Abstraction and Approximate Decision Theoretic Planning*

Richard Dearden and Craig Boutilier†
Department of Computer Science
University of British Columbia
Vancouver, British Columbia
CANADA, V6T 1Z4
email: dearden,cebly@cs.ubc.ca

Abstract

Markov decision processes (MDPs) have recently been proposed as useful conceptual models for understanding decision-theoretic planning. However, the utility of the associated computational methods remains open to question: most algorithms for computing optimal policies require explicit enumeration of the state space of the planning problem. We propose an abstraction technique for MDPs that allows approximately optimal solutions to be computed quickly. Abstractions are generated automatically, using an intensional representation of the planning problem (probabilistic STRIPS rules) to determine the most relevant problem features and optimally solving a reduced problem based on these relevant features. The key features of our method are: abstractions can be generated quickly; the abstract solution can be applied directly to the original problem; and the loss of optimality can be bounded. We also describe methods by which the abstract solution can be viewed as a set of default reactions that can be improved incrementally, and used as a heuristic for search-based planning or other MDP methods. Finally, we discuss certain difficulties that point toward other forms of aggregation for MDPs.

Keywords: Planning, decision theory, abstraction, approximation, search, heuristics, execution, Markov decision processes

---

*Some parts of this report appeared in preliminary form in "Using Abstractions for Decision-Theoretic Planning with Time Constraints," *Proc. of Twelfth National Conf. on Artificial Intelligence (AAAI-94)*, Seattle, pp.1016–1022 (1994); and in "Integrating Planning and Execution in Stochastic Domains," *Proc. of Tenth Conf. on Uncertainty in Artificial Intelligence (UAI-94)*, Seattle, pp.162–169 (1994).

†Corresponding author: Craig Boutilier

1 Introduction

The classical planning problem is that of producing a sequence of actions that guarantees the achievement of certain goal conditions when applied to a specified starting state. The unrealistic assumptions embodied in much classical planning research, such as complete knowledge of the initial state and completely predictable action effects, have been challenged in, for instance, work on conditional planning [50, 43] and probabilistic planning [32]. The problem of *decision-theoretic planning* (DTP) involves the design of plans or policies in situations where the initial conditions and the effects of actions are not known with certainty, and in which multiple, potentially conflicting objectives must be traded against one another to determine an optimal course of action. For this reason, one can view a DTP problem as a problem of optimal stochastic control. Recently, *Markov decision processes* (MDPs) [26, 54, 45] have been proposed as a semantic and computational framework in which to formulate DTP problems [15, 2, 56, 10, 8, 13, 11]. This model allows the formulation of actions with stochastic effects and the specification of states or objectives of differing value. It can also be applied to settings without obvious termination conditions, such as on-going processes [12], which cannot easily be dealt with by current goal-based planning algorithms.

While MDPs provide firm semantic foundations for much of DTP, the question of their computational utility for AI remains. Many robust methods for optimal policy construction have been developed in the operations research (OR) community, but most of these methods require explicit enumeration of the underlying state space of the planning problem, which grows exponentially with the number of variables relevant to the problem at hand. This severely affects the computational performance of these methods, the storage required to represent the problem, and (potentially) the amount of effort required by the user to specify the problem. Much emphasis in DTP research has been placed on the issue of speeding up computation by means of *approximation*. One class of methods involves restricting search or dynamic programming to local regions or *envelopes* of the state space [15, 2, 56]. This approach reduces the state space to locally accessible regions and allows OR methods to be used on reduced problems. While optimality is sacrificed, judicious choice of relevant states can lead to good approximations.

In this paper, we explore a different way of coping with the computational difficulties involved in optimal policy generation for large state spaces. First, we present a particular structured representation of MDPs using a variant of the probabilistic STRIPS operators used in [32] to describe actions and rewards. This representation is a syntactic variant of certain types of "two-stage" Bayesian networks or influence diagrams [16, 10, 8]. This in itself allows large problems to be specified and represented in a concise and natural fashion.

The key aim of this paper is the exploitation of structured representations to quickly identify appropriate dimensions for *abstraction*. We generate an *abstract* state space in which (concrete) states are clustered together, and construct an *abstract MDP*. This abstract MDP has a state space (potentially exponentially) smaller than that of the original MDP, and can be solved much more quickly. Crucial features of the aggregation process are:

(a) the construction of the abstract MDP is reasonably fast (i.e., the time required does not grow with the state space); and

(b) the abstract process is constructed in such a way that standard solution techniques may be used in this reduced space to produce an *abstract policy*.

Our approach has several advantages over the envelope method. Foremost among these is the fact that no states are ignored in abstract policy generation – each state may have some influence on the constructed policy by membership in an abstract state.[1] This allows us to prove bounds on the value of abstract policies (with respect to an optimal policy). Furthermore, finer-grained abstractions are guaranteed to increase the value of policies. Finally, abstractions can be generated quickly. These factors allow abstract policies of varying degrees of accuracy to be constructed in an anytime fashion (in particular, in the style of *contract anytime* algorithms [47, 48]).

While the abstraction method of approximating optimal policies is orthogonal to the envelope approach, the model we propose actually illustrates that the two approaches complement one another quite nicely. Off-line computation of the optimal abstract policy provides one with a set of appropriate actions, though perhaps not optimal in the concrete space. In addition, it produces an abstract *value function* that characterizes the estimated *long-term* value of every (abstract) state. However, even with a good policy in hand, an agent may find itself in a situation where computation time is available to improve its action choice. To integrate abstraction into a more online model of planning, one can treat an abstract policy as a set of *default reactions* to be executed by an agent when an action must be performed. However, local search through the concrete state space (i.e., the construction of a decision tree) can be used to refine these reactions when additional computation time is available. Such a model is reminiscent of reaction-first search [18] or real-time dynamic programming [2]. A crucial difference is the existence of an abstract value function to guide this search. This gives us two ways to view abstraction: the abstraction process is used to generate default reactions and heuristics (or static-evaluation functions) to guide and prune an online search for good actions; or the abstraction process provides a fast means for building approximately optimal plans in an off-line planning system.

[1] The envelope method of [15] in fact uses a heuristic function to estimate the value of falling out of the current envelope; but it is not clear how to construct such functions for a desired level of accuracy.

The main aims of this paper are: to present a compact and natural representation of MDPs; to describe how such a representation can be used to construct abstract MDPs that can be solved quickly to produce approximately optimal policies, and to show how such abstract solutions can be used in online planning algorithms. In Section 2, we briefly describe the aims of decision-theoretic planning and the suitability of Markov decision processes as a foundational model for DTP. We describe MDPs and various methods for constructing optimal policies, such as value iteration and policy iteration, based on the dynamic programming principle [3]. We discuss compact representations for MDPs; in particular, we adopt a variant of the probabilistic STRIPS operators used in [32] that captures independence of action effects in a manner similar to Bayes nets [41]. Such a representation allows MDPs to be specified concisely and naturally by exploiting structural regularities in the domain and in the effects of actions.

In Section 3, we begin to address the computational difficulties associated with solving MDPs optimally. Solution methods such as policy iteration tend to converge well in practice, but each iteration requires computation that is polynomial in the size of the state space. Since the state space itself grows exponentially with the number of variables present in the problem description, such methods are only feasible for reasonably small problems. We present an approximation method for MDPs based on the construction and solution of a smaller *abstract MDP*. Our method for generating abstract MDPs is based on Knoblock's [29] abstraction generation technique for classical planning: certain literals are deleted from the problem description. These literals are, roughly speaking, those whose impact on the value of a state or policy is "negligible." However, there are some critical differences in our model. First, care must be taken to ensure that the reduced problem is indeed an MDP; this guarantees that existing solution methods can be used. Second, the solution to the abstract problem can be used directly in our model. In contrast to classical abstraction, where abstract solutions can only be used to guide the search for solutions at a concrete level, in our model the abstract solution is executable. Of course, the solution may not be optimal. We therefore require that abstract MDPs be generated in such a way that the *error*, or divergence from optimality, can be bounded and that different abstractions can be quickly compared for value before they are solved. The key contribution of Section 3 is an algorithm to generate abstract MDPs and the derivation of an easily computed upper bound on the error of the abstract solution. We also discuss the generation of appropriate abstractions in terms of value of information, and show that the bounds on less abstract policies are always closer to optimal than those of more abstract policies. As such, the chosen degree of abstraction provides a parameter that can be set in a contract-anytime fashion.

In Section 4, we describe how abstract MDPs and their solutions can be exploited and improved in the planning process, in both off-line and online models of plan construction. We first describe *levels* of abstraction, or abstraction hierarchies [29]. In classical abstraction, hierarchies are generally constructed so that various *refinement properties* are satisfied; that is, the solution at a given abstraction level can be refined (without changing any of its components) to provide a solution at a less abstract level. Such properties are generally impossible to ensure in our model, since the aim is the production of optimal, useful solutions at each level. However, we demonstrate empirically that abstract solutions can sometimes be used to speed up the computation for their less abstract counterparts by "seeding" the policy iteration algorithm with abstract initial policies.

We then consider refinement of abstract solutions in an online model by using search to improve the choice of action for the *actual state* in which an agent finds itself. In contrast with methods such as policy and value iteration, which consider the appropriate action choice and value of *all states*, local search can be used to focus computational effort on only those states that directly impact the value of the current state.[2] This view is reminiscent of the envelope method of Dean *et al.* [15], but is most closely related to the real-time dynamic programming model of Barto, Bradtke and Singh [2]. There are two key advantages to integrating our abstraction method with this model: first, if action is required at any time, an abstract policy generated off-line provides useful default reactions (whose value is roughly known); second, the abstract value function generated in the solution of the abstract policy can be used to guide the local search process.

In Section 5, we consider a generalization of our abstraction model. While the original algorithm for generating abstractions will not delete any literal that can influence the probability of another literal that is deemed relevant, our method of *inexact abstraction* ignores effects with *little* (in contrast to *no*) influence in the abstract problem (e.g., effects with small probability of occurrence). We provide an algorithm and error bounds for this approach, and point out the difficulties in constructing inexact abstractions with predictable, tight bounds.

Finally, in Section 6, we conclude with a discussion of open problems and directions for future research, and describe how some existing work might be integrated with our model of abstraction. In particular, we describe other forms of state aggregation that can be used to solve MDPs more quickly. We view the work presented here as a starting point for the use of AI-style, intensional representations of DTP problems to determine irrelevant details and appropriate aggregations and abstractions that allow the reasonably fast construction of good plans.

[2] Indeed, as we elaborate below, the rollback procedure for a decision tree rooted at a given state can be viewed as a form of value iteration *directed* toward that state.

2 Markov Decision Processes and their Representation

Decision-theoretic planning generalizes classical AI planning to deal with situations of uncertainty and multiple, possibly competing objectives of different utility. The tools of decision theory should be used to determine a plan with maximum expected utility. Because one generally does not know *a priori* which final state the system should end up in (e.g., a high utility state may be achievable with only low probability and thus should be eschewed), classical goal-based techniques such as regression or partial-order planning are of little value.[3] For example, an agent required to deliver two packages before a certain deadline, but unable to do so, must decide which (if either) of the packages to pick up first. Goal-based methods require one of these objectives first be chosen; but the objective to be chosen cannot generally be known until possible plans have been considered.

The articulation of explicit goals is also frequently absent in DTP problems, which often have a process-oriented flavor. For example, our agent above may be acting in a constant loop of anticipating and performing routine tasks and achieving certain requests without consideration of termination conditions [12]. Manufacturing processes are often best viewed this way as well: the aim is not to reach some final state where a certain number of units have been produced, rather one wishes to maximize throughput subject to other considerations of importance (such as safety, labor and maintenance constraints). Each aspect of the manufacturing process is an action (whether joining two parts together, or checking a part for faults); the objectives are to fill specific orders, to operate as inexpensively as possible, to minimize the number of faulty parts produced, etc. Actions are stochastic (e.g., they may introduce faults on occasion) and there is uncertainty in the state of knowledge. A decision-theoretic planner should produce a plan of operation that includes a sequence of manufacturing steps for parts of specific types, as well as certain test and repair actions designed to deal with faulty parts. Note that as a matter of course, an optimal plan may not test for certain faults if their cost and probability of occurrence is sufficiently small. The role of decision theory in such a process is to decide which tests are "worth it"; thus *a priori* goal states such as "Part X should be free of faults with probability 0.995" are useful only in specific structured settings (see, e.g., [32, 17] where this view is pursued).

Features such as these make Markov decision processes an ideal model for modeling DTP problems. MDPs can be viewed as stochastic automata in which actions have uncertain effects, inducing stochastic transitions between states, and the precise state of the system is known only with a certain probability. In addition, the expected value of a certain course of action is a function of the

[3] At least, as currently formulated: there is some possibility that regression methods may prove useful in approximation methods for DTP [8].

transitions it induces, allowing rewards to be associated with different aspects of the problem rather than with all-or-nothing goal propositions. Finally, plans can be optimized over a fixed finite period of time, or over an infinite horizon, suitable for ongoing processes.

We describe MDPs in detail below; however, we do not present them in full generality. Certain simplifying assumptions are made that restrict the class of problems we address. The most restrictive assumption is that of *complete observability*: although actions may have uncertain effects, we assume that once the agent has performed an action, it can observe its actual outcome. In other words, the agent has full access to the state of the system being controlled. Thus, the planning algorithm need not deal with uncertainty in its knowledge of the world. While unrealistic in many domains, *fully observable MDPs* (FOMDPs) capture an interesting and useful class of problems. In addition, the computational methods for optimal policy construction for the fully observable case are much better studied and more powerful than those for *partially observable MDPs* (POMDPs). Indeed, while special purpose code for FOMDPs can often handle systems with hundreds of thousands of states, dealing with twenty-state systems is often problematic for POMDP algorithms [35, 13]. Our initial investigations of representational and abstraction methods for MDPs, described in this paper, are therefore directed toward FOMDPs. However, we fully expect these and related methods will be adaptable to POMDPs (see [11] for investigations of this point).

Primarily for reasons of presentation, we do not consider action costs in our formulation of MDPs. All utilities are associated with states (or propositions). This assumption is not especially restrictive, for our algorithms can be augmented to deal with more general reward specifications. However, explicit consideration of action costs would detract from the main points of this paper. Finally, we note that our examples are primarily goal-based, again for ease of presentation. However, our algorithms can be applied directly to process-oriented problems (see, e.g., [12] for process-oriented problems that extend the types of examples we present here).

2.1 Markov Decision Processes

For our purposes, a Markov decision process can be defined as a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where $\mathcal{S}$ is a finite set of *states* or possible worlds, $\mathcal{A}$ is a finite set of *actions*, T is a *state transition function*, and R is a *reward function*. A state is a description of the system of interest that captures all information about the system relevant to the problem at hand. In typical planning applications, the state is a possible world, or truth assignment to the logical propositions with which the system is described. The agent can control the state of the system to some extent by performing actions $a \in \mathcal{A}$ that cause *state transitions*, movement from the current state to some new state. Actions are stochastic in that the actual transition caused cannot be predicted with certainty. The transition function T describes the effects of each action at each state. $T(s, a)$ is a probability distribution over $\mathcal{S}$: $T(s, a)(t)$ is the probability of ending up in state $t \in \mathcal{S}$ when action a is performed at state s. We will write this quantity as $\Pr(t|a, s)$.[4] We require that $0 \leq \Pr(t|a, s) \leq 1$ for all s, t, and that for all s, $\sum_{t \in \mathcal{S}} \Pr(t|a, s) = 1$. The components $\mathcal{S}$, $\mathcal{A}$ and T determine the dynamics of the system being controlled. We assume that each action can be performed at each state. In general models, each state can have a different *feasible action set*, but this is not crucial here.[5]

The states that the system passes through as actions are performed correspond to the *stages* of the process. The system starts in a state s_0 at stage 0. After m actions are performed, the system is at stage m. Given a fixed "course of action", the state of the system at stage m can be viewed as a random variable S^m. Stages provide a very rough notion of time for MDPs. The system is *Markovian* due to the nature of the transition function; that is,

$$\Pr(S^m | a^{m-1}, S^{m-1}, a^{m-2}, S^{m-2}, \cdots a^0, S^0) = \Pr(S^m | a^{m-1}, S^{m-1})$$

(where a^i corresponds to the action taken at stage i). The fact that the system is fully observable means that the agent knows the true state at each stage m (once that stageis reached), and its decisions can be based on this knowledge.

A *policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$ describes a course of action to be adopted by an agent controlling the system and plays the same role as a plan in classical planning. An agent adopting such a policy performs action $\pi(s)$ whenever it finds itself in state s.[6] In a sense, π is a conditional and *universal plan* [51], specifying an action to perform in every possible circumstance. An agent following policy π can also be thought of as a reactive system. From a given start state s_0, a fixed policy induces a distribution over possible system *trajectories*. An m-stage trajectory is a sequence of states s_0 through s_m corresponding to the states of the system at stages 0 through m.

Given an MDP, an agent ought to adopt a policy that maximizes the expected value of the trajectories it admits. A number of different value measures or *optimality criteria* have been studied in the literature, most based on a bounded, real-valued, history-independent reward function $R : \mathcal{S} \rightarrow \Re$. $R(s)$ is the instantaneous reward an agent receives for entering state s. We take a *Markov decision problem* to be an MDP together with a specific optimality criterion. Optimality criteria vary with the horizon of the process being controlled

[4] This notation is merely suggestive. The term $T(s, a)(t)$ cannot be formally interpreted as a conditional probability.

[5] We could model the applicability conditions for actions using preconditions in a way that fits within our framework below. However, we prefer to think of actions as action attempts, which the agent can execute (possibly without effect or success) at any state. Preconditions may be useful to restrict the planning agent's attention to potentially "useful" actions, and thus can be viewed as a form of heuristic guidance (e.g., don't bother considering attempting to open a locked door). This will not impact what follows.

[6] In fact, such policies are *stationary* (and *Markovian*), the action choice depending only on the state of the system, not on the stage of the process or its history. For the problems we consider, optimal stationary policies always exist.

and the manner in which future reward is valued. For finite-horizon problems, the aim is typically to construct a policy that maximizes the expected total reward gained over some fixed number of stages m. The total reward for a finite trajectory is simply the sum of the rewards $\sum_{i=0}^{m} R(s_i)$. For infinite-horizon problems total expected reward will typically diverge, and other criteria are necessary. One criterion is *average reward* per stage of the process. While well-studied, and perhaps ideally suited for planning problems, such measures are often difficult to compute. In this paper, we focus on *discounted infinite horizon* problems: the *current* value of a reward received n stages in the future is discounted by some factor $\beta^n (0 < \beta < 1)$. This allows simple computations to be used, as discounted total reward will be finite. The infinite-horizon model is important because, even if a planning problem does not proceed for an infinite number of stages, the horizon is usually *indefinite*, and can only be bounded loosely. Furthermore, solving an infinite-horizon problem is typically more computationally tractable than a very long finite-horizon problem. Discounting has certain other attractive features, such as encouraging plans that achieve goals quickly, and can sometimes be justified on economic grounds, or can be justified as modeling expected total reward in a setting where the process has probability $1 - \beta$ of terminating (e.g., the agent breaks down) at each stage. We refer to [45] for further discussion of MDPs and different optimality criteria.

The expected sum of discounted future rewards for a fixed policy π depends on the state in which the process starts and is denoted by the function V_π, where $V_\pi(s)$ is the *expected value* when π is executed beginning in state s. There are several algorithms that can be used to determine V_π (see [5, 45] for details). A straightforward iterative algorithm, called *successive approximation*, proceeds by constructing the sequence of *n-stage-to-go* value functions V_π^n. The quantity $V_\pi^n(s)$ is the expected discounted future reward received when π is executed for n stages starting at state s. We set $V_\pi^0(s) = R(s)$ and inductively compute

$$V_\pi^n(s) = R(s) + \beta \sum_{t \in \mathcal{S}} \Pr(t|\pi(s), s) V_\pi^{n-1}(t) \qquad (1)$$

As $n \to \infty$, $V_\pi^n \to V_\pi$; and the convergence rate and error for a fixed n can be bounded easily [45]. We note that the right-hand side of this equation determines a contraction operator so that: a) the algorithm converges for any starting estimate V_π^0; and b) if we set $V_\pi^0 = V_\pi$, then the computed V_π^n for any n is equal to V_π (i.e., V_π is a fixed-point of this operator). We can also exactly compute the value V_π using the following formula due to Howard [25]:

$$V_\pi(s) = R(s) + \beta \sum_{t \in \mathcal{S}} \Pr(t|\pi(s), s) V_\pi(t) \qquad (2)$$

We can find the value of π for all states by solving this set of linear equations $V_\pi(s)$, $\forall s \in \mathcal{S}$.

A policy π^* is *optimal* if, for all $s \in \mathcal{S}$ and all policies π, we have $V_{\pi^*}(s) \geq V_\pi(s)$. We say the (optimal) value of a state $V^*(s)$ is its value under any optimal policy ($V_{\pi^*}(s)$). We take the problem of decision-theoretic planning to be that of determining an optimal policy (or an approximately optimal or satisficing policy). An incremental approximation method for policy construction known as *value iteration* proceeds much like successive approximation, except that a random value function V^0 is initially chosen and at each stage we choose the action that maximizes the right-hand side of Equation 1:

$$V^n(s) = R(s) + \max_{a \in \mathcal{A}}\{\beta \sum_{t \in \mathcal{S}} \Pr(t|a,s) V^{n-1}(t)\} \tag{3}$$

The sequence of value functions V^n converges to V^*, and for some finite n the actions a that maximize the right-hand side of Equation 3 form an optimal policy. As with successive approximation, V^* is a fixed point of Equation 3 and, if used as an initial value estimate, results in immediate convergence.

Policy iteration is an ingenious algorithm proposed by Howard [25] for optimal policy construction. It proceeds as follows:

1. Let π' be any policy on $\mathcal{S}$
2. While $\pi \neq \pi'$ do
   (a) $\pi = \pi'$
   (b) For all $s \in \mathcal{S}$, calculate $V_\pi(s)$ by solving the set of $|\mathcal{S}|$ linear equations given by Equation 2.
   (c) For all $s \in \mathcal{S}$, if there is some action $a \in \mathcal{A}$ such that
   $$R(s) + \beta \sum_{t \in \mathcal{S}} \Pr(t|a,s) V_\pi(t) > V_\pi(s)$$
   then $\pi'(s) = a$; otherwise $\pi'(s) = \pi(s)$
3. Return π

The algorithm begins with an arbitrary policy and alternates repeatedly (in Step 2) between an evaluation phase (Step b) in which the current policy is evaluated, and an improvement phase (Step c) in which local improvements are made to the policy. This continues until no local policy improvement is possible. The algorithm is guaranteed to converge [25] and in practice tends to do so in relatively few iterations [45]. The evaluation phase requires solving the set of $|\mathcal{S}|$ linear equations. Algorithms for solving linear equations of this kind are typically $O(n^3)$ where n is the number of variables (here $n = |\mathcal{S}|$). The improvement phase uses these values in a local computation to find an action that, if executed once at state s, followed by execution of the current policy π, results in improved value.

The main cost per iteration in the policy iteration is clearly policy evaluation. Puterman and Shin [46] have observed that the exact value of the current policy is typically not needed to check for improvement. Their *modified policy iteration* algorithm is exactly like policy iteration except that the evaluation phase uses some number of successive approximation steps instead of the exact solution method. This algorithm tends to work extremely well in practice and can be tuned so that both policy iteration and value iteration are special cases [46, 45]. We note that all three policy construction methods produce the value function V^* as well as an optimal policy. In addition, the algorithms are incremental in the sense that a sequence of improving (or roughly improving) intermediate policies is produced.

2.2 Compact Representation of MDPs

Most planning problems are described by a set of features or propositions that characterize the domain of interest, and problems typically "grow" by the addition of atomic propositions reflecting relevant features of the domain. We assume that the system to be controlled is described by some logical propositional language $\mathcal{L}$, generated by a set $\mathbf{P}$ of atomic propositions. The state space $\mathcal{S}$ is the set of all valuations over this language containing $|\mathcal{S}| = 2^{|\mathbf{P}|}$ possible states, and grows exponentially with the number of variables. This poses some difficulty for the specification and computational methods for MDPs described above, for the problem formulation requires explicit enumeration of the state space.

Focusing on representation for the moment, we notice that the transition function T requires a set of $|\mathcal{S}| \times |\mathcal{S}|$ matrices, one matrix representing the transition probabilities for each action. For a large planning problem the storage requirements for these action descriptions (as well as the reward function) can be prohibitive. For a problem with ten propositions (roughly 1000 states), a 1,000,000 element matrix may be needed to represent the effects of each action. Even though the probability matrices are typically quite sparse, (and storage methods may exploit this), the specification of a problem in this format is unattractive. In AI planning, actions are rarely described explicitly as state transitions. Natural representations such as STRIPS rules or the situation calculus specify the effects of actions on propositions rather than states. Such representations are extremely compact in normal circumstances because actions exhibit a number of regularities that can be exploited.

To represent stochastic actions compactly, we adopt a probabilistic variant of STRIPS rules very similar to that used in BURIDAN [32]. In the classical STRIPS representation [19], an action is represented using a list of *effects*, or a set of literals that become true when the action is executed. When an action is executed at a state, the effect is "applied" to the current state to determine the new state that results. More precisely, let E be the effect (a consistent set of literals) associated with action a, and let s be a state (represented as the set of literals true in that state). The state that results when a is executed at s

(denoted $a(s)$) is simply the result of applying the effect to s:

$$E(s) = (s \setminus \{p : \neg p \in E\}) \cup E$$

Note that any literal unmentioned in the effect persists in truth value and that a single effect changes many states in similar ways. Thus large classes of state transitions can be represented using a single effect. As an example, consider the action a with effect $\{P, \neg Q\}$. When applied to state $s = \{\neg P, \neg Q, R, S\}$, the resulting state is $a(s) = \{P, \neg Q, R, S\}$.

Pednault [42] generalizes these descriptions somewhat by allowing actions to have *conditional effects*, or context-dependent effects that vary with the initial state. Following [32] we assume that the conditions under which an action can have different effects are described by a finite set of *discriminants* $D = \{d^1, \cdots d^n\}$. This is a set of mutually exclusive and exhaustive logical formulae that partitions the state space. We typically assume each d^i to be a conjunction of literals, and often treat d^i as the set of literals occurring in the conjunction. We denote by $atoms(d^i)$ the set of atoms occurring in d^i (when viewed as a set). A conditional action description associates an effect E^i with each discriminant d^i. The state $a(s)$ that results from a conditional action a is given by $E^i(s)$, where d^i is the (unique) action discriminant such that $s \models d^i$. For example, suppose action a is described using two discriminants, $d^1 = \{R\}$ and $d^2 = \{\neg R\}$, with associated effects $E^1 = \{P, \neg Q\}$ and $E^2 = \{P, Q\}$. When a is applied to state $s = \{\neg P, \neg Q, R, S\}$, the resulting state is $a(s) = \{P, \neg Q, R, S\}$ (as above) since $s \models d^1$. But when applied to $t = \{\neg P, \neg Q, \neg R, S\}$, the result is $a(t) = \{P, Q, \neg R, S\}$.

To these conditional effects, we add nondeterminism by supposing that under each condition a number of possible effects might occur with a specified probability, following the BURIDAN representation. That is, with each d^i we associate a *stochastic effects list* of the form $\langle E^i_1, p^i_1; \ldots E^i_n, p^i_n \rangle$, where each E^i_j is an effect and each p^i_j is the probability that effect will occur; we require only that $\sum_{j=1}^{n} p^i_j = 1$. An action now induces a probability distribution over possible resulting states. The semantics of an action of this type is as follows:

$$\Pr(t|a, s) = \sum_j \{p^i_j : E^i_j(s) = t\}$$

where $s \models d^i$. It should be clear that this determines a well-defined stochastic transition function for each action, and that any transition function can be so represented (though perhaps not compactly).

To illustrate this representation, as well as our algorithms below, consider the following simple planning problem. We have a robot whose main objective is to deliver coffee to a user. It can move between the user's office and a coffee shop across the street, buy coffee at the coffee shop, and deliver coffee to the user in the office. If it is raining outside the robot gets wet if it moves between the two locations, unless it has an umbrella (which it can obtain in the user's

Action	*Discriminant*	*Effect*	*Prob.*
Move	*Office*	$\neg$ *Office*	0.9
		$\emptyset$	0.1
	$\neg$ *Office*	*Office*	0.9
		$\emptyset$	0.1
Move	*Rain*, $\neg$ *Umb*	*Wet*	0.9
		$\emptyset$	0.1
	$\neg$*Rain* $\vee$ *Umb*	$\emptyset$	1.0
BuyC	$\neg$ *Office*	*HRC*	0.8
		$\emptyset$	0.2
	Office	$\emptyset$	1.0
GetU	*Office*	*Umb*	0.9
		$\emptyset$	0.1
	$\neg$ *Office*	$\emptyset$	1.0
DelC	*Office*, *HRC*	*HUC*, $\neg$*HRC*	0.8
		$\neg$*HRC*	0.1
		$\emptyset$	0.1
	$\neg$ *Office*, *HRC*	$\neg$*HRC*	0.8
		$\emptyset$	0.2
	$\neg$*HRC*	$\emptyset$	1.0

Figure 1: Stochastic STRIPS-style action representation

office). The robot is penalized for getting wet, but it is penalized more if the user does not have coffee. The COFFEE domain is characterized by six propositions: *Office* (the robot is in the office, otherwise at the coffee shop); *HRC* (the robot has coffee); *HUC* (the user has coffee); *Rain* (it is raining); *Umb* (the robot has the umbrella); and *Wet* (the robot is wet). The robot has four actions at its disposal, all of which may fail: *Move* (to the opposite location); *BuyC* (buy coffee if it is in the coffee shop); *DelC* (deliver coffee in its possession to the user in the office); *GetU* (get the umbrella if it is in the office). The effects of these actions and their probabilities are listed in Figure 1. Worth noting is that the *DelC* action can fail in two different ways: ten per cent of the time the user simply fails to get the coffee and the robot retains possession (simple failure), and ten per cent of the time the the robot loses the coffee (coffee spill).

We extend the BURIDAN representation by adding *action aspects*. These are intended to represent the fact that some effects of an action only depend on certain features distinguished by the discriminant set. For example, in Figure 1, the *Move* action has two aspects. The first represents the fact that when the agent performs a *Move*, the resulting location depends on the agent's current location only. It is independent of the values of *Rain* and *Umb*. The second aspect deals with whether the agent becomes wet or not. Since this is independent of where the agent is, the discriminant only contains *Rain* and *Umb*.

Actions with multiple action aspects can be translated into actions with a single aspect by forming the "cross-product" of their effects. Figure 2 shows

Action	*Discriminant*	*Effect*	*Prob.*
Move	*Office*, *Rain*, $\neg$ *Umb*	$\neg$ *Office*, *Wet*	0.81
		Wet	0.09
		$\neg$ *Office*	0.09
		$\emptyset$	0.01
	Office, $\neg$ *Rain* $\vee$ *Umb*	$\neg$ *Office*	0.9
		$\emptyset$	0.1
	$\neg$ *Office*, *Rain*, $\neg$ *Umb*	*Office*, *Wet*	0.81
		Wet	0.09
		Office	0.09
		$\emptyset$	0.01
	$\neg$ *Office*, $\neg$ *Rain* $\vee$ *Umb*	*Office*	0.9
		$\emptyset$	0.1

Figure 2: Expansion of action aspects

the translated form of the *Move* action for this example. More precisely, an action can be specified using different aspects, each of which has the form of an action as described above (i.e., each aspect has its own discriminant set). The actual effect of an action at a state is determined by applying the effects list of the relevant discriminant for *each* aspect of that action. Let w be some state to which we apply an action with k aspects. Since each aspect has a proper discriminant set associated with it, w satisfies exactly one discriminant for each aspect. Assume the discriminants for the jth aspect are $d_j^1, \cdots, d_j^{c_j}$ and that each d_j^i has an associated effects list $\langle E_j^{i,1}, p_j^{i,1}; \ldots E_j^{i,n_j}, p_j^{i,n_j} \rangle$. An effect from each applicable list will occur with the specified probability, these probabilities being independent. Intuitively, action aspects capture the kind of independence assumptions one might find in a Bayesian network or influence diagram (as we show below). Thus, the net effect of an action A at w is the union of these effects (sets of literals), one chosen from each aspect. The probability of this combined effect is determined by multiplying these probabilities. Thus, we have

$$Pr(v|A, w) = \sum \{ p_1^{i_1,j_1} \cdot p_2^{i_2,j_2} \cdots p_k^{i_k,j_k} : E(w) = v \}$$

where E is an effect such that

$$E = E_1^{i_1,j_1} \cup E_2^{i_2,j_2} \cup \cdots \cup E_k^{i_k,j_k}$$

To ensure that actions are well-formed we impose the following consistency condition: if d_l^i and d_k^j are mutually consistent discriminants taken from distinct aspects i and j of a given action, then their effects lists must contain no atoms in common (thus, the union above is consistent).

Compact representation of the reward function can use the same techniques used for action representation. We assume a set of mutually exclusive and exhaustive reward discriminants d^i to each of which is assigned an immediate

Discriminant	*Value*	*Discriminant*	*Value*
$HUC, \neg Wet$	1.0	$\neg HUC, \neg Wet$	0.2
HUC, Wet	0.8	$\neg HUC, Wet$	0.0

Figure 3: STRIPS-style reward function representation

reward r^i. As usual, $R(s) = r^i$ for any $s \models d^i$. Such a representation is completely general. Figure 3 describes the reward function for our example: the robot is given a reward of 0.8 for ensuring the user has coffee, and a reward of 0.2 for staying dry. An alternative representation, which we do not pursue but which could exploited by our algorithms below, is the association of independent, additive rewards with a number of propositions in the manner of multi-attribute utility theory [28], and to sum the individual rewards of each proposition satisfied by s to determine $R(s)$. This would provide a very direct encoding of the reward function we adopt in this example.

A related action representation uses "two-stage" Bayes nets [16, 38, 10], in which each action is modeled with a Bayesian network with two "slices" or sets of variables. The first slice represents the values of (possibly multi-valued) variables before the action is performed while the second slice represents the value after the action. Arcs in the diagram represent probabilistic dependence between variables.[7] As with conventional Bayesian networks, each post-action node contains a table of conditional probabilities given the values of its parent variables. The Bayes net representation of the action *Move* in our example is illustrated in Figure 4, with three of its probability tables.

The dashed arcs indicate *persistence* relations: the value of the variable after the action is identical to its value prior to the action. Unlike STRIPS rules, such persistence must be expressed explicitly in the network (though they can be constructed automatically, having the prototypical form shown for the variable *HRC*). In addition, the locally exponential probability tables in the network fail to capture some of the regularities in transition probabilities that allow the STRIPS model to be specified more compactly (e.g., the table for *Wet* could be represented more compactly [21, 10, 44]). Notice however that the independence of the effect of *Move* on *Office* and *Wet* is captured naturally in the network, while standard (stochastic) STRIPS rules cannot express this independence. Our action aspects provide the means to represent such independent effects concisely and are intended to perform precisely this role. The expressiveness of stochastic STRIPS rules (with or without aspects) and two-stage Bayes nets are identical in this propositional setting, both able to express arbitrary transition relations. The relative advantages of both representations *vis-à-vis* compactness

[7] Typically, arcs from pre-action nodes can point only to post-action nodes, while arcs between post-action nodes (correlated action effects) must not induce directed cycles in the graph.

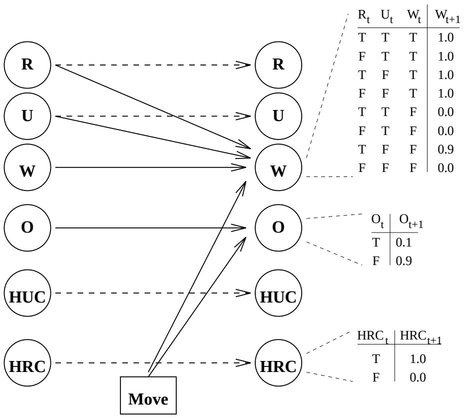

Figure 4: The influence diagram representation for *Move*

	W	$\overline{W}UR$,$\overline{W}U\overline{R}$,$\overline{WUR}$	$\overline{WU}R$
HUC	DelC — 16.0	DelC — 20.0	
$\overline{HUC}, HRC, O$	DelC — 14.73	DelC — 18.73	DelC — 18.66
$\overline{HUC}, HRC, \overline{O}$	Move — 13.92	Move — 17.92	Move — 14.46
$\overline{HUC, HRC, O}$	BuyC — 13.05	BuyC — 17.06	BuyC — 13.81
$\overline{HUC, HRC}, O$	Move — 12.34	Move — 16.34	GetU — 15.66

Table 1: Optimal policy for the COFFEE domain

and naturalness are described in some detail in [8].

The optimal policy and the corresponding value function V^* for this example are shown in Table 1, as computed by policy iteration using a discounting factor of 0.95. While policy iteration explicitly computes an action and value for each of the 64 states, the policy and value function exhibit regularities that permit the compact expression shown in the table.[8]

[8]This fact itself suggests that more reasonable implementations of policy iteration might exploit such structure — see Section 6.

3 Constructing and Solving Abstract MDPs

Even though STRIPS-style representations allow problems with large state spaces to be specified concisely, algorithms such as policy iteration still require enumeration of the exponential state space to produce optimal policies.[9] In classical planning, one technique for dealing with large problems is *abstraction*. In traditional abstraction planners, a complex problem is decomposed into a hierarchy of progressively simpler problems. The simplest problem is then solved, this solution is used to solve the next simplest problem, and so on, until the original problem is solved. While the solutions to these simpler problems are not generally executable plans, they reduce the complexity of the problem by guiding the search for a solution at less abstract levels [49, 29, 30].

We describe an abstraction method, similar in spirit to those used in classical planning, for dealing with large state spaces in solving MDPs. In particular, we adopt a method similar to ABSTRIPS [49, 29] in which an abstract problem is one where certain details of the original problem, in this case propositional atoms, are ignored. However, in contrast to this traditional work, the solutions to our abstract problems will be *directly executable*. Thus, an abstract policy (an optimal solution to an abstract problem) will be an approximately optimal solution to the original problem.[10]

To perform abstraction, we construct an *abstract MDP* that has (possibly exponentially) fewer states, but the same set of actions as the original problem. To reduce the number of states, the propositional description of the problem (i.e., actions and reward structure) is used to choose some subset of the variables that are judged less relevant than the rest, and the irrelevant variables are deleted from the problem description. The idea is to construct a problem that only captures the most important parts of the concrete MDP, find an optimal policy for this abstract MDP using standard algorithms, and apply this policy in the original problem. The key to the approach is the automatic construction of the abstract MDP.

The algorithm used is described in broad outlines in Figure 5. Automatic construction of an abstract MDP requires first that we identify the set of relevant atoms that must be retained in the abstract problem description. The procedure that makes this identification uses a form of value of information as well as a variant of Knoblock's [29] algorithm for constructing abstractions in a classical setting. The abstract state space $\widetilde{\mathcal{S}}$ is the set of states induced by the language obtained by deleting the set of irrelevant atoms. An alternative view of the abstract state space is as an *aggregation* of states: each abstract state $\widetilde{s} \in \widetilde{\mathcal{S}}$ is a collection of concrete states such that each $s \in \widetilde{s}$ is indistinguishable in the reduced language. Finally, a set of actions and a reward function suitable for

[9] For large problems, sparse matrix methods alleviate this problem to some extent, but will only reduce computation by relatively small factors.

[10] Our abstract solutions *can* be used in the traditional way, to guide search for a concrete solution, as well (see Section 4).

1. Using the probabilistic STRIPS representation of the domain, decide which atoms are most important for constructing a good policy. (This defines an abstract state space $\widetilde{\mathcal{S}}$.)
2. For each action, build an abstract transition function $\widetilde{T}$ by deleting all reference to unimportant atoms from the action description and translating the extended STRIPS representation of the action into an MDP transition function. Note that an explicit transition matrix need not be built for each action as the extended STRIPS rules can be used to generate the linear equations required for policy iteration directly.
3. Construct $\widetilde{R}$, the reward function for the abstract problem.
4. Use policy iteration to find the optimal policy $\widetilde{\pi}$ for the MDP $\langle \widetilde{\mathcal{S}}, \mathcal{A}, \widetilde{T}, \widetilde{R} \rangle$.
5. Construct the policy π such that for each state $s \in \widetilde{s} \in \widetilde{\mathcal{S}}, \pi(s) = \widetilde{\pi}(\widetilde{s})$. π is an approximately optimal policy for the original MDP.

Figure 5: Constructing an approximately optimal policy using abstraction

the new state space $\widetilde{\mathcal{S}}$ must be constructed. A key feature of our model is that the set of abstract actions is the same as the action set for the original problem, though each action description may be simplified somewhat.

With this abstract MDP in place, standard methods such as policy iteration can be used to produce an *abstract policy* $\widetilde{\pi}$ associating an action with each abstract state $\widetilde{s} \in \widetilde{\mathcal{S}}$. Finally, the abstract policy determines a concrete policy π such that $\pi(s) = \widetilde{\pi}(\widetilde{s})$ for each $s \in \widetilde{s}$: the action associated with a cluster is applied to each constituent state. (We note that Step 5 need never be performed explicitly; the abstract policy $\widetilde{\pi}$ is itself a good representation of the concrete policy π.)

We describe each of the components of the algorithm below. There are several key points that ensure the usefulness of our abstraction framework. First, the identification of relevant atoms and the construction of the abstract MDP must be very quick — the time taken must be negligible compared to the time required to solve the MDP. In particular, we require that the time grow polynomially with the size of the problem *description* rather than with the size of the state space. Second, the abstract MDP must be well-defined, so that policy construction algorithms applied to the abstract MDP produce meaningful policies. Third, we should be able to bound the error of the abstract policy, or characterize how much worse than optimal the abstract policy might be.

3.1 Constructing an Abstract MDP

In order to construct an abstract MDP, we need to select some subset of the atoms that will form the basis of the abstraction. The quality of the policy and the effectiveness of the abstraction process depend closely on the atoms chosen.

If too many atoms are selected, the policy created may be very close to optimal, but the computational savings may not be large enough to justify the loss of optimality. On the other hand, if the set of atoms chosen is too small, then the computation required to produce the approximate policy will be minimal, but the policy may be quite poor.

As well as choosing an appropriate "number" of atoms for the abstract MDP (e.g., determined by available computation time), we must consider which atoms should be selected. Obviously, if the reward for each state depends solely on the value of a single atom, it would be foolish to ignore that atom when constructing the abstract state space. However, this is not the only consideration — atoms that have relatively little effect on the reward for a state may be ignorable, while atoms that have no direct impact on the reward function (i.e., that are not mentioned in the description of R) may not.

In order to construct a set of atoms which meets the criteria described above, we first identify a set $\mathcal{IR}$ of *immediately relevant atoms*. $\mathcal{IR}$ is formed by examining the propositional model of the reward structure and selecting only those atoms which have the greatest impact on the reward for each state. The larger this set is, the more fine-grained the abstraction will be, so by varying the size of $\mathcal{IR}$, we can strike a balance between the quality of the abstraction and the computation time required.

To construct $\mathcal{IR}$ we examine each atom which appears in the reward function and calculate the maximum range of the reward function for each of its values. In general, atoms with smaller ranges have greater effect on reward than atoms with larger ranges, and should be placed in $\mathcal{IR}$ first. For example, in the COFFEE domain of Figure 1, HUC has range $0.8 - 1.0$ when true, and range $0.0 - 0.2$ when false, so it's maximum range is 0.2. This makes it a better candidate for inclusion in $\mathcal{IR}$ than Wet which has maximum range 0.8. We discuss the choice of immediately relevant atoms further in Section 3.3.

Although the set $\mathcal{IR}$ contains some of the relevant atoms needed for abstraction, it does not yet include all relevant atoms. For example, in a domain where the reward is large if atom A is true and small otherwise, $\mathcal{IR}$ would be $\{A\}$. But if an action that makes A true requires B to be true to achieve the desired effect, then clearly B is a relevant atom as well: ignoring B may not give the agent the ability to affect A as it should.[11] The set $\mathcal{R}$ is defined as the smallest set satisfying the following conditions (as before, d^i is the discriminant associated with the action effect E^i_j):

1. $\mathcal{IR} \subseteq \mathcal{R}$.
2. if $q \in \mathcal{R}$ and for some effect E^i_j, $q \in atoms(E^i_j)$, then $atoms(d^i) \subseteq \mathcal{R}$.

Only the atoms in a discriminant that might probabilistically lead to a relevant effect are deemed relevant; we will call this a *relevant discriminant*. Other

[11] In fact, if the impact of B on the control of A is marginal, we may do well to ignore B after all. We address this issue in Section 5.

```
Initialize R_old ← IR; R ← ∅; R_new ← ∅
while R_old ≠ ∅ do
    for each P ∈ R_old do
        for each action aspect A do
            for each discriminant D^i ∈ A do
                if P ∈ E^i_j for some j then
                    R_new ← R_new ∪ atoms(D^i)
                end if
            end for
        end for
    end for
    R ← R ∪ R_old
    R_old ← R_new − R
end while
```

Figure 6: Algorithm to generate set $\mathcal{R}$ of relevant atoms

conditions associated with the same action aspect are ignored (unless these are relevant for other reasons).

The only decision required from the user of the system is that of which atoms should be placed in $\mathcal{IR}$. As we shall see, this fact allows the user to specify the degree of accuracy required of the abstraction, and to have an abstract policy calculated automatically. (The set $\mathcal{IR}$ may be chosen automatically as well; see Section 3.3.)

The algorithm we use to generate the set of relevant atoms is based on Knoblock's [29] algorithm for determining constraints for problem-specific abstractions. Intuitively, the algorithm backchains through action descriptions to see what atoms influence immediately relevant atoms, what atoms influence those, and so on, until a fixed point is reached. The algorithm is described in Figure 6 and takes as input a set of action (aspect) descriptions, as described above, and a set $\mathcal{IR}$ of immediately relevant atoms. The output is a set $\mathcal{R}$ of relevant atoms. The complexity of this algorithm is $O(r \cdot a \cdot e)$, where r is the number of relevant atoms produced, a is the number of action aspects, and e is the average number of effect literals per action aspect (that is, e is the product of the number of discriminants per action, the number of effects per discriminant and the size of the effects lists). It is reasonable to assume that the "branching factor" and number of effects of a given action is bounded by some reasonably small constant, so we can take e to be constant and state the complexity to be $O(r \cdot a)$. In the worst case, each atom in the language will be considered relevant and the algorithm will take roughly $|\mathbf{P}| \cdot a$ steps. However, even in this worst case, this term is not significant compared to solving an MDP (whose state space is of size $2^{|\mathbf{P}|}$).

Having calculated $\mathcal{R}$, the abstract state space $\widetilde{\mathcal{S}}$ is that induced by clustering together all the states in the original MDP that agree on the values of the atoms in $\mathcal{R}$. By treating each cluster as a state in the abstract MDP, we ignore the irrelevant details of atoms that do not appear in $\mathcal{R}$.

Definition 3.1 The abstract state space generated by $\mathcal{R}$ is $\widetilde{\mathcal{S}} = \{\widetilde{s}_1, \ldots, \widetilde{s}_n\}$, where:

1. $\widetilde{s}_i \subseteq \mathcal{S}$.
2. $\bigcup\{\widetilde{s}_i\} = \mathcal{S}$.
3. $\widetilde{s}_i \cap \widetilde{s}_j = \emptyset$ if $i \neq j$.
4. $s, t \in \widetilde{s}_i$ iff $s \models P$ implies $t \models P$ for all $P \in \mathcal{R}$.

Note that there is no need to actually group together states in the algorithm; the construction of $\widetilde{\mathcal{S}}$ is merely conceptual.

To illustrate the construction of an abstract state space, we consider the COFFEE example shown in Figures 1 and 3. There are two atoms that influence the reward assigned to a state, *HUC* and *Wet*; but the influence of *Wet* is relatively small while that of *HUC* is more substantive. Thus, we will set $\mathcal{IR} = \{HUC\}$. To construct $\mathcal{R}$, we notice that only the action *DelC* affects *HUC*, and that the variables *Office* and *HRC* influence its truth; so *Office* and *HRC* are added to $\mathcal{R}$. When examining discriminants of actions that affect these two atoms, we see that no further atoms are deemed relevant. We end up with $\mathcal{R} = \{HUC, Office, HRC\}$. The abstract state space $\widetilde{\mathcal{S}}$ consists of those subsets of eight states that agree on the truth assignment to these three atoms, but disagree on the values of the remaining irrelevant atoms *Rain*, *Wet* and *Umb*. A portion of the abstract state space is shown in Figure 7. Note that $|\widetilde{\mathcal{S}}| = 8$ (in contrast, $|\mathcal{S}| = 64$).

We note that by breaking up the action *Move* into two independent aspects, the set $\mathcal{R}$ remains small. Had we used the expanded action effect shown in Figure 2, the atoms *Rain* and *Umb* would have been added to the relevant set although they have no impact on the probability of other relevant atoms becoming true or false. We also note that had we chosen $\mathcal{IR}$ to include *Wet*, then *Rain* and *Umb* would have been added to $\mathcal{R}$ due to the second aspect of the *Move* action, so all the atoms from the original problem would appear in $\mathcal{R}$ (and the abstract state space would be identical in size to the original).

Apart from the abstract state space, we require actions and a reward function compatible with these abstract states. In general, we can imagine that computing the transition probabilities for actions associated with an arbitrary clustering of states is computationally prohibitive, demanding that one consider the effect of each action on each state in the cluster. Furthermore, computing the probability of moving from one cluster to another requires, in general, some prior distribution over the states in the initial cluster. This cannot be realized in our setting, since such information depends on the distribution over initial

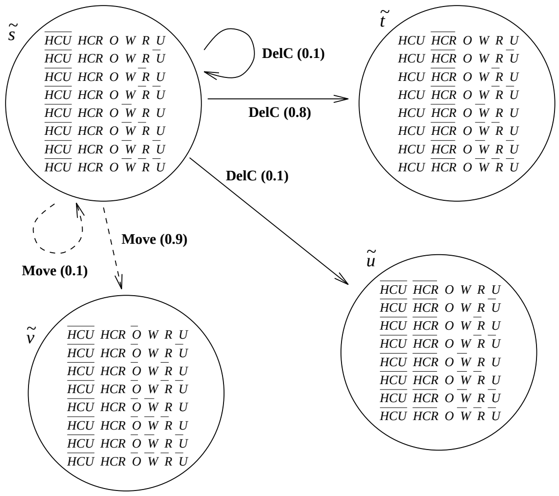

Figure 7: Portion of the abstract state space

states of the system, and knowledge of the policy adopted by the agent (which is what we wish to compute).

The clustering mechanism we have described is designed to avoid exactly these problems. The definition of $\mathcal{R}$ (in particular, the requirement that all atoms of a discriminant be added to $\mathcal{R}$ whenever part of the corresponding effect is in $\mathcal{R}$) ensures that the states in any given cluster have the same transition probabilities for each action. More accurately, each state in a fixed cluster $\widetilde{s}$ has the same probability of moving to a given cluster $\widetilde{t}$. Therefore, we may assign the (unique) transition probability for some state in the cluster to the cluster itself. For instance, in Figure 7, assigning probability 0.8 to the transition from cluster $\widetilde{s}$ to $\widetilde{t}$ (under the action $DelC$) is perfectly reasonable, since each $s \in \widetilde{s}$ will move to some state in $\widetilde{t}$ with probability 0.8. The following results illustrate the significant characteristics of the abstraction mechanism more formally.

Lemma 3.1 *If $\widetilde{s}$ is an abstract state, $s, t \in \widetilde{s}$, and d is a relevant discriminant for some action aspect, then s satisfies d iff t does.*

Proof Let $s, t \in \widetilde{s}$ and d be some relevant discriminant. Since s and t are in the same cluster, they must assign the same truth value to each atom in $\mathcal{R}$. Since d is relevant, $atoms(d) \subseteq \mathcal{R}$. So $s \models d$ iff $t \models d$. ■

Lemma 3.2 *If E_i is a possible effect of some action and $s, t \in \widetilde{s}$, then: i) If E_i is associated with an irrelevant discriminant, then $E_i(s) \in \widetilde{s}$; and ii) $E_i(s) \in \widetilde{u}$ iff $E_i(t) \in \widetilde{u}$.*

Proof i) If E_i is associated with an irrelevant discriminant, then $E_i \cap \mathcal{R} = \emptyset$ (otherwise the discriminant would be relevant). $E_i(s)$ must therefore agree with s on the truth value of all atoms in $\mathcal{R}$, and hence $E_i(s) \in \widetilde{s}$.
ii) Since s and t agree on the values of all atoms in $\mathcal{R}$, $E_i(s)$ and $E_i(t)$ must also (because E_i changes the same literals in both). So $E_i(s) \in \widetilde{u}$ iff $E_i(t) \in \widetilde{u}$. ■

Theorem 3.3 *Let $\widetilde{s}$ and $\widetilde{u}$ be clusters such that $s, t \in \widetilde{s}$. Then for any action a*

$$\sum_{u \in \widetilde{u}} \Pr(u|a,s) = \sum_{u \in \widetilde{u}} \Pr(u|a,t)$$

Proof Since s and t are in the same cluster, they either satisfy the *same* relevant discriminant d for action a or neither satisfies a relevant discriminant for a (by Lemma 3.1). In the former case, for each effect E_i associated with d, we have $E_i(s) \in \widetilde{u}$ iff $E_i(t) \in \widetilde{u}$ (by Lemma 3.2). Since $\Pr(u|a,s) = \sum\{p_i : E_i(s) = u\}$ for each u (and similarly for t), the result holds. In the latter case, let $s \models d^j$ and $t \models d^k$, where d^j and d^k are irrelevant discriminants for a. By Lemma 3.2, each possible effect E_i^j is such that $E_i^j(s) \in \widetilde{s}$, so $\sum_{u \in \widetilde{u}} \Pr(u|a,s)$ equals 1 if $\widetilde{s} = \widetilde{u}$ and 0 otherwise. Similarly, $\sum_{u \in \widetilde{u}} \Pr(u|a,t)$ equals 1 if $\widetilde{s} = \widetilde{u}$ and 0 otherwise. Thus the result holds in the latter case as well. ■

Theorem 3.3 provides justification for associating a unique transition probability $\Pr(\widetilde{t}|a,\widetilde{s})$ with the abstract MDP, namely $\sum_{t \in \widetilde{t}} \Pr(t|a,s)$. Figure 8 illustrates how the abstraction mechanism works when concrete states map to more than one state within a given cluster (in this case for the action *Move*). The bold type indicates the abstract version of the concrete transitions (in smaller type).

The results above permit a simple syntactic procedure to construct abstract action descriptions: we simply delete all reference to irrelevant atoms from the actions in the original problem. This may subsequently permit simplification of the action specification, as discriminants and effects that were different in the original problem potentially become the same in the abstract problem. For example, consider the action shown in Figure 9, where atom Q is deemed relevant and as a result so is A, leaving B and P to be irrelevant. The abstract action is created by deleting P and B from the action description. We note that this leaves the two possible effects associated with each of the first two discriminants identical (both become $\emptyset$). So the first stage of the simplification is to collapse identical action effects within a discriminant into one effect with the sum of the original probabilities. In this case, the effect becomes $\emptyset$ and is given probability 1.0. We also note that action discriminants may also become non-disjoint or

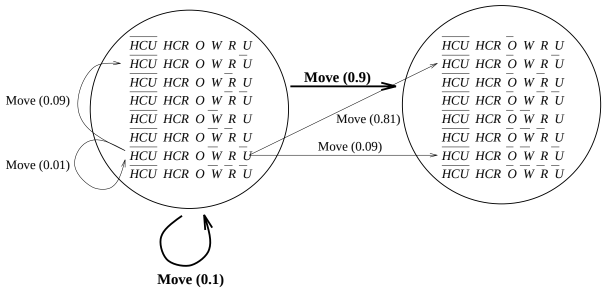

Figure 8: Summing transition probabilities for the abstract MDP

Discriminant	*Effect*	*Prob.*
A, B	P	0.9
	$\emptyset$	0.1
$A, \neg B$	$\neg P$	0.9
	$\emptyset$	0.1
$\neg A$	Q	0.9
	$\emptyset$	0.1

Figure 9: An action that simplifies when abstracted

Action	*Discriminant*	*Effect*	*Prob.*
Move	*Office*	$\neg$ *Office*	0.9
		$\emptyset$	0.1
	$\neg$ *Office*	*Office*	0.9
		$\emptyset$	0.1
BuyC	$\neg$ *Office*	*HRC*	0.8
		$\emptyset$	0.2
	Office	$\emptyset$	1.0
GetU	*True*	$\emptyset$	1.0
DelC	*Office*, *HRC*	*HUC*, $\neg$*HRC*	0.8
		$\neg$*HRC*	0.1
		$\emptyset$	0.1
	$\neg$ *Office*, *HRC*	$\neg$*HRC*	0.8
		$\emptyset$	0.2
	$\neg$*HRC*	$\emptyset$	1.0

Figure 10: Abstract actions for the COFFEE domain

even identical; however, by construction (see, e.g., Lemma 3.2) this can only be the case when the discriminants in question are irrelevant (i.e., have no relevant effects), thus only when the reduced effect is $\emptyset$ with probability 1. We can therefore collapse overlapping or identical reduced discriminants into a set of well-formed discriminants or into a single discriminant in the abstract action, this being well-defined since they must have the same "abstract" effect. The reduced action description in this small example has two discriminants, $A : \emptyset, 1.0$ and $\neg A : Q, 0.9; \emptyset, 0.1$. In simple outline, the algorithm for constructing an abstract action description, given the set of relevant atoms $\mathcal{R}$ is:

1. Delete irrelevant atoms from each d^i and E^i_j (call these the *reduced* discriminants and effects).
2. For each discriminant d^i, collapse any reduced effects E^i_j, E^i_k, etc. that have become identical into a single reduced effect with probability equal to $\sum p^i_j$ of the participating reduced effects
3. For any non-disjoint reduced discriminant (which must thus have a collapsed effect of the form $\emptyset$), simplify the action description as needed.

The set of abstract actions for the COFFEE domain (Figure 1) is shown in Figure 10. We denote by $\widetilde{\mathcal{T}}$ the new transition function constructed in this way for the abstract MDP (the set of actions $\mathcal{A}$ remains unchanged). It is easy to see by construction that:

Proposition 3.4 *For any $s \in \widetilde{s}$ and action a*

$$\Pr(\widetilde{t}|a, \widetilde{s}) = \sum_{t \in \widetilde{t}} \Pr(t|a, s)$$

Sentence	*Value*
HUC	0.9
$\neg$*HUC*	0.1

Figure 11: Abstract reward function for the COFFEE domain

A simple corollary of Theorem 3.3 and Proposition 3.4 is the fact that the abstract process is indeed Markovian. Let $\widetilde{S}^m$ be a random variable denoting the state of the abstract process at stage m. Then we have

Corollary 3.5

$$\Pr(\widetilde{S}^m | a^{m-1}, \widetilde{S}^{m-1}, a^{m-2}, \widetilde{S}^{m-2}, \cdots \widetilde{S}^0) = \Pr(\widetilde{S}^m | a^{m-1}, \widetilde{S}^{m-1})$$

The system dynamics for the abstract system is truly history-independent.

The reward function for the abstract MDP is denoted $\widetilde{R}$ and must associate an immediate reward with each abstract state or cluster $\widetilde{s} \in \widetilde{\mathcal{S}}$. We choose to assign the midpoint of the range of (concrete) rewards for the states in $\widetilde{s}$. Formally, let $\min(\widetilde{s})$ and $\max(\widetilde{s})$ denote the minimum and maximum values of the set $\{R(s) : s \in \widetilde{s}\}$, respectively. The *abstract reward function* is:

$$\widetilde{R}(\widetilde{s}) = \frac{\max(\widetilde{s}) + \min(\widetilde{s})}{2}$$

This choice of $\widetilde{R}(\widetilde{s})$ minimizes the maximum difference between $R(s)$ and $\widetilde{R}(\widetilde{s})$ for any $s \in \widetilde{s}$, and is adopted because it allows the tightest error bounds to be derived (below). Although using the average of the rewards in a cluster might result in better average-case behavior, it can lead to much worse bounds on the difference between the abstract and optimal policies. The abstract reward function for our example problem is shown in Figure 11. The general method for constructing this representation is identical to that used for creating abstract action descriptions, with the simple addition of choosing a midpoint reward for any collapsible reward discriminants.

3.2 Solution of Abstract MDPs

Once we have constructed an abstract MDP $\langle \widetilde{\mathcal{S}}, \mathcal{A}, \widetilde{T}, \widetilde{R} \rangle$, we can compute an optimal *abstract policy* $\widetilde{\pi}$ as well as an optimal *abstract value function* $\widetilde{V}^*$ using standard policy construction techniques. For example, the optimal abstract policy and value function (computed using policy iteration) for the abstract

	HUC	Val.	$\neg HUC$	Val.
$HRC, Office$	$DelC$	18.0	$DelC$	16.7
$HRC, \neg Office$	$DelC$	18.0	$Move$	15.9
$\neg HRC, Office$	$DelC$	18.0	$Move$	14.3
$\neg HRC, \neg Office$	$DelC$	18.0	$BuyC$	15.1

Table 2: The policy computed using the abstract MDP

version of the coffee problem is described in Table 2, which shows the action and value for each of the eight abstract states. When compared to the optimal policy for the original problem (Table 1), we see that an "optimal" action is chosen at all but one of the 64 states. As we would expect given the method of construction, the policy is optimal except in the state where it is raining and the robot can pick up the umbrella before going to the coffee shop — in the abstract policy the robot immediately heads for coffee ignoring the umbrella.[12] The abstract value for each cluster according to the abstract policy is close to the midpoint of the range of true (optimal) values for the states in the cluster. Moreover, the time required to compute the abstract policy is only 2 percent of that required for optimal policy construction in the original problem (both using policy iteration).

In general, the utility of a particular abstraction is a function of the time required to compute the abstract policy and the quality of the abstract policy (both relative to the same properties for the original problem). Since the time required for policy iteration is a function of the size of the state space, and the size of the state space is exponential in the number of underlying atoms, any reduction in the size of $\mathcal{R}$ will result in an exponential reduction in the size of the state space and hence in computation time. Even reducing the domain by a single atom will halve the size of the state space, and produce a large computational saving when performing policy iteration or other policy construction methods.[13]

This speed-up comes at the cost of generating possibly less-than-optimal policies. However, we can estimate the solution quality of an abstract policy by bounding the *error* associated with this policy. In particular, we are interested in two quantities, the difference between the computed value of an abstract policy and its true value in the original MDP, and the difference between this true

[12] The fact that the abstract policy agrees with the optimal policy in states where the goal has been achieved is an artifact of the robot choosing the "harmless" action $DelC$ arbitrarily. Had the robot chosen the "harmless" action $Move$ as something to do once coffee has been successfully delivered, the robot would get wet if it is raining, making the abstract policy suboptimal in all states where $Rain$ and HUC hold, losing the small reward for $\neg W$.

[13] The number of iterations required by policy iteration can be hard to predict for a given problem, but is polynomial in $|\mathcal{S}|$. Aside from the number of iterations, the *time per iteration* is generally $O(|\mathcal{S}|^3)$. See [34] for a survey of complexity results regarding the solution of MDPs.

value and the value of an optimal policy for the original MDP. More precisely, let $\widetilde{\pi}$ be the optimal policy computed for the abstract MDP and let $V_{\widetilde{\pi}}$ be the abstract value function computed for this policy. Let π be the concrete policy induced by $\widetilde{\pi}$ (i.e., $\pi(s) = \widetilde{\pi}(\widetilde{s})$ for each $s \in \widetilde{s}$) and let V_π be the value of this concrete policy. Finally, let V^* denote the optimal value function for the concrete MDP. The quality of an abstract policy will be characterized in terms of two quantities, the discounting factor β and the maximum *reward span* of the abstract MDP.

Definition 3.2 The *reward span* of a cluster $\widetilde{s} \in \widetilde{\mathcal{S}}$ is the maximum range of possible rewards for that cluster, that is

$$span(\widetilde{s}) = \max(\widetilde{s}) - \min(\widetilde{s})$$

The reward span for a cluster is twice the maximum degree to which the estimate $\widetilde{R}(\widetilde{s})$ of the immediate reward associated with a state $s \in \widetilde{s}$ differs from the true reward $R(s)$ for that state.[14]

Let δ denote the maximum reward span over all the clusters in $\widetilde{\mathcal{S}}$; that is:

Definition 3.3 The *maximum reward span* for an abstract MDP is

$$\delta = \max_{\widetilde{s} \in \widetilde{\mathcal{S}}} \{span(\widetilde{s})\}$$

Proposition 3.6 *For any* $s \in \widetilde{s}$

$$R(s) - \frac{\delta}{2} \leq \widetilde{R}(\widetilde{s}) \leq R(s) + \frac{\delta}{2}$$

Theorem 3.7 *For any* $s \in \widetilde{s} \in \widetilde{\mathcal{S}}$,

$$\left| \widetilde{V}_{\widetilde{\pi}}(\widetilde{s}) - V_\pi(s) \right| \leq \frac{\delta}{2(1-\beta)}$$

Proof We prove inductively that for all n

$$\left| \widetilde{V}^n_{\widetilde{\pi}}(\widetilde{s}) - V^n_\pi(s) \right| \leq \sum_{i=0}^{n} \frac{\delta}{2} \beta^i$$

[14] The use of utility spans to generate abstractions is proposed by Horvitz and Klein [24], who use the notion in single-step decision making. Our analysis can be applied to their framework to establish bounds on the degree to which an "abstract decision" can be less than optimal. Furthermore, the notion is useful in more general circumstances, as our results illustrate.

Since $V_\pi(s) = \lim_{n\to\infty} V_\pi^n(s)$ and $\widetilde{V}_{\widetilde{\pi}}(\widetilde{s}) = \lim_{n\to\infty} \widetilde{V}_{\widetilde{\pi}}^n(\widetilde{s})$, this suffices to prove the result.

The base of the induction is immediate: since $\widetilde{V}_{\widetilde{\pi}}^0(\widetilde{s}) = \widetilde{R}(\widetilde{s})$ and $V_\pi^0(s) = R(s)$, by Proposition 3.6 we have

$$\left|\widetilde{V}_{\widetilde{\pi}}^0(\widetilde{s}) - V_\pi^0(s)\right| \leq \frac{\delta}{2}\beta^0$$

Now assume that for all $\widetilde{s}$ and $s \in \widetilde{s}$, for some fixed k,

$$\left|\widetilde{V}_{\widetilde{\pi}}^k(\widetilde{s}) - V_\pi^k(s)\right| \leq \sum_{i=0}^{k} \frac{\delta}{2}\beta^i$$

Then

$$\begin{aligned}
\left|\widetilde{V}_{\widetilde{\pi}}^{k+1}(\widetilde{s}) - V_\pi^{k+1}(s)\right| &= \left|\left[\widetilde{R}(\widetilde{s}) + \beta \sum_{\widetilde{t}\in\widetilde{\mathcal{S}}} \Pr(\widetilde{t}|\widetilde{\pi}(\widetilde{s}),\widetilde{s})\widetilde{V}_{\widetilde{\pi}}^k(\widetilde{t})\right] - \left[R(s) + \beta \sum_{t\in\mathcal{S}} \Pr(t|\pi(s),s)V_\pi^k(t)\right]\right| \\
&\leq \left|\widetilde{R}(\widetilde{s}) - R(s)\right| + \beta \left|\sum_{\widetilde{t}\in\widetilde{\mathcal{S}}}\left[\Pr(\widetilde{t}|\widetilde{\pi}(\widetilde{s}),\widetilde{s})\widetilde{V}_{\widetilde{\pi}}^k(\widetilde{t}) - \sum_{t\in\widetilde{t}} \Pr(t|\pi(s),s)V_\pi^k(t)\right]\right|
\end{aligned}$$

By Proposition 3.6, Theorem 3.3 and Proposition 3.4, this term is no greater than

$$\frac{\delta}{2} + \beta\left|\widetilde{V}_{\widetilde{\pi}}^k(\widetilde{s}) - V_\pi^k(s)\right|$$

Therefore, by the inductive hypothesis,

$$\left|\widetilde{V}_{\widetilde{\pi}}^{k+1}(\widetilde{s}) - V_\pi^{k+1}(s)\right| \quad\leq\quad \frac{\delta}{2} + \beta\sum_{i=0}^{k}\frac{\delta}{2}\beta^i \quad\leq\quad \sum_{i=0}^{k+1}\frac{\delta}{2}\beta^i$$

■

This result shows the maximum difference between the computed value of the abstract policy and the actual value of that policy when implemented in the original decision problem. In effect, this determines the confidence we may adopt in this computed value. Intuitively, this result shows that the value of the policy differs from the computed value by no more that $\frac{\delta}{2}$ per stage of the process.

Of use in determining the loss of value one might expect by focusing on the abstract problem is the following result (where $\widetilde{V}^*$ denotes the optimal value function for the abstract MDP):

Lemma 3.8 *For any $s \in \widetilde{s}$,*

$$\left| V^*(s) - \widetilde{V}^*(\widetilde{s}) \right| \leq \frac{\delta}{2(1-\beta)}$$

Proof We prove inductively that for all n

$$|\widetilde{V}^n(\widetilde{s}) - V^n(s)| \leq \sum_{i=0}^{n} \frac{\delta}{2}\beta^i$$

Since $V^*(s) = \lim_{n\to\infty} V^n(s)$ and $\widetilde{V}^*(\widetilde{s}) = \lim_{n\to\infty} \widetilde{V}^n(\widetilde{s})$, this suffices to prove the result.

The base of the induction is immediate: since $\widetilde{V}^0(\widetilde{s}) = \widetilde{R}(\widetilde{s})$ and $V^0(s) = R(s)$, by Proposition 3.6 we have

$$|\widetilde{V}^0(\widetilde{s}) - V^0(s)| \leq \frac{\delta}{2}\beta^0$$

Now assume that for all $\widetilde{s}$ and $s \in \widetilde{s}$, for some fixed k,

$$|\widetilde{V}^k(\widetilde{s}) - V^k(s)| \leq \sum_{i=0}^{k} \frac{\delta}{2}\beta^i$$

Let a be any action that maximizes the value of $\widetilde{V}^{k+1}(\widetilde{s})$ in its definition (see Equation 3); i.e.,

$$\widetilde{V}^{k+1}(\widetilde{s}) = \widetilde{R}(\widetilde{s}) + \beta \sum_{\widetilde{t}\in\widetilde{\mathcal{S}}} \Pr(\widetilde{t}|a,\widetilde{s})\widetilde{V}^k(\widetilde{t})$$

Similarly, let b be a maximizing action for $V^{k+1}(s)$, so that

$$V^{k+1}(s) = R(s) + \beta \sum_{t\in\mathcal{S}} \Pr(t|b,s) V^k_\pi(t)$$

Thus, we have

$$\begin{aligned}
\left|\widetilde{V}_{k+1}(\widetilde{s}) - V^{k+1}(s)\right| &= \left| \left[\widetilde{R}(\widetilde{s}) + \beta\sum_{\widetilde{t}\in\widetilde{\mathcal{S}}} \Pr(\widetilde{t}|a,\widetilde{s})\widetilde{V}^k(\widetilde{t})\right] - \left[R(s) + \beta\sum_{t\in\mathcal{S}} \Pr(t|b,s)V^k(t)\right]\right| \\
&\leq \left|\widetilde{R}(\widetilde{s}) - R(s)\right| + \beta\left|\sum_{\widetilde{t}\in\widetilde{\mathcal{S}}} \Pr(\widetilde{t}|a,\widetilde{s})\widetilde{V}^k(\widetilde{t}) - \sum_{t\in\mathcal{S}} \Pr(t|b,s)V^k(t)\right|
\end{aligned}$$

We introduce the following defined terms; let

$$\begin{aligned}
\mathbf{A} &= \sum_{t\in\mathcal{S}} \Pr(t|b,s)V^k(t) \\
\mathbf{B} &= \sum_{t\in\mathcal{S}} \Pr(t|a,s)V^k(t) \\
\mathbf{C} &= \sum_{\widetilde{t}\in\widetilde{\mathcal{S}}} \Pr(\widetilde{t}|a,\widetilde{s})\widetilde{V}^k(\widetilde{t}) \\
\mathbf{D} &= \sum_{\widetilde{t}\in\widetilde{\mathcal{S}}} \Pr(\widetilde{t}|b,\widetilde{s})\widetilde{V}^k(\widetilde{t})
\end{aligned}$$

and let $f = \sum_{i=0}^{k} \frac{\delta}{2}\beta^i$. Now, by Theorem 3.3 and Proposition 3.4, and the inductive hypothesis, it is easy to verify that both $|\mathbf{B}-\mathbf{C}| \le f$ and $|\mathbf{A}-\mathbf{D}| \le f$. Furthermore, by the choice of actions a and b (which maximize their value respective functions), we have both $\mathbf{A} \ge \mathbf{B}$, and $\mathbf{C} \ge \mathbf{D}$. Reasoning with these inequalities, we obtain that: a) if $\mathbf{C} \ge \mathbf{B}$, we have $\mathbf{C}-\mathbf{A} \le f$, and (since $\mathbf{C} \ge \mathbf{D}$ and $|\mathbf{A}-\mathbf{D}| \le f$) we have $\mathbf{A}-\mathbf{C} \le f$ — thus, $|\mathbf{A}-\mathbf{C}| \le f$; b) if $\mathbf{C} < \mathbf{B}$, we have $\mathbf{A} > \mathbf{C}$, and (since $\mathbf{C} \ge \mathbf{D}$ and $\mathbf{A}-\mathbf{D} \le f$), we have $|\mathbf{A}-\mathbf{C}| \le f$. In either case, $|\mathbf{A}-\mathbf{C}| \le f = \sum_{i=0}^{k} \frac{\delta}{2}\beta^i$. Plugging these quantities into the inequality above, we obtain

$$\begin{aligned}
\left|\widetilde{V}_{k+1}(\widetilde{s}) - V^{k+1}(s)\right| &\le \left|\widetilde{R}(\widetilde{s}) - R(s)\right| + \beta\,|\mathbf{C}-\mathbf{A}| \\
&\le \frac{\delta}{2} + \beta\sum_{i=0}^{k}\frac{\delta}{2}\beta^i \\
&\le \sum_{i=0}^{k+1}\frac{\delta}{2}\beta^i
\end{aligned}$$

■

The true bound of solution quality is given by the following result:

Theorem 3.9 *For any $s \in \mathcal{S}$,*

$$|V_{\pi*}(s) - V_{\pi}(s)| \le \frac{\beta\delta}{1-\beta}$$

Proof This follows immediately from Theorem 3.7 and Lemma 3.8, with the combined error reduced by the added term β due to the fact that the initial reward received at stage 0 will be identical in both cases. ■

This is the main result regarding our abstraction mechanism. By adopting an abstract version of the original decision problem, we can guarantee that an agent implementing the abstract policy will lose no more than a reward of δ per stage of the process — the error introduced by abstraction is simply additive over time. In addition, the smaller the reward span of the clusters used in the abstract process, the better the performance guarantees on the abstract solution. Clearly, if the abstraction is such that no atoms that impact on the reward function are deleted, the abstract solution will be optimal (since $\delta = 0$).

Of course, the error bound here is absolute, not relative. While the most one could lose is δ per stage, there is a possibility that this is "all the value" we could have obtained by behaving optimally. For instance, in our example it might have been that case that getting coffee turned out to be impossible, in which case staying dry is the best the robot could have done; yet the abstraction prevented even this. The relative error in this case may be unacceptable. However, tight relative error bounds, while not computable *a priori* can be determined once the value function has been computed.

We note that should a more refined abstraction be used, the generated policy will have tighter error bounds. Although one cannot guarantee improvement of the abstract policy at each state (with respect to the performance of its concrete counterpart) when moving to a less abstract version of an MDP, the bound on the maximum possible error will be tighter and we can typically expect better policies as a result.

Finally, we point out that the reliance of the error term on the discounting factor β is of little import. As mentioned, this simply indicates that value loss accumulates over time. Since value itself accumulates over time, it is the relative value loss that is crucial. If the problem is undiscounted (i.e., $\beta = 1$), then the error is unbounded, but generally so is value for the types of problems we consider. In such a case, an average reward analysis could be performed. We do not pursue this here, but expect that our ideas can be extended this way in a relatively straightforward fashion.

3.3 Choosing an Abstraction

We have described how an abstract MDP is generated and solved given some set $\mathcal{IR}$ of immediately relevant atoms. The time required to solve the abstract MDP and the accuracy of the policy produced will both depend on the chosen $\mathcal{IR}$, or more directly on the size of the induced set $\mathcal{R}$ of relevant atoms. A smaller relevant set is desirable since computation time grows exponentially with the number of relevant atoms (in fact, in a polynomial of this exponential factor), but a larger relevant set is desirable since the error bound will be tighter. This produces the tension between computation time and solution quality characteristic of most AI problems, especially those for which anytime algorithms are designed. The key question then arises: what is the "right" set of relevant atoms to use given this tradeoff?

The answer to this question depends of course on the time pressure under which a planning agent finds itself and the relative values of quick solutions versus good solutions. This type of issue is addressed in the work of Boddy and Dean [6, 7], Horvitz [23], Russell and Wefald [47] and other work on anytime methods. It is important that we provide techniques for estimating solution time and solution quality as well as methods for improving solution quality in a way that can interact with a module assessing the time-quality tradeoffs. We briefly sketch ways to compute the error bound associated with a particular abstraction as well as improve this bound through judicious selection of new relevant atoms.

Deciding *which* atoms to add to the set $\mathcal{IR}$ is essentially a *value of information* calculation [27, 41]. We could imagine for example being given a time bound and wanting the best possible solution computable within that time. Since a time bound restricts the number of atoms that can be considered, we want a set of relevant atoms that satisfies the size restriction and has the lowest possible error bound. Thus we want a set $\mathcal{IR}$ that has the largest value of information for our decision problem among all sets of atoms of the appropriate size. Alternatively, we may simply want a solution of fixed quality (whose error is under some threshold). In this case, we want the smallest set of relevant atoms whose error bound is under threshold.

To estimate the computation time associated with a given set $\mathcal{IR}$, we must first generate the set $\mathcal{R}$ induced by $\mathcal{IR}$. As described above, this operation is relatively efficient.[15] The maximum span $\mathit{span}(\mathcal{IR})$ provides an estimate of the value of information associated with $\mathcal{IR}$: the smaller $\mathit{span}(\mathcal{IR})$ is, the better we expect the solution of the abstract MDP to be. Given a set $\mathcal{IR}$, let $\mathit{Val}(\mathcal{IR})$ be the set of $2^{|\mathcal{IR}|}$ truth valuations over this set of atoms (i.e., the set of clusters induced by $\mathcal{IR}$). Assume that the reward function is represented using a set D of reward discriminants. We compute $\mathit{span}(\mathcal{IR})$ as follows:

1. For each $v \in \mathit{Val}(\mathcal{IR})$, compute $\mathit{span}(v)$ to be

$$\max_{d \in D}\{R(d) : v \not\models \neg d\} - \min_{d \in D}\{R(d) : v \not\models \neg d\}$$

2. Let $\mathit{span}(\mathcal{IR}) = \max\{\mathit{span}(v) : v \in \mathit{Val}(\mathcal{IR})\}$.

Testing the span of any cluster v requires $|D|$ satisfiability tests (i.e., testing whether v intersects each utility discriminant). The satisfiability tests themselves will be $O(|v| \cdot |d|)$ when each discriminant $d \in D$ is represented as a set of literals (as v is). Assuming the size of the reward description to be bounded, this becomes essentially linear in the number of immediately relevant atoms. This ensures that computing $\mathit{span}(\mathcal{IR})$ is exponential in $|\mathcal{IR}|$, but linear in the

[15] We note that appropriate preprocessing of actions – e.g., constructing an operator graph as described in [55] – can make this much more efficient.

size of the induced (abstract) state space.[16]

To determine the best set of immediately relevant atoms, say, of a given size is equivalent to determining the subset of atoms of that size with the greatest value of information. Value of information has certain nice properties such as monotonicity: the value of knowing the assignments of a set of variables $S \supseteq S'$ is at least as great as that of knowing S'. In our setting, this is reflected in the fact that adding new relevant atoms will not worsen (and will generally improve) bounds on solution quality. Unfortunately, value of information (and span) have other less desirable features. For example, the atom with the best single value may not be an element of the set of two atoms with highest value. Thus we cannot guarantee that determining the most important single atom will aid in constructing the most valuable set of two (or more) atoms.

Determining a set of variables of fixed size with greatest information value generally requires exhaustive search through the space of possible sets [27]. If time restrictions require a set $\mathcal{IR}$ of size k, we potentially have to enumerate all size k subsets of the set of atoms P and determine their span, choosing a set with smallest span. Of course, atoms not mentioned in the reward discriminant can have no impact on span, so we can restrict attention to subsets of the atoms mentioned in D (i.e., to $\cup\{d \in D\}$). If we want to find the smallest set $\mathcal{IR}$ with an error bound under some threshold, this may require exhaustive search through all subsets of the atoms in D. In either case, constructing optimal approximations is computationally prohibitive, and grows exponentially with the number of immediately relevant atoms one is willing to consider.

One may alleviate this problem by adopting a greedy approach to abstraction selection. Techniques of this type are the norm when value of information is involved. For each atom $p \in \cup\{d \in D\}$, one can estimate its value by computing $span(\{p\})$ as above. For each atom p, this is an $O(|D|)$ operation. This captures the rough value of abstracting based on p only and provides an ordering of all atoms that might potentially be added to $\mathcal{IR}$. We can then add atoms to $\mathcal{IR}$ incrementally based on this ordering, adding atoms with smaller span first.

If we are attempting to find the set of k atoms with the lowest error bound, we can simply add the best k atoms (considered individually) to the set $\mathcal{IR}$ and solve the problem. This greedy strategy will not generally guarantee that it is indeed the best set of k atoms (considered collectively), but may work well in practice.[17] In addition, under certain conditions, such a greedy strategy can produce optimal abstractions — for instance, when the reward function encodes

[16] The estimated $span(\mathcal{IR})$ may in fact be too liberal. When determining the set of relevant atoms $\mathcal{R}$ based on $\mathcal{IR}$, additional atoms may be added to $\mathcal{R}$ that impact reward, and may in fact reduce the span further. In this case, the error bounds on solution quality will be even tighter than indicated by $span(\mathcal{IR})$. One could, when adding atoms to $\mathcal{IR}$, compute the span of the induced set $\mathcal{R}$ if one is willing to construct $\mathcal{R}$ for the different candidate sets $\mathcal{IR}$. All the methods for choosing $\mathcal{IR}$ described here can be applied using $Val(\mathcal{R})$ instead of $Val(\mathcal{IR})$.

[17] We again emphasize that we are ignoring the fact that the set $\mathcal{R}$ may have a tighter span than $\mathcal{IR}$. In general, one can use $span(\mathcal{R})$ to get more accurate estimates if one is willing to compute $\mathcal{R}$ repeatedly.

additive independent rewards for certain atoms or sets of atoms. If we want to find the smallest set of atoms within a certain error threshold, we can construct $\mathcal{IR}$ incrementally. We add atoms to $\mathcal{IR}$ according to their span, at each stage testing the span of the current set $\mathcal{IR}$. If the error is below threshold we use $\mathcal{IR}$ as it stands; if not, we add the next best atom to $\mathcal{IR}$ and test again. Similar remarks apply here: under certain conditions this may guarantee an optimal (smallest) abstraction, but not generally.

We note that if the size of $\cup\{d \in D\}$ is relatively small (in relation to the problem size $|\mathbf{P}|$), then the computation involved in determining an optimal (or roughly optimal) abstraction that satisfies the time or solution quality constraints imposed by the problem is relatively trivial compared with the time to solve the abstract MDP itself. We expect that computation time spent in careful abstraction selection using value of information considerations will be time well spent in typical domains.

3.4 Experimental Results

The error bounds described in Section 3.2 are worst-case bounds described in terms of the maximum reward span. However, if certain clusters have smaller span than maximum we can expect better performance. In addition, unless we visit states at each stage whose reward actually differs maximally from the abstract reward, we will generally not achieve these worst case results. In this section we relate some initial experimental results that examine the performance of our abstraction mechanism.

The COFFEE domain is an extension of the running COFFEE example we have been using. It contains 2048 states described by ten variables, and seven actions (see Appendix A for a description of the problem). There are three possible abstractions for this domain, with 32, 64 and 256 abstract states respectively. Policy iteration on the 2048-state complete problem required 1588 seconds and 113 iterations. The optimal values for this problem range from 22.4 to 42.0. The results of the abstractions are summarized in Table 3 which compares the computed abstract policy value with its true value, as well as this true value with the optimal value for the original problem.[18] As the table shows, the more fine-grained the abstraction, the better the resulting policy is. The number of "correct" actions chosen by the abstract policies improves until over 93 per cent of states agree with the optimal policy using the 256 state abstraction. Of course, this measure is less crucial[19] than the loss in expected value accrued

[18] Number of errors refers to the number of (concrete) states at which the true value differs from the optimal (or computed) value. The average, deviation and maximums refer to the magnitudes of these differences (and percentage of the range of optimal state values). Time and number of iterations refers to the time taken by policy iteration to compute the optimal policy for each of the abstractions (and percentage of time compared to optimal solution).

[19] Indeed, the extent to which an "approximate" policy agrees with the optimal policy may not measure anything like the quality of the approximate policy; even changing one action in the optimal policy may require drastic changes in the rest of the policy for it to retain

Abstract value vs. true policy value	32 state domain	64 state domain	256 state domain
Average error	6.96	3.14	0.99
Standard Deviation	3.28	1.00	0.19
Maximum Error	11.0	4.0	1.0
Predicted Bound	11.0	4.0	1.0
Time required to compute policy	0.34s (0.021%)	1.03s (0.065%)	9.74s (0.61%)
Iterations	5	9	13
true abstract policy value vs. optimal policy			
No. of errors in action to choose	879 42.9%	425 20.8%	136 6.6%
Average error in value of state	8.26 (42.1%)	3.27 (16.7%)	0.22 (1.1%)
Standard Deviation	5.21	1.92	0.21
Maximum Error	19.6	7.39	1.9
Predicted Bound	20.9	7.6	1.9

Table 3: Results of abstraction for the COFFEE domain.

by adopting the abstract policy rather than the optimal policy. As we see, the average loss in value per state is quite good, dropping to 0.22 (with possible optimal values ranging from 22.4 to 42.0) in the finest-grained abstraction, with a maximum error of 1.9. The 32-state and 64-state abstractions also produce reasonable policies, and require trivial amounts of computation time (0.34 and 1.07 seconds, respectively) compared to the 1588 seconds required to solve the full problem.

The second domain, the BUILDER domain, involves an agent that must join two objects together and is adapted from standard job-shop scheduling problems used to test partial-order planners like SNLP [36] and BURIDAN [32]. It is not designed with the ability to construct good abstractions in mind. For maximum reward, the objects must be machined to the correct shape, clean, painted, and joined together. The reward for any given state is simply the sum of the individual rewards for all of these attributes. The state contains nine propositions (512 states) and ten actions (see Appendix A for a description of the problem). Policy iteration on the entire state space required 27.1 seconds and eight iterations. State values range from 0.0 to 20.0. The results of the single possible abstraction are summarized in Table 4. The abstraction was again good, especially considering the small size of the abstraction (only 32 states). The average error in the value of a state is 5.99, which is quite large, spanning 30 per cent of the possible range of optimal values; but the abstract policy re-

reasonable value. However, in this domain the measure is of some interest since many of the same tasks must be performed at different levels of abstraction.

Abstract value vs. true policy value	
Average error in state value	2.69
Standard Deviation	1.86
Maximum Error	6.0
Predicted Bound	6.0
Time required to compute policy	$0.35s = 1.29\%$
Iterations required	4
true abstract policy value vs. optimal policy	
No. of errors in action	$309 = 60.4\%$
Average error	5.99
Standard Deviation	2.16
Maximum Error	10.00
Predicted Bound	11.40

Table 4: Results of abstraction for the BUILDER domain.

quired only about one percent of the computation time required for the optimal policy. For this domain, since the abstract state space is so much smaller than the concrete one, some local way of improving the policy, such as the search procedure described in the next section, may be very valuable. Since there is no abstraction that is more fine-grained than this, we cannot choose another abstraction if the bound on the difference between the abstract and optimal policies of 11.4 is unacceptable.

4 Using Abstract Policies and Value Functions

Many problems may prove amenable to our abstraction procedure, and allow approximately optimal policies to be computed much more quickly than one could construct an optimal policy. Problem characteristics that will give rise to good abstractions include the existence of variables that are irrelevant to the objectives at hand (or only marginally relevant — see Section 5); a multi-attribute utility function in which the various attributes (subgoals) may be achieved or maintained relatively independently; and especially the existence of subgoals whose contributions to the value function are considerably larger than those of other subgoals. However, we expect that for many problems, abstractions of the type described here may not produce abstract policies with acceptable error bounds. In such cases, our abstraction mechanism may still prove useful, for it can be integrated with a number of other planning strategies.

One way to take advantage of the abstraction procedure is to use multiple levels of abstraction to produce an optimal policy in a way analogous to classical abstraction planners: a solution to an abstact problem is used to find the solution to a less abstract problem more efficiently, perhaps proceeding through a hierarchy of more and more fine-grained abstractions. This can prove useful, even when good abstract solutions exist, if an optimal solution is required.

MDP to solve	Initial policy	Time	Iterations
32 state	Greedy	0.34s	5
64 state	Greedy	1.03s	9
	32 state	1.29s	9
256 state	Greedy	9.75s	13
	32 state	10.40s	12
	64 state	8.37s	9
2048 state	Greedy	1588.62s	113
	32 state	1401.72s	81
	64 state	588.23s	20
	256 state	1106.32s	71

Table 5: Abstraction sequences for optimal solution

Another way to exploit the information generated by our abstraction mechanism is to use the abstract value function as a heuristic estimate of the true (long-term) value of individual states. This is an invaluable source of information when planning is viewed as forward or progressive search through the state space. In addition, should real-time constraints force actions to be selected and executed at different intervals, the abstract policy provides a reasonable set of *default reactions*. We discuss each of these uses of the abstraction mechanism in turn.

4.1 Levels of Abstraction

The performance of the policy iteration algorithm tends to be very sensitive to the initial (seed) policy that is used. We can take advantage of this by using the solution for an abstract MDP to seed the application of policy iteration to a more concrete problem, which should then require fewer iterations to solve.[20] The results of performing this experiment in the COFFEE domain described above, compared with using a one-step greedy policy as the initial seed, are given in Table 5. As the table shows, considerable savings can be gained by using a series of abstractions to solve the concrete problem. The fastest way to compute the optimal value for the concrete domain is to compute the optimal value for the 64 state abstraction, and then use that to find the optimal policy. This requires only 37 percent of the computation time of computing the optimal policy directly. Perhaps surprisingly, computing the optimal policy using the 256 state policy is less effective than using the 64 state policy. At present, we have no way of predicting which abstractions to use for the best possible performance. If multiple processors are available, a number of different sequences of abstractions

[20]Similar remarks apply to value iteration, where the initial value estimate adopted, typically the immediate reward function, can have a dramatic impact on convergence. In this case, the abstract value function can be used as the initial estimate. We elaborate on this point in the section on state-space search below.

could be run in parallel, and computation halted as soon as any processor returns an optimal policy. This method guarantees that computing the optimal policy using a series of abstractions will be no worse than computing it directly, should the original problem be included as a (trivial) abstraction sequence.

4.2 Abstract Value Function as a Heuristic Function

Perhaps the most straightforward planning algorithm is forward search through the state space, or *progression planning*. In a decision-theoretic setting such as ours, state space search amounts to the construction of a *decision tree*, familiar from decision analysis [27, 20, 41]. The value of taking an action a at a state s is the (discounted) weighted average of the value of all possible states that may result from a. The action selected for s is that with the highest expected value. Of course, determining the value of the outcome states requires the evaluation of actions that may be taken at those states, and so on, until terminal states are reached, and the values at these leaves are propagated through the tree via the familiar *rollback* procedure.

Unfortunately, discounted infinite-horizon problems do not have terminal states, so the rollback procedure cannot be applied to true terminal values.[21] In this sense, every branch of the actual decision tree is infinite and search becomes more like AI problem-solving or game-tree search: the tree must be cut off at some finite-horizon and an estimated or *heuristic* value must be assigned to the leaves. The only difference from standard game tree search is the existence of *chance* nodes, at which expectations are taken, instead of *adversary* nodes, at which minimum values are backed up. This is the basis of, for instance, Ballard's $\star$-minimax search [1].

We discuss the relationship of our abstraction mechanism and decision tree search below, but we first describe the search mechanism in slightly more detail. The search algorithm constructs a partial decision tree rooted at the current state to determine the best action to perform. The decision tree is built to a fixed depth, and a heuristic function is used to estimate the value of the leaf nodes. Although fixed-depth search is not necessary for the algorithm to function, it allows the use of depth-first search, which tends to perform well in practice. Using breadth-first search (or one of its variations) would make some of the pruning methods we describe below more efficient, but these techniques often require considerable extra book-keeping costs.

Let s and t be states, let β be the discounting factor as before, and let $\mathcal{V}(t)$ be the value of the heuristic function at state t. Then the *estimated* expected utility of action a_i in state s is:

$$U(a_i|s) = \sum_{t \in \mathcal{S}} \Pr(t|a_i, s)V(t)$$

[21] One exception might be when a goal state s, or other absorbing state is reached, for which the optimal value function can be determined analytically as $R(s)/(1-\beta)$.

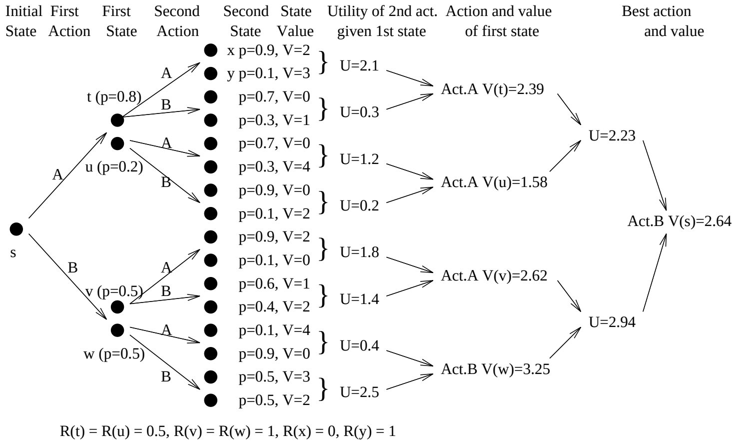

Figure 12: An example of a two-level search for the best action from state s.

where $V(s)$, the value (estimated by the search process) of a state is:

$$V(s) = \begin{cases} \mathcal{V}(s) & \text{if } s \text{ is a leaf node} \\ R(s) + \beta(\max\{U(a_j|s) : a_j \in \mathcal{A}\}) & \text{otherwise} \end{cases}$$

Figure 12 illustrates the search process with a partial tree of actions two levels deep, and a discounting factor β of 0.9. As the figure shows, the value of action A in state s is based on the values of states t and u, which have values of 2.39 and 1.58 respectively (each has A as its best action). The weighted average of these is less than that of action B, so B would be chosen as the best action to perform in state s.

As available computation time varies, the depth of the search tree can also vary. In practice, an interruptible search using an iterative deepening technique may well be used, so that at any time the algorithm can be interrupted and the current best action performed. We cannot guarantee that deeper search will produce better results [40]; however, the deeper the tree is expanded, the more accurate the estimates of action utility will tend to be, and the more confidence we should have that the action selected approaches optimality.

One can view this search process as a form of "directed" value iteration: if one uses the immediate reward function R as the heuristic $\mathcal{V}$, it is easy to see that the computed value $V(s)$ for the state at the root of a tree of depth n is precisely $V^n(s)$ defined in Equation 3 (the optimal n-stage-to-go value for s). This is the basis for the known relationships between heuristic search techniques and stochastic dynamic programming [5, 2]. The advantage of forward search over dynamic programming (at least for short horizon problems) lies in the fact that we need only compute the relevant n-stage value for states reachable from the initial state s (at the appropriate stage n). This can provide a significant advantage should a plan be needed only for a *specific* start state (or small set of start states), as much of the state space may remain unexplored.

Our abstraction mechanism is relevant to off-line decision-tree search for two reasons. First, an abstract MDP can generally be solved off-line relatively quickly, but may not produce a policy whose performance is acceptable. One by-product of this process is the abstract value function that provides an estimate of the value of each state. The values $\widetilde{V}(s)$ can then be used as heuristic estimates for the leaves of the search tree. This is especially important when the connection to value iteration is taken into account. A reasonable heuristic function (or more accurately, a static evaluation function) can cause a dramatic performance increase in a state-space search planner. On this view, our abstraction mechanism can be thought of as a method of automatically generating heuristic functions for decision tree evaluation. In addition, the amount of time spent on the construction of this heuristic and its accuracy can be controlled, to some extent, by adopting an abstraction at a particular level of granularity.

The second advantage of using an abstract value function $\widetilde{V}$ as a heuristic function is that we have considerable knowledge of its range of values and its accuracy. This allows the deployment of several pruning strategies in decision tree construction. *Utility pruning* is very similar to α- and β-cuts in minimax search, and requires knowledge of the maximum and minimum values of the heuristic function. For heuristics produced by the abstraction algorithm described in Section 3, these can be bounded as follows. Define the quantities M^+ and M^- as follows:

$$M^+ = \frac{\max\{R(s) : s \in \mathcal{S}\}}{1-\beta} \quad \text{and} \quad M^- = \frac{\min\{R(s) : s \in \mathcal{S}\}}{1-\beta}$$

These quantities are quickly computable (assuming R is represented compactly) and it is easy to see [5, 45] that $M^- \leq V^*(s) \leq M^+$ for all states s. In addition, the value function $\widetilde{V}^*$ must also satisfy the same relation (since the range of the abstract reward function $\widetilde{R}$ can only be tighter than that of R). For the second type of pruning, *expectation pruning*, we require bounds on the error associated with the heuristic function. Again, for heuristics based on our abstraction algorithm, these bounds can be computed using Lemma 3.8.

Utility Pruning We can prune the search at an AVERAGE step if we know that no matter what the value of the remaining outcomes of this action, we can never exceed the utility of some other action at the preceding MAX step. For example, consider the search tree in Figure 13 (a). We assume that the maximum value that the heuristic function can take is 10. When evaluating action b, since we know that the value of the subtree rooted at T is 5, and the best that the subtrees below U and V could be is 10, the expanded value of action b cannot be larger than 6.5 ($= 5 \times 0.7 + 10 \times 0.3$), so neither of nodes U and V need be expanded. This type of pruning

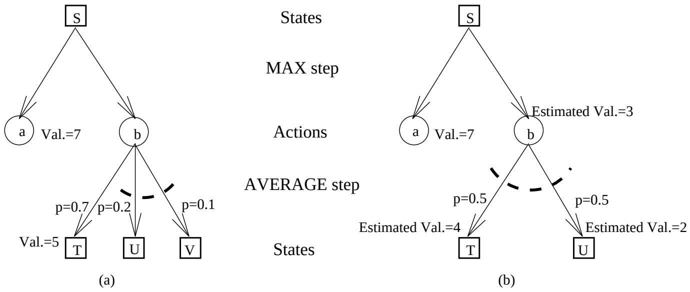

Figure 13: (a) Utility pruning; and (b) Expectation pruning

requires knowledge of the maximum value of the heuristic. We can use the minimum value in a more restricted fashion. If, for example, the value of action a was 3, and the minimum value of the heuristic was 0, then the value of b must be at least 3.5, so we can tell that b is the best action without searching nodes U and V. This process will only work if b is the last action to be evaluated, and we are at the topmost level of the search tree. For the maximum amount of pruning to be performed, the possible action outcomes should be searched in order of their probability of occurrence, with high probability outcomes expanded first.

Expectation Pruning For this type of pruning we need to know the maximum error associated with the heuristic function. Suppose that, at a maximizing node in the search tree, the action we are investigating cannot have as high a utility as some other action for which the utility is already known (even taking into account the error in the heuristic function). Then we do not need to expand this action further. For example, consider Figure 13 (b), where we assume that for all states s, $\mathcal{V}(s)$ is within ± 1 of the true (optimal) value of s. Having determined that $U(a|S) = 7$, we know that any potentially better action must have a heuristic value of at least 6. Since $\Pr(T|b,S)\mathcal{V}(T) + \Pr(U|b,S)\mathcal{V}(U) \leq 4$, b cannot be better than a, so there is no need to search the subtrees below states T and U.

The formal details of the pruning strategies are straightforward and we omit them here. For a depth-first search algorithm, utility pruning is simple to implement as it requires very little extra computation. Expectation pruning requires a more significant modification of the search algorithm to check all the outcomes of an action to see if they require searching, but in domains where the heuristic function is quite accurate, it can still offer a performance improvement, as the results below show. Expectation pruning is quite closely related to what Korf [31] calls alpha-pruning. The difference is that while Korf relies on a property of the heuristic that it is always increasing, we rely on an estimate of the actual error in the heuristic. Using iterative deepening rather than depth-first search seriously limits the applicability of utility pruning since the final value of an action is only known when the last round of deepening is performed. On the other hand, iterative deepening removes the additional computation requirements that make expectation pruning more expensive to perform.

The advantage of planning via search is that the complexity of the algorithm does not depend on the size of the state space. If n is the number of actions, and b is the maximum number of possible outcomes for any action and state, then for an unpruned search tree of depth d, the number of nodes (states) expanded while calculating the best action for a single state is $1+bn+(bn)^2+\ldots+(bn)^d = ((bn)^{d+1}-1)/(bn-1)$. Over a series of such calculations, the cost is slightly less than this because we can reuse previous calculations, but the complexity is $O((bn)^d)$. Thus the size of the state space has no effect on the algorithm; only the number of states visited determines the cost. In many domains this number may be considerably less than the total number of states. More importantly, the complexity of the algorithm is constant (with regard to the number of states), and execution time per action can be bounded for a fixed search depth and branching factor.

We have performed experiments to test the effectiveness of the searching algorithm in several domains. Table 6 summarizes the effects of search for three different problems, where for each problem, search is performed at each state in the MDP.

As the first and third tables show, deeper search generally leads to improved performance. The number of states for which an optimal action has not been found[22] drops steadily as search depth increases, and the state values quickly approach optimal even though a few high-error states remain even after four step search. The results from the BUILDER domain also illustrate an important point; the search procedure doesn't always perform better as search depth increases. The two-step search is better than searching to three or four steps, at least in terms of the average value of a state. More detailed analysis of the policies produced by each depth of search reveals that for almost all states the value of the policy continues to improve as search depth increases, but there are a small number of pathological states for which the search algorithm performs very badly. This phenomenon is well-documented in the search literature [40]. Search in the COFFEE domain using a heuristic derived from the 256 state abstract MDP finds a very close to optimal policy; but even four step search is unable to improve on it since the heuristic is so good.

Figure 14 shows the effects of pruning for both fine- and coarse-grained heuristics in the COFFEE domain. It describes the number of states examined and time required as a percentage of the same values for full (unpruned) search to the same depth. In this case, the fine-grained heuristic was produced from

[22] A non-zero error means that some reachable state has a suboptimal action.

COFFEE domain, heuristic from 64 state abstraction

	Optimal policy	1 step search	2 step search	3 step search	4 step search
Average state value	35.35	32.0	33.13	33.24	34.90
Percentage of optimal	100.0	90.5	93.7	94.0	98.7
Maximum error	0	7.6	5.51	5.22	4.94
Average error	0	3.35	2.22	2.11	0.45
No. of non-zero errors	0	1786	1719	1621	1451
Average non-zero error	0	3.84	2.64	2.67	0.64
Time to search for one state	-	0.8ms	7.6ms	97.2ms	944.8ms

COFFEE domain, heuristic from 256 state abstraction

	Optimal policy	1 step search	2 step search	3 step search	4 step search
Average state value	35.35	35.31	35.34	35.34	35.34
Percentage of optimal	100.0	99.9	99.9	99.9	99.9
Maximum error	0	1.71	0.18	0.18	0.18
Average error	0	0.24	0.18	0.18	0.18
No. of non-zero errors	0	2048	2048	2048	2048
Average non-zero error	0	0.24	0.18	0.18	0.18
Time to search for one state	-	0.8ms	7.6ms	97.2ms	944.8ms

BUILDER domain

	Optimal policy	1 step search	2 step search	3 step search	4 step search
Average state value	18.17	12.23	18.11	18.01	18.02
Percentage of optimal	100.0	67.3	99.7	99.1	99.2
Maximum error	0	10.003	0.702	5.050	5.050
Average error	0	5.947	0.062	0.166	0.152
No. of non-zero errors	0	512	207	141	91
Average non-zero error	0	5.947	0.153	0.602	0.857
Time to search for one state	-	1.4ms	37.8ms	1.19s	31.13s

Table 6: Comparison of induced policies for various search depths for the BUILDER domain.

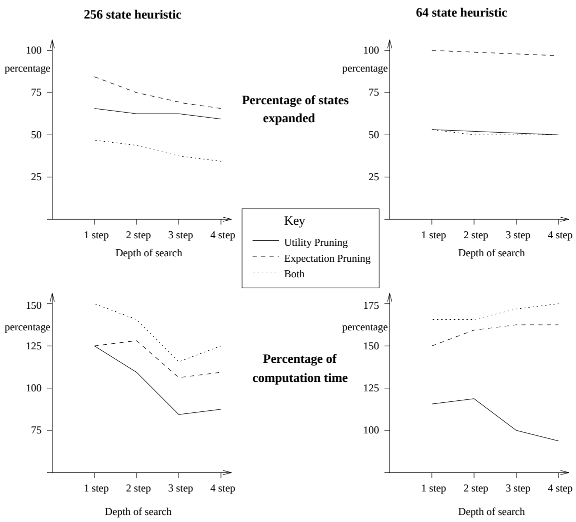

Figure 14: Pruning with (a) 256- and (b) 64- state abstractions

the optimal policy for the 256 state abstraction described in Section 3, while the coarse-grained heuristic was produced from the 64 state abstraction. For the fine-grained heuristic, both pruning algorithms result in a considerable reduction in the number of states searched, while only utility pruning is effective with the coarse-grained heuristic. This is due to the large error associated with the heuristic which largely prevents expectation pruning from being applied. As a general guide, expectation pruning should only be used when the heuristic is reasonably accurate and tight bounds can be placed on the error.

As figure 14 shows, despite the large number of states pruned from the search tree in both domains, there is little or no saving in computation time. This is due to the additional cost of pruning, and suggests that the tree should only be pruned if a sufficiently large subtree will be removed to justify this extra cost. Only allowing pruning close to the root of the tree gives the desired effect, and Table 7 shows the computational savings achieved using this method. For the best results, we suggest allowing pruning to a depth one less than the search tree.

4.3 Integrating Planning and Execution

The type of search described above provides an online, anytime method for planning and action selection. These types of considerations form, for instance, the basis of Korf's [31] RTA* algorithm. Real-time dynamic programming (RTDP)

Search depth	Prune depth	Percentage of states pruned	Percentage of search time
2	1	18.6	88.2
	2	55.9	138.2
3	1	13.1	70.6
	2	26.3	67.2
	3	59.5	112.0
4	1	5.3	101.7
	2	9.5	75.2
	3	24.0	84.3
	4	67.1	140.2

Table 7: Effects of limiting the depth to which pruning is performed. Values are compared with those for no pruning.

[2] generalizes RTA* to deal with MDPs, essentially adapting a form of *asynchronous value iteration* [5] to a real-time setting. The search procedure described above can be viewed as a form of RTDP, where a search tree is used determine which backups are to be performed. The key difference in our algorithm is the existence of a heuristic function that will cause faster convergence of the search and will generally cause better actions to be chosen using the same amount of search.

Time-critical domains provide a minimum of computation time in which to plan, hence it is important to restrict the space to be searched as much as possible. This suggests another advantage to integrating planning and execution in stochastic domains (apart from the real-time aspect). By executing the current best action, the agent resolves any uncertainty about the next state. We note that the presence of a heuristic function with error bounds may cause search to terminate quickly (through pruning), or cause fairly rapid convergence. To this end, the agent should execute each action as soon as it has been selected. This may result in considerable computational savings. Performing an action in a certain state can leave the system in a number of different states, so a planning algorithm that constructs a sequence of actions would need to find an action to perform for each of the possible outcomes of the action it selects first. By executing actions as soon as they are selected, we know (since the MDP is completely observable) which of the possible outcomes actually occurred, and need only search for the next action to perform from a single state rather than from many.

An online search-based planning algorithm can be viewed at the highest level as follows:

1. Calculate the best action for the current state, using the heuristic function as needed.

2. Execute the best action when it is known, or the current estimated best

Search depth	Execution Interleaved		No Execution	
	Caching	No cache	Caching	No cache
1	0.01	0.02	5.19	26.7
2	0.04	0.06	5.41	281
3	0.42	0.51	7.14	2780
4	4.48	5.68	15.4	-
5	55.9	56.9	102	-
6	219	230	272	-

Table 8: Search time for ten actions varying caching and execution

action when required due to time pressure

3. Observe the new state of the system and return to step 1.

Although the algorithm as presented never terminates, this is consistent with the process-like domains for which MDPs are ideally suited. If the domain contains goal states or other terminal (e.g., absorbing) states, the algorithm may terminate when such a state is reached. In general, however, the agent will continue planning and acting indefinitely.

As we would expect, the saving in computation gained by interleaving execution with planning is considerable. Let b be the maximum number of outcomes of any action. In a search for a sequence of n actions, search without execution will require an action to be selected for $\sum_{i=0}^{n-1} b^i$ states compared with only n states for search with execution. We have also performed experiments to investigate the value of caching previously computed best actions (similar to LRTA* [31] or LRTDP [2]), and the value of interleaving execution with search. Table 8 summarizes the results of the search for a sequence of ten actions in a small version of the COFFEE domain. The columns where execution is interleaved with search show the standard algorithm as described above. For search without execution, the agent performs the standard search, determines the best action and then, rather than executing it, searches again to find the best action to perform for all possible outcomes of the action. Unsurprisingly, cached search interleaved with execution is the most efficient method. The size of the domain will have a considerable effect on the value of caching. In this case, the domain contains 256 states but we are only performing ten actions, so caching has relatively little effect. However, if more search is performed and space is available, caching appears to be worthwhile. One surprising aspect of this table is how well the cached search without execution performs. This is due to the small number of states in the example. Because the algorithm caches the best computed action for each state, the interleaved execution algorithm will only cache values for at most ten states. In comparison, if each action has m possible outcomes, the no-execution algorithm can cache actions for up to $m^{10}/2$ states. In practice this means that for a domain of this size, the algorithm quickly finds

Action	*Discriminant*	*Effect*	*Prob.*
DelC	*Office*, *HRC*, *Wet*	*HUC*, $\neg$*HRC*	0.7
		$\neg$*HRC*	0.3
	Office, *HRC*, $\neg$*Wet*	*HUC*, $\neg$*HRC*	0.8
		$\neg$*HRC*	0.2
	$\neg$*Office*, *HRC*	$\neg$*HRC*	1.0
	$\neg$*HRC*		1.0

Figure 15: Action with marginally relevant factors

itself looking for actions for states it has already evaluated. The column for search without caching or execution gives an idea of how badly search without execution and with caching would perform if the state space were large enough to prevent sufficient reuse of cached values.

5 Inexact Abstraction

The method of abstraction presented in Section 3 takes as a starting point those propositions deemed to have the largest impact on immediate reward and then determines the set of atoms that can, under some action choice, influence the truth of these propositions. However, this is a very cautious approach to generating relevant atoms for a given abstraction, since it does not account for the *degree of relevance* of the atoms in question. In particular, an atom that has only a marginal influence on the probability of an immediately relevant proposition (under some action) should be considered less relevant than an atom that completely determines the truth or falsity of that proposition.

A simple variant of our COFFEE example illustrates this point. Imagine that the problem description is exactly as in Figure 1 except that the successful delivery of coffee is influenced slightly by the fact *Wet*: if the robot is wet, there is slightly increased chance (0.3 vs. 0.2) it will drop the coffee (the new *DelC* action is shown in Figure 15 — we ignore the possibility of the nothing happening, a possibility in the original formulation). In the original abstraction of this problem we ignored the impact of the variable *Wet* on immediate reward and generated an abstract MDP based on literals *L*, *HRC* and *HUC*. In this slightly altered problem, our abstraction generation mechanism will now notice that *Wet* is relevant to the achievement of *HUC*; subsequently, the literals *Rain* and *Umb* will be deemed relevant (since they influence *Wet*) and the abstract MDP for this slightly altered problem offers no compression of the state space at all.

Just as we may be willing to ignore small distinctions in immediate reward, we may accept small errors in transition probabilities if it opens up the possibility of a much smaller state space. In this example, the difference in the

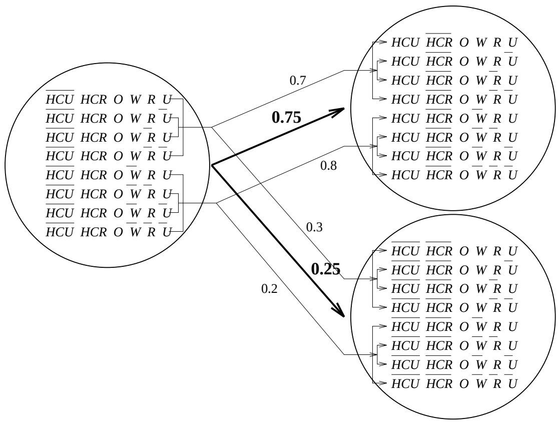

Figure 16: Abstract state space with inexact abstraction (action *DelC*)

probabilities of making *HUC* true when *Wet* is true or false is 0.1. The relative impact of making the distinction *Wet* is roughly 0.08 unit of utility in a one-step decision problem (since the reward for *HUC* is 0.8). If we can accept such an error in the policy, then it makes sense to ignore *Wet* in the abstract MDP. This allows us to also ignore *Rain* and *Umb*, generating a very small MDP (as in the original problem) with a small increase in error.

Certain difficulties arise with this type of *inexact abstraction*. The first is illustrated in Figure 16, which shows the clustering induced by the above considerations over part of the state space, along with the transition probabilities for the altered *DelC* action (in the lighter type). In contrast with exact abstraction (see Figure 7 and Theorem 3.3), we now have states in clusters that do not have identical transition probabilities in the concrete model for a given action. To deal with this, we must assign transition probabilities to these clusters that in some sense "average" the different probabilities associated with the states in that cluster. In the diagram, we have assigned the midpoint probabilities 0.75 and 0.25 (the heavier transition arrows) to the abstract version of *DelC*. A related difficulty is the construction of the abstract action descriptions. In the exact method, if an action discriminant had a single relevant atom, the entire discriminant (i.e., all atoms in the discriminant) was deemed relevant. With inexact abstraction, we may delete specific literals from a discriminant; for example, in the abstract version of *DelC* the atom *Wet* will be deleted, resulting in partial, *non-exclusive* discriminants characterizing the action effects, some of which have contradictory probabilities. We describe an algorithm that deals with both phenomena below.

It is worth pointing out that clustering of this type induces an abstract MDP of a fundamentally different character than those built during exact abstraction: by assigning a single transition probability to cluster $\widetilde{s}$, our predictions are based on less than accurate information. In particular, if we know the prior history of the abstract process (for example, the cluster visited prior to $\widetilde{s}$), we may gain insight into which of the states we are actually at within $\widetilde{s}$. But if we do know this, we can make more accurate predictions about the effects of actions performed in $\widetilde{s}$. In other words, if we keep track of the history of the abstract process, we can generally make more informed decisions. For instance, it may well be that $Pr(\widetilde{t}|a, \widetilde{s})$ is not the same as the probability of moving from $\widetilde{s}$ to $\widetilde{t}$ under action *a given that the process was in cluster $\widetilde{u}$ prior to cluster $\widetilde{s}$.* Therefore, the abstract stochastic process induced by inexact abstraction may not be Markovian. By assigning *history-independent* probabilities $Pr(\widetilde{t}|a, \widetilde{s})$ to clusters in the abstract MDP, we are necessarily losing information relevant to optimal decision making (information that is contained in the abstract MDP itself). However, treating the abstract process as Markovian allows standard, computationally feasible history-independent solution techniques to be used. We simply have to analyze the potential loss in decision quality associated with treating a non-Markovian model as an MDP. We describe techniques for doing so and prove certain error bounds below.

A final difficulty associated with inexact abstraction has to do with choosing relevant atoms. As suggested above, given a particular relevant proposition P, we deem an atom R to be more or less relevant depending on its probabilistic influence on P under some action. Of course, this probabilistic influence must be traded against the relative importance of P. If R has a fairly strong impact on P, we may consider R to be relevant; but we may decide to ignore R if the utility of P is sufficiently small. In contrast, R may make only a small difference in the ability to predict P accurately; but if P is extremely important, R may be judged relevant. Unfortunately, the degree of relevance of P must be quantified in order to make such a decision; and in general the impact of P on immediate reward is not an appropriate measure. Consider the following example, with two actions a_1 and a_2. If a_1 is executed when R, then P becomes true with probability 0.9 (no effect with probability 0.1), and if executed when $\neg R$, P becomes true with probability 0.8. If a_2 is executed when P, then Q becomes true with probability 0.6 and false with probability 0.4; and if executed when $\neg P$, Q becomes true with probability 0.1 and false with probability 0.9. Suppose also that immediate reward depends only on whether Q is true (reward 10) or false (reward 0). Certainly Q will be deemed relevant, and presumably P will be deemed relevant because it has an important impact on the probability of Q. Now when deciding whether to include R among the relevant atoms, we must determine whether the 0.1 difference in the predictability of P permitted by distinguishing R from $\neg R$ is large enough to merit an increase in the abstract state space. This in turn depends on the relative importance of P itself. Note that the impact of P is not a function of its impact on immediate reward (it has none); rather it depends on the atoms *it* influences (in this case Q).

Constructing inexact abstractions requires a method of quantifying the impact of atoms on the value of the optimal decision. This is a difficult issue, exacerbated by the fact that we are dealing with infinite horizon problems. We do not have a wholly satisfactory method for solving this problem, but we do make some suggestions below.[23] We first describe the construction of an abstract MDP and prove certain error bounds. This makes it clear just what factors should be accounted for when assembling the set of relevant atoms.

5.1 Constructing an Abstract MDP

The algorithm for generating a set of relevant atoms in the case of inexact abstraction is similar to the algorithm used for exact abstraction presented in Section 3.1. The only difference is that atoms in discriminants with effects containing relevant atoms are not automatically deemed relevant; rather some criterion is used to determine relevance based on the importance of the atom in the effect list and the "predictive power" of the atom under consideration with respect to the affected atom. We defer discussion of possible criteria to the next sections. We first present an algorithm that constructs a new abstract action from an existing action description assuming the set of relevant atoms is given.

Assume a set $\mathcal{R}$ of relevant atoms has been determined using some method of inexact abstraction and let a be an action of the form

$$\begin{array}{ll} d^1: & E^1_1, p^1_1; E^1_2, p^1_2; \cdots \\ d^2: & E^2_1, p^2_1; E^2_2, p^2_2; \cdots \\ \cdots & \\ d^n: & E^n_1, p^n_1; E^n_2, p^n_2; \cdots \end{array}$$

The algorithm to construct an abstract action corresponding to a proceeds by first deleting irrelevant atoms from the action description and collapsing common effects within discriminants (as with exact abstraction). However, special steps must be taken to combine the possibly different effects of the new partial discriminants that are no longer completely disjoint, or have perhaps become identical. It proceeds in four stages:

1. Delete irrelevant atoms from each d^i, E^i_j (call these the *reduced* discriminants and effects)

2. For each discriminant d^i, collapse reduced effects E^i_j that have become identical into a single reduced effect with probability equal to $\sum p^i_j$ of the participating reduced effects

3. For any two reduced discriminants that are not mutually exclusive (i.e., do not have complementary literals), but that have not become identical,

[23] In particular, we suggest in Section 6.2.1 that *dynamic* aggregation methods will be better suited to this problem.

split (one of) the discriminants so they become exclusive, giving each split discriminant the same effects and probabilities as the original discriminant

4. For each (maximal) set of reduced discriminants $ID = \{d^i, d^j, \cdots\}$ that are identical, collapse into a single discriminant with a unique effect list as follows (for ease of presentation, assume $ID = \{d^1, \cdots d^k\}$):

   (a) Replace the set ID with the single discriminant $\widetilde{d} = d^1$

   (b) For each $d^i \in ID$ and E^i_j associated with d^i, add the effect-probability pair $\langle E^i_j, \frac{p^i_j}{k} \rangle$ to the new effect list for $\widetilde{d}$

   (c) Collapse identical effects E^i_j associated with d in the usual way (summing probabilities – let the probability of any effect in the simplified abstract action be denoted $\widetilde{p}^i_j$)

Steps 1 and 2 of this process proceed exactly as in exact abstraction. Steps 3 and 4 are required because discriminants need not be mutually exclusive. Note that in exact abstraction, two discriminants can become simpler only if all possible effects contain no relevant atoms — no conflicting effects are possible in the abstract space — so Steps 3 and 4 were not necessary.

Step 3 is intended to deal with a situation where an atom is deleted from an action description leaving two non-exclusive discriminants. For instance, imagine an action of the form

$$\begin{aligned} A \wedge B: &\quad e^1, 1.0 \\ A \wedge \neg B: &\quad e^1, 0.95; e^2, 0.05 \\ \neg A: &\quad e^1, 0.9; e^2, 0.1 \end{aligned}$$

where B is deemed relevant for certain reasons, but the difference between effects e^1 and e^2 is not judged important enough to warrant the distinction between A and $\neg A$ (i.e., the probability difference of 0.1 is too small). Deleting A from the action description results in the three non-exclusive discriminants B, $\neg B$ and $\top$. Before combining probabilistic effects, we split discriminants such as $\top$ into two parts B and $\neg B$ so that each pair of discriminants is mutually exclusive or identical, and copy the effect list of the original discriminant into each of its components. This results in the new action description

$$\begin{aligned} B: &\quad e^1, 1.0 \\ \neg B: &\quad e^1, 0.95; e^2, 0.05 \\ B: &\quad e^1, 0.9; e^2, 0.1 \\ \neg B: &\quad e^1, 0.9; e^2, 0.1 \end{aligned}$$

The details of such a procedure are straightforward in the case of discriminants represented as sets of literals, so we do not elaborate here.

Step 4 captures the essence of inexact abstraction, approximating the transition probabilities for states that have become clustered despite having slightly different probabilistic effects on relevant atoms. The effect of this collapsing in the action above would be

$$\begin{array}{rl} B: & e^1, 0.95; e^2, 0.05 \\ \neg B: & e^1, 0.925; e^2, 0.075 \end{array}$$

In the case of the new *DelC* action in Figure 15, the abstract action produced, should atom *Wet* be judged irrelevant, is given by:

$$\begin{array}{rl} \mathit{Office}, \mathit{HRC}: & \mathit{HUC}, \neg \mathit{HRC}, 0.75; \neg \mathit{HRC}, 0.25 \\ \neg \mathit{Office}, \mathit{HRC}: & \neg \mathit{HRC}, 1.0 \\ \neg \mathit{HRC}: & \emptyset, 1.0 \end{array}$$

We note that the particular procedure described in Step 4 for "blurring" probabilities is adopted primarily for convenience. In general, any procedure can be used to assign probabilities to effects, as long as the effect probabilities sum to one for the combined discriminant. This particularly simple approach has this property and works well in cases where the same effects occur in the elements of *ID*, just with different probabilities. In such a case, the effects are assigned the average probability. Approximate midpoints might also be assigned to each effect, so long as care is taken to ensure the effect probabilities sum to one (e.g., a sophisticated minimization procedure might be adopted). We leave open the possibility of more sophisticated but computationally demanding blurring techniques. In general, we want to minimize the difference between the new assigned probability p and the true probability of the effect under any of the original discriminants. More precisely, as we will see below, the errors in transition probabilities within a single discriminant can *accumulate* to produce errors in the computation of value and in action selection. Thus, we want to ensure that the total error is kept small. We will assume below (roughly) that the accumulated error in the new transition probabilities for any action discriminant is bounded by some factor ρ: that is, for any discriminant d^i, $\sum_j |\widetilde{p}^i_j - p^i_j| \leq \rho$. Clearly, this factor will influence which atoms are actually deleted from the action description.

We note without proof the following rather obvious properties of new abstract actions constructed in this way:

Proposition 5.1 *Let action $\widetilde{a}$ be an abstract action*

$$\begin{array}{rl} \widetilde{d}^1: & E^1_1, \widetilde{p}^1_1; E^1_2, \widetilde{p}^1_2; \cdots \\ \widetilde{d}^2: & E^2_1, \widetilde{p}^2_1; E^2_2, \widetilde{p}^2_2; \cdots \\ \cdots & \\ \widetilde{d}^n: & E^n_1, \widetilde{p}^n_1; E^n_2, \widetilde{p}^n_2; \cdots \end{array}$$

constructed from a concrete action a by inexact abstraction as described above. If a is well-formed (i.e., has mutually exclusive, exhaustive discriminants and probabilities that sum to one for each discriminant), then

(a) *The set of discriminants* $\{\widetilde{d}^1, \cdots \widetilde{d}^n\}$ *is mutually exclusive and exhaustive.*

(b) *For each* j, $\sum_i \{\widetilde{p}_i^j\} = 1$.

5.2 Error Bounds for Inexact Abstraction

Before considering means by which to construct the set of relevant atoms, it is instructive to determine error bounds for inexact abstraction and the features of the abstraction that affect solution quality. To begin, we consider the error associated with determining the value of the one-stage policy *DelC* at the initial cluster in the inexact abstraction depicted in Figure 16. The abstract (zero-stage) value of the two clusters $\widetilde{t}$ and $\widetilde{u}$ is simply the abstract reward function — $\widetilde{V}^0(\widetilde{t}) = 0.9$ and $\widetilde{V}^0(\widetilde{u}) = 0.1$. The error introduced in the abstract one-stage value function $\widetilde{V}^1(\widetilde{s})$ by blurring the immediate reward function is exactly as characterized in exact abstraction. However, the fact that the transition probabilities used to derive the abstract value function are imprecise introduces further error in the estimated value. The true value of the policy *DelC* at state s is a function of the transition probabilities 0.8 and 0.2, whereas the abstract value function adopts probabilities 0.75 and 0.25 in its calculation. Ignoring the errors in the reward function, a simple calculation reveals that

$$|\widetilde{V}^1(\widetilde{s}) - V^1(s)| \leq \beta \cdot 0.05(V^0(t) - V^0(u))$$

Here 0.05 is the error in the probability estimates for each of the effects associated with action *DelC*, while intuitively the quantity $V^0(t) - V^0(u)$ is the difference in (zero-stage) value for the states reachable from s. We could also replace this part of the term by $\widetilde{V}^0(\widetilde{t}) - \widetilde{V}^0(\widetilde{u})$, which suggests that the possible values of the different reachable clusters may be used to bound error as well.

To determine the possible error in the abstract value function $\widetilde{V}^i$ introduced by inexact abstraction, we must know the possible ranges of values (at least for reachable clusters) in the function $\widetilde{V}^{i-1}$. Not surprisingly, knowing the errors in the abstract transition probabilities is not enough. The difficulty is that the error bound cannot be computed directly using the local information in the problem specification (such as the reward function). For an infinite horizon problem, we must have estimates of the (optimal) value function before we can compute the error bounds for possible inexact abstractions. However, the optimal value function is one of the things we are trying to determine by solving the abstract MDP. This circularity makes generating inexact abstractions with (*a priori*) known error bounds very difficult. However, we can prove error bounds based on certain knowledge of the abstract (or concrete) value function; this suggests certain crude methods for bounding the error of inexact abstraction, which we address in the next section.

As noted, the error in the value estimate of an abstract policy depends on both the accumulated error in transition probabilities and the value differences in the reachable clusters for which inexact probabilities are used. We assume that, for a specific action a and discriminant d associated with a in its inexact abstraction, the total error introduced in the probability of any effect associated with that discriminant is $\rho_{d,a}$; that is, $\sum_j |\widetilde{p}_j - p_j| \leq \rho_{d,a}$ for all effects E_j in the effect list for d. Characterized in terms of state transitions, we assume the inexact abstraction is such that, for any $\widetilde{s}$ satisfying discriminant d of action a and $s \in \widetilde{s}$:

$$\sum_{\widetilde{t} \in \widetilde{\mathcal{S}}} |\Pr(\widetilde{t}|a, \widetilde{s}) - \sum_{t \in \widetilde{t}} \Pr(t|a, s)| \leq \rho_{d,a}$$

Of course, even small errors in prediction can be disastrous if they have important consequences. Thus, we also assume that we have certain information about the value function for any blurred transition probability. Let $\widetilde{s}$ be any abstract state satisfying discriminant d. We assume that

$$[\max\{\widetilde{V}(\widetilde{t}) : \Pr(\widetilde{t}|a, \widetilde{s}) > 0\} - \min\{\widetilde{V}(\widetilde{t}) : \Pr(\widetilde{t}|a, \widetilde{s}) > 0\}] \cdot \rho_{d,a} \leq \Delta$$

In other words, for any transition probability error introduced, the values of the clusters to which that error applies lie within a range r such that $r \cdot \rho_{d,a} \leq \Delta$. If we assume that an inexact abstraction is created such that, whenever a transition probability is approximated within a certain discriminant, this condition is observed, then we can bound the errors introduced by inexact abstraction. As usual, we take β to be the discounting factor and δ to be the maximum utility span for the (inexact) abstract MDP.

Theorem 5.2 *For any $s \in \widetilde{s}$,*

$$|\widetilde{V}_{\widetilde{\pi}}(\widetilde{s}) - V_\pi(s)| \leq \frac{\delta + \beta\Delta}{2(1-\beta)}$$

Proof This result is proved in an inductive fashion similar to the proof of Theorem 3.7. The crucial difference in the proof lies in the inductive step where the abstract and concrete probabilities may differ. The inductive hypothesis is

$$|\widetilde{V}^k_{\widetilde{\pi}}(\widetilde{t}) - V^k_\pi(t)| \quad \leq \quad \sum_{i=0}^{k} \frac{\delta}{2}\beta^i + \sum_{i=1}^{k} \frac{\Delta}{2}\beta^i$$

which clearly holds for $k = 0$ (where the error is $\frac{\delta}{2}$ as in the case of exact abstraction). The induction step proceeds as follows (we assume $\widetilde{s}$ satisfies discriminant d of action $\widetilde{\pi}(\widetilde{s})$).

$$\left|\widetilde{V}^{k+1}_{\widetilde{\pi}}(\widetilde{s}) - V^{k+1}_\pi(s)\right| \quad = \quad \left|\left[\widetilde{R}(\widetilde{s}) + \beta \sum_{\widetilde{t} \in \widetilde{\mathcal{S}}} \Pr(\widetilde{t}|\widetilde{\pi}(\widetilde{s}), \widetilde{s})\widetilde{V}^k_{\widetilde{\pi}}(\widetilde{t})\right] - \left[R(s) + \beta \sum_{t \in \mathcal{S}} \Pr(t|\pi(s), s)V^k_\pi(t)\right]\right|$$

$$\leq \quad \left| \widetilde{R}(\widetilde{s}) - R(s) \right| + \beta \left| \sum_{\widetilde{t} \in \widetilde{\mathcal{S}}} \left[\Pr(\widetilde{t} | \widetilde{\pi}(\widetilde{s}), \widetilde{s}) \widetilde{V}^k_{\widetilde{\pi}}(\widetilde{t}) - \sum_{t \in \widetilde{t}} \Pr(t | \pi(s), s) V^k_\pi(t) \right] \right|$$

Now we have

$$\left| \sum_{\widetilde{t} \in \widetilde{\mathcal{S}}} \left[\Pr(\widetilde{t} | \widetilde{\pi}(\widetilde{s}), \widetilde{s}) \widetilde{V}^k_{\widetilde{\pi}}(\widetilde{t}) - \sum_{t \in \widetilde{t}} \Pr(t | \pi(s), s) V^k_\pi(t) \right] \right|$$

$$\leq \quad \left| \sum_{\widetilde{t} \in \widetilde{\mathcal{S}}} \left[\Pr(\widetilde{t} | \widetilde{\pi}(\widetilde{s}), \widetilde{s}) \widetilde{V}^k_{\widetilde{\pi}}(\widetilde{t}) - \sum_{t \in \widetilde{t}} \Pr(t | \pi(s), s) \widetilde{V}^k_\pi(\widetilde{t}) \right] \right| + \left| \sum_{\widetilde{t} \in \widetilde{\mathcal{S}}} \sum_{t \in \widetilde{t}} \Pr(t | \pi(s), s) \left(\widetilde{V}^k_\pi(\widetilde{t}) - V^k_\pi(t) \right) \right|$$

$$\leq \quad \left| \sum_{\widetilde{t} \in \widetilde{\mathcal{S}}} \left(\Pr(\widetilde{t} | \widetilde{\pi}(\widetilde{s}), \widetilde{s}) - \sum_{t \in \widetilde{t}} \Pr(t | \pi(s), s) \right) \widetilde{V}^k_\pi(\widetilde{t}) \right| + \sum_{i=0}^{k} \frac{\delta}{2} \beta^i + \sum_{i=1}^{k} \frac{\Delta}{2} \beta^i$$

We know that, for $s \in \widetilde{s}$, the following relationships hold:

$$\left| \Pr(\widetilde{t} | \widetilde{\pi}(\widetilde{s}), \widetilde{s}) - \sum_{t \in \widetilde{t}} \Pr(t | \pi(s), s)) \right| \leq \rho_{d, \widetilde{\pi}(\widetilde{s})},$$

$$\sum_{\widetilde{t} \in \widetilde{\mathcal{S}}} \Pr(\widetilde{t} | \widetilde{\pi}(\widetilde{s}), \widetilde{s}) = 1 \quad \text{and} \quad \sum_{\widetilde{t} \in \widetilde{\mathcal{S}}} \sum_{t \in \widetilde{t}} \Pr(t | \pi(s), s) = 1$$

Each summand in the term

$$\left| \sum_{\widetilde{t} \in \widetilde{\mathcal{S}}} \left(\Pr(\widetilde{t} | \widetilde{\pi}(\widetilde{s}), \widetilde{s}) - \sum_{t \in \widetilde{t}} \Pr(t | \pi(s), s) \right) \widetilde{V}^k_\pi(\widetilde{t}) \right| \tag{4}$$

has the form $(p - p')V$ where the ps sum to 1, and the p's sum to 1, hence

$$\sum_{\widetilde{t} \in \widetilde{\mathcal{S}}} \left(\Pr(\widetilde{t} | \widetilde{\pi}(\widetilde{s}), \widetilde{s}) - \sum_{t \in \widetilde{t}} \Pr(t | \pi(s), s) \right) = 0$$

That is, the sum of the transition probability errors is zero. Therefore Equation 4 takes its maximum value when the positive half of the error in probability $\rho_{d, \widetilde{\pi}(\widetilde{s})}/2$ occurs at extreme values of $\widetilde{V}^k$ (either its maximum or minimum), and the negative half of the error occurs at the other extreme. Thus we have

$$\left| \sum_{\widetilde{t} \in \widetilde{\mathcal{S}}} \left(\Pr(\widetilde{t} | \widetilde{\pi}(\widetilde{s}), \widetilde{s}) - \sum_{t \in \widetilde{t}} \Pr(t | \pi(s), s) \right) \widetilde{V}^k_\pi(\widetilde{t}) \right| \leq$$

$$\frac{\rho_{d, \widetilde{\pi}(\widetilde{s})}}{2} \left[\max\{\widetilde{V}^k(\widetilde{t}) : \Pr(\widetilde{t} | \widetilde{\pi}(\widetilde{s}), \widetilde{s}) > 0\} - \min\{\widetilde{V}^k(\widetilde{t}) : \Pr(\widetilde{t} | \widetilde{\pi}(\widetilde{s}), \widetilde{s}) > 0\} \right]$$

By construction, the l.h.s. of this last inequality is bounded by $\Delta/2$. Putting these components together we get

$$\begin{aligned}\left|\widetilde{V}_{\widetilde{\pi}}^{k+1}(\widetilde{s}) - V_{\pi}^{k+1}(s)\right| &\leq \frac{\delta}{2} + \beta\left[\frac{\Delta}{2} + \sum_{i=0}^{k}\frac{\delta}{2}\beta^i + \sum_{i=1}^{k}\frac{\Delta}{2}\beta^i\right] \\ &\leq \sum_{i=0}^{k+1}\frac{\delta}{2}\beta^i + \sum_{i=1}^{k+1}\frac{\Delta}{2}\beta^i \\ &\leq \sum_{i=0}^{k+1}\frac{\delta+\beta\Delta}{2}\beta^i\end{aligned}$$

Taking this in the limit yields the result. ■

Thus, by introducing small errors in the abstract transition function, we introduce additional errors in the computed value function. However, as in the case of errors in the abstract reward function, the contribution of this error to the overall error in the value is additive, introducing additional error of at most $\Delta/2$ at each stage of the process. We note that similar bounds can be derived using the concrete value function in the definition of Δ instead of the abstract value function $\widetilde{V}$.

Similar considerations apply to the loss in solution quality introduced by adopting an inexact abstraction. Taking π to be the concrete policy induced by solution of the abstract MDP, we have:

Theorem 5.3 *For any $s \in \mathcal{S}$*

$$|V^*(s) - V_\pi(s)| \leq \frac{\beta(\delta+\Delta)}{1-\beta}$$

Proof We omit details. The proof proceeds exactly like that for Theorem 3.7. In particular, the proof of Lemma 3.8 can be adapted using considerations identical to those described in the proof of Theorem 5.2. ■

5.3 Determining Relevant Atoms

As mentioned above, the difficulty associated with inexact abstraction is that one is required to know the value of a particular state or cluster before being able to determine the degree of relevance of any atom when considering it for deletion from the problem description. Naturally, this value will not generally be known since it is usually determined by the generation of an optimal policy (whose computation *will* be based on the abstraction we generate). However, there are methods one can use to estimate or bound the value function in a way that can be applied to this problem.

The simplest mechanism for bounding the error is to use the maximum and minimum rewards to bound the maximum and minimum values of any state which can then be plugged into the formulae above. The quantities M^+ and M^-, defined in Section 4, can thus be used to bound the error introduced (in the definition of Δ) and can thus be used to decide when small probabilistic influences should be ignored in the generation of the set of relevant atoms. In particular, suppose that the error in transition probability for any abstract action at any state is bounded by a term ρ. The following easily computable error bound then holds:

Proposition 5.4 *Let $\rho_{d,a} \leq \rho$ for all actions a and discriminants d. Then for any $s \in \mathcal{S}$*

$$|V^*(s) - V_\pi(s)| \leq \frac{\beta(\delta + \rho(M^+ - M^-))}{1-\beta}$$

This fact can be used in the generation of the relevant set whenever a small probabilistic distinction is to be ignored. If the collapsing of a set of discriminants for a given action introduces an error in transition probability ρ such that the error term above is acceptable, then the deletion of the distinction can be made. In essence, considerations of this type introduce a threshold in transition probability error that is simple from both a conceptual and implementational standpoint.

This simple and loose error bound induces a strategy for deleting marginally relevant atoms that depends *solely* on the difference in probabilities of an action's effects, not on the relative value of the effects. While easy to implement, this may provide only crude estimates of degrees of relevance and will tend to be extremely cautious (ignoring only small probability errors). In general, larger errors in transition probability will be acceptable if the value of the target clusters is reasonably close. In order to track this, one could augment the algorithm for determining relevant atoms with additional machinery to determine degree of relevance. This would be able to take into account the influence of atoms on other relevant atoms by considering the degree to which they affect the control of the relevant atoms using particular actions; it can also account for the "distance" of this influence. If the impact of a particular atom A on the value of an immediately relevant atom B is removed through a sequence of n actions, then the relative importance of A can be scaled down by a factor β^n to reflect this "effect at a distance".

A backchaining algorithm similar to the one described in Section 3 could be adapted in this way. Unfortunately, to determine degrees of relevance with any accuracy will no longer be an operation linear in the problem description: the more accurate these values must be, the more such backchaining must implement the steps of the dynamic programming solution algorithms. We therefore do not present any algorithm for estimating the relative importance of atoms for use by our inexact abstraction mechanism. We feel that the appropriate manner in which to deal with these considerations requires the integration of the abstraction mechanism with dynamic programming solution methods. In particular, to deal with these problems we suggest that *adaptive* abstraction mechanisms must be adopted. We have begun investigations of such techniques in [10]; we elaborate further in Section 6.2.1.

6 Concluding Remarks

6.1 Summary

We have argued that Markov decision processes provide a useful foundation for understanding decision theoretic planning, and that computational tools for optimal policy construction may be adapted for DTP. In particular, we have shown that AI representational techniques allow the compact and natural specification of DTP problems as MDPs, and that the regularities and independencies made explicit by the representation can be exploited to develop abstractions or aggregations that allow approximately optimal policies to be developed with greatly reduced computation time in appropriate domains. This aspect of our work also adopts techniques from classical AI planning, in particular, work on the generation of abstraction hierarchies. Finally, we have shown several ways in which the solutions to abstract MDPs can be used and locally improved in both online and off-line models of plan construction.

The keys to our approach to abstraction are the fact that abstractions can be generated quickly (in time roughly linear in the size of the problem description rather than in time that grows with the state space), and the fact that one can determine upper bounds on the loss in solution quality associated with a given abstraction (also very quickly). In addition, the trade-offs between abstractions and their quality can be characterized in terms related to the notion of value of information.

We consider this work to represent some first steps toward the development of practical and theoretically-sound solution methods based on the use of intensional, AI-style representations of decision and planning problems. However, we do not claim that the particular model adopted here will prove useful in all settings — it seems most appropriate when there are objectives with additive and independent value, some of which are more important than others, and for which some features of the domain are only (or primarily) relevant in the achievement of less important objectives. Combined with local improvement methods (such as search) or global improvement methods (such as abstraction hierarchies), our abstraction model provides faster and potentially very tractable (exact or approximate) solution methods for such domains.

There are two key weaknesses in this model. The first is the difficulty in providing tight, *a priori* error bounds on inexact abstractions. The second is the fact that the aggregation method used requires the prior, uniform deletion of literals from the problem description. Thus, the aggregation is *fixed* and *uniform*. Other aggregation methods will prove more useful (as we describe below). However, this work provides the conceptual foundation and techniques for proving error bounds for other abstraction methods based on intensional representation.

6.2 Future Directions

6.2.1 Other Aggregation Methods

There are a number of very interesting directions in which this work can be extended, some of which we are currently exploring. One of the most promising avenues appears to be the use of more general forms of aggregation. While our method of aggregation, exploiting intensional problem representations, is novel, the notion of aggregation of states to solve MDPs has been explored previously [4, 52]. For instance, Bertsekas and Castanon [4] propose an adaptive aggregation method that allows one to group together states in the evaluation phase of policy iteration such that the value produced for any cluster of states approximates the value for each constituent state. Unfortunately, with this method one typically must examine properties of individual states to determine an appropriate aggregation (thus, this does not preclude explicit enumeration of the state space). However, dynamic aggregation using structured representations to approximate the solution of MDPs should prove extremely valuable.

To elaborate, we can roughly classify aggregation methods along three dimensions (among others): *adaptivity*, *uniformity* and *accuracy*. Aggregations can be either *dynamic* (*adaptive*) or *fixed*, referring to whether or not the clustering of states can change according the state of the computation of a solution. They can also be *uniform* or *nonuniform*, depending on whether the distinctions used to partition the state space are identical everywhere. Finally, they can be *exact* or *approximate*, where by exact aggregation we refer to a clustering in which the states within any particular cluster are known to have the *same* value or best action, in contrast to an approximate aggregation where these states may share similar but not identical values.

Our abstraction procedure is a fixed, uniform and approximate aggregation method. The first two characteristics are drawbacks in may cases. For example, suppose a reward function describes two objectives o_1 and o_2 such that o_1 is somewhat more important that o_2, but o_2 is important enough to merit consideration. In the course of developing a policy, a planner may notice that the achievement of both o_1 and o_2 is impossible and that an optimal policy ignores o_2 completely in favor of o_1. It often turns out that the irrelevance of o_2 can be detected early in the development of an optimal policy [10]. More generally, the relevance of literals can vary dramatically with the policy adopted. This means that dynamic rather than fixed aggregations should be adopted: one should often allow the aggregation to vary with the current policy in policy iteration.

The uniform deletion of literals from the problem description may also be inappropriate: if an agent receives a reward for ensuring B whenever A is true, but reward is independent of B when A is false, then a uniform aggregation scheme requires either than B be deleted everywhere (thus ignoring the possible reward difference when A is true), or the distinction on dimension B be made everywhere (although the distinction is irrelevant where A is false). Intuitively, relevance is a *conditional* notion: a literal may be relevant in certain circumstances and irrelevant in others. In this example, the state space should be aggregated into three clusters corresponding to the propositions $\neg A$, $A \wedge B$ and $A \wedge \neg B$. In contrast to a *uniform* aggregation, where clusters are of the same "size" and make the same distinctions, a nonuniform aggregation would be appropriate here.

We have begun explorations of the use of intensional representations for creating dynamic, nonuniform aggregation techniques for solving MDPs. In [10] we describe an algorithm in which a decision-tree representation is used to represent value functions and policies so that regularities in these functions can be exploited and the functions themselves can be represented compactly. It is also possible to apply of the approximation methods developed in this paper to such dynamic, nonuniform methods [9].

An advantage of an adaptive scheme like that described in [10, 9] is that the problem of estimating the impact of ignoring marginally relevant atoms (the bottleneck in Section 5) is obviated. Because the aggregation being used is reconstructed at each step of the computation, evaluating the impact of ignoring one of these distinctions is essentially a local operation whose error bounds are locally computable. We need not determine the global impact of such a decision since the decision may be re-evaluated at a later stage of computation. Our expectation is that combining the ideas from this paper with other aggregation methods will result in very robust and tractable dynamic, nonuniform, approximate abstraction mechanisms.

Aggregation methods and function approximation have also been studied to a large extent in the reinforcement learning community, albeit not usually based on intensional problem descriptions, and particular ideas in that work can also play a crucial role in determining good abstractions [37, 14, 53]. It is important to point out that many aggregation algorithms are useful for dealing with metric state spaces, such as robot navigation domains, where states can be clustered according to their distance from each other. For example, a grid world may be broken into geographic regions such that the construction of a policy may be computed separately for each region or (less commonly addressed) the same action can be performed with good results at each state within a region. Such clusterings cannot be developed within our approach — close locations on a map do not share properties of the type we exploit in our method. In particular, if locations are described by x, y-coordinates, then no two locations share useful properties (having the same x-dimension is not likely to be meaningful for abstraction). Of course, such approaches cannot deal with regularity in "propositional structure".

In addition, good theories of action aggregation may prove useful. To some extent, our abstraction mechanism does aggregate actions, either by collapsing "branches" when discriminants within an action become the same, or (implicitly) combining actions themselves. However, this action simplification is driven by abstraction in the state space. Considerations unique to actions themselves may also be applied (see, e.g., [22] where hand-crafted action abstractions are analyzed), but the automatic aggregation of (components of) action descriptions remains largely unexplored.

6.2.2 Other Directions

Other directions in which this work can be extended include the development of problem-specific abstraction mechanisms. For instance, if one is generating a policy for repeated use over finite-horizon problems, it may be possible to take advantage of knowledge of the typical starting states (e.g., in the form of a distribution over the state space) and exploit this in constructing a single abstraction that gives good average-case performance. Another way of exploiting known starting states is the adoption of envelope methods [15, 56], where (likely) reachability from the start state is used to cluster states into a set of IN states and OUT states. Dynamic programming is restricted to the IN states, and if a transition leads out of the envelope, a estimated value of "being OUT" is used to determine the value of related IN states. Our abstraction techniques can be used by such a method, for example, to construct the estimate for being OUT (perhaps refining the set of OUT states into regions) and provide initial estimates for policy construction over the IN states. The combination of complementary approximation techniques should prove fruitful.

Finally, methods such as these must be extended to partially-observable settings if they are to be applied to general DTP problems. The computational difficulty of solving POMDPs optimally is well-documented, so the use of approximation becomes crucial. Some methods based on function approximation are described in [39, 33]; and preliminary investigations of the use of intensional representations to determine dynamic, nonuniform, exact aggregations are described in [11].

References

[1] Bruce W. Ballard. The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, 21:327–350, 1983.

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

[3] Richard E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, 1957.

[4] D. P. Bertsekas and D. A. Castanon. Adaptive aggregation for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, 34:589–598, 1989.

[5] Dimitri P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, 1987.

[6] Mark Boddy and Thomas L. Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 979–984, Detroit, 1989.

[7] Mark Boddy and Thomas L. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67:245–285, 1994.

[8] Craig Boutilier, Thomas Dean, and Steve Hanks. Planning under uncertainty: Structural assumptions and computational leverage. In *Proceedings of the Third European Workshop on Planning*, Assisi, Italy, 1995.

[9] Craig Boutilier and Richard Dearden. Approximating value trees in structured dynamic programming. In *Proceedings of the Thirteenth International Conference on Machine Learning*, Bari, Italy, 1996. to appear.

[10] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1111, Montreal, 1995.

[11] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, 1996. to appear.

[12] Craig Boutilier and Martin L. Puterman. Process-oriented planning and average-reward optimality. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1096–1103, Montreal, 1995.

[13] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1023–1028, Seattle, 1994.

[14] David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 726–731, Sydney, 1991.

[15] Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 574–579, Washington, D.C., 1993.

[16] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[17] Denise Draper, Steve Hanks, and Daniel Weld. A probabilistic model of action for least-commitment planning with information gathering. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 178–186, Seattle, 1994.

[18] Mark Drummond, Keith Swanson, John Bresina, and Richard Levinson. Reaction-first search. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1408–1414, Chambery, 1993.

[19] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[20] Simon French. *Decision Theory.* Halsted Press, New York, 1986.

[21] Dan Geiger and David Heckerman. Advances in probabilistic reasoning. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 118–126, Los Angeles, 1991.

[22] Peter Haddawy and Anhai Doan. Abstracting probabilistic actions. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 270–277, Seattle, 1994.

[23] Eric J. Horvitz. Computation and action under bounded resources. Technical Report KSL-90-76, Stanford University, Stanford, December 1990. Ph.D. thesis.

[24] Eric J. Horvitz and Adrian C. Klein. Utility-based abstraction and categorization. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pages 128–135, Washington, D.C., 1993.

[25] Ronald A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, 1960.

[26] Ronald A. Howard. *Dynamic Probabilistic Systems.* Wiley, New York, 1971.

[27] Ronald A. Howard and James E. Matheson, editors. *Readings on the Principles and Applications of Decision Analysis.* Strategic Decision Group, Menlo Park, CA, 1984.

[28] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs.* Wiley, New York, 1978.

[29] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.

[30] Craig A. Knoblock, Josh D. Tenenberg, and Qiang Yang. Characterizing abstraction hierarchies for planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 692–697, Anaheim, 1991.

[31] Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[32] Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic least-commitment planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1073–1078, Seattle, 1994.

[33] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370, Lake Tahoe, 1995.

[34] Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 394–402, Montreal, 1995.

[35] William S. Lovejoy. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28:47–66, 1991.

[36] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634–639, Anaheim, 1991.

[37] Andrew W. Moore and Christopher G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces. *Machine Learning*, 1995. To appear.

[38] Ann E. Nicholson and Leslie Pack Kaelbling. Toward approximate planning in very large stochastic domains. In *AAAI Spring Symposium on Decision Theoretic Planning*, pages 190–196, Stanford, 1994.

[39] Ronald Parr and Stuart Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1088–1094, Montreal, 1995.

[40] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA, 1984.

[41] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, 1988.

[42] Edwin Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332, Toronto, 1989.

[43] Mark A. Peot and David E. Smith. Conditional nonlinear planning. In *Proceedings of the First International Conference on AI Planning Systems*, pages 189–197, College Park, MD, 1992.

[44] David Poole. Exploiting the rule structure for decision making within the independent choice logic. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 454–463, Montreal, 1995.

[45] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.

[46] Martin L. Puterman and M.C. Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24:1127–1137, 1978.

[47] Stuart J. Russell and Eric Wefald. *Do the Right Thing: Studies in Limited Rationality*. MIT Press, Cambridge, 1991.

[48] Stuart J. Russell and Shlomo Zilberstein. Composing real-time systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 212–217, Sydney, 1991.

[49] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[50] Earl D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 206–214, 1975.

[51] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1039–1046, Milan, 1987.

[52] Paul L. Schweitzer, Martin L. Puterman, and Kyle W. Kindle. Iterative aggregation-disaggregation procedures for discounted semi-Markov reward processes. *Operations Research*, 33:589–605, 1985.

[53] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Reinforcement learning with soft state aggregation. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 7*. Morgan-Kaufmann, San Mateo, 1994.

[54] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[55] David E. Smith and Mark A. Peot. Postponing threats in partial-order planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 500–506, Washington, D.C., 1993.

[56] Jonathan Tash and Stuart Russell. Control strategies for a stochastic planner. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1079–1085, Seattle, 1994.

Acknowledgements

This work has been aided substantially by our discussions with a number of people. Our initial investigations into abstraction arose from conversation with Moisés Goldszmidt. Thanks also to Tom Dean, Eric Horvitz, Keiji Kanazawa, Daphne Koller, Ann Nicholson, David Poole, Marty Puterman, Stuart Russell and Jonathan Tash for discussions and pointers to relevant work. We are grateful to the anonymous referees for their very detailed comments. This research was supported by NSERC Research Grant OGP0121843 and IRIS Phase II Project IC-7.

A Example Problem Descriptions

The 2048-State COFFEE Domain

Action	Discriminant	Effect	Prob.
MoveLeft	Loc = Off	Loc = Lab	0.9
		$\emptyset$	0.1
	Loc = Lab	Loc = Shop	0.9
		$\emptyset$	0.1
	Loc = Shop	Loc = Mail	0.9
		$\emptyset$	0.1
	Loc = Mail	Loc = Off	0.9
		$\emptyset$	0.1
MoveLeft	R,$\neg$U	W	0.9
		$\emptyset$	0.1
	R,U	$\emptyset$	1.0
	$\neg$R	$\emptyset$	1.0
MoveRight	Loc = Off	Loc = Mail	0.9
		$\emptyset$	0.1
	Loc = Lab	Loc = Off	0.9
		$\emptyset$	0.1
	Loc = Shop	Loc = Lab	0.9
		$\emptyset$	0.1
	Loc = Mail	Loc = Shop	0.9
		$\emptyset$	0.1
MoveRight	R,$\neg$U	W	0.9
		$\emptyset$	0.1
	R,U	$\emptyset$	1.0
	$\neg$R	$\emptyset$	1.0
Deliver	Loc = Off,RhC	$\neg$RhC,UhC	0.8
		$\neg$RhC	0.1
		$\emptyset$	0.1
	Loc = Off,$\neg$RhC,RhB	$\neg$RhB,UhB	0.8
		$\neg$RhB	0.1
		$\emptyset$	0.1
	Loc = Off,$\neg$RhC,$\neg$Rhb	$\emptyset$	1.0
	Loc = Lab	$\emptyset$	1.0
	Loc = Shop	$\emptyset$	1.0
	Loc = Mail	$\emptyset$	1.0

Action	Discriminant	Effect	Prob.
BuyCoffee	Loc = Shop,$\neg$RhB	RhC	0.8
		$\emptyset$	0.2
	Loc = Shop,RhB	RhC,$\neg$RhB	0.7
		$\neg$RhB	0.2
		$\emptyset$	0.1
	Loc = Off	$\emptyset$	1.0
	Loc = Lab	$\emptyset$	1.0
	Loc = Mail	$\emptyset$	1.0
BuyBun	Loc = Shop,$\neg$RhC	RhB	0.8
		$\emptyset$	0.2
	Loc = Shop,RhC	RhB,$\neg$RhC	0.7
		$\neg$RhC	0.2
		$\emptyset$	0.1
	Loc = Off	$\emptyset$	1.0
	Loc = Lab	$\emptyset$	1.0
	Loc = Mail	$\emptyset$	1.0
GetMail	Loc = Mail,MW	RhM,$\neg$MW	0.9
		$\emptyset$	0.1
	Loc = Mail,$\neg$MW	$\emptyset$	1.0
	Loc = Off	$\emptyset$	1.0
	Loc = Lab	$\emptyset$	1.0
	Loc = Shop	$\emptyset$	1.0
DelMail	Loc = Off,RhM	$\neg$RhM	0.9
		$\emptyset$	0.1
	Loc = Off,$\neg$RhM	$\emptyset$	1.0
	Loc = Lab	$\emptyset$	1.0
	Loc = Shop	$\emptyset$	1.0
	Loc = Mail	$\emptyset$	1.0

Rewards in this domain are additive as follows:

Proposition	Value	Proposition	Value
UhC	1.0	$\neg$UhC	0.0
UhB	0.7	$\neg$UhB	0.0
W	0.0	$\neg$W	0.1
MW,RhM	0.0	MW,$\neg$RhM	0.0
$\neg$MW,RhM	0.0	$\neg$MW,$\neg$RhM	0.3

The BUILDER Domain

Action	Discriminant	Effect	Prob.
PaintA	AClean	APainted	0.75
		¬AClean	0.20
		∅	0.05
	¬AClean	∅	1.00
PaintB	BClean	BPainted	0.75
		¬BClean	0.20
		∅	0.05
	¬BClean	∅	1.00
ShapeA	¬Joined	¬APainted,AShaped	0.80
		¬APainted,¬AClean,¬AShaped,¬ADrilled	0.10
		¬APainted	0.10
	Joined	¬BPainted,¬APainted	1.00
ShapeB	¬Joined	¬BPainted,BShaped	0.80
		¬BPainted,¬BClean,¬BShaped,¬BDrilled	0.10
		¬BPainted	0.10
	Joined	¬BPainted,¬APainted	1.00
DrillA	¬Joined	ADrilled	0.90
		∅	0.10
	Joined	∅	1.00
DrillB	¬Joined	BDrilled	0.90
		∅	0.10
	Joined	∅	1.00
WashA		AClean	0.90
		∅	0.10
WashB		BClean	0.90
		∅	0.10
Bolt	BShaped,AShaped, BDrilled,ADrilled	Joined	0.80
		∅	0.20
	¬ADrilled	∅	1.00
	¬BDrilled,ADrilled	∅	1.00
	¬AShaped,BDrilled,ADrilled	∅	1.00
	¬BShaped,AShaped, BDrilled,ADrilled	∅	1.00
Glue	BShaped,AShaped	¬BClean,¬AClean,Joined	0.35
		Joined	0.35
		¬BClean,¬AClean	0.15
		∅	0.15
	¬AShaped	¬BClean,¬AClean	0.50
		∅	0.50
	¬BShaped,AShaped	¬BClean,¬AClean	0.50
		∅	0.50

Rewards in this domain are additive as follows:

Proposition	Value	Proposition	Value
AClean	0.1	¬AClean	0.0
BClean	0.1	¬BClean	0.0
APainted	0.2	¬BPainted	0.0
APainted	0.2	¬BPainted	0.0
Joined	0.4	¬Joined	0.0