Prioritized Goal Decomposition of Markov Decision Processes: Toward a Synthesis of Classical and Decision Theoretic Planning

Craig Boutilier, Ronen I. Brafman, and Christopher Geib
Department of Computer Science
University of British Columbia
Vancouver, BC, CANADA, V6T 1Z4
email: {cebly,brafman,geib} @cs.ubc.ca

Abstract

We describe an approach to goal decomposition for a certain class of Markov decision processes (MDPs). An abstraction mechanism is used to generate abstract MDPs associated with different objectives, and several methods for merging the policies for these different objectives are considered. In one technique, causal (least-commitment) structures are generated for abstract policies and plan merging techniques, exploiting the relaxation of policy commitments reflected in this structure, are used to piece the results into a single policy. Abstract value functions provide guidance if plan repair is needed. This work makes some first steps toward the synthesis of classical and decision theoretic planning methods.

1 Introduction

Markov decision processes (MDPs) have become a standard conceptual and computational model for *decision theoretic planning* (DTP) problems, allowing one to model uncertainty, competing goals, and process-oriented objectives. One of the key drawbacks of MDPs, *vis-a-vis* classical planning methods, is the exponential nature of standard, dynamic programming policy construction algorithms. These generally require explicit state space enumeration.

Recent work in DTP has focused on techniques that exploit problem structure to solve MDPs in efficient ways. Such methods include problem *decomposition* based on reachability analysis [4] and exact and approximate *abstraction* algorithms [6, 2] in which states are (implicitly) aggregated depending on whether they agree on the values of certain problem variables. Indeed, these abstraction methods apply an insight fundamental to classical planning: feature-based search methods are usually much more tractable than state-based methods. This underlies, for instance, regression and causal link planning algorithms [13].

In this paper, we investigate another way of decomposing MDPs for efficient solution, namely, *goal decomposition.* Specifically, if utility is assigned to the achievement of certain features (as in multi-attribute utility theory [8]), we will consider "components" of the utility function separately. In rough outline, we decompose the utility function, construct an *abstract policy* for each component that maximizes performance with respect to its objectives, and then attempt to *merge* the abstract policies into a single, coherent, approximately optimal policy for the entire set of objectives.

We view *goal decomposition* of this type as a step toward the further synthesis of decision-theoretic and classical planning methods; goal decomposition is essential in least-commitment planning. If two subgoals are noninteracting, we can plan for each separately and "merge" the results. If the solutions interact, least-commitment planners facilitate the merging process by not committing to the ordering of actions required for a given subproblem when the order is not relevant. In fact, a representation using *causal links* makes clear the *reasons* for an action's presence in the plan.

This second insight seems to offer genuine benefits to planning, but it has not been exploited in any proposed solution methods for MDPs. Indeed, a major impediment to this approach is the requirement for some form *of policy merging* as opposed to simple plan merging. This is difficult because the concept of a policy for an MDP has no explicit flexibility: it simply assigns actions to every state.

We note that the *value function* associated with an MDP implicitly determines a set of optimal policies, not just a single policy; thus it offers some flexibility. Intuitively, it can be used to tell when an alternative action can be used instead of that dictated by the policy with no or (under certain conditions) little loss in value. We will describe a state-based search process that allows us to merge policies using value functions in this way.

We also consider intuitions from least-commitment planning to provide an alternative means of policy merging. We provide a method for extracting causal structure from a policy, under certain assumptions, that reveals the aspects of the policy that are essential to its performance and those that are not. This flexibility permits policy merging to proceed more readily. Since merging is not guaranteed to succeed, we also give *priority* to abstract policies associated with more important objectives. This *prioritization* of goals approximates the workings of optimal policy construction.

In Section 2 we overview MDPs and their representation, nonlinear plans, and the MDP abstraction technique of [6]. Section 3 outlines the major components of our framework—we describe: the construction of prioritized goals or objectives; the extraction of a nonlinear plan from a given policy; and methods for merging policies, including the use of least-commitment structure. We describe directions for future research, including challenges facing the generalization of this approach, in Section 4.

2 MDPs and Least-Commitment Plans

2.1 Markov Decision Processes

We assume that the system to be controlled can be described as a fully-observable, discrete state *Markov decision process* [1, 7, 12], with a finite set of system states S. The controlling agent has available a finite set of actions A which cause stochastic state transitions: we write $\Pr(s, a, t)$ to denote the probability action a causes a transition to state t when executed in state s. A real-valued reward function R reflects the objectives of the agent, with $R(s)$ denoting the (immediate) utility of being in state s. A (stationary) *policy* $\pi : S \rightarrow A$ denotes a particular course of action to be adopted by an agent, with $\pi(s)$ being the action to be executed whenever the agent finds itself in state s. We assume an infinite horizon (i.e., the agent will act indefinitely) and that the agent accumulates the rewards associated with the states it enters.

In order to compare policies, we adopt *expected total discounted reward* as our optimality criterion; future rewards are discounted by rate $0 \leq \beta < 1$. The value of a policy π can be shown to satisfy [7]:

$$V_\pi(s) = R(s) + \beta \sum_{t \in S} Pr(s, \pi(s), t) \cdot V_\pi(t)$$

The value of π at any initial state s can be computed by solving this system of linear equations. A policy π is *optimal* if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in S$ and policies π'. The *optimal value function* V^* is simply the value function for any optimal policy. We refer to [1, 7, 12] for a description of policy construction techniques.

2.2 Action and Reward Representation

Several good representations for MDPs, suitable for DTP, have been proposed. These include stochastic STRIPS operators [9, 6] and dynamic Bayes nets [5, 2] for representing actions, and related methods for describing reward functions. We will adopt the STRIPS-style specification used in [6].

We assume a set of atomic propositions used to describe the planning problem of interest. Each action a is represented by two components, a *precondition* and a *probabilistic effect set*. A precondition P is a list of literals, interpreted in the usual way: a can only be performed in states satisfying P. A probabilistic effect set $\mathcal{E}$ comprises a set of *context-effect* pairs of the form $\langle C, EL\rangle$. Each C is a *context* under which action a has a different (stochastic) effect. We require that the set of contexts be mutually exclusive and exhaustive given P. The associated *effects list EL* describes the possible effects action a could have in context C and the probability with which they occur. More precisely, $EL = \langle E_1, p_1; \ldots; E_n, p_n\rangle$ where each E_i is a list of literals (an *effect*) and each p_i is the probability of effect E_i occurring. If E_i occurs, the resulting state change is such that the literals mentioned in E_i become true and any literal whose atom does not occur in E_i retains its truth value [9] (i.e., E_i is a STRIPS-style change list).

To concisely represent actions that have multiple, uncorrelated stochastic effects, we use *action aspects* [6]. Each aspect for action a is represented exactly as a probabilistic effect set, with its own set of contexts and effects. However, the effects lists in different aspects must be disjoint (i.e., have no atoms in common). To determine the net effect of a in state s, the condition C_i satisfied by s in each aspect $\mathcal{E}_i$ is determined, and a *joint effect* consisting of the union of one effects list from each aspect occurs with the product of the probabilities associated with these lists.[1]

Action	*Context*	*Effect*	*Prob.*
drill(p) (Asp.1)	$\emptyset$	*hole*(p)	1.0
drill(p) (Asp.2)	*pressed*(p)	*hot*	0.9
		$\emptyset$	0.1
	$\neg$*pressed*(p)	$\emptyset$	1.0
drill(p) (Asp.3)	$\emptyset$	$\neg$*painted*(p)	0.9
		$\emptyset$	0.1
paint(p)	$\emptyset$	*painted*(p)	1.0
press(p) (Asp.1)	$\emptyset$	*pressed*(p)	0.95
		$\emptyset$	0.05
press(p) (Asp.2)	$\emptyset$	$\neg$*painted*(p)	0.95
		$\emptyset$	0.05
mdrill(p) (Asp.1)	$\emptyset$	*hole*(p)	0.7
		$\emptyset$	0.3
mdrill(p) (Asp.2)	$\emptyset$	$\neg$*painted*(p)	1.0

Figure 1: Stochastic STRIPS actions with Aspects

Feature	*Reward*
hole($P1$) $\wedge$ *pressed*($P1$)	10
painted($P1$)	7
hole($P2$)	2

Figure 2: Additive Reward Function

This representation is illustrated in Figure 1. We have several job shop operations that can be applied to parts, such as drilling, manual drilling, pressing, and painting. The effects of these operations are shown in the table. For instance, *drill(p)* has several independent effects (aspects): it produces a hole in p, it might cause the drill bit to become hot if *p* is pressed, and tends to destroy p's paint job. If one drills a pressed, painted part p, the different aspects combine to produce the following effects:

$$\langle hole(p), hot, \neg painted(p)\rangle : 0.81; \;\; \langle hole(p)\rangle : 0.01;$$
$$\langle hole(p), \neg painted(p)\rangle : 0.09; \;\; \langle hole(p), hot, \rangle : 0.09$$

Pressing can sometimes fail, and also destroys paint. Manual drilling is not guaranteed to produce a hole and is guaranteed to destroy paint. No preconditions are shown in the table, but we assume that the drilling action has the precondition *-hot* (the drill bit is not hot).

The reward function R can also be represented compactly in many circumstances. We assume the reward function associates additive, independent rewards with different domain propositions [8]: the reward associated with a state is simply the sum of the rewards given for the propositions it satisfies. Figure 2 shows such a function, where reward is given based on the features of two distinct parts *P1* and *P2.* We assume reward 0 is associated with all unmentioned features.

[1] This representation reflects the same kinds of independence assumptions one finds in Bayes net action representations [5, 2], and offers the same conciseness of representation when there are multiple uncorrclated effects.

2.3 Approximate Abstraction

The abstraction technique developed in [6] can reduce the size of the MDP one needs to solve to produce an approximately optimal policy. Using regression-like operations, we can discover which atoms influence the probability of other propositions becoming true when certain action sequences are executed. When a reward function has independent components, we can sweep through the action descriptions determining which propositions (contexts and preconditions) influence the ability to make the reward proposition true or false, which propositions, in turn, influence these, and so on. Such atoms are deemed *relevant* atoms, the set of which is denoted R. Once this process is completed, any atoms not in R (i.e., that do not influence the probability of a reward proposition through any action sequence) can be ignored. Their truth values cannot influence the optimal choice of action.

For instance, suppose the only atom that influences reward is *hole(Pl).* Since drilling affects this, its precondition *-hot* dictates that atom *hot* is relevant. Since *hot* is affected by drilling if *pressed(Pl),* atom *pressed(PJ)* is also added to R. No atom influences any action's effect on *pressed{Pl)* thus, a fixed point is reached, with the only relevant atoms being *hole(PJ), hot* and *pressed(PJ).* The abstract MDP over this reduced space has only 8 states, and can be more easily solved to produce an optimal policy.

More importantly, one can ignore certain reward propositions to produce an *approximate abstraction* (see [6]). For instance, in Figure 2, atoms *hole(PJ), pressed(Pl),* and *painted(PJ)* have a larger influence on reward than *hole{P2).* By ignoring the *hole(P2),* we create an abstract space much smaller than the one needed to solve the MDP optimally—only atoms pertaining to *Pl* (and *hot)* are introduced into R hence the resulting abstract MDP. This MDP has only 16 states and is easier to solve. One optimal policy (among several possibilities) for this MDP has the form:

If $\neg pressed(P1)$:	$press(P1)$
If $pressed(P1) \wedge \neg hole(P1) \wedge \neg hot$:	$drill(P1)$
If $pressed(P1) \wedge \neg hole(P1) \wedge hot$:	$mdrill(P1)$
If $pressed(P1) \wedge hole(P1) \wedge \neg painted(P1)$:	$paint(P1)$
If $pressed(P1) \wedge hole(P1) \wedge painted(P1)$:	*no-op*

This computational benefit comes at the cost of optimality: the policy makes no attempt to machine $P2$, since any value associated with it has been ignored. However, if the value lost is small enough, the tradeoff for a (potentially exponential) decrease in computation time may be acceptable.

Formally, we can describe the approach as follows: 1) Select a subset, $\mathcal{IR}$, of *immediately relevant* atoms that influence immediate reward. These will generally be the atoms that have the largest impact on reward. 2) Generate the set $\mathcal{R}$ of *relevant* atoms satisfying the properties: a) $\mathcal{IR} \subseteq \mathcal{R}$; and b) if $P \in \mathcal{R}$ "occurs" in an effect list of some action aspect, each atom occurring in the corresponding context, or in the precondition for a, is in $\mathcal{R}$. 3) Solve the MDP associated with this reduced set of atoms $\mathcal{R}$. We note that generating $\mathcal{R}$ given the set $\mathcal{IR}$ can be done in $O(E|\mathcal{R}|)$ time, where E is the size of the set of action descriptions (i.e., the number of context-effect pairs).

2.4 Least-Commitment Plans

We will use *least-commitment* (or partially-ordered) plans to represent the relevant parts of policies in a way that allows them to be more easily "merged." We briefly review the relevant concepts here, but refer to [13] for more details.

A *least-commitment plan* (LCP) consists of a set of action instances *A,* a set of *causal links* involving these actions and a set of *ordering constraints* (i.e., using the relations < and >) over these actions. We ignore issues of quantification (e.g., involving codesignation constraints, etc.) for simplicity. A causal link is a tuple (a, p, 6) where a and *b* are actions and *p* is a proposition. Intuitively, such a link represents the fact that *a* has effect p, *b* has precondition (or a context) p, and that no action that might occur between *a* and *b* will change the truth of *p.* We say that *a produces p* and that 6 *consumes p.* When constructing apian, we may introduce an action *c* in order to achieve a subgoal q, where c has the additional effect of making *p* false. If *c* can be consistently ordered between *a* and 6, we say that *c threatens (a, p,* 6).[2] To ensure the plan is valid, we resolve the threat by insisting that *c* occur before *a* or after 6. Such constraints restrict the legal sequences of action instances. We can think of an LCP as representing its set of *linearizations:* the sequences of action instances in *A* consistent with the ordering constraints.

We do not discuss here algorithms for producing LCPs, but refer to [13] for a survey and to some of the key developments presented in [10, 11].

3 Goal Decomposition

The aim of goal decomposition is similar to that of abstraction: to solve smaller MDPs by ignoring variables. Unlike the abstraction method of [6], however, we do not solve a single abstract MDP (e.g., for *Pl* -objectives) and ignore other objectives completely. Instead, we solve *several* abstract MDPs, each pertaining to different sets of objectives (e.g., one for *Pl* and one for *P2,* and combine the results into a single policy of higher value than any individual abstract policy.

There are two major challenges that must be met to solve this task. First, we must decide how to decompose the reward function in order to generate (and solve) small, abstract MDPs (Section 3.1). Second, we must merge policies, which itself involves several difficulties. Chief among these is the fact that policies offer no flexibility. In Section 3.2, we describe how to extract essential *causal structure* from abstract policies that reveals the flexibility inherent in a policy. In Section 3.3, we discuss policy merging. We first describe a merging procedure that uses the value functions of two abstract policies, rather than the policies themselves, to guide a search through state space. We then propose a method for merging that uses the causal structure of the first policy to constrain search through state space.

The details of the various algorithms described in this paper are based on several assumptions. We assume that our problems are *goal-oriented;* that is, we are only concerned with the achievement of propositions in the "final state" that

[2] We refer to [13] for a discussion of the complexities introduced by the existence of conditional effects.

results from executing a sequence of actions.[3] We also assume that each action has a *primary* or most probable effect, and that an initial state has been given. These assumptions legitimize the use of causal links and classical LCPs. We conjecture, however, that the basic framework for MDP decomposition and policy merging can be applied in much more general settings with appropriate extensions.

3.1 Decomposing the Reward Function

Decomposing the reward function does not involve much more than partitioning the set of reward attributes in some way, say into sets IR_1, IR_2, • • •, and generating the appropriate sets of relevant atoms R_1, R_2, • • •• For instance, Figure 2 naturally suggests the decomposition described in Section 2.3, (using the rewards associated with *PI* in one abstraction and the reward for *P2* in another). Several factors will influence the choice of abstract MDPs.

When merging policies (Section 3.3), we assume that certain abstract policies are given priority over others; that is, abstractions should be orderable in terms of *importance.* Partitioning of reward attributes should therefore be such that features with large impact on reward should lie within more important partitions. In our example, the rewards associated with the completion of *PI* are more critical, so the partitions are ordered so that *PI's* success is given priority over *P2.* When objectives conflict and policies cannot be satisfactorily merged (e.g., if *PI* and *P2* require the same consumable resource), this *prioritization* allows the more important objectives to be achieved. This approximates the behavior of an optimal policy, which would make a similar tradeoff when faced with conflicting objectives.

Attributes whose effect on reward is not independent should obviously be placed in the same abstraction. In our example, the contribution to reward made by *hole(PI)* and *pressed(PJ)* is highly correlated. Achieving one without accounting for the other makes little sense.

The odds of successful policy merging can be increased if there is little interaction between atoms in different abstract spaces; this can be a factor in the choice of IR sets. If an action influences an atom in two different spaces, the policies may interact in undesirable ways. The potential for interaction can be detected by a simple extension of the algorithm that generates R sets. Such interaction cannot generally be avoided, but if the overlap is substantial one might consider merging different IR sets, or breaking them up differently. Of course, larger MDPs usually result and the "degree of potential interaction" should be balanced against the increased size of the MDP and the potential loss in value if one of the objective sets cannot be accommodated by merging. We refer to [6] for further discussion of reward attribute selection for abstraction and the tradeoff with abstract MDP size.

3.2 Extracting Causal Structure from Policies

Once the reward function has been decomposed, the abstract MDPs associated with feature sets R_1, R_2,... can be formulated and solved, and optimal policies $\pi_1, \pi_2, \ldots$ and value functions V_1, V_2,... determined. We note that this process can proceed in an incremental anytime fashion, with attention focused on the highest priority MDPs, leaving lower priority MDPs to be solved as time permits.

As we will see, it will be useful to extract the essential causal structure of a policy π in the form of a LCP: this flexibility will aid in merging. We assume each action has a unique most likely or *primary* effect. We define the *most likely execution path* (MLEP) of an abstract policy to be the sequence of actions and states, given the initial state /, that is most likely. We assume that this path ends in some abstract goal state *G* (see below). We generate the *LCP for policy IT* by extracting causal link information from the actions that occur along this path and the objectives that are actually achieved in *G.* Any linearization of this LCP is guaranteed to be "optimal" in the sense that its MLEP leads to the same abstract goal state. Furthermore, should execution of the plan fall "off path" with an unlikely action effect, π can be used to dictate a new action choice (a new LCP can also be generated at this point if replanning/remerging is desired).

LCP extraction for *n* proceeds as follows. We first generate the MLEP for ∏ given initial state / to goal state *G.*[4] We then call a least-commitment planning algorithm, such as Weld's [13] UCPOP, with initial state / and goals consisting of those reward propositions in the current R set satisfied by *G.* Planning will be very constrained (and efficient) however: when a subgoal *q* for action *b* is selected, the most recent action *a* (in MLEP) that produces *q* is chosen as the producer and causal link (a, *q, b)* is generated. Any requirements for action *a* (preconditions and specific context propositions) are then added as subgoals. We note that there is no ambiguity in choice of context for *a* (and hence a's conditional effects) as these are uniquely determined by the state in which *a* occurs in MLEP. Finally, threats to causal links are resolved respecting the ordering of MLEP

At each step, the LCP produced by this algorithm has π**'s** MLEP as one of its linearizations. Since this path is a solution to the planning problem we constructed, UCPOP will return a plan without backtracking: MLEP acts as an oracle. Further efficiency can be gained by generating causal link information during MLEP generation, leaving only threat resolution to complete the planning process.

To illustrate, consider the optimal abstract policy described in Section 2.3 (pertaining to *PL)* with initial state:

$$\neg hot, \neg pressed(PI), \neg hole(PI), \neg painted(PI)$$

The action sequence *{press(PJ),drill(PI), paint(PI))* appears on the MLEP, and the *goals pressed (PI), hole(PI)* and *painted(PI)* are determined. UCPOP then uses the actions above to generate the LCP shown in Figure 3: causal links are denoted by solid lines and ordering constraints by dashed lines. Note that *press(PI)* and *drill(PI)* are ordered before *paint(PI)* because they threaten the effect *painted(PJ).* On the other hand, *press(PJ)* and *drill(PJ)* are unordered: though the policy commits to their ordering, this commitment

[3] We refer to [4] to see how such objectives can be encoded in an MDP using absorbing states. For example, we can have a distinguished *stop* action that precludes further actions.

[4] A terminal state *G* can be identified as one whose value $V_\pi(G)$ is not significantly greater than the sum of the reward propositions it satisfies, or one for which a *stop* action has been selected.

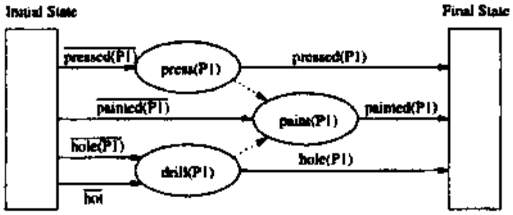

Figure 3: Causal structure with ordering constraints

is discovered to be unessential to goal achievement in the planning process. Execution in either order (assuming most likely outcomes) will be optimal.

As we will see, plan merging benefits from having LCPs with as few constraints and causal links as possible, increasing the odds of success. The LCP extraction algorithm can be altered to produce "relaxed" plans that approximately correspond to MLEP. In particular, we may ignore certain context conditions consumed by an action if these do not substantially influence goal achievement. For example, suppose *smooth(Pl),* while potentially important at certain states, is of marginal value on the current MLEP for π. If, say, *smooth(Pl)* is produced by the initial state and consumed by *drill* and only marginally increases the probability of *hole,* the link from the initial state to *drill* could be dropped. Merging new plans with this *relaxed* LCP might result in a (slightly) less than optimal execution of π if the new plan destroys *smooth(PJ)* before drilling; however, the added flexibility makes merging easier and may be worthwhile.

33 Prioritized Policy Merging

Using State Space Search Suppose we now have a set of abstract MDPs that have been solved. For concreteness, assume two such MDPs whose optimal policies π_1, π_2 and value functions *V1*, V2 have been determined. Our goal is to produce a new policy π that behaves well with respect to both sets of objectives, without solving a new MDP that combines both abstract spaces. To do this, we can use the value functions as heuristics to guide a search through state space. Each value function tells us how good a state is with respect to its own objectives: if *Vi(s)* is high, the goals in MDP i are achievable. Note, however, that this does not mean the goals are achievable jointly with those of some other MDP *j.*

More precisely, we define a search process whose nodes are states, whose arcs are labeled with actions, and whose initial state is the start state. At any active node of the search tree, we consider the effect of each possible action; that is, we add children corresponding to the unique state resulting from each action. Each such state *s* has values Vi(s) The most promising such state is that with the highest such values.5 Even if no action dominates all others, under the assumption that the reward function is additive, the sum of values Vi(s) can be used as the heuristic value of state s.

How this search process proceeds is rather flexible; but a best-first approach seems most appropriate. Each value V, (s) is a "perfect" heuristic in that, should the agent consider only objectives in space t, it could attain Vi(s). If the agent attempts to integrate other objectives, this can only cause the value *Vi(s)* to go down (or, at best, stay the same). Thus, the "multivalued" heuristic (or the sum of values V, (s)) overestimates the true value of the state s, and is admissible—a simple best first search using this as the heuristic function will be appropriate.6 The search process terminates once an appropriate goal state is reached.

Merging does require that we consider the "combined" abstract space containing the relevant atoms from all abstract MDPs involved. However, while this space is considerably larger than the individual abstract spaces, we never need to construct policies, or plan in substantial ways, in the combined space. Thus, the exponential "blow up" in state space size is not a factor in policy merging.

To illustrate the process, consider the example above, where the value functions with respect to parts *P1* and *P2* are denoted *V* and *V2* Assume *pressed(P2)* holds at the start state (along with the relevant propositions for *Pl).* Suppose *press(Pl)* is chosen as the first action: it causes no drop in *V* or V2.7 However, at any subsequent point *drill(Pl)* or *drill(P2)* will cause a drop in value because *hot* becomes true, preventing drilling of the other part. But an alternative initial action, *drill(Pl),* exists and subsequent search will lead directly to an appropriate sequence of actions. Notice that this process leads to a policy in which the actions *drill(Pl)* and *press(Pl)* are reversed (w.r.t. the original π_1). Note, however, that policy π_1 does not play a role in the search process.

Modifying the example slightly, suppose that drilling causes *hot* to become true no matter what. In this case, *drill(P2)* causes a drop in V1, while *drill(Pl)* causes a drop in V2 (regardless *of pressed (P1)).* Since *P1* has higher value, the first drop is greater and we will "commit" to drilling *Pl,* and using *mdrill* for *P2.*

Finally, we note that pseudo-determinism is exploited by the search to find a merged *action sequence* rather than a policy *per se.* However, as mentioned, should any action have an unlikely effect, the agent can always resort to one of the abstract policies to decide how to act at the resultant, "off-path" state (e.g., choose to execute π_i for that abstract policy with highest value *Vi(s)).* We could also imagine applying the search process to the most probable "unlikely" branches, or remerging online as circumstances warrant.

Using Plan Structure to Guide Search One difficulty with the search process described above is the fact that the actual search space explored can be quite large. Furthermore, a number of alternatives may be explored before a genuine

[5] The unique state associated with an action is due to the "pseudo-deterministic" assumption we made, but, in fact, the value of this resulting state is not the only consideration. If an action is likely to be successful but has some risk associated with it, the fact that its likely outcome is better than that of another action is not sufficient reason to select it (note that the abstract policies do take such considerations into account). Thus we use the *expected value* of an action w.r.t. V, (or *Q-value of the action)* as a guide.

[6] Multiple path (or cycle) checking is also appropriate in our pseudo-deterministic setting to ensure progress in being made.

[7] To be precise, a slight drop in value, corresponding to the discount factor will occur. We shall ignore this effect in our discussion.

conflict is detected (as in the second case above). A second difficulty lies in the fact that this process does not lend itself to revision of policies or to the incremental construction of merged policies in an anytime manner. For example, if we merge policies for several abstract spaces, and then later solve another abstract MDP, merging the new policy requires revisiting the entire search process, unless some form of dependency information is recorded.

We now describe a process that alleviates these difficulties to some extent: when merging π_1 and π_2, we will first extract an LCP P1 for π_1 and restrict our search to action sequences that have high value with respect to V2 (on MDP 2). Any such action sequence must, however, respect the causal links in plan P1. This makes the search process easier for two reasons. First, the flexibility in ordering constraints in *P1* allows us to consider several different optimal policies (w.r.t. *V1*) without explicit backtracking. Second, we will not consider policies that are inconsistent with P1. This means we are *committing* to a particular means of achieving objectives in the first space. As such, the resulting merged policy may not be as good as one that doesn't adhere to such restrictions; but the computational benefits may make this worthwhile. Once we've merged π_1 and π_2, we can extract a new LCP P2 that can be used when considering subsequent objectives. Recording *plans* allows the merging process to proceed in an incremental fashion. We now make this more precise.

Assume that an LCP P1 has been extracted for a particular set of objectives,[8] and that an abstract value function V2 has been generated for a new abstract MDP. Our merging process is *prioritized:* as before, we generate a path starting at the start state. Only now, this path must be consistent with Pi (i.e., some linearization of P1 is a subsequence of this path), and *V2* alone is used as the heuristic function. Hence, we search for the best policy with respect to V2 subject to the constraints imposed by P1. This prioritization is necessary in instances where all objectives cannot be met. Since some must be "sacrificed," we commit to the plan pertaining to higher valued objectives.[9] The search proceeds as follows:

1. Each search node corresponds to a state and is annotated by a set of *open links* and *permissible actions* from P1.
2. Each arc in the search tree is annotated by an action.
3. We use *V2* as the heuristic function.
4. We search from the start state: its open links are produced by *start,* and permissible actions are those that can consistently be executed first in P1.
5. Each node is expanded using only actions that are on the permissible list from P1, or that do not threaten any link on the open list. Irrelevant actions (that do not belong to P1 and do not influence atoms in R2) are not considered.
6. If an action *a* from P1 is used, the child node is annotated as follows: all links consumed by *a* are removed from the open list and those produced by *a* are added; *a* is removed from the permissible list; and any new action, all of whose predecessors (w.r.t. ordering constraints of P1) now lie on the search path, is added to the permissible list.

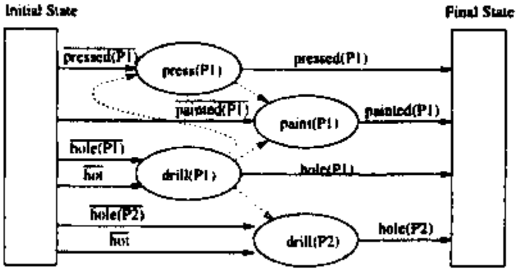

Figure 4: Merged causal structure (1)

The *open list* keeps tracks of links that must be respected by any action inserted to achieve objectives in \/2, while the *permissible list* serves as a "plan counter" for P1. A best-first search using *V2* can be used. Once this search is complete, we can extract (backtrack-free) a new LCP P2 from the chosen path. The process can then be repeated using P2 should one wish to consider further objectives (abstract MDPs).

It is often the case that more than one optimal policy exists. Although our use of LCPs allows us to capture the causal link structure of a number of policies, there may be different policies involving substantially different actions (e.g., a different way of making holes). Hence, at any point where merging results in a greatly reduced value in V2, we have the option of backtracking to a different optimal (or possibly suboptimal) policy for the earlier abstract MDP, generating a new plan for it and attempting to merge with this new plan. Thus, our commitment to P1 need not be absolute.

To illustrate the merging process, let our first abstract plan Pi be that in Figure 3 (corresponding to P/), and let the second value function *V2* correspond to the sole objective pertaining to *P2* (attained by *drill(P2)).* Assume again that *pressed(P2)* holds. Search using V2 suggests a number of potential actions: *drill(P2)y drill(P1),* and *press(Pl).* Intuitively, the first action is desirable relative to *V2* and the other two actions are from P1. (Action *mdrill(P2)* is suboptimal w.r.t. V2, and all actions involving *P1* are irrelevant to V2.) The action *drill(P2)* threatens *drill(Pl)* in P1 because of its effect *hot.* Hence, we consider the next possible action, *drill(Pl).* Suppose we choose to follow this path. As a subsequent action, we can choose either *drill(P2)* or *press(PJ).* The conditional effect *hot* of *drill(P2)* is no longer a threat. We can continue from here on in a straightforward manner, eventually generating the LCP shown in Figure 4. Note that merging *reversed* the ordering of *press(Pl)* and *drill(Pl)* in the original abstract policy without backtracking, showing the benefits of extracting causal structure. In addition, *mdrill(P2)* is never considered because of its low value.

As a second example, again imagine that *hot* becomes true *whenever* drilling occurs. Any attempt to use *drill(P2)* as the first action will threaten *drill(Pl)* as above; but *drill(P1)* precludes subsequent execution of *drill(P2).* The commit-

[8] This need not be for a single abstract MDP: a set of such MDPs may have been merged as in the previous section.

[9] We assume without further mention that if abstract policy π_1 has very low value, compared to the set of objectives it is designed to achieve, it can be ignored completely. Priority can be altered dynamically in this and other ways.

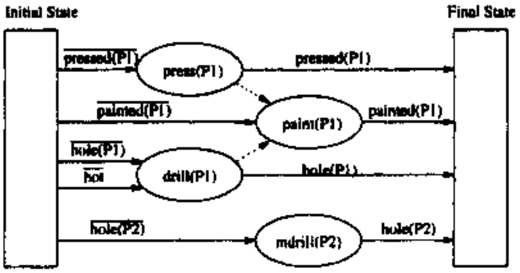

Figure 5: Merged causal structure (2)

ment to the initial plan means that an alternative action must be chosen: the value function V_2 dictates that action *mdrill(P2)* should be performed and the goal state is then reached (though with lower probability). An LCP can then be extracted which achieves both sets of objectives, as shown in Figure 5. We note that action *mdrill(P2)* is not on the MLEP for the optima] policy π_2; but commitment to the first plan forced us to consider a suboptimal way of achieving the second objective. In general, the algorithm—using the optimal value function V_2—effects "repair" of an unmergeable policy. Indeed, if at that point the optimal policy π_2 dictated that a different (conflicting) objective should be attempted (e.g., shipping a different part instead of P2), the merging algorithm would have stirred us to a completely new "goal."

The key difference between merging using only abstract value functions and merging using a plan is that the latter *commits* to a particular way of achieving the higher priority objectives and attempts to work around these commitments when determining how to achieve lower priority objectives. This commitment has some clear computational advantages, but comes at the cost of perhaps sacrificing some lower priority objectives because one is not willing to consider (substantially) different ways of attaining high priority goals.

The plan-based merging approach shares much of the motivation underlying work on intentions and resource-bounded reasoning in planning [3]. Although it may result in inferior policies as compared to the first, less constrained search method, the commitment embodied by the plan-based approach is suitable when, due to uncertainty and resource constraints, we prefer an anytime approach in which we first generate a plan for the most important objectives. Later, we can modify such plans, taking into account other, less important objectives. In addition, efficient planning may often require us to accept certain assumptions about the state of the world or its dynamics. Once we learn that one of these assumptions is false, it may not be possible to replan from scratch, as required by the first merging approach. Revision of a plan to reflect changing assumptions may prove more suitable in such circumstances.

This use of commitment could be taken even further: we could extract an LCP for both (sets of) objectives and attempt to merge them, thus committing to a specific means of attaining both. We note that it may be possible to use and modify existing plan merging techniques (e.g., see [14]) in this regard. However, strong commitment to earlier (higher priority) plans remains essential if plans prove unmergeable.

4 Concluding Remarks

We have suggested a method of goal decomposition for MDPs based on abstraction and least-commitment planning. Our technique allows certain classes of MDPs to be solved efficiently, though suboptimally, through the solution of a set of considerably smaller MDPs associated with different objectives. There are a number of questions relating classical planning, DTP and resource-bounded reasoning that may, in the future, be addressed satisfactorily within this framework. These include the derivation of *goals* from general decision-theoretic considerations, the utility of classical plans in DTP settings, and the derivation of intentions or commitments from the decomposition of objective functions.

We are currently exploring generalizations of this approach, in particular, the development of least-commitment, causal link representations of complete policies and methods of extracting such from standard, state-based policies. This will enable the restrictive assumptions of goal-based, pseudo-deterministic problem structure to be dropped. We are also exploring other means of merging and methods for bounding the error associated with prioritized merging.

References

[1] R. E. Bellman. *Dynamic Programming.* Princeton, 1957.

[2] C Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *IJCAJ-95,* pp. 1104-1 111, Montreal, 1995.

[3] M E. Bratman. *Intentions, Plans, and Practical Reason.* Harvard, 1987.

[4] T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In *AAAI-93,* pp.574-579, Washington, D.C., 1993.

[5] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Comp. Intel.,* 5(3): 142-150,1989.

[6] R. Dearden and C. Boutilier. Abstraction and approximate decision theoretic planning. *Artif. Intel.,* 89:219-283, 1997.

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

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

[9] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic least-commitment planning. In *AAA/-94,* pp. 1073-1078, Seattle, 1994.

[10] D. McAilester and D. Rosenblitt. Systematic nonlinear planning. In AAA/-9/, pp.634-639, Anaheim, 1991.

[11] J. S. Penberthy and D. S. Weld. UCPOP: A sound, complete, partial order planner for adl. In *KR-92,* 1992.

[12] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, NY, 1994.

[13] D. S. Weld. An introduction to least commitment planning. *AI Magazine,* Winter 1994:27-61,1994.

[14] Q. Yang, D. S. Nau, and J. Hendler. Merging seperately generated plans in limited domains. *Comp. Intel,* 8(4), 1992.