Decomposition Techniques for Planning in Stochastic Domains

Thomas Dean" Shieu-Hong Lin
Department of Computer Science
Box 1910, Brown University
Providence, RI 02906, USA
Email {tld,shl}@cs brown edu

Abstract

This paper is concerned with modeling planning problems involving uncertainty as discrete-time, finite-stale stochastic automata Solving planning problems is reduced to computing policies for Markov decision processes Classical methods for solving Markov decision processes cannot cope with the size of the state spaces for typical problems encountered in practice As an allernative, we investigate methods that decompose global planning problems into a number of local problems solve the local problems separately and then combine the local solutions to generate a global solu tion We present algorithms that decompose planning problems into smaller problems given an arbitrary partition of the state space The local problems are interpreted as Markov decision processes and solutions to the local problems are interpreted as policies restricted to the subsets of the state space defined by the partition One algorithm relies on constructing and solving an abstract version of the original de cision problem A second algorithm itcratively approximates parameters of the local problems to converge to an optimal solution We show how properties of a specified partition affect the time and storage required for lhese algorithms

1 Introduction

We are concerned with solving planning problems posed as Markov decision processes Spec Ifically, given a dynamical model described as a stochastic process (t *g* Markov chain) with a large, discrete state space and a performance criterion *(e g ,* minimize the expected time or cost to reach a goal), construct a policy (plan) mapping states to actions that realizes the specified performance criterion or approximates it to within some specified tolerance

This work was supported by the Air Force and the Advanced Research Projects Agency of the Department of De Tense under Contract No F30602-91 C-0041 and by the National Science Foundation in conjunction with the Advanced Research Projects Agency under C ontract No IRI-89054J6

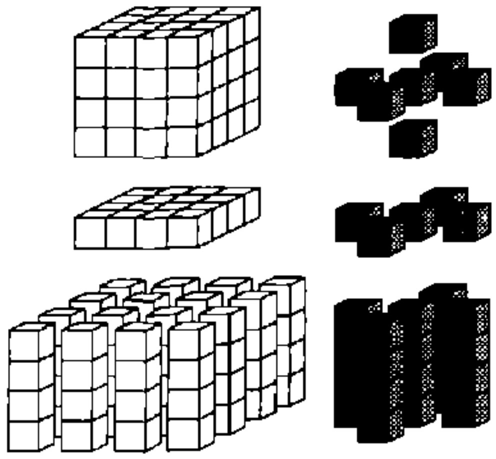

Figure l Three views of a three dimensional state space with the corresponding local strut-turre of spnce shown to the right The top view representh the unstructured stale space the middle view represent an abstraction obtained b\ projection and the bottom view represents a decomposition obtained by partitioning the stales into aggregate states

1 1 Factoring Large State Spaces

large state spaces present a number of cnallenges lo begin with, the problem has to be efficiently encoded A *factored* state-space representation uses state variables to represent different aspects of the overall state of the system ' Co npact encodings for stochastic professes can be achieved for many applications using fat l ore d state space representations where the size of the model is usually logarithmic in The size of the state space [Dean and Kanazawa, 1989] Similarly, policies for large faetored state spaces can often be efficiently encoded using decision trees that branch on state variables [Bou tiher *it at* 1995] Assuming that both the problem (a stochastic process) and the solution (a policy) can be encoded in a compact form, we would like to generate solutions in Lime bounded by some small factor of the problem and solution size

'Propositions representing fluents in STRIPS opualors [Fikes and Nilsson 197l] correspond lo state variables in a factored state-apace representation

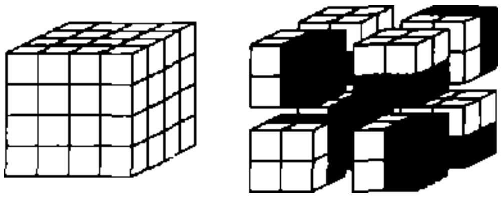

Figure 2 Region by-region dimensionality reduction A three-dimensional space is represented as the union of two-dirnensional abstract subspaces shaded dark grav

A factored state-space representation with it boolean state variables represents an n-dimcnsional state space with $O(2^n)$ states We assume that all of the state variables are relevant in at least some portion of the stale space and so the dimensionality of the problem cannot be reduced without suffering some lOSS in performance However, it is very likely that not all statt variables are relevant in all portions of the state space I(e g , when you are planning to take a walk in southern Florida you ran neglect the possibility of snow) Figure 1 Illustratis thru VIWS, of a multi-dirnensional state space The bottom view in winch the state space is partitioned into aggregate states is the view we are most interested in

1 2 Dimensionality Reduction

In this paper, we assume that a domain expert has partitioned the state space into m regions such that in each region only a small subset (of size no more than $r, r \ll n$) of the set of all state variables, is relevant for decision making In other words, for each region, we are concerned with an abstract subspace of size no more than 2^r The size of the union of these abstract subspaces is no more than $m2^r$ The problem of automatically constructing such a partition is *not* addressed in this paper, but see [Lin and Dean, 1994] for some relevant techniques Figure 2 illustrate* how a three-dimensional state space might be represented as the union of two-dimensiona] abstract subspaces

There exist methods for computing policies that are polynomial in the size of the state and action spaces [Papadimitriou an d Tsitsiklis 19H7] [Puterman, 1994], but these methods are impractical for large state spaces ($e g > 10^6$ states given 20 state variables) Instead of considering the large state space as a whole, we are interested in decomposition methods that deal with the smaller subspaces of individual regions

1 3 Combining Local Solutions

Our framework is a special case of divide and conquer given a Markov decision process and a partition of the state space into regions, (i) reformulate the problem m terms of smaller Markov decision processes over the subspaces of the individual regions., (n) solve each of these ^ubproblenib and then (m) combine the solutions to obtain a solution to the original problem

In the best case, all of the subproblems are independent and combination is trivial *(e g ,* a manufacturing task that involves assembling and testing several components each of which is assembled independently) In such cases, it does not matter how you enter a region of the partition or how you leave, the only thing that matters is the cost accrued while in that portion of tht state space In the more likely case, the subproblems are weakly coupled to one another so that, for example, what you do in one region only affects what you do in a few neighboring regions Examples of weakly-coupled systems include staged manufacturing and military planning problems and robot navigation tasks

We associate a set of topologically motivated param eters with each region *R* These parameters summarize the interactions between *R* and the other regions in the partition A specific estimate of the parameter values associated with region R allows us to construct a Markov decision process over the subspace associated with *R* B} solving such a Markov decision process, we determine a local policy on the region *R,* which is a solution to the subproblem associated with *R* In this paper we describe two basic methods for combining solutions to subproblems in order to generate a solution to the global problem

The first combination method is illustrated in an algorithm called *hierarchical policy construction* which considers particular sets of parameter values that have an intuitive topological interpretation These sets of pa rameter values give us a set of candidate local policies associated with each region We then construct an abstract Markov decision process by considering individual regions as abstract states and their candidate local policies as abstract actions The solution to this abstract Markov decision process assigns a particular candidate policy to each region thus yielding a policy on the entire state space Hierarchical policy construction produces an optimal policy only in special cases however it does so relatively efficiently and has an intuitive interpretation that makes it particularly suitable for robot navigation domains

The second method of combining solutions involves it erative approximation of the optimal parameter values i *i* , those values associated with optimal solutions to the global problem On each iteration of the iterative approximation method we consider for each region *R* a specific estimate of the parameter values for region *R* and solve the resulting Markov decision process to obtain a local policy By examining the resulting local poh eies, we obtain information to generate a new estimate of the parameter values that is guaranteed to improve the global solution This information about local policies also tells us when the current solution is optimal or within some specified tolerance, and therefore when it is appropriate to terminate the iterative procedure

1 4 Overview of the Paper

In Section 2, we provide a brief introduction to Markov decision processes Section 3 describee the parameters modeling the inter-regional interactions, and the construction of a Markov decision process on a region *R* given a specific estimate of the parameter values for *R* Section 4 presents the hierarchical policy construction algorithm We describe the construction of abstract decision processes from a base-level process, and the use of such abstract decision processes to construct policies for the base-level process Section 5 illustrates the method of the iterative approximation and briefly addresses is sues concerning convergence, optimally, and complexity Details are available in a longer version of the paper [Dean and Lin, 1995]

2 Markov Decision Processes

Let $M = (\Omega_X, \Omega_A, p, c)$ be a Markov decision process with *finite state space* Ω_X, *actions* Ω_A, *state transition* matrix p and cost matrix c Let $\Omega_X = \{1, 2, \ldots, N\}$ X_t (A_t) is a variable indicating the state (action) at time t For all $i, j \in \Omega_X$ and $a \in \Omega_A$ we have

$$p_{ij}(a) = \Pr(X_t = j \mid X_{t-1} = i, A_{t-1} = a)$$
$$c_{ij}(a) = C(X_t = j \mid X_{t-1} = i, A_{t-1} = a)$$

where $\Pr(\,|\,)$ is a conditional probability distribution and $C(\,|\,)$ is a real-valued cost function A *policy* π is a function mapping states to actions $\pi : \Omega_X \rightarrow \Omega_A$

To completely define a Markov decision process we also need a performance criterion Two criteria that we consider are *expected discounted cumulative cost* and *expected cost to reach a specified goal* In the former, the task is to find a policy minimizing the expected cumulative cost function,

$$\mathrm{E}_\pi(\Sigma_\gamma|i) = \sum_{j\in\Omega_X} p_{ij}(\pi(i))\,[c_{ij}(\pi(i)) + \gamma \mathrm{E}_\pi(\Sigma_\gamma|j)]$$

for all $i \in \Omega_X$ where $0 < \gamma < 1$ is the *discount rate*, Σ_γ represents the discounted cumulative cost, and $\mathrm{E}_\pi(\,|\,)$ denotes an expectation with respect to the policy π For the criterion of expected cost to reach a specified goal a subset of Ω_X is designated as a target and performance is measured as the expected cost until arriving in some target state Informally, we can model each target state as a sink (all transitions out have probability zero $p_{ij\neq i}(a) = 0$) and proceed as in the case of expected discounted cumulative cost but with $\gamma = 1$

We mention two standard methods for solving Markov decision processes Bellman's value iteration method [Bellman, 1961] iterates by computing the optimal expected cumulative cost function accounting for n steps of lookahead using the optimal expected cumulative cost function accounting for $n-1$ steps of lookahead Value iteration is guaranteed to converge in the limit to the optimal expected cumulative cost function accounting for an infinite lookahead Howard's policy iteration [Howard 1960] iterates by first computing the expected cumulative cost function for the current policy and then improving the policy by using this cost function Policy iteration is guaranteed to converge to the optimal policy in time polynomial in N Puterman [1994] provides an up-to-date overview of algorithms for solving Markov decision processes

In iterative methods it is often useful to be able to compute a bound on the difference between the value of the current solution and the optimum Let π^* denote an optimal policy for M Suppose ξ_i is the probability of starting out in state i and π is the current policy then $\sum_{i\in\Omega_X} \xi_i \mathrm{E}_\pi(\Sigma_\gamma|i)$ is the value of the current solution and $\sum_{i\in\Omega_X} \xi_i \mathrm{E}_{\pi^*}(\Sigma_\gamma|i)$ is the optimum The algorithm described in Section 5 relies on computing such a bound

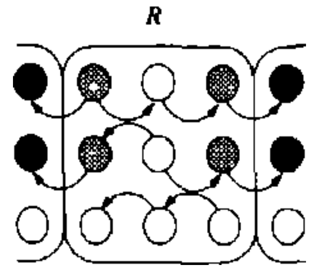

Figure 3 Boundary states (light gray) and periphery states (darker gray) of region R

3 Decomposing Markov Processes

In this section, we describe a general method of decomposing a Markov decision process defined on a large *state space into smaller Markov decision processes* defined on local regions Based on this regional decomposition framework we develop two approaches in the following two sections that combine the solutions to the smaller Markov decision processes into a solution to the original Markov decision process

Let P be any partition of Ω_X, $P = \{R_1, \ldots, R_m\}$ such that $\Omega_X = \bigcup_{i=1}^{m} R_i$ and $R_i \cap R_j = \emptyset$ for all $i \neq j$ We refer to a region $R \in P$ as an *aggregate* state We refer to a state in Ω_X as a *base-level* state

The *periphery* of an aggregate state R (denoted Periphery(R)) is the set of all base-level states not in R but reachable in a single transition from a base-level state in R $\{j | j \notin R \wedge \exists i \in R, a \in \Omega_A, p_{ij}(a) > 0\}$ We say that aggregate state R in P is *adjacent* to aggregate state S in P (denoted $R \rightsquigarrow S$) just in case Periphery$(R) \cap S \neq \emptyset$

The *boundary* of an aggregate state R (denoted Boundary(R)) is the set of all base level states in R from which you can reach a base-level state not in R in a single transition $\{i | i \in R \wedge \exists j \notin R, a \in \Omega_A, p_{ij}(a) > 0\}$ In Figure 3, the boundary states are shaded light gray and the periphery states are shaded darker gray

Next we introduce a set of parameters that we will use to model interactions among regions Let $U = \bigcup_{R\in P}$ Periphery(R) and λ_i for each $i \in U$ denote a real-valued parameter Let $\bar{\lambda} \in \Re^{|U|}$ denote a vector of all such λ_i parameters and $\bar{\lambda}|_R$ denote a subvector of $\bar{\lambda}$ composed of λ_i where i is in Periphery(R) U is the medium for inter-regional interactions, a region R can only communicate with the other regions through the states in U Parameter λ_i serves as a measure of the expected cumulative cost of starting from a periphery state and $\bar{\lambda}|_R$ provides an abstract summary of how the other regions affect R

Given a particular $\bar{\lambda}$, the original Markov decision process can be decomposed into smaller Markov decision processes, each of which determines a local policy on a local region For a region R and the subvector $\bar{\lambda}|_R$ of $\bar{\lambda}$, we define a Markov decision process $M_{\bar{\lambda}|_R} = (R \cup \text{Periphery}(R), \Omega_A, q, k)$ and the corresponding local policy $\pi_{\bar{\lambda}|_R}$ as follows

1 $R \cup$ Periphery(R) is the (local) state space for $M_{\bar{\lambda}|_R}$

2. q_{ij} is the (local) state transition matrix for $M_{\bar{\lambda}|_R}$, where
   - $q_{ij} = p_{ij}$ for $i \in R$
   - $q_{ii} = 1$ for $i \in$ Periphery(R)
3. k_{ij} is the (local) cost matrix for $M_{\bar{\lambda}|_R}$, where
   - $k_{ij} = c_{ij}$ for $i, j \in R$
   - $k_{ij} = \lambda_j + c_{ij}$ for $i \in R$ and $j \in$ Periphery(R)
   - $k_{ii} = 0$ for $i \in$ Periphery(R)
4. $\pi_{\bar{\lambda}|_R}$ corresponds to the local policy that is optimal for $M_{\bar{\lambda}|_R}$ with performance criterion expected cost to goal and target set Periphery(R)

$M_{\bar{\lambda}|_R}$ is the subproblem we associate with region R given $\lambda|_R$ as an abstract summary of R's interaction with the other regions. $\pi_{\bar{\lambda}|_R}$ is the solution to the subproblem $M_{\bar{\lambda}|_R}$. A particular $\bar{\lambda}$ determines a set of local policies (abstract actions) which in turn determines a policy on the entire state space. Let π^* denote an optimal policy for M. If $\lambda_i = \mathrm{E}_{\pi^*}(\Sigma_\gamma|i)$, then the resulting local policies as defined above define an optimal policy on the entire state space. The algorithms considered in the following two sections offer various methods for either guessing or successively approximating $\mathrm{E}_{\pi^*}(\Sigma_\gamma|i)$ for all $i \in U$.

4 Hierarchical Policy Construction

In this section, we first describe a general method for constructing an abstract decision process from a base-level process given a fixed partition of the state space. We then present a hierarchical policy construction method for using an abstract decision process to construct policies for the base-level process.

4.1 Abstract Decision Processes

Let $P = \{R_1, \ldots, R_m\}$ be any partition of Ω_X. Each region $R \in P$ is considered as an *abstract state*. A particular local policy $\pi_{\bar{\lambda}|_R}$ on region R is considered as an *abstract action* on R, which reflects our bias toward different periphery states described by $\bar{\lambda}|_R$. Abstract actions for stochastic domains are the rough equivalent of macro operators for deterministic domains. The abstract action $\pi_{\lambda|_R}$ indicates how to act optimally in region R if the interactions with the other regions are captured in λ. For example, a large value for λ_j naturally discourages us from entering the periphery state j since it induces a large cost in k_{ij}. The family of all abstract actions is denoted $\mathcal{F} = \{\pi_{\lambda|_R} | R \in P, \bar{\lambda} \in \Re^{|U|}\}$.

The probability of ending up in S starting in R and following an abstraction $\pi_{\lambda|_R}$ is defined by

$$p'_{RS}(\pi_{\bar{\lambda}|_R}) = \frac{1}{|\text{Boundary}(R)|}\left[\sum_{i \in \text{Boundary}(R)} \varphi_i\right]$$

$$\varphi_i = \left[\sum_{j \in S \cap \text{Periphery}(R)} p_{ij}\right] + \sum_{j \in R} p_{ij}\varphi_j$$

where $p_{ij} = p_{ij}(\pi_{\lambda|_R}(i))$. Note that we assume here that there is an equal probability of starting in any state in the boundary of R.

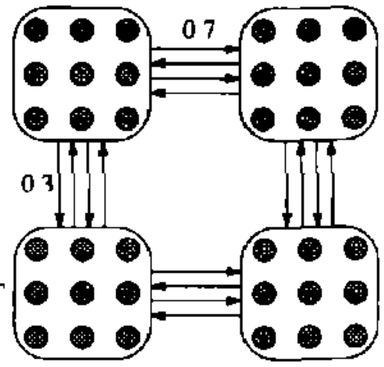

Figure 4: Abstract Markov decision process

The cost of ending up in S starting in R and following $\pi_{\lambda|_R}$ is defined by

$$c'_{RS}(\pi_{\bar{\lambda}|_R}) = \frac{1}{|\text{Boundary}(R)|}\left[\sum_{i \in \text{Boundary}(R)} \vartheta_i\right]$$

$$\vartheta_i = \left[\sum_{j \in S \cap \text{Periphery}(R)} p_{ij}c_{ij}\right] + \sum_{j \in R} p_{ij}[c_{ij} + \vartheta_j]$$

where $c_{ij} = c_{ij}(\pi_{\bar{\lambda}|_R}(i))$.

The resulting abstract decision process is then defined by $M_{\bar{\lambda}} = (P, \mathcal{F}, p', c')$. It is important to note that M_λ need not be Markov. In some cases, we may simply accept this as one of the inevitable consequences of abstraction and proceed as if the process is Markov; in other cases, we may attempt to ameliorate this condition (at some increase in computational cost) by using one of several standard techniques.

4.2 Hierarchical Policy Construction

In the following algorithm, called *hierarchical policy construction*, we restrict our attention to a finite subset of $\mathcal{F}$. For each R and each S adjacent to R, we construct a local policy $\pi_{R \to S}$ by setting $\lambda_i = 0$ for $i \in S \cap$ Periphery(R) and setting $\lambda_i = \kappa$ for $i \in$ Periphery$(R) - S$, where κ is some fixed constant. If the performance criterion for the base-level process is expected cost to goal, then for each R containing one or more target states, we add an additional action in which all the peripheral states get $\lambda_i = \kappa$ and all of the target states are made into sinks with $k_{ii} = 0$. The abstract decision process is $(P, \mathcal{F}_{\leadsto}, p', c')$ where $\mathcal{F}_{\leadsto} = \{\pi_{R \to S} | R, S \in P \wedge R \leadsto S\}$ and the performance criterion is expected discounted cumulative reward for a discount rate γ.

The local policies have the interpretation that $\pi_{R \to S}$ is the policy to take starting in R if you want to get to S. The larger κ is, the more incentive there is to get to S and avoid the rest of the periphery of R. Figure 4 illustrates an example in which $p'_{RS}(\pi_{\bar{\lambda}|_R}) = 0.7$ and $p'_{RT}(\pi_{\bar{\lambda}|_R}) = 0.3$. The abstract policy has the interpretation of providing a global perspective and indicating for each region the best local policy to use. Generally, it is best to set γ very close to one or use an alternative performance criterion such as average expected cost per step [Derman, 1970]

The following is an algorithm to construct a global policy using the abstract decision process $(P, \mathcal{F}_{\sim}, p', c')$

1. Set κ and compute $\pi_{R\to S}$ for $R\rightsquigarrow S \in P$
2. Calculate the abstract transition probabilities p' and abstract costs c'
3. Set γ and solve the abstract decision process to obtain an abstract policy $\Pi : P \to \mathcal{F}_{\sim}$
4. To determine the action to take in base-level state i, determine $R \in P$ such that $i \in R$ and take action $\Pi(R)(i)$

The above algorithm for constructing and solving abstract processes can be applied recursively and hence applies to hierarchical partitions

Hierarchical policy construction produces an optimal policy only in special cases, however, it does so relatively efficiently and has an intuitive interpretation that makes it particularly suitable for robot navigation domains For a simple partition of the state space with no aggregation within regions, standard algorithms [Puterman, 1994] on the base-level state space would be dominated by a factor quadratic in the size of the state space ($|\Omega_{\mathcal{X}}|$) while hierarchical policy construction would be dominated by the number of regions in the partition ($|P|$) times the maximum number of neighbors for any region ($\max_{R\in P}|\{S|S \in P \wedge R\rightsquigarrow S\}|$) times the square of the size of the largest region ($\max_{R\in P}|R|$)

5 Iterative Improvement Approach

Given a particular $\bar{\lambda}$, the base-level process is decomposed into local processes $\{M_{\lambda|_R}\}$ By solving these local processes, we derive the corresponding local policies (abstract actions) $\{\pi_{\lambda|_R}\}$ The solution policy π, to the base-level process is then derived by combining these local policies together The quality of π critically depends on the choice of λ Instead of relying on a particular λ we can successively modify λ and proceed through multiple iterations of decomposition and localized computation to determine a policy for the base-level process There are three issues in realizing this iterative improvement framework

- how to modify $\bar{\lambda}$ iteratively so that the solution quality can be improved on each iteration and is guaranteed to converge to an optimal solution in a finite number of iterations,
- how to determine it is time to terminate the computation when the solution is optimal or within some specified tolerance, and
- how to combine the previously generated local policies into a policy of the base-level process when we terminate the computation

In this section, we focus on a particular iterative method that resolves these three issues This iterative method is based on a reduction to the methods of Kushner and Chen [1974] that demonstrate how to solve Markov decision processes as linear programs using Dantzig-Wolfe decomposition [Dantzig and Wolfe, 1960]

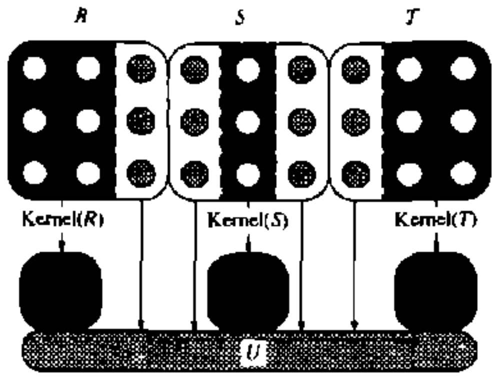

Figure 5 Relationship between the partitions P and Q

The details of the material presented in this section depend on some understanding of linear programming [Chvatal 1980] and methods for decomposing and solving large systems [Lasdon, 1970] Rather than assume this understanding we refer the reader to the longer version of the paper [Dean and Lin, 1995] for the details and just sketch the method in the following

1. Given an arbitrary partition $P = \{R_1, \ldots, R_m\}$ of size m we first transform P into a new partition $Q = \{T_0, T_1, \ldots, T_m\}$ of size $m+1$ as follows For each region R define $\mathrm{Kernel}(R)$ to be $R - U$ (recall that $U = \bigcup_{R\in P} \mathrm{Periphery}(R)$) Q is defined by $T_0 = U$ and $T_i = \mathrm{Kernel}(R_i)$ for $1 \le i \le m$ The resulting structure induces a star topology that is critical in applying the techniques in [Kushner and Chen 1974] $T_0 = U$ is called the *coupling region*, removing the states in T_0 separates the state space into isolated regions, each of which corresponds to a T_i, $i > 0$ Figure 5 illustrates the relationship between the partitions P and Q
2. In the ith iteration, we consider the particular $\bar{\lambda}$ determined at the end of last iteration We decompose the original base-level process into local processes $\{M_{\lambda|_R}|R \in Q\}$, and derive the corresponding local policies $\{\pi_{\lambda|_R}|R \in Q\}$ Let Π_i denote the global policy formed by gluing together these local policies
3. We need to maintain a policy repository of at most $|U|+1$ previously generated global policies at a time As soon as policy Π_i is generated, it replaces some policy Π_j, $j < i$, in the repository The information associated with the current policy repository tells us an upper bound on the gap between the value of the current solution and the optimum This bound allows us to determine whether we have reached the optimum or are within a specified range of the optimum, and whether we should continue for another iteration or terminate and report a final solution
4. If we decide to terminate the computation we generate a policy as a final solution by properly combining the policies in the current policy repository using its associated information, otherwise, we determine a new $\bar{\lambda}$ according to the information asso-

ciated with the current policy repository

Proposition 1 *The iterative method described above improves the solution quality on each iteration, and converges. to an optimal solution* in *a finite number of steps*

For a proof of this proportion and a more detailed description of the algorithm see the longer version of this paper [Dean and Lin, 1995] Our approach to analyse involves (1) reformulating this iterative method in terms of solving large linear programs and (n) applying a reduction to the methods of Kushner and Chen [1974] that solve these large linear programs for Markov decision processes using Dantzig-Wolfe decomposition In the following, we briefly discuss the convergence rate and the time and space complexity of this iterative method

- Empirical experience suggests that the Dantzig-Wolfe method of decomposition upon which our analysis of the iterative method is based converges to within 1-5% of the optimum fairly quickly, although the tail convergence rate can be very slow (see page 325 m [M S Bazaraa 1990]) In other words it is likely that after only a small number of iterations in the iterative method we are able to produce a solution of good quality, hut it may not be worthwhile to continue after reaching 1-5% of the optimum

- The computational task in an iteration is decomposed into two subtasks (i) deriving and solving local processes over local regions as subproblems and (ii) maintaining a policy repository of up to $|U| + 1$ previously generated policies where *V* is the union of Periphery *(R)* for the regions *R* in the original partition *P*

  The computational cost in an iteration is critically affected by the structure of the partition *Q* (I) the maximum number of base-level states in a region in the partition Q, and (n) $|U|$, the total number of base-level states in the coupling region for the partition *Q*

m (ompared with solving the original base-level process as a whole, the first subtask can be achieved more efficiently by applying the standard techniques for Markov decision processes over individual regions This is a natural advantage of decomposition trchniques, which divide large problems into suliprohlems of traclable size

  The second subtask is achieved by maintaining a $(|U| + 1) \times (|U| + 1)$ matrix whose computational efficiency critically depends on the topology of the given partilion *P* The second subtask can be performed efficiently if the size of *V is* relatively small This is the additional cost to pay for decomposition techniques since we need to combine the solutions to subproblems

- In other words this iterative method is promising if the given partition *P* evenly divides the whole state space into many regions, and the number of states in the peripheries of the regions in *P* is small

In the longer version of this paper, we also describe other iterative methods that do not necessarily converge to an optimal solution but allow intuitive interpretation and more computational efficiency We are currently testing these algorithms on a set of benchmark problems Since the discussion is somewhat lengthy and requires some understanding of both Howard's policy iteration [Howard I960] and Bellman 8 value iteration [Bellman 1961] for solving Markov decision processes we refer the interested reader to the longer version of the paper [Dean and Lin, 1995]

6 Related Work

The related work on abstraction and decomposition is extensive In the area planning and search assuming, deterministic action models, there is the work on macro operators [horf, 1985] and hierarchies of state-space operators [Sacerdoti, 1974] [knoblock, 1991] C losely related is the work on decomposing discrete event systems modeled as (deterministic) finite state machines [Zhong and Wonham 1990] [Caines and Wang, 1990]

In the area of reinforcement learning, then, is work on deterministic action models and continuous state spaces [Moore and Atkeson, 1995] and stochastic models and discrete state spaces [Kaelbling, 1993] The hierarcl ncal policy construction method described in Section 4 provides an alternative formulation of Kaelbling s hierarchical learning algorithm [Kaelbling, 1991] and suggests how Moore and Atkeson's parti-game algorithm might be extended to handle discrete state spaces

The analysis hinted at in thih paper and found in the longer version of the paper borrows heavily from lh< work in operations research and combinatorial optimization for representing Markov decision processes as 1m ear programs [D Epenoux 1963] [D erman, 1970] [Rush ner and Kleinman, 1971] and decomposing large sys teint generally [Dantzig and Wolfe, 19G0] [Lasdon, 1970] and Markov decision processes specifically [Kushner and Chen, 1974] The approach described in [Dean el *al* 1993] [Dean *et at* , 1995] represents a special case of the framework presented here, in which the partition consists of singleton sets for all of the states in the envelope and a set for all the states in the complement of the envelope

7 Conclusion

The benefit of decomposition, techniques is that we are able to deal with subproblems of smaller size, the tradeoff is that extra effort is required to combine the solutions to these subproblems into a solution to the original problem The leverage of decomposition techniques is orthogonal to that of standard techniques used to solve the original problem In the case of problem instances of very large size, decomposition techniques are often valuable even if standard polynomial-time algorithms are available

We provide decomposition techniques for Markov decision processes, given an arbitrary partition of the state space into regions Subproblems correspond to local Markov decision processes over regions associated with a A parameter that provides an abstract summary of the interactions among regions We present two methods for combining the solutions to subproblems a hierarchical construction approach and an iterative improvement approach The hierarchical construction method provides a quick solution with an intuitive interpretation The iterative method is guaranteed to converge to an optimal solution in a finite number of iterations For practical purpose*, a small number of iterations should be suffi-(nnt for a solution of near optimal quality

Acknowledgements

We thank (raig Bontiher, Moises Goldszinidt Steve Hanks, David Smith, and Mike Williamson for their comments on the content and presentation of the ideas described in this paper

References

[Bellman I96I] Richard Bellman Adaptivt (ontrol Processes Princeton University Press Princeton New Jersey, 1961

[Boutiher tt al , 1995] Craig Boutiher Richard Dearden and Moises Goldszmidt Exploiting structure in policy construction In Procecdings of the 1995 International Joint Conference on Artificial Intelligence, 1995

[(aines and Wang 1990] Peter E (aines and S Wang (OCOLOG A conditional observer and controller logic for finite machines In Proceedings of tht 29th IEEE Conference on Decision and Control Hawaii 1990

[(hvatal, 1980] Vasek Chvalal / irn-ar Programmmq W H Freeman and (oinpany, 1980

[Danlzig and Wolfe, 1960] George Dantzig and Philip Wolfe Decomposition principle for dynamic programs Operations Research 8(I) 101 111,1900

[E)ean and Kanazawa, 1989] Thomas Dean and Keiji Kanazawa A model for reasoning about persistnce and causation Computational Intelliquence, 5(3) 112-150 1989

[Dean and Lin 1995] Thomas Dean and Shieu-Hong 1 in Decomposition techniques for planning in stochastic domains Technical Report (S-95-08, Brown University Department of Computer Science 1995

[Dean tt al, 1993] Thomas Dean, Leslie Kaelbling, lak Rinnan, and Ann Nicholson Planning with deadlines in stochastic domains In Proceedings AAAI 93 pages 574 579 AAAI 1993

[Dean et al , 1995] Thomas Dean, Leslie Kaelbhng, Jak Kirman aud Ann Nicholson Planning under time constraints in stochastic domains To appear in Artificial Intelligence, 1995

[D'Epenoux, 1963] F D'Epenoux Sur un probleme de production et de stockage dans 1 aleatoire Management Science, 10 98-108, 1963

[Derman, 1970] Cyrus Derinan Finite State Markovtan Decision Processes Cambridge University Press New York, 1970

[Fikes and Nilsson 1971] Richard Fikes and Nils J Nilsson Strip* A new approach to the application of theorem proving to problem solving Artificial Intdligenct, 2 189-208, 1971

[Howard I960] Ronald A Howard Dynamic Program mmg and Markov Processes MIT Press, Cambridge, Massachusetts 1960

[Kaelbhng, 1993] Leslie Park Kaelbhng Hierarchical learning in stochastic domains A preliminary report In Proceedings Tenth International C onftrtnce on Ma chme Learning, 1993

[Knoblock, 199l] Craig A Knoblork Search reduction in hierarchical problem solving In Proceedings 4 4 4/-91 pages 686 691 AAAI, 1991

[Korf, 1985] Richard Korf Macro-operators \ weak method for learning Artificial InUlligtnet 26 35 77 1985

[Kushnei and Chen 1974] Harold J Kushncr and C hing-Hui Chen Decomposition of systems governed by Markov chains IEEE Transactions on Automatic (ontrol AC-19(5) 501-507,1974

[Kushner and Kleinman, 1971] H 1 Kushner and A) Kleinman Mathematical programming and the ton trul of Markov chains International Journal on Con trot 13(5) 801-820, 1971

[Lasdon, 1970] Leon S las-don Optimization Theory for I argt Systems. Macrinllan (ornpany 1970

(/in and Dean 1991] Shreu u-Hong inand I nomas Dean Exploiting locality in temporal reasoning In E Sandewall and C Backstrom edilors (inrent Trends in 1/ Planning, Amsterdam, 1994 10S Press

[Moore and Alkeson, 1995] Andrew W Moore and (linstoplier G Atkeson The parti-game algorithm for variable resolution reinforcement learning in multich mensional state spares To appear in Machine Learning 1995

[MS Be-araa, 1990] H D Sherah MS Bazaraa J I larvis Linear Programming and Ntwork Flours john Wilev & Sons New "York 1990

[Papadnnitriou and Tsitsikhs 1987] C hnstos H Papadimitnou and John N Isltsiklis The complexily of Markov chain decision processes Mathematu of Opt rations Research 12(3)441-4^0 1987

[Puterman, 1994] Martin L Pu term an Maihoi I)tasion Processes John Wiley &. Sons, New York 1994

[Sacerdoh, 1974] Earl Sacerdoti Planning in a hierarchy of abstraction spaces Artificial Intethqence 7 231-272 1974

[Zhong and Wonharn, 1990] H Zhong and W M Wonham On the consistency of hierarchical supervision in discrele-event systems IEEE Transactions on 4utomatic Control, 35(10) 1125-11 14, 1990