the states.

Theorem 5 *Every STATE-DISAMBIGUATION instance is equivalent to another STATE-DISAMBIGUATION instance with a uniform prior p , and to another with a constant utility function u ($u(\theta_t) = 1$ for all $\theta_t \in \Theta$). Moreover, these equivalent instances can be constructed in linear time.*

Proof: The only relevance of p and u in STATE-DISAMBIGUATION is to the policy's expected utility $\sum_{1 \leq t \leq r} p(\theta_t)\pi(\theta_t)u(\theta_t)$. So, only the products $p(\theta_t)u(\theta_t)$ matter; adding a constant factor to them also makes no difference if we correct G accordingly. Hence, any instance is equivalent to one where we replace p and u by $p'(\theta_t) = \frac{1}{|\Theta|}$ and $u'(\theta_t) = |\Theta|p(\theta_t)u(\theta_t)$. It is also equivalent to one where we replace p , u and G by $p''(\theta_t) = \frac{p(\theta_t)u(\theta_t)}{\sum_{1 \leq i \leq r} (p(\theta_i)u(\theta_i))}$, $u''(\theta_t) =$

$$1, G'' = \frac{G}{\sum_{1 \leq i \leq r} (p(\theta_i)u(\theta_i))}. \quad \blacksquare$$

5 Conclusion and future research

In most real-world settings, due to limited time or other resources, an agent cannot perform all potentially useful deliberation and information gathering actions. This leads to the metareasoning problem of selecting such actions carefully. Decision-theoretic methods for metareasoning have been studied in AI for the last 15 years, but there are few theoretical results on the complexity of metareasoning.

We derived hardness results for three metareasoning problems. In the first, the agent has to decide how to allocate its deliberation time across anytime algorithms running on different problem instances. We showed this to be $\mathcal{NP}$ -complete. In the second, the agent has to (dynamically) allocate its deliberation or information gathering resources across

multiple actions that it has to choose among. We showed this to be $\mathcal{NP}$ -hard even when evaluating each individual action is very simple. In the third, the agent has to (dynamically) choose a limited number of deliberation or information gathering actions to disambiguate the state of the world. We showed that this is $\mathcal{NP}$ -hard under a natural restriction, and $PSPACE$ -hard in general.

Our results have general applicability in that most metareasoning systems must somehow deal with one or more of these problems (in addition to dealing with other issues). The results are not intended as an argument against metareasoning or decision-theoretic deliberation control. However, they do show that the metareasoning policies directly suggested by decision theory are not always feasible. This leaves several interesting avenues for future research: 1) investigating the complexity of metareasoning when deliberation (and information gathering) is costly rather than limited, 2) developing optimal metareasoning algorithms that usually run fast (albeit, per our results, not always), 3) developing fast optimal metareasoning algorithms for special cases, 4) developing approximately optimal metareasoning algorithms that are always fast, and 5) developing meta-metareasoning algorithms to control the meta-reasoning, etc.

References

- [1] Eric B Baum and Warren D Smith. A Bayesian approach to relevance in game playing. *Artificial Intelligence*, 97(1–2):195–242, 1997.
- [2] Mark Boddy and Thomas Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67:245–285, 1994.
- [3] Irving Good. Twenty-seven principles of rationality. In V Godambe and D Sprott, eds, *Foundations of Statistical Inference*. Toronto: Holt, Rinehart, Winston, 1971.
- [4] E Hansen and S Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126:139–157, 2001.
- [5] Eric Horvitz. Principles and applications of continual computation. *Artificial Intelligence*, 126:159–196, 2001.
- [6] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. *Workshop on Uncertainty in AI*, pp. 429–444, Seattle, Washington, 1987. American Assoc. for AI. Also in L. Kanal, T. Levitt, and J. Lemmer, eds, *Uncertainty in AI 3*, Elsevier, 1989, pp. 301–324.
- [7] Ronald Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, 1966.
- [8] Kate Larson and Tuomas Sandholm. Bargaining with limited computation: Deliberation equilibrium. *Artificial Intelligence*, 132(2):183–217, 2001. Early version: AAAI, pp. 48–55, Austin, TX, 2000.
- [9] Kate Larson and Tuomas Sandholm. Costly valuation computation in auctions. *Theoretical Aspects of Rationality and Knowledge (TARK)*, pp. 169–182, 2001.
- [10] James E Matheson. The economic value of analysis and computation. *IEEE Transactions on Systems Science and Cybernetics*, 4(3):325–332, 1968.
- [11] C Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31:288–301, 1985.
- [12] David Parkes and Lloyd Greenwald. Approximate and compensate: A method for risk-sensitive meta-deliberation and continual computation. In *AAAI Fall Symposium on Using Uncertainty within Computation*, 2001.
- [13] S Russell and D Subramanian. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, 1:1–36, 1995.
- [14] S Russell and E Wefald. *Do the right thing: Studies in Limited Rationality*. 1991.
- [15] Tuomas Sandholm and Victor Lesser. Utility-based termination of anytime algorithms. *ECAI Workshop on Decision Theory for DAI Applications*, pp. 88–99, Amsterdam, 1994. Extended version: UMass CS tech report 94-54.
- [16] Tuomas Sandholm and Victor Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94:99–137, 1997. Early version: IJCAI-95, 662–669.
- [17] Herbert A Simon. *Models of bounded rationality*, volume 2. MIT Press, 1982.
- [18] Shlomo Zilberstein, François Charnipillet, and Philippe Chassaing. Real-time problem solving with contract algorithms. *IJCAI*, pp. 1008–1013, 1999.
- [19] Shlomo Zilberstein and Abdel-Ilhah Mouaddib. Reactive control of dynamic progressive processing. *IJCAI*, pp. 1268–1273, 1999.
- [20] Shlomo Zilberstein and Stuart Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1–2):181–213, 1996.

Approximating Optimal Policies for Agents with Limited Execution Resources

Dmitri A. Dolgov and Edmund H. Durfee

Department of Electrical Engineering and Computer Science

University of Michigan

Ann Arbor, MI 48109

{ddolgov, durfee}@umich.edu

Abstract

An agent with limited consumable execution resources needs policies that attempt to achieve good performance while respecting these limitations. Otherwise, an agent (such as a plane) might fail catastrophically (crash) when it runs out of resources (fuel) at the wrong time (in midair). We present a new approach to constructing policies for agents with limited execution resources that builds on principles of real-time AI, as well as research in constrained Markov decision processes. Specifically, we formulate, solve, and analyze the policy optimization problem where constraints are imposed on the probability of exceeding the resource limits. We describe and empirically evaluate our solution technique to show that it is computationally reasonable, and that it generates policies that sacrifice some potential reward in order to make the kinds of precise guarantees about the probability of resource overutilization that are crucial for mission-critical applications.

1 Introduction

Optimality is the gold standard in rational decision making (e.g., [Russell and Subramanian, 1995]), and, consequently, the problem of constructing optimal policies for autonomous agents has received considerable attention over the years. Traditionally, this problem has been viewed separately from the problem of actually carrying out the policies. However, real agents have limitations as to what they can execute, and, clearly, a policy is less useful if an agent might run out of resources while carrying out the policy.

In this paper, we present a new approach for constructing policies for agents that have limited consumable resources where running out of the resources can have negative consequences. Whereas AI research has mostly focused on (PO)MDP [Boutilier *et al.*, 1999; Dean *et al.*, 1993; Howard, 1960; Puterman, 1995] methods for formulating policies for agents without emphasizing constraints on their execution resources, the Operations Research literature has developed constrained MDP (CMDP) [Altman, 1999; Puterman, 1995] techniques that can account for resource constraints. CMDP methods are particularly useful for domains

where the current resource amounts are unobservable and cannot be easily estimated by the agent, or where modeling resource amounts in the state description is computationally infeasible. In an aircraft scenario, some resources and situations where such methods are beneficial could include an airplane with a broken fuel gauge (fuel amount is unobservable), pilot fatigue (attention is a resource that cannot be easily estimated), or a combination of non-critical resources (ex. various refreshments) that should nevertheless not be exhausted, but explicit modeling of which unnecessarily complicates the optimization problem and increases policy size. In the rest of the paper, we will use the fuel example, simply because it is a very intuitive instance of a consumable resource.

However, as we will explain later, standard risk-neutral CMDP optimization techniques are not applicable to problems where violating the constraints can have negative or, in the limit, catastrophic¹ consequences. The main contribution of this work is that it extends the standard CMDP techniques to handle the types of hard constraints that naturally arise in problems involving critical resources. In particular, we formulate an optimization problem where constraints are imposed on the *probability* of resource overutilization, and show how the problem can be solved using standard linear programming algorithms. Our formulation yields sub-optimal solutions to the constrained problem because it sacrifices potential reward to make guarantees about the probability of violating the resource constraints. As we later show, when violating constraints incurs dire costs, these guarantees are worthwhile ([Musliner *et al.*, 1995] and references therein).

We introduce our model in section 2, where we review Markov models, introduce notation, and specify our assumptions about the problem domain. Section 3 describes and compares several candidate solutions for addressing aspects of our problem. We then present in section 4 our new method, and empirically evaluate it in section 5. We conclude with a discussion about the strengths and limitations of our results, and about our future directions.

2 The Model

The stochastic properties of the environments in the problems that we are addressing lead us to formulate our op-

¹The term catastrophic is, of course, relative. We assume that the system designer is willing to accept certain risks to receive payoff.

timization problem as a stationary, discrete-time, Markov decision process. In this section, we review some well-known facts from the theory of standard [Puterman, 1995; Boutilier *et al.*, 1999] and constrained [Altman, 1999] fully-observable Markov decision processes and also discuss the assumptions that are particular to the class of problems that we address in this work. This section provides the necessary background for the subsequent sections, where we discuss resource-constrained optimization problems.

2.1 Markov Decision Processes

A standard Markov decision process can be defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathbf{P}, \mathbf{R} \rangle$, where $\mathcal{S}$ is a finite set of states that the agent can be in, $\mathcal{A}$ is a finite set of actions that the agent can execute, $\mathbf{P} = [p_{ij}^a] : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ defines the transition function (p_{ij}^a is the probability that the agent will go to state j if it executes action a in state i), and $\mathbf{R} = [r_{ia}] : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function (agent receives a reward of r_{ia} for executing action a in state i).

Clearly, the total probability of transitioning out of a state, given a particular action, cannot be greater than 1, i.e. $\sum_j p_{ij}^a \leq 1$. As we discuss below, we are actually interested in domains where there exist states for which $\sum_j p_{ij}^a < 1$.

A policy is defined as a procedure for selecting an action in each state. A policy that makes its choices according to a probability distribution over the set of actions is called *randomized* and can be described as a mapping of states to probability distributions over actions: $\pi = [\pi_{ia}] : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. A *deterministic* policy that always chooses the same action for a state is, of course, a special case of a randomized policy. It can be seen (similarly to the case of standard CMDPs, as described in [Kallenberg, 1983; Puterman, 1995]) that under our optimization criterion and constraints (section 4), deterministic policies can be suboptimal. Therefore, in this work, we focus on approximating optimal policies in the class of randomized ones.

If, at time 0, the agent has an initial probability distribution $\alpha = [\alpha_i]$ over the state space, a Markov system follows the following trajectory:

$$\rho_j^{t+1} = \sum_i \left(\sum_a p_{ij}^a \pi_{ia} \right) \rho_i^t, \quad \rho_i^0 = \alpha_i, \quad (1)$$

where $\rho^t = [\rho_i^t]$ is the probability distribution at time t .

2.2 Assumptions

Typically, Markov decision processes are divided into two categories: *finite-horizon* problems, where the total number of steps that the agent spends in the system is finite and is known *a priori*, and *infinite-horizon* problems, where the agent is assumed to stay in the system forever (see [Puterman, 1995] for a detailed discussion of both types of models).

In this work we concentrate on dynamic real-time domains, where agents have tasks to accomplish. For example, consider an agent flying a plane, whose goal is to safely get to its destination and land there. This example does not naturally correspond to a finite-horizon problem, because the duration of executing various policies is not predetermined (unless we artificially impose such a finite duration, which is not easily

justifiable). On the other hand, the problem does not naturally fit the definition of the infinite-horizon model, because the plane obviously cannot keep on flying forever.

This leads us to make a slightly different and less common (although, certainly, not novel) assumption about how much time the agent spends executing its policy. We assume that there is no predefined number of steps that the agent spends in the system, but that optimal policies always yield *transient* Markov processes (decision problems of this type were extensively studied by Kallenberg [1983]). A policy is said to yield a transient Markov process if the agent executing that policy will eventually leave the corresponding Markov chain, after spending a finite number of time steps in it. Given a finite state space, this assumption implies that there has to be some “leakage” of probability out of the system, i.e. there have to exist some state-action pairs (i, a) for which $\sum_j p_{ij}^a < 1$. One particular case where the above assumption holds is in a system in which all trajectories lead to *absorbing* states. Once an agent enters an absorbing state, it has finished (or failed to finish) some task and has nowhere else to go, i.e. the probability of transitioning out of an absorbing state i is zero: $\sum_j \sum_a p_{ij}^a = 0$.² In the plane-flying example, all trajectories lead to either a safe landing or a crash, and once the agent enters one of these states, the probability of transitioning to other states is zero.

The transient nature of our problems leads us to adopt the expected total reward as the policy evaluation criterion. Given that an agent receives a reward whenever it executes an action, the total expected utility of a policy can be expressed as:

$$V(\pi, \alpha) = \sum_{t=0}^T \sum_i \rho_i^t \sum_a \pi_{ia} r_{ia}, \quad (2)$$

where T is the number of steps during which the agent accumulates utility. For a transient system with bounded rewards, the above sum converges for any T .

3 Related Work

In this section, we briefly survey several approaches (based on well-known techniques) to solving the problem of finding optimal policies for agents with limited resources, and point out the assumptions, strengths, and limitations of these methods. This establishes a landscape of solution algorithms, to which we can compare our method (presented in the next section) in terms of complexity, efficiency, and solution quality.

3.1 Fully Observable MDPs

The most straightforward way of handling resource constraints in a MDP framework is to explicitly model the resources by making their current amounts a part of the state description. This yields a standard FOMDP, which can be solved by a wide variety of efficient methods. The benefit

²An alternative way of handling these states is to treat them as infinitely-recurrent, i.e. once the agent gets there, it infinitely transitions back to itself. We do not adopt this model, because it is less natural for our domains and also leads to unnecessary complications in the optimization problems.

of this approach is that it allows one to make use of all relevant information to construct the best possible policy, as the agent is able to base its action choice on the current state and resource amounts. The downside of this approach is that it requires an *a priori* discretization of resource amounts to be made when the world model is constructed. Also, there is an additional burden of specifying the rewards and state transitions as functions of current resource amounts. Furthermore, in this model, the size of the state space and, consequently, the policy size, explodes exponentially in the number of resources, as compared to the state space where the resource amounts are not folded into the state description.

The FOMDP approach relies on the fact that the agent can observe the exact amounts of all resources at runtime. However, this may not hold, especially in multiagent domains with shared resources, when an agent does not know what other agents have been doing and how much of the shared resources they have been consuming.

3.2 Constrained MDPs

An alternative to the FOMDP approach described above is to treat the problem as a constrained Markov decision process [Altman, 1999], where the resources are not explicitly modeled, but rather are treated as constraints that are imposed on the space of feasible solutions (policies).

A constrained MDP for a resource-limited agent differs from a standard MDP in that the agent, besides getting rewards for executing actions, also incurs resource costs. Consequently, a constrained MDP (CMDP) can be described as a tuple $\langle S, \mathcal{A}, \mathbf{P}, \mathbf{R}, \mathbf{C}, \mathbf{Q} \rangle$, where $\mathbf{C} = [c_{iak}] : S \times \mathcal{A} \rightarrow \mathbb{R}^K$ defines a vector of actions' costs (c_{iak} units of resource $k \in [1, K]$ are used when action a is executed in state i), and $\mathbf{Q} = [q_k] \in \mathbb{R}^K$ is the vector of amounts of available resources (there is q_k units of resource k initially available).

The benefit of the CMDP is that it does not require one to explicitly model how the resources affect the world. Indeed, if all policies satisfy the resource constraints, one does not have to worry about what happens when the resources are overutilized. Consequently, the state space and the resulting policies are exponentially smaller than the ones in the FOMDP model. The standard CMDP formulation constrains the expected usage of all resources to be below a certain limit and can be formulated as the following linear program:

$$\max \sum_i \sum_a x_{ia} r_{ia} \quad \left| \begin{array}{l} \sum_a x_{ja} - \sum_i \sum_a x_{ia} p_{ij}^a = \alpha_j \\ \sum_i \sum_a x_{ia} c_{iak} \leq q_k, \quad x_{ia} \geq 0, \end{array} \right. \quad (3)$$

where the variables x_{ia} have the interpretation of the expected number of times action a is executed in state i .

A weakness of this approach is that it yields policies that can be suboptimal, as compared to the ones constructed by the FOMDP method, because the agent does not base its decision on the current resource amounts, but rather completely ignores that aspect of the state. However, as mentioned in the previous section, in domains where the resources are not observable, or if the policy size is of vital importance (for example if the agent's architecture imposes constraints on policies that it can store), this approach could prove fruitful.

However, the biggest problem with this approach (as pointed out, for example, by Ross and Chen in the telecom-

munication domain [1988]) is that a standard CMDP imposes constraints on the *expected* amounts. Clearly, this method does not work for critical resources, whose overutilization has negative consequences. Indeed, an agent that pilots aircraft would not be satisfied with a policy that *on average* uses an amount of fuel that does not exceed the tank capacity.

3.3 Sample Path MDPs

As just mentioned, Ross and Chen pointed out the weakness of the CMDP approach with constraints on the expected amounts. As a possible solution, Ross and Varadarajan [1989; 1991] propose an approach where constraints are placed on the actual *sample-path* costs. In their work, the space of feasible solutions is constrained to the set of policies whose probability of violating the constraints (overutilizing the resources) in the long run is zero. However, their work concentrates on the average per-step costs and rewards, whereas we are interested in the total amounts, whose distributions are not easily calculable. The approach of Ross and Varadarajan has the same benefits as the standard CMDP method, i.e. no explicit modeling of resources is required, and the state space and policies are small. In addition, unlike the standard CMDP, this method is suitable for problems with critical resources. However, a weakness of this approach is that for some problems it might be too restrictive, in that it allows no possibility of overutilizing the resources. Indeed, policies produced by the sample path method might have significantly lower payoff, as compared to policies that have some probability of resource overutilization. Furthermore, for some domains it might be desirable for the system designer to be able to control the probability of resource overutilization, as a means of balancing optimality and risk.

3.4 MDPs With Constraints on Variance

Another approach to handling deviations from the expectation in Markov models is to impose additional constraints on (or to assign additional cost to) the variance. Sobel [1985] proposed to constrain the expected cost and to maximize the mean-variance ratio of the reward. Huang and Kallenberg [1994] proposed a unified approach to handling variance via an algorithm based on parametric linear programming. These approaches have the same benefits as the standard CMDP and the sample-path methods (compared to the FOMDP formulation) in terms of state space and policy size, as well as the complexity of constructing the initial world model. Additionally, they allow one to somewhat balance payoff and the deviation from the expected. However, these methods do not allow one to make hard guarantees about the probability of overutilizing the resources.

4 Linear Approximation

As hinted at in the previous sections, we would like to be able to constrain the feasible solution space to the set of policies whose probability of overutilizing the resources is below a user-specified threshold ($\mathbf{p}_0 = [p_{0k}]$). In other words, we would like to be able to solve the following math program:

$$\max \sum_i \sum_a x_{ia} r_{ia} \quad \left| \begin{array}{l} \sum_a x_{ja} - \sum_i \sum_a x_{ia} p_{ij}^a = \alpha_j \\ P(c_k > q_k) \leq p_{0k}, \quad x_{ia} \geq 0, \end{array} \right. \quad (4)$$

where c_k is the total amount of resource k that is used by the policy, and q_k is the upper bound on that resource (as before).

The trouble is that the optimization is in the space of x_{ia} , which can be interpreted as the *expected* number of times for executing the actions in the corresponding states. However, it is difficult to express $P(c_k > q_k)$ as a simple function of the optimization variables (x_{ia}), because the latter contain no information about the probability distribution of the random variable of the number of times the action is actually executed in the corresponding state – only the expected values.³ In this section, we present a linear approximation to the above program, based on the Markov inequality:⁴

$$P(c_k \geq q_k) \leq E[c_k]/q_k \quad (5)$$

Using this inequality and the fact that the expected resource usage can be expressed as $E[c_k] = \sum_i \sum_a x_{ia} c_{iak}$, the optimization problem can be formulated as a linear program:

$$\max \sum_i \sum_a x_{ia} r_{ia} \quad \left| \begin{array}{l} \sum_a x_{ja} - \sum_i \sum_a x_{ia} p_{ij}^a = \alpha_j \\ \frac{1}{q_k} \sum_i \sum_a x_{ia} c_{iak} \leq p_0 \\ x_{ia} \geq 0 \end{array} \right. \quad (6)$$

A potential weakness of this approach is that the Markov inequality gives a very rough upper bound, and the policies that correspond to solutions to this LP can be too conservative, in that their real probability of overutilizing the resources can be significantly lower than p_0 . However, if making hard guarantees about the probability of overutilizing the resources is of vital importance, this method might prove valuable, as it yields policies that, in general, would have higher payoff than the ones obtained by the sample-path method (as the latter is often too restrictive). On the other hand, unlike the standard CMDPs that impose constraints on the expected resource usage, and the MDPs that constrain the variance, this method allows one to *explicitly* bound the allowable probability of resource overutilization, and to make precise guarantees about the behavior of the system in that respect.

It is also worth noting that, unlike the sample-path method or the methods that constrain on variance, this method relies on solving a standard linear program, whereas the former require solving quadratic or parametric linear programs. Therefore, the above formulation appears to be a reasonable approximation, because it should be no harder to solve than the standard CMDP (see section 5 for experimental results), while providing a means of balancing solution quality with the precisely quantifiable risk of resource overutilization.

5 Evaluation

To verify our hypotheses about the properties of the approximation described in the previous section, we have performed a set of numerical experiments that compare its behavior to a standard CMDP with constraints on the expected resource amounts (section 3), and to an unconstrained MDP.

³Generally speaking, the total cost is a sum of a random number of differently-distributed random variables, and calculating its probability distribution is a nontrivial task.

⁴This inequality only holds for nonnegative random variables. However, for a transient system, we can make the assumption that all costs are nonnegative, without any loss of generality.

To reduce the bias that might arise from using a small number of hand-crafted test cases, we have instead used a large number of randomly-generated constrained MDPs. All of the generated problems shared some common properties, among which the most interesting ones are the following (the values for our main experiments are given in parentheses):

N_s, N_a, N_r The total number of states, actions, and resources, respectively. (20, 20, 2)

$M_R = \max(R)$ Maximum reward. Rewards are assigned from a uniform distribution on $[0, M_R]$. (10)

$M_C = \max(C)$ Maximum action cost. Resource costs are assigned to state-action pairs from $[0, M_C]$ based on a distribution of R and $\rho(C, R)$ (described below). (10)

$\rho = \rho(C, R)$ Correlation between rewards and resource costs; better actions are typically more costly. ([0.8, 1])

$O_k = [\min Q_k, \max Q_k]$ Upper bounds on resource amounts are assigned according to a uniform distribution from this range. ([200, 300])

γ Dissipation of probability - the probability that the agent exits the system at each time step. ([0.95, 0.99])

The last parameter was used to ensure a transient chain. Instead of providing a small number of sink states, we have chosen to use a uniform dissipation of probability, in order to avoid additional randomization in our experiments, as the latter choice provides a more stable domain.

Our main concern was the behavior of our approximation, as a function of the probability threshold p_0 . Therefore, we have run a number of experiments for various values of $p_0 \in [0, 1]$. To be more precise, we have gradually increased p_0 from 0 to 1 in increments of 0.05, and for each value, generated 50 random models and solved them using the three methods: 1) an unconstrained MDP without any regard for resource limitations, 2) a standard CMDP with constraints on the expected usage of resources, and 3) our CMDP with constraints on the probability of overutilizing the resources (eq. 6). We then evaluated (using a Monte-Carlo simulation) each of the three solutions (policies) in terms of expected reward and probability of overutilizing the resources.

Figure 1 shows a plot of the actual probability of overutilizing the resources for the policies obtained via the three methods as a function of the probability threshold p_0 . The data points are averaged over the runs for a particular value of p_0 . The curve that corresponds to the method that bounds the overutilization probability also shows the standard deviation for the runs. The other data have very similar variance, so we will use the plots of means (averaged over the runs for a given p_0) for our analyses.

Obviously, p_0 has no effect on the unconstrained and the standard CMDP (which maintain more or less a constant probability of overutilization), but it does affect to a large extent the solutions to the problem with constraints on overutilization probability. One can see that the overutilization probability for the solutions produced by our approach is always below p_0 (as it should be). Also, it is worth noting that when p_0 approaches 1, our approximation yields the same results as the other methods, which is good, since setting $p_0 = 1$ should not constrain the space of feasible solutions.

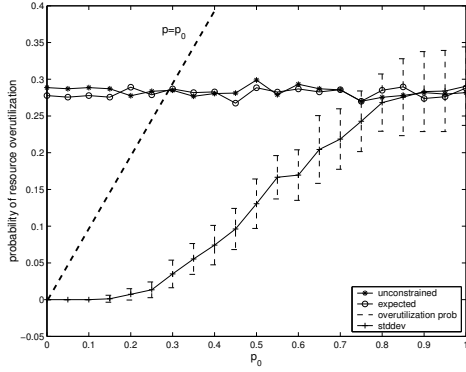


Figure 1: Probability of resource overutilization.

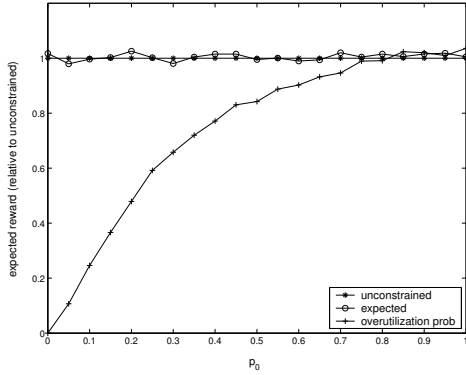


Figure 2: Average rewards for solutions to the three problems.

The rewards obtained by these policies are shown in Figure 2. These *actual* rewards do not necessarily equal the *expected* rewards (which are used during the optimization process). This is because only the runs that did not overutilize the resources were included in the average, and policies were not penalized for violating the resource constraints. This also explains why the total rewards received by solutions to the standard CMDPs were sometimes greater than the ones obtained by solutions to the unconstrained problems.

However, realistically, an agent always incurs a penalty for overutilizing a critical resource, where the penalty amount is based on the consequences of overutilization. For example, if the agent is flying a plane, and the resource in question is fuel, the consequences of trying to use too much of that resource are catastrophic, so the penalty is very high. If we take this into account by assigning a fixed penalty to policies that overutilize the resources, we can update the graph in Figure 2 to get a more realistic picture. Figure 3 shows the average rewards for solutions obtained via the three methods, recalculated to reflect the penalties: new rewards are $R' = (1 - p_{ou})R + p_{ou}W$, where p_{ou} is the overutilization probability, R is the average reward for successful runs of the given policy (as in Figure 2), and W is the penalty for overutilization. We see that, if we take the penalty into account, there is an interval of p_0 where the conservative policy produced by our linear approximation outperforms the other policies.

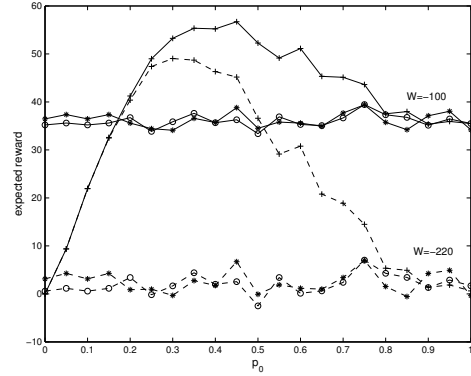


Figure 3: Average rewards with penalties for overutilization.

Moreover, for large penalties ($W = -220$), the conservative policy outperforms the others for all values of p_0 . Note that here, the policy is re-evaluated *post-factum*. Section 6 briefly discusses an approach that explicitly models the penalties in the optimization program.

As we mentioned in section 4, the linear approximation should be no harder to solve than the standard constrained MDP, because both are formulated as linear programs with the same number of constraints. To experimentally verify this, we have timed the runs of all our experiments. Figure 4 contains a plot of the times that it took to solve the problems in all our experiments. One can see that the running times for all three methods are not appreciably different.⁵ In particular, the average ratio of the running time for the standard CMDP to the running time of the unconstrained method is 1.25; the ratio of the running time of our linear approximation with constraints on overutilization probability to the running time of the unconstrained method is 1.06. The slight downward curvature of the plot of the running time of our approximation method appears to be a consequence of the specific implementation of the linear programming algorithm that we used in our experiments.

6 Discussion and Future Work

Our experiments substantiate the claims that our approximation provides an effective and efficient method that agents can use to formulate policies that not only consider limitations on execution resources, but that also explicitly bound the probability of resource overutilization. Our new approximation achieves the constraints on overutilization, and is essentially no more expensive to use than more standard CMDP and unconstrained MDP methods. Because our method constructs policies that are more careful about avoiding resource overutilization, the rewards associated with its policies when resources are not overutilized tend to be less than the rewards for the other methods' policies when resources are not overutilized. However, as we illustrated in Figure 3, when overutilization incurs penalties our new method can outper-

⁵Here we have to note that we used the linear programming approach to solving the unconstrained problem. A different algorithm such as value or policy iteration might be more efficient.

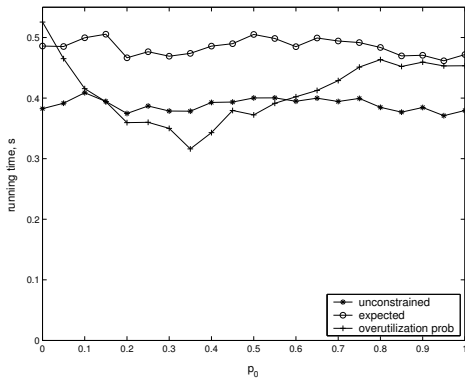


Figure 4: Running time for the three methods

form previous techniques. Thus, our new method is particularly suited to agents engaged in mission-critical domains.

Furthermore, if penalties for resource overutilization are known at design-time and can be expressed in the same units as the rewards, an interesting modification of our LP formulation is to include the penalties in the policy evaluation criterion, as opposed to modeling the constraints on the overutilization probability. This would yield an LP with the following objective function:

$$\max \left(\sum_i \sum_a x_{ia} r_{ia} - \sum_k \frac{W_k}{q_k} \sum_i \sum_a x_{ia} c_{iak} \right), \quad (7)$$

where W_k is the penalty (in units of r_{ia}) incurred for overutilizing resource k . The maximization is subject to just the standard “conservation of probability” constraints as in (eq. 3). A benefit of this formulation is that, for certain initial probability distributions, deterministic policies are optimal. A downside is that the formulation does not allow one to explicitly control the acceptable overutilization probabilities.

As can be seen in Figure 1, our linear approximation is in general overly conservative. For example, given permission to overutilize the resource 20% of the time ($p_0 = 0.2$), the method generates a policy that overutilizes the resource only about 1% of the time. Since rewards and resource usage are typically correlated, we would expect that a policy that undershoots the permitted resource overutilization probability by a lower amount would also yield a higher expected reward. Toward this end, we have formulated a quadratic programming approximation, based on the Chebyshev inequality, which allows us to put a better upper bound on the probability of resource overutilization:

$$\sum_i \sum_a x_{ia} c_{iak} \leq q_k - \Delta_k, \quad \sigma_k^2 / \Delta_k^2 \leq p_{0k}, \quad (8)$$

where $\sigma_k^2 = \text{VAR}[c_k] = E[c_k^2] - E[c_k]^2$ is the variance in the amount of used resource c_k , and Δ_k specify the widths of the allowable regions for the amounts of used resources.

We are also investigating another formulation of the optimization program that should give a more accurate estimate of the resource overutilization probability. The method is based on a polynomial approximation of the pdf of the total resource-usage cost and uses the moments of the cost as

the optimization variables. This approximation and the one based on the Chebyshev inequality are more costly to compute than the linear one. Our current efforts are to encode the approximations and evaluate their strengths and weaknesses.

7 Acknowledgments

This paper is based upon work supported by DARPA/ITO and the Air Force Research Laboratory under contract F30602-00-C-0017 as a subcontractor through Honeywell Laboratories. The authors thank Kang Shin, Haksun Li, and David Musliner for their valuable contributions, as well as one of the anonymous reviewers whose comments inspired (eq. 7).

References

- [Altman, 1999] Eitan Altman. *Constrained Markov Decision Processes*. Chapman and HALL/CRC, 1999.
- [Boutilier et al., 1999] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.
- [Dean et al., 1993] Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Planning with deadlines in stochastic domains. In *Proc. of the Eleventh National Conf. on Artificial Intelligence*, pages 574–579, CA, 1993.
- [Howard, 1960] R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960.
- [Huang and Kallenberg, 1994] Y. Huang and L.C.M. Kallenberg. On finding optimal policies for Markov decision chains. *Math. of Operations Research*, 19:434–448, 1994.
- [Kallenberg, 1983] L.C.M. Kallenberg. *Linear Programming and Finite Markovian Control Problems*. Mathematisch Centrum, Amsterdam, 1983.
- [Musliner et al., 1995] David John Musliner, James Hendler, Ashok K. Agrawala, Edmund H. Durfee, Jay K. Strosnider, and C. J. Paul. The challenges of real-time AI. *IEEE Computer*, 28(1), 1995.
- [Puterman, 1995] M. L. Puterman. *Markov decision processes*. New York, 1995. John Wiley & Sons.
- [Ross and Chen, 1988] K.W. Ross and B. Chen. Optimal scheduling of interactive and non-interactive traffic in telecommunication systems. *IEEE Transactions on Auto Control*, 33:261–267, 1988.
- [Ross and Varadarajan, 1989] K.W. Ross and R. Varadarajan. Markov decision processes with sample path constraints: the communicating case. *OR*, 37:780–790, 1989.
- [Ross and Varadarajan, 1991] K.W. Ross and R. Varadarajan. Multichain Markov decision processes with a sample path constraint: A decomposition approach. *Math. of Operations Research*, 16:195–207, 1991.
- [Russell and Subramanian, 1995] S. J. Russell and D. Subramanian. Provably bounded optimal agents. *JAIR*, 2:575–609, 1995.
- [Sobel, 1985] M.J. Sobel. Maximal mean/standard deviation ratio in undiscounted mdp. *OR Letters*, 4:157–188, 1985.

Belief, Awareness, and Two-Dimensional Logic*

Hu Liu and Shier Ju¹

Institute of Logic and Cognition
Zhongshan University
Guangzhou, China

Abstract

Belief has been formally modelled using doxastic logics in recent decades. The possible worlds model provides an intuitive semantics for these logics. But it also commits us to the problem of logical omniscience. A number of logics have been introduced to circumvent the problem. Of particular interest is the logic of awareness. In this paper we present a new method to put awareness into doxastic logic so as to get a flexible way to model actual belief. The underlying logics are two-dimensional logics. Two two-dimensional doxastic logics are given. In the first logic, a quite limited concept of actual belief is presented. In the second logic, two-dimensional and classical semantics are combined into a hybrid system.

1 Introduction

There has been a long story in philosophy to find a suitable semantics for logics of knowledge and belief since the twentieth century. The subject was picked up by researchers in the area of artificial intelligence, in which human reasoning or resource-bounded agents reasoning is considered.

The standard approach for characterizing knowledge and belief is based on the possible-worlds model. The intuitive idea, which was discovered and labelled by Hintikka [Hintikka, 1962], is that an agent considers a number of situations as possible. Under this interpretation, an agent is said to believe a fact if it is true in all the states that the agent regards as possible. Thus belief is modelled by means of accessibility relations as they are present in possible worlds models. The model is a structure M of the form $(S, \pi, \mathcal{R}_1, \dots, \mathcal{R}_n)$, where S is the set of all worlds, π is the truth assignment for every atom and every world, and $\mathcal{R}_i$, $i = 1, \dots, n$, are the binary relations in S .

On the basis of the possible worlds model a logic of belief can be devised. To this end, introduce modal operators $L_1, \dots, L_n$, L_i to be interpreted as “the agent i believes that”,

and give them formal semantics by a clause: $M, s \models L_i\varphi$ if and only if for each world t such that $\mathcal{R}_i(s, t)$ it holds that $M, t \models \varphi$. To complete the logic, assume that, besides propositional atoms, formulas can also be composed by means of the usual propositional connectives $\sim, \wedge, \vee, \rightarrow$, and $\leftrightarrow$.

In order to be sure that certain properties that intuitively hold of belief are valid in this setting, some constraints should be put on the accessibility relation $\mathcal{R}_i$. The standard way to do that is requiring that $\mathcal{R}_i$ be serial, transitive, and Euclidean. From these constraints, three intuitive properties of belief can be captured: consistency, positive introspection, and negative introspection (they are reflected as the below axioms A5, A3, and A4 respectively).

This notion of belief is completely characterized by the following sound and complete axiom system, traditionally called KD45.

A1: All tautologies of propositional logic

A2: $L_i(\varphi \rightarrow \psi) \rightarrow (L_i\varphi \rightarrow L_i\psi)$

A3: $L_i\varphi \rightarrow L_iL_i\varphi$

A4: $\sim L_i\varphi \rightarrow L_i \sim L_i\varphi$

A5: $\sim L_i(\varphi \wedge \sim \varphi)$

R1: $\varphi, \varphi \rightarrow \psi / \psi$

R2: $\varphi / L_i\varphi$

Modelling knowledge and belief of this kind yields what Hintikka called *logical omniscience*. Logical omniscience presupposes that an agent’s beliefs are closed under logical consequence. Furthermore, valid sentences are always believed. It is clear that rational agents can never be so intelligent that they become omniscient.

To avoid these undesired properties, something non-standard is needed. In the literature there appear quite a number of drastically varying approaches. Of these, of particular interest to us is *awareness logic*.

A number of ways of modelling awareness and actual belief have been suggested in the literature, among them we would like to refer to [Rantala, 1982], [Fagin and Halpern, 1988], [Wansing, 1990] and [Thijssse, 1996].

The main problem of logical omniscience is that it forces an agent to believe too much. Fagin and Halpern suggested a variation of the standard Kripke model [Fagin and Halpern, 1988], named the logic of general awareness (abbreviated

* Authors were supported in part by the logical structure of cognition in open worlds project grant under ministry of education, China grant 2000DXM720.40002.

¹Corresponding author. E-mail address: hssjse@zsu.edu.cn

GAL). The underlying idea is to use awareness as a “sieve” to remove the undesired parts of the logic. Though an agent *implicitly* believes a fact, the agent may not believe it *explicitly* if it is not aware of it. GAL distinguishes between *implicit belief* and *explicit belief* by the following equation:

$$\text{Explicit Belief} = \text{Implicit Belief} + \text{Awareness}$$

The model is endowed with function $\mathcal{A}_i$ that act as a kind of sieve, filtering out explicit beliefs from the bulk of implicit beliefs. The model is a tuple M of the form $(S, \pi, \mathcal{R}_1, \dots, \mathcal{R}_n, \mathcal{A}_1, \dots, \mathcal{A}_n)$, where S , π , and $\mathcal{R}_1, \dots, \mathcal{R}_n$ have their usual meanings. For each possible world s , $\mathcal{A}_i(s)$ is an arbitrary set of formulas of the language, indicating the formulas that the agent i is aware of in s .

The language contains operators $A_1, \dots, A_n$, which reflect awareness, as well as the implicit belief operators $L_1, \dots, L_n$ and explicit belief operators $B_1, \dots, B_n$.

The semantics of the language is as usual, except for adding two clauses for the operators A_i and B_i . Note that the one for A_i is not expressed recursively.

- $M, s \models A_i \varphi$ iff $\varphi \in \mathcal{A}_i(s)$;
- $M, s \models B_i \varphi$ iff $\varphi \in \mathcal{A}_i(s)$ and $M, s \models L_i \varphi$.

The explicit beliefs are just those implicit beliefs that belong to the awareness set. The sentence

$$\mathbf{A6} \quad B_i \varphi \leftrightarrow A_i \varphi \wedge L_i \varphi$$

is valid in the model. Fagin and Halpern showed in [Fagin and Halpern, 1988] that by adding A6 to KD45, the resultant system completely characterize the semantics of GAL.

GAL, despite its simplicity, is a very flexible and powerful tool. Wansing exhibited in [Wansing, 1990] that a slight generalization of GAL (Let $\mathcal{R}_i$ be any binary relation in S) can characterize every modal system that contains classical logic.

GAL has also some disadvantages. Useful information is neglected in this semantics. There are two reasons that commit an agent not to explicitly believe a sentence φ : one is that the agent is not aware of φ ; the other is that the agent does not implicitly believe φ . GAL cannot express these different situations. Even if we know the fact that an agent does not explicitly believe a sentence, we still do not know where it comes from. In our proposal, the difference between them is able to be distinguished.

In GAL, it is possible for a sentence to be true even if the agent is not aware of some parts of the sentence. For example, consider the sentence “if an agent believes that it rains, it will take an umbrella.” (formalized as $B_i \varphi \rightarrow \psi$). The sentence can be true in a world s even if the agent is not aware of anything about “rain” in s . In a different opinion of belief and awareness, it may be argue that the above sentence seems nonsensical to the agent if it does not know what the “rain” is. From that opinion, the sentence can never be simply true. We will introduce a method such that different opinions of belief and awareness can be easily modelled.

It is important to distinguish between objective truth and subjective truth. It is reasonable for the above sentence to be objective true (as a truth of fact), but not subjective true (from the point of view of the agent). We leave the item aside for the moment. We will present two logics below. The first

logic only concern about subjective truth. The notion will be picked up in the second logic.

We use a different way to add awareness sieve to doxastic logics. What we use is two-dimensional logics. In two-dimensional logics, the value of a formula φ ranges over the set $\{(1, 1), (0, 1), (1, 0), (0, 0)\}$. Each member of the set is called a truth-degree. Every truth degree has two dimensions. Intuitively, the first dimension of a truth-degree, called a truth-value, represents the truth value of φ and the second, called an awareness-value, keeps track of the value of the awareness condition (we will clarify the notion below) of φ . Though the logic looks like a type of four-valued logic, it is in fact two-valued. Whichever way it is read, we are still left with two-values, with 1 as *true* and 0 as *false*.

We got the idea of using two-dimensional logics in this way from theoretical linguistics, where two-dimensional logics are used to give the semantic concept of presupposition [Bergmann, 1991]. Usually, the sentence “ a exists” is looked upon as the presupposition of the sentence “ a is P ”. We interpret “an agent is aware of φ ” as “ φ exists in the agent’s memory”. And then we treat the sentence as the presupposition of the sentence “the agent explicitly believe φ ”. Here the presupposition is indeed the existence of a denotation of φ in the agent’s memory. We call it the *awareness condition*. The denotation is subjective. It denotes things in the given memory, unlike in the case of the sentence “ a exists”, whose usual meaning is that a has an objective denotation. Another difference between them is that the subjective denotation is a denotation of a sentence other than a denotation of a term.

Two-dimensional logics allow us to calculate the truth value of a sentence and its awareness condition independently. This gives us flexibility to construct different models of actual belief. Intuitively, the awareness condition of a sentence should affect its truth value. However, it may be no agreement on what the effect should be. By amending its matrices, two-dimensional logics are flexible enough to characterize different concepts of actual belief.

It is easy to see that GAL can be represented by a two-dimensional logic. The only thing we need to do is to make the awareness condition of a sentence does not disturb its truth value.

We would like to emphasize again that what we present in this paper is not only a more rigorous concept of actual belief than GAL’s, but also a method by which different opinions to actual belief can be modelled readily. We will give two doxastic logics use two-dimensional logics in the rest of the paper. The first logic can be viewed as an example of using our method, in which a more limited concept of actual belief is presented. The second logic extend the first logic to a hybrid system.

To be concise, we only consider one agent mode and use operators B, L, A substitute for B_i, L_i, A_i respectively. It is easy to extend the results of the paper to multi-agent mode.

We think many-dimensional logics may play an important role in many topics. The problems in many topics are in the following style: There are several factors that affect the outcome event’s situation. To construct a logical structure of such problem, many-dimensional logics are preferred. Each factor or outcome is put in a different dimension of the truth

degree. In such a many-dimensional semantics, it can be clearly indicated that how the factors affect the outcome. Of course, it is an “out of focus” topic in the present paper.

By the way, we got some ideas of this paper from our work on “Open Worlds”. Interested readers may refer to [Shier and Hu, 2001].

2 Two-Dimensional Awareness Logics

2.1 Semantics

By awareness different people can understand in quite different ways. In this section, we present a logic in which a sentence’s awareness condition is considered as a very rigorous limitation to the sentence: If the awareness condition is not satisfied, then the sentence is nonsensical to the agent and is invariably false. We call the resultant logic *two-dimensional awareness logic*, abbreviated TDAL.

The language of TDAL is the same as GAL’s. A TDAL model is a tuple $M = (\mathcal{S}, \pi, \mathcal{R}, \mathcal{A})$, where,

- $\mathcal{S}$ is a non-empty set of possible worlds;
- π is a truth degree assignment function from the set of atoms to $\{(1, 1), (0, 1)\}$ per possible world. By convention, the parentheses and commas of truth degrees are dropped;
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is a binary relation that is serial, transitive, and Euclidean;
- $\mathcal{A}$ is the awareness function, assigning each possible world a set of formulas.

The matrices are:

A	$\sim A$	$A \vee B$	11 01 10 00
11	01	11	11 11 00 00
01	11	01	11 01 00 00
10	00	10	00 00 00 00
00	00	00	00 00 00 00

$A \wedge B$	11 01 10 00	$A \rightarrow B$	11 01 10 00
11	11 01 00 00	11	11 01 00 00
01	01 01 00 00	01	11 11 00 00
10	00 00 00 00	10	00 00 00 00
00	00 00 00 00	00	00 00 00 00

Notice that $\vee$ and $\wedge$ can be defined from $\sim$ and $\rightarrow$. The latter two connectives are considered as primary.

The truth degree conditions for the modal operators are:

- $\pi(s, A\varphi) = 11$ if $\varphi \in \mathcal{A}(s)$; otherwise, $\pi(s, A\varphi) = 01$.
- $\pi(s, L\varphi) = 11$ if $\pi(t, \varphi) = 11$ or $\pi(t, \varphi) = 10$ for all t such that $(s, t) \in \mathcal{R}$; otherwise, $\pi(s, L\varphi) = 01$.
- $\pi(s, B\varphi) = 11$ if $\pi(s, A\varphi) = 11$ and $\pi(s, L\varphi) = 11$;
 $\pi(s, B\varphi) = 01$ if $\pi(s, A\varphi) = 11$ and $\pi(s, L\varphi) = 01$;
 $\pi(s, B\varphi) = 00$ if $\pi(s, A\varphi) = 01$.

φ is satisfied in s if and only if $\pi(s, \varphi) = 11$, denoted by $M, s \models \varphi$. φ is valid if and only if $M, s \models \varphi$ for every model M and every possible world s , denoted by $\models \varphi$.

The semantics of TDAL needs some explanations. The assignment function π is indeed two-valued. It assigns only

two values, 11 and 01, to each atom. The primary awareness-value is always assigned as 1. Only an occurrence of the B operator can introduce falseness as awareness-value. The reason is this: The awareness condition only makes sense when explicit belief is considered. An atom has no awareness condition at all. It is natural to assign atoms a trivial awareness-value, i.e. an always true value. It is easy to see that this definition is sound: an always true awareness-value does not disturb the calculation of truth value. This also explains the semantic definition of A and L operators, where the awareness-value of each formula of the form $A\varphi$ or $L\varphi$ is always true.

Awareness-value is contagious: the awareness-value of a sentence governed by one of the binary connectives is false if the awareness-value of one of its immediate components is false. If the awareness-value is always true, the truth-value is calculated classically.

Clearly, TDAL is along skeptical lines. It characterizes a cautious agent: the agent deems a sentence to be true only if every propositional part of it has no uncertainty.

Note that the semantics of TDAL can be reduced to a three-valued logic because the truth degree 10 can never result, neither from logical connectives nor from modal operators. 10 can be dropped safely from the matrices. At first sight, the resulting three-valued propositional logic looks the same as weak Kleene logic. If we substitute the truth values t, f and u for the truth degrees 11, 01 and 00 respectively, then we get the matrices of weak Kleene logic. However, the two logics are quite different. In TDAL, the assignment function is indeed two-valued, unlike Kleene’s. 00 can only result from application of the modal operator B . This characteristic makes TDAL very different than weak Kleene logic. For example, there are no propositional tautologies in weak Kleene logics, but in TDAL, some of them are preserved.

2.2 Discussion And Formalization

Definition 2.1 An occurrence of a B operator in a formula φ is bound if that occurrence is in the scope of one of the modal operators; otherwise, such an occurrence of B is free. A formula φ is bound iff each occurrence in it of B is bound, and φ is free iff it is not bound.

Some classical tautologies are not valid in TDAL. A limitation is required in order to make them valid.

Proposition 2.2 The following sentences are valid in TDAL.

(A7) all bound classical tautologies.

Proof. Trivial. $\square$

The next proposition shows that the axioms with respect to the properties of implicit belief are all preserved in TDAL, as they should be.

Proposition 2.3 A2-A5 are all valid in TDAL.

Proof. The proof is as usual. $\square$

Intuitively, the axiom A6, which characterize explicit belief, should not hold in TDAL because we have a different view to awareness condition here. It is just the case. This is easy to see by taking a model that contains a worlds s such that the agent is not aware of φ in s . Instead of A6, we have

the following axioms and rules to characterize properties of explicit belief.

We use φ^* to denote the formula obtained from φ by substituting $A\phi \wedge L\phi$ for all free occurrences of $B\phi$ in φ . Clearly, φ^* is bound.

Proposition 2.4 *The following sentences and rules are all valid in TDAL:*

- (R3) $A\phi_1, \dots, A\phi_n / \varphi \leftrightarrow \varphi^*$, where for all free occurrences of $B\phi$ in φ , $A\phi \in \{A\phi_1, \dots, A\phi_n\}$.
- (R4) $\varphi / A\phi$, where $B\phi$ occurs free in φ .
- (A8) $(LA\phi_1 \wedge \dots \wedge LA\phi_n) \rightarrow L(\varphi \leftrightarrow \varphi^*)$, where for all free occurrences of $B\phi$ in φ , $A\phi \in \{A\phi_1, \dots, A\phi_n\}$.
- (A9) $L\varphi \rightarrow LA\phi$, where $B\phi$ occurs free in φ .

Proof. The proof is as usual. $\square$

These sentences are quite intuitive to the semantics of TDAL. The system corresponding to the semantics of TDAL consists of axioms A2-A5, A7-A9, and rules R1-R4. Call it the system T.

These sentences and rules are in quite generalized forms. To put a clearer explanation of TDAL, we consider those that are close to A6. As we have known, A6 is not valid in TDAL. However, A6 will be satisfied if a condition is satisfied. It is easy to verify that the rule $\frac{A\varphi}{A\varphi \wedge L\varphi \leftrightarrow B\varphi}$ is valid in TDAL, as well as the sentence $LA\varphi \rightarrow L(A\varphi \wedge L\varphi \leftrightarrow B\varphi)$ (Also, they can be derived from system T). We may see the rule as a process: In the beginning, the agent is not aware of a sentence φ so that the substitute instance of A6 of φ is not true; if by some reasons the agent is aware of φ , then it will deem the substitute instance of A6 of φ to be true. Thus, if the agent implicitly believes that it is able to be aware of φ , then it also implicitly believes that the substitute instance of A6 of φ is true. That is just the above sentence's meaning.

Note that the sentence $L(A\varphi \wedge L\varphi \leftrightarrow B\varphi)$ is not valid in TDAL. Thus, in general, A6 is not an implicit belief of the agent. That is to say, the agent may never be aware of some sentences (maybe they are too long and too complicate). So, for $A\varphi \wedge L\varphi \leftrightarrow B\varphi$ to be an implicit belief, there has to put a condition: The agent (implicitly) believes $A\varphi$.

In general, the deduction theorem does not hold. A limited version of the deduction theorem is presented.

Theorem 2.5 *If $\Gamma \vdash_T \varphi$, then $\vdash_T (L\psi_1 \wedge \dots \wedge L\psi_n) \rightarrow L\varphi$, where $\psi_1, \dots, \psi_n$ are members of Γ that are used in the derivation $\Gamma \vdash_T \varphi$.*

Proof. Consider a derivation $\varphi_1, \dots, \varphi_n$ of φ from Γ . We show by induction on i ($i \leq n$) that $\vdash_T (L\psi_1 \wedge \dots \wedge L\psi_n) \rightarrow L\varphi_i$. The inductive steps for φ_i an axiom, for $\varphi_i \in \Gamma$, and for φ_i a conclusion from R1 are clear.

Suppose $\varphi_i = L\varphi_j$ is obtained from φ_j by R2. By induction hypothesis, $\vdash_T (L\psi_1 \wedge \dots \wedge L\psi_n) \rightarrow L\varphi_j$. By A2, $\vdash_T L\varphi_j \rightarrow LL\varphi_j$. Thus $\vdash_T (L\psi_1 \wedge \dots \wedge L\psi_n) \rightarrow L\varphi_i$.

Suppose that $\varphi_i = \chi \leftrightarrow \chi^*$ is obtained from $A\phi_1, \dots, A\phi_n$ by R3. By induction hypothesis, $\vdash_T (L\psi_1 \wedge \dots \wedge L\psi_n) \rightarrow LA\phi_1, \dots, \vdash_T (L\psi_1 \wedge \dots \wedge L\psi_n) \rightarrow LA\phi_n$. Then by A8, we have $\vdash_T (L\psi_1 \wedge \dots \wedge L\psi_n) \rightarrow L\varphi_i$. The inductive steps for R4 is similar to R3's. $\square$

A set of formulas is *consistent* if one cannot derive from it φ and $\sim \varphi$ for some formula φ . A set of formulas is *maximal consistent* if it is consistent and any strict superset is inconsistent.

Lemma 2.6 (1) *Any consistent set is included in a maximal consistent set.* (2) *If Γ is maximal consistent, then for all formulas φ and ψ the following hold: (a) If $B\phi$ occurs free in φ infers that $A\phi \in \Gamma$, then either $\varphi \in \Gamma$ or $\sim \varphi \in \Gamma$; (b) if $\varphi \in \Gamma$ and $\varphi \rightarrow \psi \in \Gamma$, then $\psi \in \Gamma$; (c) if φ is provable, then $\varphi \in \Gamma$.*

Proof. The proof is left to the reader. $\square$

Theorem 2.7 *The system T is sound and complete with respect to TDAL.*

Proof. The soundness is easy to see. We prove completeness using standard techniques with modifications to fit the context.

It is sufficient to show that every consistent formula is satisfiable. We do so by constructing a canonical Kripke model M , containing a possible world s_Γ for every maximal consistent set Γ , such that $M, s_\Gamma \models \varphi$ iff $\varphi \in \Gamma$. Since every consistent formula is contained in some maximal consistent set, this suffices. Let $M = (\mathcal{S}, \pi, \mathcal{R}, \mathcal{A})$, where

- $\mathcal{S} = \{\Gamma \mid \Gamma \text{ is a maximal consistent set}\}$;
- $\pi(s_\Gamma, p) = 1$ if $p \in \Gamma$; $\pi(s_\Gamma, p) = 0$ if $p \notin \Gamma$;
- $\mathcal{A}(s_\Gamma) = \{\varphi \mid A\varphi \in \Gamma\}$;
- $(s_{\Gamma_1}, s_{\Gamma_2}) \in \mathcal{R}$ iff $\{\varphi \mid L\varphi \in \Gamma_1\} \subseteq \Gamma_2$.

Axioms A3, A4 and A5 guarantee that the binary relation $\mathcal{R}$ as defined is indeed transitive, Euclidean, and serial.

By induction on the structure of φ , we show that $M, s_\Gamma \models \varphi$ iff $\varphi \in \Gamma$. The cases when φ is an atom or $\varphi = A\psi$ is directly proved from the definition of M .

Suppose $\varphi = \sim \psi$. (1) Suppose $\varphi \in \Gamma$. Because the inference rule $\frac{\varphi, \sim \varphi}{\psi}$ can be derived for the system T, $\psi \notin \Gamma$ and so, by inductive hypothesis, $s_\Gamma \not\models \psi$. By R4, if $B\phi$ occurs free in ψ , then $A\phi \in \Gamma$. Then by R3, $s_\Gamma \models \psi \leftrightarrow \psi^*$. We get $s_\Gamma \not\models \psi^*$. Because ψ^* is bound, $s_\Gamma \models \sim \psi^*$. Thus $s_\Gamma \models \sim \psi$. (2) Suppose $s_\Gamma \models \sim \psi$. Then $s_\Gamma \not\models \psi$ and for every free occurrence of $B\phi$ in ψ , $s_\Gamma \models A\phi$ and so, by inductive hypothesis, $\psi \notin \Gamma$. Also we have $A\phi \in \Gamma$. Then by Lemma 2.6, $\sim \psi \in \Gamma$. The inductive step for $\rightarrow$ is similar.

Suppose $\varphi = L\psi$. (1) Suppose $L\psi \in \Gamma$. It is directly obtained from the definition that $s_\Gamma \models L\psi$. (2) Suppose $L\psi \notin \Gamma$. Consider the set $\Gamma' = \{\chi \mid L\chi \in \Gamma\}$. We show that $\Gamma' \vdash_T \psi$ does not hold. Indeed, by assuming otherwise, we would have $\Gamma' \vdash_T \psi$ and so, by theorem 2.5, $\{L\chi_1, \dots, L\chi_n\} \vdash_T L\psi$, where $\chi_1 \dots \chi_n$ are all distinct formulas in Γ' that are used in the derivation $\Gamma' \vdash_T \psi$. So $\Gamma \vdash_T L\psi$, contrary to the consistency of Γ . Thus $\Gamma' \cup \{\sim \psi\}$ is consistent and so, it is contained in some maximal consistent set Γ'' . By the definition of Γ' , we have $s_\Gamma \mathcal{R} s_{\Gamma''}$ and $\psi \notin \Gamma''$. By inductive hypothesis, $s_{\Gamma''} \not\models \psi$ and so, $s_\Gamma \not\models L\psi$.

Suppose $\varphi = B\psi$. Note that the two rules, $\frac{A\varphi, L\varphi}{B\varphi}$ and $\frac{B\varphi}{A\varphi \wedge L\varphi}$ can be derived from R3 and R4. Using the two rules, a process similar to the inductive step for L can demonstrate that $B\psi \in \Gamma$ iff $s_\Gamma \models B\psi$. $\square$

Proof. Since the sub-model of M obtained by restricting it to S_T is just a canonical model of the system T , the proof of (i) has already been given in Theorem 2.7.

The proof of (ii) is by induction on the structure of φ . The basic steps, φ is an atom and $\varphi = A\psi$, are direct. Because H contains all classical tautologies, the induction step for $\sim$ and $\rightarrow$ is carried out by the standard techniques of propositional reasoning.

Suppose $\varphi = L\psi$. (1) Suppose $L\psi \in \Sigma$. It follows from the definition that for all s_Γ such that $s_\Sigma \mathcal{R} s_\Gamma$, $\psi \in \Gamma$ and so, $M, s_\Gamma \models \psi$ and then, $M, s_\Sigma \Vdash L\psi$. (2) Now focusing on the other direction, suppose $L\psi \notin \Sigma$. Consider the set $\Gamma = \{\chi \mid L\chi \in \Sigma\}$. By proposition 3.2, using steps similar to those in the proof of theorem 2.7, it can be shown that $\Gamma \vdash_T \psi$ does not hold. Thus $\Gamma \cup \{\sim \psi\}$ is T -consistent and so it is contained in some T -maximal consistent set Γ' . By the definition of Γ , we have $\Sigma \mathcal{R} \Gamma'$ and $\psi \notin \Gamma'$. Then $s_{\Gamma'} \not\models \psi$ and so, $s_\Sigma \not\Vdash L\psi$ does not hold. The inductive step for the B operator is similar. $\square$

Notice that the axioms A3-A5 are contained in all worlds of the H -canonical model. So by proposition 3.1(i), $M, s \Vdash \varphi$ for all worlds s in M , where $\varphi \in \{A3, A4, A5\}$. Thus the relation $\mathcal{R}$ defined in the H -canonical model is indeed transitive, Euclidean, and serial.

Theorem 3.4 *System H is sound and complete for the semantics of HTAL.*

Proof. The soundness is easy to check. To show completeness, suppose $\vdash_H \varphi$ does not hold. Then there exists a H -maximal consistent set Σ such that $\varphi \notin \Sigma$. Construct a H -canonical model of Σ with designated world Σ . By lemma 3.3, $M, s_\Sigma \not\Vdash \varphi$ does not hold. $\square$

4 Conclusion

Many doxastic logics have been suggested in the literature, reflecting quite different concepts of actual belief. One may wonder why there are so many proposals. Were all of us to agree on what human belief is, an “unique right” model of belief would be possible. However, it is not the case, as can be seen from philosophical controversies continuing from long ago. Thus all of the logics have their own claim to “correctness”, if they have clear explanations.

The method using two-dimensional logic to model awareness and actual belief has, in our opinion, particular benefits. Processing awareness condition independently gives us more flexibility in controlling the model that is constructed. Different opinions about the relation between awareness and actual belief will be reflected in different models, which can be obtained by amending the underlying two-dimensional logic. Of course, there are other, maybe many, alternatives to the logics presented in this paper. For example, it is possible to give a doxastic logic with the two-dimensional logic presented by Bergmann in [Bergmann, 1991], in which the logic is intended to be along Russellian lines. Examination of this or other forms of doxastic logics and the relations among them would call for further research.

There has begun to be use of awareness logic for net protocol verification [Accorsi, 2001]. We think the method pre-

sented in this paper may be useful in that purpose. This also calls for further research.

5 Acknowledgements

We would like to thank Nuel Belnap for his detailed reading of the earlier version of this paper and numerous pertinent comments and criticisms.

References

- [Accorsi, 2001] R. Accorsi, D. Basin, and L. Vigano. Towards an awareness-based semantics for security protocol analysis. In *Post-CAV Workshop on Logical Aspects of Cryptographic Protocol Verification*, 2001.
- [Bergmann, 1981] Bergmann, M., Only, even, and clefts in two-dimensional logic, in *11th Internat. Symp. Multiple-valued logic*, Oklahoma 1981, Proceedings, IEEE, Long Beach, 117-123, 1981.
- [Bergmann, 1991] M. Bergmann. Presupposition and two-dimensional logic. In *Meaning and Truth: The Essential Readings in Modern Semantics*, eds. Jay L. Garfield and K. Murray, New York: Paragon House, 189-211, 1991.
- [Fagin and Halpern, 1988] R. Fagin and J. Y. Halpern. Belief, awareness and limited reasoning. *Artificial Intelligence*, 34, 39-76, 1988.
- [Fagin and Halpern, 1995] R. Fagin, J. Y. Halpern, Y. Moses and M. Y. Vardi. Reasoning about Knowledge. MIT press, 1995.
- [Gottward, 2001] S. Gottwald. A treatise on Many-valued Logic. research studies press, 2001.
- [Hintikka, 1962] J. Hintikka. Knowledge and Belief. Cornell University Press, 1962.
- [Konolige, 1986] K. Konolige. What awareness isn't: a sentential view of implicit and explicit belief. In *Proceedings of the First Conference on Theoretical Aspects of Reasoning about knowledge*, 241-250, 1986.
- [Levesque, 1984] H. Levesque. Logic of implicit and explicit belief,. In *Proceeding of the Fourth National Conference on Artificial Intelligence(AAI84)*. Menlo Park, CA, 198-202, 1984.
- [Rantala, 1982] V. Rantala. Quantified modal logic: non-normal worlds and propositional attitudes. *Studia Logica*, 41, 47-67, 1982.
- [Shier and Hu, 2001] J. Shier, L. Hu. The logical structure of open worlds. . Proceedings of 863 plan on intelligent computer. Tsinghua University Press, 2001.
- [Thijsse, 1992] E. Thijsse. Partial Logic and Knowledge Representation. Eburon Publisher, 1992.
- [Thijsse, 1996] E. Thijsse. Combining partial and classic semantics: a hybrid approach to belief and awareness. In *Partiality, Modality, and Nonmonotonicity*, Doher(eds.), 223-249, 1996.
- [Wansing, 1990] H. Wansing. A general possible worlds framework for reasoning about knowledge and belief. *Studia Logica*, 49, 523-539, 1990.