Decomposition Techniques for Planning in Stochastic Domains

Thomas Dean and Shieu-Hong Lin

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-95-10
June 1995

Decomposition Techniques for Planning in Stochastic Domains

Thomas Dean[1] Shieu-Hong Lin
Department of Computer Science
Brown University
115 Waterman Street
Providence, RI 02906, USA
Phone: (401) 863-7600
Fax: (401) 863-7657
Email: {tld,shl}@cs.brown.edu

Abstract

This paper is concerned with modeling planning problems involving uncertainty as discrete-time, finite-state 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 alternative, 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 solution. 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 decision problem. A second algorithm iteratively approximates parameters of the local problems to converge to an optimal solution. We show how properties of the specified partition impact on the time and storage required for these algorithms.

Keywords: planning, reasoning under uncertainty, abstraction, decomposition, Markov decision processes

[1]This work was supported in part by a National Science Foundation Presidential Young Investigator Award IRI-8957601, by the Air Force and the Advanced Research Projects Agency of the Department of Defense under Contract No. F30602-91-C-0041, and by the National Science foundation in conjunction with the Advanced Research Projects Agency of the Department of Defense under Contract No. IRI-8905436.

Contents

1 Introduction **4**
- 1.1 Factoring Large State Spaces 4
- 1.2 Decomposing High-Dimensional State Spaces 4
- 1.3 Combining Local Solutions 6
- 1.4 Overview of the Paper 7

2 Markov Decision Processes **7**

3 Decomposing Markov Decision Processes **8**

4 Abstraction and Hierarchical Policy Construction **10**
- 4.1 Abstract Decision Processes 11
- 4.2 Hierarchical Policy Construction 11

5 The Iterative Approximation Approach **13**

6 Iterative Approximation and Linear Programming **16**
- 6.1 Linear Programming and the Dantzig-Wolfe Decomposition Principle 16
  - 6.1.1 The Master Problem in the DW Decomposition 17
  - 6.1.2 The Subproblems in the DW Decomposition 18
  - 6.1.3 Conducting an Iteration of the DW Decomposition . . . 19
- 6.2 The DW Decomposition and Markov Decision Processes 20
  - 6.2.1 Formulating Markov Decision Processes as Linear Programs 20
  - 6.2.2 The DW Approach in Solving Markov Decision Processes 21

7 The Details of the Iterative Approximation Method **22**
- 7.1 Relationship between the DW Approach and Iterative Approximation 23
- 7.2 Overview of the Iterative Method 24
- 7.3 Partitioning a State Space into Local Regions 24
- 7.4 The Abstract Action Space 25
- 7.5 Deriving Local Policies 25
  - 7.5.1 Deriving the Local Policies over the Kernels 25
  - 7.5.2 Deriving the Local Policies over the Peripheries 28
- 7.6 Further Information Associated with Local Policies 28
- 7.7 Updating the Policy Repository 30
- 7.8 Bounds, Stopping Criteria, and New Abstract Actions 31
- 7.9 The Initialization and Termination of the Iterative Method . . . 32
  - 7.9.1 Initializing the Policy Repository and Abstract Actions . 32
  - 7.9.2 Generating a Solution Policy from the Policy Repository 33

8 A Robot Navigation Example **34**
8.1 Modeling a Robot Navigation Problem as a Markov Decision Process 34
8.2 Exploiting the Locality Structure 36
8.3 Using the Hierarchical Construction Approach 37
8.4 Using the Iterative Approximation Approach 38

9 More Theoretical Analysis in Robot Navigation Domains **39**
9.1 Modeling the Computation Complexity of the Standard Techniques 40
9.2 Computation Efficiency of the Hierarchical Construction Approach 41
9.3 Computation Efficiency of the Iterative Approximation Approach 41

10 Related work **42**

11 Conclusion **42**

List of Figures

1 Three views of a three-dimensional state space with the corresponding local structure of space shown to the right. The top view represents the unstructured state space, the middle view represents an abstraction obtained by projection, and the bottom view represents a decomposition obtained by partitioning the states into aggregate states. 5

2 Region-by-region dimensionality reduction. A three-dimensional space is represented as the union of two-dimensional abstract subspaces shaded dark gray. 5

3 Boundary (light gray) and periphery (darker gray) states of region R 9

4 R is adjacent to S ($R \rightsquigarrow S$) 9

5 Abstract Markov decision process 12

6 Abstract action for moving from R to S. Base-level action for moving from i to j. 12

7 Relationship between the partitions P and Q 14

8 The relationship between actual and biased action cost 27

9 A building of three floors and two elevators 34

10 Uncertainty in action outcomes 35

11 An $N \times N$ area tessellated into N^2 squares and M^2 regions where $N = 8$ and $M = 4$ 39

1 Introduction

We are concerned with solving planning problems posed as Markov decision processes. Specifically, given a dynamical model described as a stochastic process (*e.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.

1.1 Factoring Large State Spaces

Large state spaces present a number of challenges. To 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.[2] Compact encodings for stochastic processes can be achieved for many applications using factored state-space representations, where the size of the model is usually logarithmic in the size of the state space [6]. Similarly, policies for large factored state spaces can often be efficiently encoded using decision trees that branch on state variables. 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 time bounded by some small factor of the problem and solution size.

A factored state-space representation with n boolean state variables represents an n-dimensional state space with $O(2^n)$ states. We assume that all of the state variables are relevant in at least some portion of the state 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 state variables are relevant in all portions of the state space (*e.g.,* when you are planning to take a walk in southern Florida you can neglect the possibility of snow). Figure 1 illustrates three views of a multi-dimensional state space. The bottom view in which the state space is partitioned into aggregate states is the view we are most interested in.

1.2 Decomposing High-Dimensional State Spaces

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

[2] In artificial intelligence, such factored representations are often called *intensional* representations. Propositions representing fluents in STRIPS operators [9] correspond to state variables in a factored state-space representation.

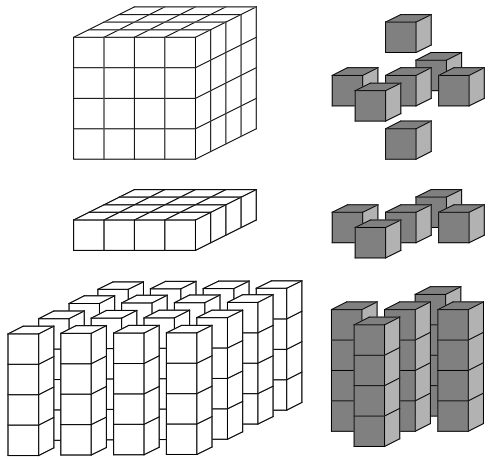

Figure 1: Three views of a three-dimensional state space with the corresponding local structure of space shown to the right. The top view represents the unstructured state space, the middle view represents an abstraction obtained by projection, and the bottom view represents a decomposition obtained by partitioning the states into aggregate states.

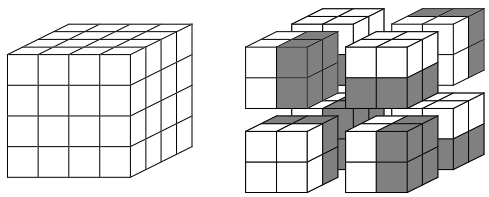

Figure 2: Region-by-region dimensionality reduction. A three-dimensional space is represented as the union of two-dimensional abstract subspaces shaded dark gray.

than $m2^r$. The problem of automatically constructing such a partition is *not* addressed in this paper, but see [17] for some relevant techniques. Figure 2 illustrates how a three-dimensional state space might be represented as the union of two-dimensional abstract subspaces.

There exist methods for computing policies that are polynomial in the size of the state and action spaces [20] [21], 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 in terms of smaller Markov decision processes over the subspaces of the individual regions, (ii) solve each of these subproblems, and then (iii) 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 costs accrued while in that portion of the 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 various robot navigation problems.

We associate a set of topologically motivated parameters with each 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. By 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 parameter 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 iterative approximation of the optimal parameter values, *i.e.,* 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 of region R, and solve the resulting Markov decision process to obtain a local policy. By examining the resulting local policies, 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 it is therefore 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 describes 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 of R. Section 4 presents the hierarchical 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 sketches the method of the iterative approximation and briefly addresses issues concerning convergence, optimality, and complexity. Section 6 draws the connection between the iterative approximation method and the Dantzig-Wolfe decomposition for solving linear programs. Section 7 provides the details of the iterative approximation method sketched in Section 5. In Section 8, we describe the use of decomposition techniques in robot navigation domains through an example. In Section 9, we theoretically analyze the computation complexity of the use of decomposition techniques in robot navigation domains, and indicates when it is advantageous to use decomposition techniques.

2 Markov Decision Processes

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

$$p_{ij}(a) = \Pr(X_t = j | X_{t-1} = i, A_{t-1} = a) \quad i, j \in \Omega_X \quad a \in \Omega_A$$
$$c_{ij}(a) = \mathrm{C}(X_t = j | X_{t-1} = i, A_{t-1} = a) \quad i, j \in \Omega_X \quad a \in \Omega_A$$

where $\Pr(.|.)$ is a conditional probability distribution and $\mathrm{C}(.|.)$ 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,

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

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 [1] 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 [10] 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 $O(N^3)$ iterations. Puterman [21] 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 $\mathcal{M}$. 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.

3 Decomposing Markov Decision 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

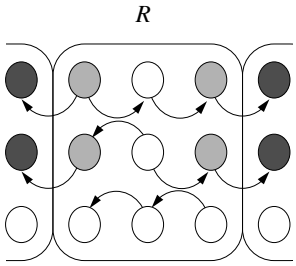

Figure 3: Boundary (light gray) and periphery (darker gray) states of region R

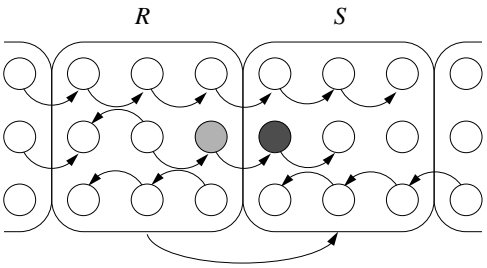

Figure 4: R is adjacent to S ($R \rightsquigarrow S$)

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.

$$\text{Periphery}(R) = \{j \,|\, j \notin R \wedge \exists i \in R, a \in \Omega_A, p_{ij}(a) > 0\}$$

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.

$$\text{Boundary}(R) = \{i | i \in R \wedge \exists j \notin R, a \in \Omega_A, p_{ij}(a) > 0\}$$

In Figure 3, the boundary states are shaded light gray and the periphery states are shaded darker gray.

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) $\cap\, S \neq \emptyset$ (see Figure 4).

Next, we introduce a set of parameters that we will use to model interactions among regions. Let $U = \bigcup_{R \in P} \text{Periphery}(R)$, and β_i for each $i \in U$ denote a real-valued parameter. Let $\vec{\beta} \in \mathbf{R}^{|U|}$ denote a vector of all such β_i

parameters, and $\bar{\beta}|_R$ denote a subvector of $\bar{\beta}$ composed of β_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 β_i serves as a measure of the expected cumulative cost of starting from a periphery state, and $\bar{\beta}|_R$ provides an abstract summary of how the other regions affect R.

Given a particular $\bar{\beta}$, 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 $\bar{\beta}|_R$ of $\bar{\beta}$, we define a Markov decision process $\mathcal{M}_{\bar{\beta}|_R} = (R \cup \text{Periphery}(R), \Omega_A, q, k)$ and the corresponding local policy $\pi_{\bar{\beta}|_R}$ as follows.

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

$\mathcal{M}_{\bar{\beta}|_R}$ is the subproblem we associate with region R given $\bar{\beta}|_R$ as an abstract summary of R's interaction with the other regions. $\pi_{\bar{\beta}|_R}$ is the solution to the subproblem $\mathcal{M}_{\bar{\beta}|_R}$. A particular $\bar{\beta}$ 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 $\mathcal{M}$. If $\beta_i = \text{E}_{\pi^*}(\Sigma_\gamma|i)$, 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 $\text{E}_{\pi^*}(\Sigma_\gamma|i)$ for all $i \in U$.

4 Abstraction and 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 $R \in P$ can be considered as an *abstract state*. A particular local policy $\pi_{\bar{\beta}|_R}$ on region R can be considered as an *abstract action* on R, which reflects our bias toward different periphery states described by $\bar{\beta}|_R$. Abstract actions for stochastic domains are the rough equivalent of macro operators for deterministic domains. The abstract action $\pi_{\bar{\beta}|_R}$ indicates how to act optimally in region R if the interactions with the other regions are captured in $\bar{\beta}|_R$. For example, a large value for β_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 $\mathcal{F} = \{\pi_{\bar{\beta}|_R} | R \in P, \bar{\beta} \in \mathbf{R}^{|U|}\}$.

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

$$\begin{aligned} p'_{RS}(\pi_{\bar{\beta}|_R}) &= \frac{1}{|\text{Boundary}(R)|} \left[\sum_{i \in \text{Boundary}(R)} \varphi_i \right] \\ \varphi_i &= \left[\sum_{j \in S \cap \text{Periphery}(R)} p_{ij} \right] + \sum_{j \in R} p_{ij} \varphi_j \end{aligned}$$

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

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

$$\begin{aligned} c'_{RS}(\pi_{\bar{\beta}|_R}) &= \frac{1}{|\text{Boundary}(R)|} \left[\sum_{i \in \text{Boundary}(R)} \vartheta_i \right] \\ \vartheta_i &= \left[\sum_{j \in S \cap \text{Periphery}(R)} p_{ij} c_{ij} \right] + \sum_{j \in R} p_{ij} [c_{ij} + \vartheta_j] \end{aligned}$$

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

The resulting abstract decision process is then defined by $\mathcal{M}_{\bar{\beta}} = (P, \mathcal{F}, p', c')$. It is important to note that $\mathcal{M}_{\bar{\beta}}$ 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

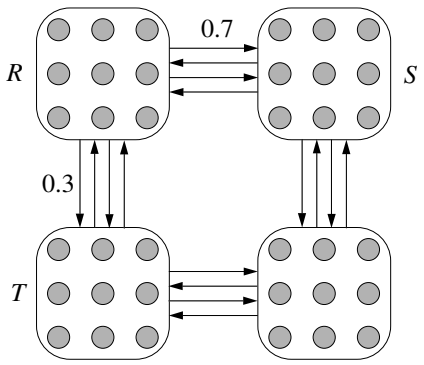

Figure 5: Abstract Markov decision process

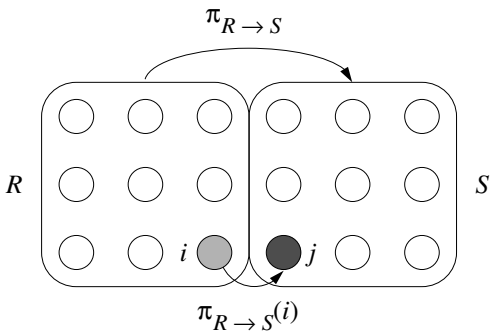

Figure 6: Abstract action for moving from R to S. Base-level action for moving from i to j.

construct a local policy $\pi_{R \to S}$ by setting $\beta_i = 0$ for $i \in S \cap \text{Periphery}(R)$ and setting $\beta_i = \kappa$ for $i \in \text{Periphery}(R) - S$, 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 goal states, we add an additional action in which all the peripheral states get $\beta_i = \kappa$ and all of the goal states are made into sinks with $k_{ii} = 0$. The abstract decision process is $(P, \mathcal{F}_{\rightsquigarrow}, p', c')$ where $\mathcal{F}_{\rightsquigarrow} = \{\pi_{R \to S} | R, S \in P \wedge R \rightsquigarrow S\}$ 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 5 illustrates an example in which $p'_{RS}(\pi_{\bar{\beta}|_R}) = 0.7$ and $p'_{RT}(\pi_{\bar{\beta}|_R}) = 0.3$. 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 [8].

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

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 $\Pi : P \to \mathcal{F}_{\rightsquigarrow}$.

4. To determine the action to take in base-level state i, determine $R \in P$ such that $i \in R$ and take action $\Pi(R)(i)$.

Figure 6 illustrates the difference between abstract actions defined as local policies ($\pi_{\bar{\beta}|_R} \in \mathcal{F}$) and base-level actions ($\pi_{\bar{\beta}|_R}(i) \in A$). The above algorithm for constructing and solving abstract processes can be applied recursively and hence applies to hierarchical partitions.

Hierarchical policy construction does not guarantee to produce an optimal policy; however, it does so relatively efficiently and has an intuitive interpretation that makes it suitable for robot navigation domains. For a simple partition of the state space with no aggregation within regions, standard algorithms [21] on the base-level state space would be dominated by a factor quadratic in the size of the state space ($|\Omega_X|$), 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 The Iterative Approximation Approach

Given a particular $\bar{\beta}$, the base-level process is decomposed into local processes, $\{\mathcal{M}_{\bar{\beta}|_R}\}$. By solving these local processes, we derive the corresponding local policies (abstract actions) $\{\pi_{\bar{\beta}|_R}\}$. A global policy, Π, to the base-level process is then derived by directly gluing these local policies together. The quality of Π critically depends on the choice of $\bar{\beta}$. Instead of relying on a particular $\bar{\beta}$, we can successively modify $\bar{\beta}$ and proceed through multiple iterations of decomposition and localized computation. Let $\Pi^{(i)}$ denote the global policy determined by a particular $\bar{\beta}$ adopted on the ith iteration of computation. Rather than simply using one of the global policy $\Pi^{(i)}$ as the final solution policy, we may also cleverly combine these global polcies, $\{\Pi^{(i)}\}$, to generate an even better solution policy. There are several issues in realizing this iterative approximation framework:

- how to efficiently maintain useful information about the global policies, $\{\Pi^{(i)}\}$, derived on previous iterations,

- how to combine $\{\Pi^{(i)}\}$ to generate a solution policy if we need to terminate the computation at the end of a specific iteration,

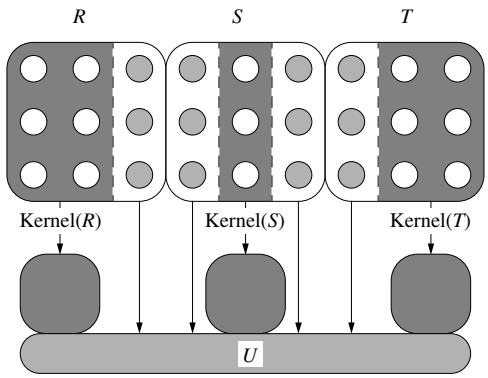

Figure 7: Relationship between the partitions P and Q

- how to modify $\bar{\beta}$ 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, and
- how to determine it is time to terminate the computation when the solution is optimal or within some specified tolerance of the optimum.

In this paper, we focus on a particular iterative method that resolves these issues. This iterative method is based on a reduction to the methods of Kushner and Chen [15] that demonstrate how to solve Markov decision processes as linear programs using the Dantzig-Wolfe decomposition principle [4]. The details of the material presented in this section depend on some understanding of linear programming [3] and methods for decomposing and solving large systems [16]. We sketch the iterative method in the following, and refer the readers to the following two sections for more details.

1. Given an arbitrary partition $P = \{R_1, \ldots, R_h\}$ of size h, we first transform P into a new partition $Q = \{U, K_1, \ldots, K_h\}$ of size $h+1$ as follows.
   - U is $\bigcup_{R_i \in P} \text{Periphery}(R_i)$, which represents the peripheries of the regions R_i's.
   - For each region R_i, we have $K_i = \text{Kernel}(R_i) = R_i - U$, which represents the kernel of region R_i.

   This resulting partition Q induces a star topology that is critical in applying the techniques in [15]. U is the *coupling region* in the new partition Q; removing the states in U separates the state space into isolated regions, each of which corresponds to a K_i, $1 \leq i \leq h$. In other words, the kernels of the regions R_i's in P are encapsulated and separated from one another by their peripheries, whose union is U. Figure 7 illustrates the relationship between the partitions P and Q.

2. On the ith iteration, we consider a particular $\bar{\beta}$ determined at the end of last iteration. We decompose the original base-level process into local processes $\{\mathcal{M}_{\bar{\beta}|_R} | R \in Q\}$, and derive the corresponding local policies $\{\pi_{\bar{\beta}|_R} | R \in Q\}$. Let $\Pi^{(i)}$ 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, $\{\Pi^{(i)}\}$, at a time. As soon as policy $\Pi^{(i)}$ is generated, it replaces some policy $\Pi^{(j)}$, $j < i$, in the repository. The information associated with the current policy repository also allows us to determine an upper bound on the gap between the value of the current solution and the optimum. This bound allows us to decide whether we have reached the optimum or within a specified range of the optimum, or we should continue for another iteration of computation.

4. If we decide to terminate the computation, we are able to generate a final solution policy by properly combining the policies, $\{\Pi^{(i)}\}$, in the current policy repository using its associated information; otherwise, we are able to determine a new $\bar{\beta}$ to go through another iteration of computation.

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 iterations.*

Proof. The iterative method is equivalent to (i) adopting the partition Q, and then (ii) applying the methods of Kushner and Chen [15] that solve Markov decision processes as large linear programs using the Dantzig-Wolfe decomposition principle, which have the properties described above. A more detailed description of the algorithm and its correspondence to the methods of Kushner and Chen [15] is provided in the next two sections. □

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 decomposition principle, which the analysis of the iterative method is based on, allows convergence to within 1–5% of the optimum fairly quickly, although the tail convergence rate can be very slow (see page 325 in [19]). In other words, it is likely that after a few number of iterations in the iterative method we are able to produce a solution of good quality, but 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 Markov decision processes over local regions as subproblems, and (ii) maintaining a policy repository of up to $|U| + 1$ previously generated policies, where U is the union of Periphery(R) for the regions R in the original partition P.

The computation efficiency of each iteration is critically affected by the structure of the given partition P: (i) $|\text{Kernel}(R_i)|$, the number of base-level states in the kernel of each region R_i in the partition P, and (ii) $|U|$, the total number of base-level states in the peripheries of the regions in the partition P.

- Compared 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 techniques, which divide large problems into subproblems of tractable sizes.

  The second subtask can be achieved by maintaining a $(|U|+1) \times (|U|+1)$ matrix, whose computational efficiency critically depends on the topology of the given partition P. The second subtask can be performed efficiently if the size of U 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 divides the whole state space into many small regions, and the number of the states in the peripheries of the regions in P is also small.

6 Iterative Approximation and Linear Programming

In this section, we explore the relationship between the iterative approximation framework described in Section 5 and the use of the Dantzig-Wolfe decomposition principle in solving Markov decision processes as linear programs. This relationship then provides interesting insight into the iterative approximation framework. In the remainder of this paper, we abbreviate the Dantzig-Wolfe decomposition principle as the *DW decomposition*, and refer to the work of Kunshner and Chen [15] as the *DW approach* for Markov decision processes.

6.1 Linear Programming and the Dantzig-Wolfe Decomposition Principle

To cope with linear programs of very large sizes, decomposition principles [16] that divide a large linear program into many correlated linear programs of smaller sizes have been well studied, among which the DW decomposition may be the most well known. In the following, we describe the general form of the DW decomposition without assuming additional structure in linear programs.

We refer the readers to [16] [19] for more details about linear programming and the DW decomposition.

Consider a linear program of standard form

$$\left.\begin{array}{rcl} \min z & = & cX \\ AX & = & b \\ X & \geq & 0 \end{array}\right\} \ldots(1),$$

where X is a $N \times 1$ column vector composed of N variables $x_1, x_2, \ldots, x_N$; c is a $1 \times N$ row vector of N scalar values representing a cost function of X; A is a $M \times N$ *constraint matrix* representing M linear constraints on X; and $X \geq 0$ represents sign constraints enforcing $x_1, x_2, \ldots, x_N$ to be nonnegative.

Using the DW decomposition, we can arbitrarily partition the set of M linear constraints $AX = b$ into two smaller sets of linear constraints $A_1X = b_1$ and $A_2X = b_2$. Let m be the number of linear constraints in $A_1X = b_1$. Rather than directly solving the linear program in (1), the DW decomposition implicitly reformulate the linear program in (1) as a master problem concerning the m linear constraints in $A_1X = b_1$, and iteratively generate a series of subproblems concerning the $M - m$ linear constraints in $A_2X = b_2$ to attain an optimal solution of the linear program in (1).

6.1.1 The Master Problem in the DW Decomposition

Note that all feasible solutions to $AX = b$ also satisfy $A_2X = b_2$. Therefore each feasible solution X to $AX = b$ can be properly represented as

$$X = \sum_{1 \leq i \leq r} \lambda_i X^{(i)} + \sum_{1 \leq j \leq t} \bar{\lambda}_j \bar{X}^{(j)},$$

where

- X^i's, $1 \leq i \leq r$, and $\bar{X}^j$'s, $1 \leq j \leq t$ are $N \times 1$ column vectors representing the finite number of extreme points and extreme rays [3] [16] [19] respectively of the set $\{x | A_2X = b_2\}$, and
- λ_i's and $\bar{\lambda}_j$'s must satisfy the following constraints

$$\left.\begin{array}{rcll} A_1(\sum_{1 \leq i \leq r} \lambda_i X^i + \sum_{1 \leq j \leq t} \bar{\lambda}_j \bar{X}^j) & = & b_1 & \\ \sum_{1 \leq i \leq r} \lambda_i & = & 1 & \\ \lambda_i \geq 0, & & \forall i = 1 \ldots r & \\ \bar{\lambda}_j \geq 0, & & \forall j = 1 \ldots t. & \end{array}\right\} \ldots(2).$$

[3] The set of extreme rays $\{\bar{X}^1, \ldots, \bar{X}^t\}$ is any maximal set of linearly independent solutions to $A_2X = 0$.

We can factor A_1 into the summations in the first equation in (2), which renders (2) as a set of linear constraints on λ_i's and $\bar{\lambda}_j$'s. Similarly, for the cost function $z = cX$ over X we can rewrite it as

$$cX = c(\sum_{1 \leq i \leq r} \lambda_i X^{(i)} + \sum_{1 \leq j \leq t} \bar{\lambda}_j \bar{X}^{(j)}),$$

which renders a cost function over λ_i's and $\bar{\lambda}_i$'s when we factor c into the summations in the above equation. Let m be the number of linear constraints in $A_1X = b_1$. We denote $A_1X^{(i)}$ as q_i, $A_1\bar{X}^{(j)}$ as $\bar{q}_j$, $cX^{(i)}$ as d_i and $c\bar{X}^{(j)}$ as $\bar{d}_j$, where q_i and $\bar{q}_j$ are m column vectors while d_i and $\bar{d}_j$ are scalar values. This allows us to rewrite the linear program in (1) as a linear program over the nonnegative variables λ_i's and $\bar{\lambda}_j$'s in the following form

$$\left.\begin{array}{rcl} \min z & = & \sum_{1 \leq i \leq r} \lambda_i d_i + \sum_{1 \leq j \leq t} \bar{\lambda}_j \bar{d}_j \\ \sum_{1 \leq i \leq r} \lambda_i q_i + \sum_{1 \leq j \leq t} \bar{\lambda}_j \bar{q}_j & = & b_1 \\ \sum_{1 \leq i \leq r} \lambda_i & = & 1 \\ \lambda_i \geq 0, & & \forall i = 1 \ldots r \\ \bar{\lambda}_j \geq 0, & & \forall j = 1 \ldots t. \end{array}\right\} \ldots (3).$$

We refer to the linear program in (3) as the *master problem* in the DW decomposition in solving the linear program in (1). Let m be the number of linear constraints in $A_1X = b_1$. Note that we have $m + 1$ linear constraints in the master problem in addition to those sign constraints that enforce λ_i's and $\bar{\lambda}_j$'s to be nonnegative. Theoretically, we can directly apply the simplex method to solve the master problem, where the simplex method iteratively updates a basis of $m+1$ columns of the constraint matrix of the master problem in (3). On each iteration, a profitable column is selected to enter the basis and replace an old one in the current basis, which corresponds to an λ_i or an $\bar{\lambda}_j$ variable associated with an extreme point or an extreme ray of the set $\{X | A_2X = b_2\}$. In this way, the solution quality can be iteratively improved until an optimal solution is reached.

6.1.2 The Subproblems in the DW Decomposition

In order to solve the master problem using the simplex method, we need to determine a profitable column corresponding to an extreme point or an extreme ray of the set $\{X | A_2X = b_2\}$ to iteratively update the basis. However, it is infeasible to exhaustively enumerate all these extreme points and extreme rays in general. To cope with this difficulty, a *subproblem* is devised according to the current *simplex multipliers* [4] [16] [19] associated with the $m + 1$ linear constraints in the master problem (3). By solving the subproblem, we can find

[4] On each iteration of the simplex method, we can determine for each linear constraint an associated simplex multiplier.

an extreme point or an extreme ray that gives us a profitable column to enter the basis.

Let $\bar{\beta}$ denote a $1 \times m$ row vector composed of the simplex multipliers associated with the m linear constraints

$$\sum_{1 \le i \le r} \lambda_i q_i + \sum_{1 \le j \le t} \bar{\lambda}_j \bar{q}_j = b_1.$$

Let β_0 denote the simplex multiplier associated with the linear constraint

$$\sum_{1 \le i \le r} \lambda_i = 1.$$

The corresponding subproblem is

$$\left.\begin{array}{rcl} \min z & = & (c - \beta A_1)X \\ A_2 X & = & b_2 \\ X & \ge & 0 \end{array}\right\} \ldots (4).$$

For such a subproblem, we use z^*_β to denote the optimum solution of the subproblem, and δ_β to denote the quantity $\beta_0 - z^*_\beta$.

6.1.3 Conducting an Iteration of the DW Decomposition

On each iteration, we conduct the following steps to either (i) derive an entering column to enter the basis of the master problem and go through another iteration, or (ii) determine that the current solution for the master problem is optimal or within a specified tolerance ϵ of the optimum, report the current solution, and terminate the computation.

- Set up a subproblem according to the current simplex multipliers of the master problem. Solve the subproblem to determine an optimal solution z^*_β of the subproblem and $\delta_\beta = \beta_0 - z^*_\beta$.
-
  - If $\delta_\beta = \infty$, the cost function $z = (c - \beta A_1)X$ is unbounded along an extreme ray $\bar{X}^{(j)}$ and we can find a corresponding column

    $$\begin{bmatrix} A_1 \bar{X}^{(j)} \\ 0 \end{bmatrix} = \begin{bmatrix} \bar{q}^{(j)} \\ 0 \end{bmatrix}$$

    to enter the basis.
  - If δ_β is finite and greater than ϵ, z^* must be achieved by an extreme point $X^{(i)}$ and we can find a corresponding column

    $$\begin{bmatrix} A_1 X^{(i)} \\ 1 \end{bmatrix} = \begin{bmatrix} q^{(i)} \\ 0 \end{bmatrix}$$

to enter the basis.

In both situations, we determine an entering column to enter the basis. We then (i) update the current basis through an pivot operation according to the selected entering column, (ii) determine a new solution to and new simplex multipliers of the master problem, and (iii) go through another iteration of the DW decomposition.

- If δ_β is no more than ϵ, we can terminate the computation and report the current solution of the master problem as the final solution. In this situation, the current solution is no greater than ϵ plus the optimum of the master problem. In particular, if δ_β is zero or negative, we have reached an optimal solution of the master problem.

Both the master problem and the subproblems center around the use of simplex method in solving the linear program in (3). We refer the readers to [19] for more details about (i) cycling prevention and (ii) the initialization of the simplex method using big-M method. We briefly summarize two important properties about the DW decomposition that are useful in the remainder of this paper.

Proposition 2 *Using the DW decomposition, the solution quality is improved iteratively and converge to an optimum solution in a finite number of iterations.*

Proposition 3 *For a subproblem in the DW decomposition, the value of the current solution is less than or equal to the value of an optimal solution plus* δ_β *where* $\delta_\beta = \beta_0 - z^*_\beta$.

6.2 The DW Decomposition and Markov Decision Processes

It is well known that we can solve Markov decision processes as linear programs [7] [8] [14] [21] using standard techniques like the simplex method. Kushner and Chen [15] investigate the use of the DW decomposition in solving Markov decision processes as linear programs. In the following, we briefly introduce their results, which provide interesting insight into the iterative approximation framework described in Section 5.

6.2.1 Formulating Markov Decision Processes as Linear Programs

Let $\mathcal{M} = (\Omega_X, \Omega_A, p, c)$ be a Markov decision process with finite state space Ω_X, finite action space Ω_A, state transition matrix p, and cost matrix c. For

convenience, let $\Omega_X = \{1, 2, \ldots, N\}$. X_t (A_t) is a variable indicating the state (action) at time t.

$$p_{ij}(a) = \Pr(X_t = j | X_{t-1} = i, A_{t-1} = a) \quad i, j \in \Omega_X \quad a \in \Omega_A$$
$$c_{ij}(a) = \mathrm{C}(X_t = j | X_{t-1} = i, A_{t-1} = a) \quad i \in \Omega_X \quad a \in \Omega_A$$

where $\Pr(.|.)$ is a conditional probability distribution and $\mathrm{C}(.|.)$ is a real-valued cost function. A *policy* π is a function mapping states to actions $\pi : \Omega_X \rightarrow \Omega_A$. We consider two kinds of performance criteria, namely (i) expected cumulative cost to reach a specified goal and (ii) expected discounted cumulative cost.

We further define $Cost(i, a) = \sum_j p_{ij}(a) c_{ij}(a)$. For the criterion of expected cumulative 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 at some target states. Assuming that the target can be reached with probability 1 from any other state, we can reformulate $\mathcal{M} = (\Omega_X, \Omega_A, p, c)$ as the following linear program [7] [14]:

$$\left.\begin{array}{rcll} \min z & = & \sum_{i,a} Cost(i,a) E_{i,a} & \\ \sum_a E_{i,a} & = & \xi_i + \sum_{j,a} P_{ji}(a) E_{j,a}, & \forall i \in \Omega_X \\ E_{i,a} & \geq & 0, & \forall i \in \Omega_X, \forall a \in \Omega_A \end{array}\right\} \ldots (5),$$

where $E_{i,a}$ denote the average number of time that action a is taken in state i, and ξ_i is the probability that state i is the initial state.

Similarly, we have the following linear program for the criterion of expected discounted cumulative cost [7] [14].

$$\left.\begin{array}{rcll} \min z & = & \sum_{i,a} Cost(i,a) E_{i,a} & \\ \sum_a E_{i,a} & = & \xi_i + \gamma \sum_{j,a} P_{ji}(a) E_{j,a}, & \forall i \in \Omega_X \\ E_{i,a} & \geq & 0 & \forall i \in \Omega_X, \forall a \in \Omega_A \end{array}\right\} \ldots (6),$$

where $E_{i,a}$ denote the discounted average number of time that action a is taken in state i, and ξ_i is the probability that state i is the initial state.

Given a solution to (5) or (6), we can determine a nondeterministic policy where $E_{i,a} / \sum_a E_{i,a}$ is the probability of mapping state i to action a.

6.2.2 The DW Approach in Solving Markov Decision Processes

Kushner and Chen [15] investigate the use of the DW decomposition in solving Markov decision processes formulated as the linear programs in (5) and (6). Given a Markov decision process $\mathcal{M} = (\Omega_X, \Omega_A, p, c)$, the following kind of partition $Q = \{U, K_1, \ldots, K_h\}$ is required in applying their techniques:

- For any two states i and j in two different K_l's, $1 \leq l \leq h$, both $p_{ij}(a)$ and $p_{ji}(a)$ are zero for any action a in Ω_A.

- The states in two different K_l's, $1 \leq l \leq h$, can only communicate with one another through the states in U. Ω_X divides into isolated pieces K_l's, $l \geq 1$ when we remove U from Ω_X.

Note that each state in Ω_X contributes a linear constraint to the linear programs in (5) or (6). Given a partition $Q = \{U, K_1, \ldots, K_h\}$ as required, we can partition the set of linear constraints into $A_1X = b_1$ and $A_2X = b_2$, where (i) $A_1X = b_1$ composed of the constraints contributed by the states in U and (ii) $A_2X = b_2$ composed of the constraints contributed by the states in $K_l, 1 \leq l \leq h$. This gives us a master problem involving the states in U and subproblems involving the states in K_l's.

Instead of solving subproblems directly, their dual linear programs are considered. Due to the special structure of the partition Q and the linear programs in (5) or (6), the dual linear program of a given subproblem can be reduced to the local Markov decision processes over individual regions K_l's, $1 \leq l \leq h$ rather than the entire state space Ω_X. We briefly summarize the use of DW approach in solving Markov decision processes as linear programs in the following and refer the readers to [15] for more details.

1. Choose a partition $Q = \{U, K_1, \ldots, K_h\}$ such that Ω_X divides into isolated pieces K_l's, $1 \leq l \leq h$ when we remove U from Ω_X.

2. Derive the simplex multipliers associated with the master problem that is concerned with U, and determine the corresponding subproblem.

3. Reduce the subproblem to local Markov decision processes over K_l's, $1 \leq l \leq h$, and solve these Markov decision processes.

4. Manipulate the solutions to the local Markov decision processes derived in step 3 to generate an entering column for the master problem. Then update the basis and the simplex multipliers accordingly.

5. Check the stopping criteria. If the stopping criteria are met, stop and report a solution; otherwise, go to step 2.

7 The Details of the Iterative Approximation Method

In this section, we provide the details of the iterative method sketched in Section 5. We establish the relationship between the iterative method and the use of the DW decomposition in solving Markov decision processes as linear programs. This relationship then provides interesting insight into the iterative approximation framework, and also demonstrate the convergence of the iterative method described in Section 5.

7.1 Relationship between the DW Approach and Iterative Approximation

Consider the steps sketched in Section 6.2 about the DW approach that solves Markov decision processes as linear programs using the DW decomposition. They can be well interpreted as an iterative method as we sketched in Section 5. This demonstrates that the iterative method converges to the optimum in a finite number of iterations.

- Given a Markov decision process $\mathcal{M} = (\Omega_X, \Omega_A, p, c)$ and an arbitrary partition $P = \{R_1, \ldots, R_h\}$ of the state space Ω_X, there is always a closely related partition that allows the use the DW approach. As depicted in Figure 7, $U = \bigcup_{R_l} \text{Periphery}(R_l)$, which is the union of the peripheries of the regions R_i's, and each $K_l = \text{Kernel}(R_l)$, which is the kernel of each individual region R_l, $1 \leq l \leq h$, naturally yield such a partition $Q = \{U, K_1 \ldots, K_h\}$. When we remove U from Ω_X, Ω_X divides into disjoint pieces K_l's, $1 \leq l \leq h$.

- Using such a partition $Q = \{U, K_1 \ldots, K_h\}$ in the DW approach, the number of linear constraints in the master problem is $|U|+1$. The simplex multipliers associated with this master problem can be interpreted as the β_i's assigned to each state i in the peripheries of the regions in P, which determine the the $\bar{\beta}$ vector described in Section 3. On each iteration, such a $\bar{\beta}$ vector determine a subproblem. that is further reduced to smaller Markov decision processes over K_l's, $1 \leq l \leq h$. Each of these smaller Markov decision processes can be interpreted as the kind of local Markov decision process $\mathcal{M}_{\bar{\beta}|K_l}$, $1 \leq l \leq h$, described in Section 3.

- In Step 2 and Step 3 of the DW approach, local Markov decision processes over K_l's, $1 \leq l \leq h$ are derived and solved according to the current simplex multipliers. In Step 4 the solutions to these local Markov decision processes in turn allow us to update the simplex multipliers and then reiterate Step 2 and Step 3. In this way, the DW approach provides a control mechanism to iteratively solve the local Markov decision processes and update the $\bar{\beta}$ vector to improve the solution quality.

- In other words, the DW approach that solves Markov decision processes as linear programs can be interpreted as a particular iterative method. According to Proposition 2 and Proposition 3 in Section 6.1, the DW decomposition iteratively improves the solution quality of a linear program and converges to an optimal solution in a finite number of iterations. Therefore the corresponding iterative method also converges to an optimal solution of the corresponding Markov decision process in a finite number iterations.

7.2 Overview of the Iterative Method

By appropriately interpreting the DW approach in [15] under the iterative approximation framework, it yields the following iterative method.

1. Given a Markov decision process $\mathcal{M} = (\Omega_X, \Omega_A, p, c)$, we choose an arbitrary (but fixed) partition P of the state space Ω_X in advance which divides the state space into disjoint regions.

2. On the tth iteration, we decompose the original Markov decision process into smaller Markov decision processes over the kernels of the regions in P, and derive a policy $\Pi^{(t)}$ by gluing together the policies of these smaller Markov decision processes.

3. The information associated with $\Pi^{(t)}$ also tells us (i) a bound on the value of the current solution, and (ii) whether we should go through another iteration of Step 2.

4. Let m be the number of the states in the peripheries of the regions in partition P. We need to keep a policy repository of at most $m + 1$ such $\Pi^{(t)}$'s at a time. As soon as policy $\Pi^{(t)}$ is generated, it properly replaces one of the policy $\Pi^{(i)}, i < t$, in the repository. The current solution is a combination of such policies $\Pi^{(t)}$'s stored in the current policy repository.

5. If we need to go through another iteration, the information associated with the current policy repository tells us how to go through another iteration of step 2. The solution quality is improved iteratively, and converge to optimum.

In the following, we (i) provide the entire picture of the iterative method, and (ii) indicate its equivalence to the DW approach that uses the DW decomposition to solve Markov decision processes as linear programs [15]. In the remainder of this paper, we focus on the criterion of expected cumulative cost to reach target states. For the expected discounted cumulative cost criterion, it is all the same except that the transition probability $p_{ij}(a)$ is multiplied by a discount factor γ and the set of target states is empty. This is true because the linear programming formulations in (5) and (6) for these two performance criteria are basically the same except for the presence of the discount factor in the linear program in (6).

7.3 Partitioning a State Space into Local Regions

- We adopt an arbitrary but fixed partition $P = \{R_1, \ldots, R_h\}$ of the state space throughout all iterations of the iterative method. In particular, the union of the peripheries of the regions R_l's, $U = \bigcup_{R_l \in P} \text{Periphery}(R_l)$,

and the kernel of each region R_l, $K_l = R_l - U$, play important roles in the iterative method.

- Let m be the number of the states in U. We number these m states in the coupling region U from 1 to m, and number the other states in $K_i, i > 0$ from $m+1$ to N. Using such a fixed numbering on states, we can more conveniently name the elements of the matrices and the vectors that are related to these states.

7.4 The Abstract Action Space

An abstract action is a local strategy imposed on a region R to encourage certain pattern of system evolution when leaving region R to enter R's adjacent regions. An abstract action of a region R is parameterized by a set of bias values assigned to the states in U.

- $\beta_i, 1 \leq i \leq m$ denotes the bias value assigned to a state i in U as described in Section 3. In addition, we define β_0 to be a measure of the current solution quality. Let

$$\beta = [\beta_1, \ldots, \beta_m] \text{ and } \bar{\beta} = [\beta_1, \ldots, \beta_m, \beta_0].$$

- Intuitively, we can interpret β_i as a measure of the expected cumulative cost since the first time of leaving region R from state i. A big β_i has the effect of discouraging the use of state i as an exit to leave R to enter R's adjacent regions.

- Refer to Section 7.8 for the initialization of $\bar{\beta}$ and refer to Section 7.9 for the iterative modifications of $\bar{\beta}$. In the remainder of this paper, we use $\bar{\beta}^{(t)}$, $\beta_i^{(t)}$, and $\beta_0^{(t)}$ respectively to denote the specific $\bar{\beta}$, β_i, and β_0 appearing on the tth iteration of the computation.

Remark The $\bar{\beta}$ vector is composed of the simplex multipliers on a specific iteration of the master problem in the DW decomposition. Consider the $m+1$ constraints in the master problem, the first m of which concern with the states in U. On each iteration, the simplex multipliers associated with these m constraints then provide these bias values β_i's, $1 \leq i \leq m$. The simplex multipliers associated with the $(m+1)$th constraint in the master problem provides the β_0 value.

7.5 Deriving Local Policies

7.5.1 Deriving the Local Policies over the Kernels

Given a Markov decision process $\mathcal{M} = (\Omega_X, \Omega_A, p, c)$, and a particular $\bar{\beta}$ vector, we can derive local Markov decision processes over the kernels K_l's.

For each region R_l in the partition P, we define a Markov decision process $\mathcal{M}_{\bar{\beta}|K_l} = (K_l \bigcup U, \Omega_A, \tilde{p}, \tilde{c})$.

1. $K_l \cup U$ is the (local) state space with respect to $\mathcal{M}_{\bar{\beta}|K_l}$, where the states outside the kernel K_l are considered as additional target states. *In other words, the union of U and the set original target states of $\mathcal{M}$ forms the set of target states of $\mathcal{M}_{\bar{\beta}|K_l}$.*

2. $\tilde{p}_{ij}$ is the (local) conditional probability distributions about action outcomes. Let $\tilde{p}_{ij}(a)$ and $p_{ij}(a)$ denote the probabilities of ending up in state j if we take action a when we are in state i with respect to $\mathcal{M}_{\bar{\beta}|k_l}$ and $\mathcal{M}$ respectively. We define

   - $\tilde{p}_{ii}(a) = 1$ if i is a target state of $\mathcal{M}_{\bar{\beta}|K_l}$, and
   - $\tilde{p}_{ij}(a) = p_{ij}(a)$ for for a non-target state i in K_l.

3. $\tilde{c}_{ij}$ specifies the biased action costs with respect to $\mathcal{M}_{\bar{\beta}|K_l}$, which is properly biased according to $\bar{\beta}$. Let $\tilde{c}_{ij}(a)$ and $c_{ij}(a)$ denote the action costs of ending up in state j if we take action a when we are in state i with respect to $\mathcal{M}_{\bar{\beta}|k_l}$ and $\mathcal{M}$ respectively. We define

   - $\tilde{c}_{ij}(a) = 0$ where i is a target state of $\mathcal{M}_{\bar{\beta}|K_l}$,
   - $\tilde{c}_{ij}(a) = c_{ij}(a)$ where i is a non-target state and j is in K_l, and
   - $\tilde{c}_{ij}(a) = c_{ij}(a) + \beta_j$ where i is a non-target state and j is in U.

4. In the following, we use

   - $Cost(i,a) = \sum_j p_{ij}(a)c_{ij}(a)$ to denote the cost of taking action a in state i with respect to $\mathcal{M}$,
   - $ActionBias(i,a) = \sum_{j \in U} \tilde{p}_{ij}(a)\beta_j$ to denote the bias regarding taking action a in state i, and
   - $BiasCost(i,a) = \sum_j \tilde{p}_{ij}(a)\tilde{c}_{ij}(a)$ to denote the biased cost of taking action a in state i with respect to $\mathcal{M}_{\bar{\beta}|K_l}$.

   We then have:

   - For a target state i of $\mathcal{M}_{\bar{\beta}|_R}$, $BiasCost(i,a) = 0$.
   - For a non-target state i in K_l,

$$\begin{aligned} BiasCost(i,a) &= \sum_j \tilde{p}_{ij}(a)\tilde{c}_{ij}(a) = \sum_j p_{ij}(a)\tilde{c}_{ij}(a) \\ &= \sum_{j \notin U} p_{ij}(a)c_{ij}(a) + \sum_{j \in U} p_{ij}(a)(c_{ij}(a) + \beta_j) \\ &= \sum_j p_{ij}(a)c_{ij}(a) + \sum_{j \in U} p_{ij}(a)\beta_j \\ &= Cost(i,a) + ActionBias(i,a). \end{aligned}$$

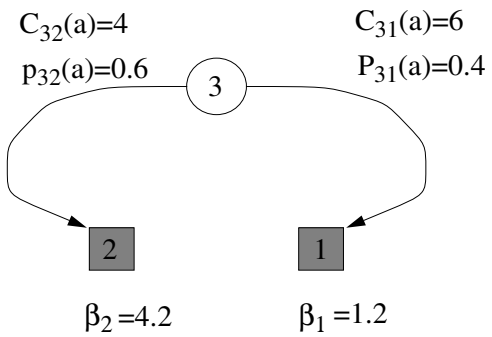

Figure 8: The relationship between actual and biased action cost

Figure 8 depicts three states 1, 2 and 3. State 1 and state 2 are in the peripheries of regions while state 3 is in the kernel of a region. When taking action a in state 3, we end in state 1 (2) with a cost of 6 (4) and with probability 0.4 (0.6). The bias associated with state 1 (2) is 1.2 (4.2). The biased action cost of taking action a in state 3 is then determined by

$$\begin{aligned} BiasCost(3,a) &= Cost(3,a) + Bias(3,a) \\ &= (0.6 \times 4 + 0.4 \times 6) + (0.6 \times 4.2 + 0.4 \times 1.2) \\ &= 7.8. \end{aligned}$$

Given $\mathcal{M}_{\bar{\beta}|K_l}$, we can derive the optimal local policy and the resulting local values as follows.

- $\pi_{\bar{\beta}|K_l}$ is a local policy that is optimal with respect to $\mathcal{M}_{\bar{\beta}|K_l}$, which maps the non-target states in K_l to actions [5].
- We use the term $LocalValue(i)$ to denote the value of a state i in K_l with respect to the local policy $\pi_{\bar{\beta}|K_l}$, which are determined by the following system of equations: For each i in K_l,

$$LocalValue(i) = BiasCost(i, \pi_{\bar{\beta}|k_l}(i)) + \sum_{j \in K_l} P_{ij}(\pi_{\bar{\beta}|k_l}(i)) \times LocalValue(j).$$

Remark The biased action costs actually correspond to the coefficients in the the cost function $(c - \beta A_1)X$ of the subproblem on a particular iteration of the DW approach, where c is the vector of actual action costs; A_1 is the constraint submatrix that gives rise to the master problem; and β are the simplex multipliers concerning those m states in U. $ActionBias(i,a)$'s corresponds to the components of $-\beta A_1$. The dual linear program of the subproblem is reduced to the local Markov decision processes described above.

[5] The actions taken in the target states of $\mathcal{M}_{\bar{\beta}|K_l}$ are not important since are modeled as absorbing states with zero action cost.

7.5.2 Deriving the Local Policies over the Peripheries

A local policy $\pi_{\bar{\beta}|K_l}$ maps the states in K_l, the kernel of region R_l, to actions. Given the set of local policies $\{\pi_{\bar{\beta}|K_l}\}$, we can also determine a local policy $\pi_{\bar{\beta}|U}$ that maps the states in U, the union of the peripheries, into actions as follows.

- We first determine the biased cost of taking an action a in a state i in U as follows:

$$ActionBias(i,a) = -\beta_i + \sum_{j \in U} P_{ij}(a)\beta_j,$$

  and

$$BiasCost(i,a) = Cost(i,a) + ActionBias(i,a).$$

- We then determine

$$MinStay = \min_{i,a} BiasCost(i,a) + \sum_{j \notin U} P_{ij}(a) LocalValue(j).$$

- – Let

$$(i^*, a^*) = \arg\min_{i,a} [BiasCost(i,a) + \sum_{j \notin U} P_{ij}(a) \times LocalValue(j)].$$

  – If $MinStay$ is negative, let $\pi_{\bar{\beta}|U}$ map the state i^* to action a^* ($\pi_{\bar{\beta}|U}(i^*) = a^*$) and arbitrarily map the other states in U to actions.

  – If $MinStay$ is positive or zero, let $\pi_{\bar{\beta}|U}$ arbitrarily map the states in U to actions.

Remark The dual linear program associated with the subproblem on an iteration of the DW decomposition is reduced to the local Markov decision processes. $MinStay$ measures the feasibility of the dual linear program. If $MinStay$ is negative, the dual linear program is infeasible and the optimal solution of the subproblem is unbounded; otherwise, the dual linear program is bounded. Deriving these local policies corresponds to solving the dual linear program, which allows us to find an entering column for the master problem.

7.6 Further Information Associated with Local Policies

On the tth iteration, we can derive a global policy $\Pi^{(t)}$ by gluing together (i) the set of local policies $\{\pi_{\bar{\beta}|K_l}\}$ over the kernels K_l's, and (ii) the local policy $\pi_{\bar{\beta}|U}$ over the peripheries. $\Pi^{(t)}$ can provide the following information:

- A weight vector $Weight^{(t)}$, which allows us to combine the policies in the policy repository to generate a solution policy if we decide to terminate at this moment.
- A feedback vector $Feedback^{(t)}$, which is a feedback on current solution quality that guides us to update the policy repository and conduct another iteration of computation.
- A value $\Delta^{(t)}$, which indicates the rate of the improvement in solution quality if $\Pi^{(t)}$ is incorporated to generate a new solution policy.

Again, let

$$\begin{aligned} MinStay &= \min_{i,a} BiasCost(i,a) + \sum_{j \notin U} P_{ij}(a) LocalValue(j), \text{and} \\ (i^*, a^*) &= \arg\min_{i,a} (BiasCost(i,a) + \sum_{j \notin U} P_{ij}(a) \times LocalValue(j)). \end{aligned}$$

We can derive $Weight^{(t)}$, $Feedback^{(t)}$, and $\Delta^{(t)}$ in following way.

- $Weight^{(t)}$ is a $N \times 1$ column vector, where $Weight^{(t)}(i)$ is a measure of the utility of taking action $\Pi^{(t)}(i)$ in state i.
  - First for each state i in U, we determine $Weight^{(t)}(i)$ as follows:
    - If $MinStay$ is negative,
      set $Weight^{(t)}(i^*)$ to be 1 and $Weight^{(t)}(i)$ to be 0 for the other states i's in U.
    - If $MinStay$ is positive or zero,
      set $Weight^{(t)}(i)$ to be zero for each state i in U.
  - Second for the kernel K_l of a region R_l,
    - if $MinStay$ is negative,
      we determine $Weight^{(t)}(i)$ for the states in K_l according to the following equations:

      $$\forall i \in K_l, \quad Weight^{(t)}(i) = p_{i^* i}(a^*) + \textstyle\sum_{j \in K_l} P_{ji}(\Pi^{(t)}(j)) Weight^{(t)}(j);$$

    - if $MinStay$ is positive or zero,
      we determine $Weight^{(t)}(i)$ for the states in K_l according to the following equations:

      $$\forall i \in K_l, \quad Weight^{(t)}(i) = \xi_i + \textstyle\sum_{j \in K_l} P_{ji}(\Pi^{(t)}(j)) Weight^{(t)}(j).$$

- $Feedback^{(t)}$ is a $(m+1) \times 1$ column vector. For each state i in U ($1 \leq i \leq m$), we can determine $Feedback^{(t)}(i)$'s in the following way:

$$Feedback^{(t)}(i) = Weight^{(t)}(i) - \sum_{j \in \Omega_X} P_{ji}(\Pi^{(t)}(j)) \times Weight^{(t)}(j).$$

If $MinStay$ is negative,

$$Feedback^{(t)} = \begin{bmatrix} Feedback^{(t)}(1) \\ \vdots \\ Feedback^{(t)}(m) \\ 1 \end{bmatrix};$$

otherwise,

$$Feedback^{(t)} = \begin{bmatrix} Feedback^{(t)}(1) \\ \vdots \\ Feedback^{(t)}(m) \\ 0 \end{bmatrix}.$$

- We can determine the unit profit associated with this policy by :

$$\Delta^{(t)} = \sum_{i \in \Omega_X} Cost(i, \Pi^{(t)}(i)) \times Weight^{(t)}(i).$$

Remark $Weight^{(t)}$ is a solution to the subproblem that is determined by the simplex multipliers $\bar{\beta}^{(t)}$. $Feedback^{(t)}$ is the corresponding entering column for the master problem. $\Delta^{(t)}$ is the unit cost of this entering column. When $MinStay$ is negative, the subproblem is unbounded and the entering column $Feedback^{(t)}$ is related to an extreme ray; otherwise, the subproblem is bounded and the entering column $Feedback^{(t)}$ is related to an extreme point.

7.7 Updating the Policy Repository

We keep a policy repository that is an array of up to $m+1$ different global policies $\Pi^{(i)}$'s generated on the previous iterations. We denote the rth policy stored in the policy repository as $\tilde{\Pi}^{(r)}$.

On the tth iteration, we maintain the following information in the policy repository:

- B is a $(m+1) \times (m+1)$ basis matrix. The rth column of B is the feedback vector of $\tilde{\Pi}^{(r)}$, the rth policy stored in the policy repository.
- C_B is a $1 \times (m+1)$ row vector. The ith element in C_B represents the rate of improvement in solution quality regarding $\tilde{\Pi}^{(i)}$, the ith policy stored in the policy repository.
- Note that $\Pi^{(t)}$ and $Weight^{(t)}$ can be reproduced from $\bar{\beta}^{(t)}$ by solving the corresponding local Markov decision processes when needed. Instead of explicitly storing $\Pi^{(t)}$ and $Weight^{(t)}$, we can just store $\bar{\beta}^{(t)}$ in the policy repository to compactly represent $\Pi^{(t)}$ and $Weight^{(t)}$.

On the tth iteration, we update the information in the policy repository in the following way:

- Let $Y = B^{-1}\bar{\xi}$ and $Z = B^{-1}Feedback^{(t)}$, and

$$r = \arg\min_{i, Z_i > 0}(Y_i/Z_i),$$

  where $\bar{\xi} = \{\xi_1, \ldots, \xi_m, 1\}$, ξ_i is the probability that the state i in U is the initial state, and Y_i (Z_i) is the ith element of Y (Z).

- On the tth iteration, we bring in the newly generated $\Pi^{(t)}$ to replace $\tilde{\Pi}^{(r)}$, the rth policy stored in the policy repository. We store $\bar{\beta}^{(t)}$ as the rth element in the repository to represent $\Pi^{(t)}$.

- We update B by replacing the rth column of B by $Feedback^{(t)}$. Similarly, we update C_B by replacing the rth element of C_B by $\Delta^{(t)}$.

Remark The policy repository corresponds to the simplex tableau when we carry out the simplex method on the master problem of the DW decomposition. On the tth iteration, $\Pi^{(t)}$ generates an entering column $Feedback^{(t)}$ to enter the basis. The rule of deciding the leaving policy $\tilde{\Pi}^{(r)}$ actually corresponds to the ratio test in the simplex method. In other words, an update on the policy repository actually corresponds to an pivot operation on the simplex tableau.

7.8 Bounds, Stopping Criteria, and New Abstract Actions

At the end of the tth iteration, we can determine (i) a bound on the quality of the current solution, and (ii) whether we should terminate the computation or go through another iteration of computation.

Bounds on the solution quality :

Let

$$\delta_\beta^{(t)} = \beta_0^{(t)} - \sum_{i \in \Omega_X} BiasCost(i, \Pi^{(t)}(i)) Weight^{(t)}(i),$$

and

$$\delta^* = \min_{1 \le i \le t} \delta_\beta^{(i)}.$$

The value of the current solution is better than or equal to the optimum plus δ^*.

Stopping Criteria :

If δ^* is no more than ϵ, we have reached an ϵ-optimal solution. In particular if δ^* is less than or equal to zero, we have reached an optimal

solution. In both situations, we should generate a solution policy of the required quality in the way described in Section 7.9.2; otherwise, we can derive new abstract actions of local regions as follows and go through a new iteration.

New Abstract Actions and Convergence :

If we determine to go through one more iteration, we derive a new bias vector by

$$\bar{\beta}^{(t+1)} = C_B B^{-1}.$$

This gives us the new abstract actions for individual regions for the next iteration. The solution quality are guaranteed to be improved from iteration to iteration. δ^* converges to 0 as we approach an optimal solution.

Remark The bounds on solution quality and the convergence property directly come from Proposition 2 and Proposition 3 in Section 6.1 about the DW decomposition. The stopping criteria actually corresponds to the optimality criteria of the master problem in the DW approach. The new bias vector corresponds to the new simplex multipliers derived by a pivot operation on the simplex tableau of the master problem.

7.9 The Initialization and Termination of the Iterative Method

7.9.1 Initializing the Policy Repository and Abstract Actions

- At the beginning, we let the policy repository contain $m + 1$ dummy polices and set

$$\begin{aligned} C_B &= [M, M, \ldots, M], \text{and} \\ B &= I \end{aligned}$$

where I is an identity matrix and M is a large positive value [6].

- Consequently we have $\bar{\beta}^{(0)} = C_B B^{-1} = [M, M, \ldots, M]$, which determines the initial abstract actions for individual regions. We then iteratively conduct the computation steps described in Section 7.5 to Section 7.8 until all the dummy policies are replaced with actual policies.

Remark Such an initialization corresponds to the derivation of an initial solution for the master problem using big-M method.

[6] M is exactly the value used in big-M method to initiate the simplex method. We refer the readers to [19] about the choice of an appropriate M.

- The $m+1$ dummy policies correspond to $m+1$ artificial variables added into the master problem. The basis matrix B is initialized as the the $m+1$ columns associated with these $m+1$ artificial variables, which is an identity matrix. Every artificial variable is associated the cost M, which is a large positive value. M is exactly the value used in big-M method to initiate the simplex method. We refer the readers to [19] for the details of the big-M method and the issue of choosing an appropriate M.

- When applying the big-M method, some of the artificial variables may leave and enter the basis more than one time. Similarly, we allow the ith dummy policy to reenter the policy repository after it has been excluded from the policy repository. This happens when (i) on the rth iteration the stopping criteria $\delta_{\beta^{(r)}} < 0$ is satisfied but (ii) $M - \beta_i^{(r)}$ is also negative, which indicates the possibility of quality improvement by bringing back the dummy policy to kick off some actual policy stored in the current repository.

7.9.2 Generating a Solution Policy from the Policy Repository

When we meet the stopping criteria, we can properly combine the policies in the policy repository to produce an optimal or ϵ optimal policy in the following way.

- Let $\bar{\xi}$ denote the column vector $(\xi_1, \xi_2, \ldots, \xi_m, 1)^T$, where ξ_i is the probability that a state i in U is the initial state. Let $Ratio = B^{-1}\bar{\xi}$ and $Ratio_j$ be the jth component of $Ratio$. $\tilde{\Pi}^{(j)}$ denote the jth policy stored in the current policy repository. We determine

$$Frequency(i,a) = \sum_{1 \le j \le m+1, \tilde{\Pi}^{(j)}(i)=a} Ratio_j \times Weight^{(t)}(i).$$

  We then derive a random policy $\tilde{\Pi}$ where $Frequency(i,a)/\sum_a Frequency(i,a)$ is the conditional probability that we take action a when we are in state i.

- $\tilde{\Pi}$ can be improved into a deterministic policy Π^* by (i) first determining the values of states given the policy $\tilde{\Pi}$, and (ii) then conducting an iteration of policy improvement :

$$\Pi^*(i) = \arg\min_a [Cost(i,a) + \sum (P_{ij}(a) \times value(j))],$$

  where $value(j)$ is the value of state j according to $\tilde{\Pi}$. Output the policy Π^* as the solution policy.

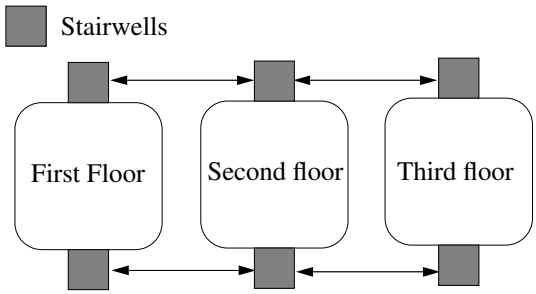

Figure 9: A building of three floors and two elevators

Remark The $Frequency$ vector corresponds to a final solution for the master problem and actually $Frequency(i, a)$ corresponds to the variable $E_{i,a}$. Note that $E_{i,a}$ and thus $Frequency(i, a)$ also is the expected number of times when we are in state i and we take action a. Therefore $Frequency(i, a)/\sum_a Frequency(i, a)$ is the probability that we map state i to action a.

8 A Robot Navigation Example

In the following, we use a robot navigation example to illustrate the basic principle of the decomposition techniques described in this paper.

8.1 Modeling a Robot Navigation Problem as a Markov Decision Process

Figure 9 displays a building of three floors. Two elevators in the building transport people from one floor to another. Each floor is tessellated into one thousand squares that can be entered, two of which correspond to the stairwells for the two elevators.

A robot moves around this building by taking actions to (i) reach squares adjacent to its current position in the same floor, and (ii) use an elevator to enter another floor. All these actions take time and involve uncertainty in their outcomes:

Moving in different directions: As long as there is no obstacle blocking its way, the robot can move in four directions, namely east, west, north and south. To move in a specific direction, the robot first orients itself toward that direction, and then move one square forward. The expected amount of time spending in such an action depends on the orientation of the robot immediately prior to taking the action. When the robot take an action to move forward, it ends in the adjacent square in front of it most of the time with some chances that it may end in the square to the right or the left of the target square. The probability distribution

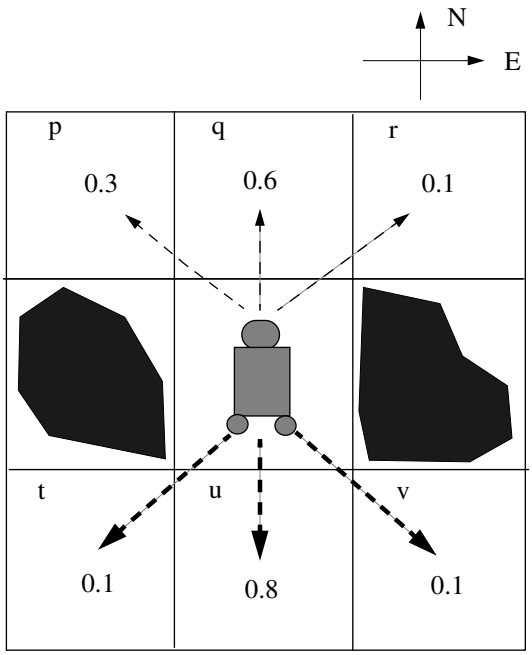

Figure 10: Uncertainty in action outcomes

over the possible outcomes is conditional on the position of the robot immediately prior to taking the action.

Entering an elevator: When standing in a square adjacent to a , the robot can take an action to (i) turn itself toward the entrance to the elevator, (ii) push the button to call for the elevator, (iii) wait until the elevator comes up, and (iv) enter the elevator and turn around. The expected amount of time spending in such an action depends on (i) the floor where the robot is before entering the elevator, and (ii) the orientation of the robot immediately prior to taking the action. The robot enters the elevator in the long with probability one.

Heading toward another floor: When the robot enters an elevator, it can take an action to (i) push a button to close the door, (ii) choose a specific floor to enter, and (iii) steps out of the elevator when the elevator stops. The robot may end in a floor different from the target floor in its mind, since people in the other floor may push the button to stop the elevator before the elevator actually reaches the target floor. Both the probability distribution over the possible outcomes and the expected amount of time spending in such an action depend on (i) the floor where the robot is before entering the elevator, and (ii) the target floor in the action.

In Figure 10, the robot is in position s and is oriented toward the north. Both the east side and the west ride of s are blocked by obstacles. If the robot

decides to go north (south), it takes five (ten) seconds on average to reach one of the positions p, q and r (t, u and v) with probability 0.3, 0.6 and 0.1 (0.1, 0.8 and 0.1) respectively. This is because the robot need to spend five extra seconds to orient itself toward the south before it can actually move to the south.

In each floor, there is a site to recharge the battery of the robot. As soon as the voltage of the battery is below certain level, the robot enters a panic mode to head for one of the site to be recharged. We would like to determine a policy that maps each given pair of the position and orientation of the robot into an action to take. In particular, we would like to determine an optimal policy that minimizes the expected amount of time for the robot to reach one of the charging sites. This can be modeled as a Markov decision process $\mathcal{M} = (\Omega_X, \Omega_A, p, c)$ as follows:

The state space: Each state (s, o) in Ω_X models a specific position s and a specific orientation o of the robot. This gives us a state space of 12,000 ($3 \times 1000 \times 4$) states, since (i) we have three floors, (ii) each floor is tessellated into one thousand squares, and (iii) there are four possible orientations when the robot is in a specific square.

The action Space: Ω_X is the action space composed of the actions of (i) moving in four directions when staying in a floor, (ii) getting into an elevator, and (iii) moving to a specific floor by elevator. The robot is enabled to move in a specific direction only if there is no obstacles blocking its way.

Cost functions and conditional probabilities: c and p specifies the cost functions and conditional probability distributions associated with actions and possible action outcomes. $c_{ij}(a)$ is the expected amount of time when the robot takes action a in a state $i = (s, o)$ and ends in another state $j = (s', o')$. $p_{ij}(a)$ is the probability of ending in another state $j = (s', o')$ when the robot takes action a in a state $i = (s, o)$.

The performance criteria: $\{i = (s, o) | s$ is the position of a charging site.$\}$ is the set of target states. We would like to minimize the expected cumulative cost to reach the target states.

8.2 Exploiting the Locality Structure

In the following, we describe the locality structure in this robot navigation example.

Local regions: Since the elevators are accessible only in a small fraction of the entire state space Ω_X, the robot tends to walk around the same floor

for quiet a while before it either (i) enters the elevator to leave for another floor or (ii) reaches one of the charging sites. Therefore, the activities of the robot are highly localized in the individual floors. In the remainder of this section, we partition the state space into three regions R, S and T, which are composed of the states in the three floors respectively.

Loose coupling among regions: The connections among the three floors are also highly localized. Only if the robot is in a stairwell position on a floor, it can (i) transfer by elevator to another floor and (ii) only end in a correspond stairwell position on that floor. In other words, the peripheries of the three regions are very *thin*. The union of the peripheries, U, is composed of only twenty four states that correspond to the combination of six stairwell positions and four possible orientations.

Local interactions: Standing in a specific position beyond the stairwells, the robot can only move to the adjacent positions on the same floor. Therefore, each state in the kernels of the regions R, S, and T can only transition to no more than thirty two other states that correspond to the combinations of eight adjacent positions and four possible orientations. We exploit this property in the theoretical analysis of the feasibility of the decomposition techniques.

8.3 Using the Hierarchical Construction Approach

Using the hierarchical construction approach, we first construct the abstract decision process $(\{R, S, T\}, \mathcal{F}_{\rightsquigarrow}, p', c')$.

1. Determine the nine abstract actions in $\mathcal{F}_{\rightsquigarrow}$: $\pi_{S\to S}$, $\pi_{S\to T}$, $\pi_{S\to R}$, $\pi_{T\to S}$, $\pi_{T\to T}$, $\pi_{T\to R}$, $\pi_{R\to S}$, $\pi_{R\to T}$, and $\pi_{R\to R}$. Each abstract action corresponds to a local policy to be used on a floor that intends to increase the possibility of ending up in a specific floor in the long run. In other words, the abstract actions specify the most preferred next floors to go for individual floors.

2. To derive these abstract actions, we first choose an appropriate value κ to parameterize the pace the robot should take to transfer from one floor to the most preferred next floor to go. Second, for each floor F, we (i) uniformly assign zero to β_i's for those periphery states i's of F that are in the preferred floor, and (ii) uniformly assign κ to β_i for those periphery states i's of F that are not in the preferred floor. Third, we construct and solve these local Markov decision processes, each of which is concerned with 4,000 states instead of 12,000 states. The optimal policies for these local Markov decision processes are the abstract actions to be considered in individual floors.

3. Calculate the abstract transition probabilities p' and abstract costs c', which represent the estimates of the probabilities and costs of ending in another region when we adopt an abstract action (i.e. a local policy) in a specific region. For example, $p'_{ST}(\pi_{S\to R})$ is an estimate of the probability of entering region T from region S when we adopt the abstract action (the local policy) $\pi_{S\to R}$. Similarly, $c'_{ST}(\pi_{S\to R})$ is an estimate of the cost of entering region T from region S when we adopt the abstract action (the local policy) $\pi_{S\to R}$.

4. Solve the abstract decision process $(\{R, S, T\}, \mathcal{F}_{\rightsquigarrow}, p', c')$ to obtain an abstract policy $\Pi : P \to \mathcal{F}_{\rightsquigarrow}$. Π indicates which abstract action in $\mathcal{F}_{\rightsquigarrow}$ to take (i.e. which floor the robot should head for) when the robot is on a specific floor.

5. The robot takes the action $\Pi(F)(i)$, when (i) the robot is in a specific region F and (ii) i is its current state, which is determined by its current position and orientation.

Remark: In the hierarchical construction approach, our underlying intuition is to, for each floor, (i) first choose a specific floor (maybe the same floor) as the most preferred next floor to go, and (ii) second minimize the expected cumulative cost in achieving (i). We accomplish the first task by solving the abstract decision process that heuristically choose the most preferred next floor to go among all floors. We accomplish the second task by solving the local Markov decision processes associated with the state subspaces concerning individual floors.

8.4 Using the Iterative Approximation Approach

1. Using the iterative approximation approach, we first initialize the policy repository and the $\bar{\beta}$ vector as we describe in Section 7, and then reiterate the following steps.

2. Since there are only twenty four (m=24) states in the peripheries, we only need to store twenty five (m+1=25) global policies at a time, each of which is compactly represented as a 25×1 column vector. Compared with the state space of size 12,000, the space taken by the policy repository is negligible.

3. $\bar{\beta}$ vector represents our current estimate of how valuable it is to enter a stairwell position. On each iteration, we solve the local Markov decision processes according to the current $\bar{\beta}$ vector. These solutions are the local policies on each floor, which are then glued together to yield a global policy. We then store this global policy in the policy repository to replace an old one there.

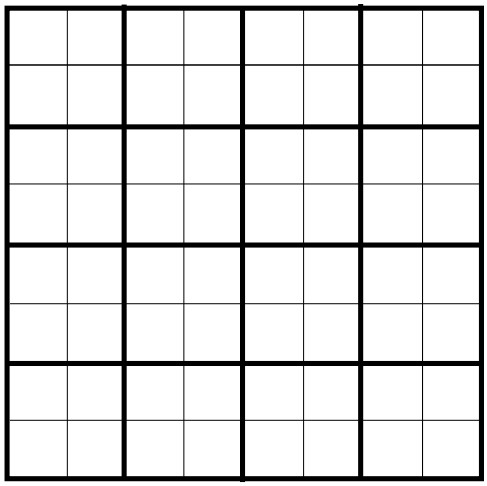

Figure 11: An $N \times N$ area tessellated into N^2 squares and M^2 regions where $N = 8$ and $M = 4$

4. If the optimum or ϵ-optimum is not yet achieved, we (i) update the policy repository, (ii) determine a new $\bar{\beta}$ vector, and (iii) go back to Step 2. Otherwise, we combine the twenty five polices stored in the policy repository to generate the final policy.

Remark: Two major simplifications or assumptions in the hierarchical approach are (i) that each floor is restricted to have a unique most preferred next floor to go, and (ii) that each state in the peripheries is assigned either the value zero or the value κ. In the iterative approximation approach, we relax these restrictions. We allow each state in the peripheries to have arbitrary bias value. In other words, we numerically specify our preference over the states in the peripheries, rather than choosing the most preferred floor to go. In addition, we do not rely on a specific $\bar{\beta}$ vector and its resulting global policy. Instead, we (i) repeatedly modify $\bar{\beta}$ to generate a sequence of global policies, (ii) keep a policy repository of 25 such global policies at a time, and (iii) combine the policies stored in the policy repository to generate a solution policy when we decide to terminate the computation.

9 More Theoretical Analysis in Robot Navigation Domains

In the following, we provide more theoretical analysis to compare the computational efficiency of the decomposition techniques in deriving an optimal policy or a near optimal policy. We focus on the kind of robot navigation problem described in this section, where (i) a state is determined by the position and orientation of a robot and (ii) only a small set of actions are enabled in a

state. We assume and exploit the general property that a robot can only move from one position to geometrically neighboring positions. Rather than assuming additional structure like floors loosely coupled by elevators, we consider a robot moving around an $N \times N$ area that is tessellated into N^2 square units as depicted in Figure 11.

Suppose we adopt a partition that divides the area into M^2 regions, each of which is a $\frac{N}{M} \times \frac{N}{M}$ local area as displayed in Figure 11. Therefore each region has no more than $\frac{4N}{M}$ neighboring squares. In summary, we have the following properties regarding the partitioning of state space:

- We assume a robot can be oriented in four different ways, namely east, west, south, and north. This yields a state space of size $4N^2$, given the N^2 squares there. Each region R is composed of $\frac{4N^2}{M^2}$ states and is adjacent to no more than $\frac{16N}{M}$ states in other regions.

- For each region R, Kernel(R) is of size $\Theta(\frac{N^2}{M^2})$. The size of U, the union of the peripheries of regions, is $M^2\Theta(\frac{N}{M}) = \Theta(MN)$.

In the following, we (i) compare the computational efficiency of different approaches, and (ii) indicate when we should apply the decomposition techniques.

9.1 Modeling the Computation Complexity of the Standard Techniques

Note that

- Standard techniques like the policy iteration method and the value iteration method converge to optimal policies in linear and quadratic convergence rate respectively [21], and

- Each iteration of the policy iteration method and the value iteration method takes time cubic and quadratic respectively in the size of the state space [8] [21].

- We model the computational complexity of directly applying the standard techniques to the state space as cn^k, where (i) both c and k are constants and (ii) n is the size of the state space. k is 3 and 2 respectively when applying the policy iteration method and the value iteration method respectively, while in applying the value iteration method c is much greater than that in applying the policy iteration method.

- Therefore, the overall computation complexity of directly applying the standard techniques in this robot navigation domain is $\Theta(N^{2k})$, since we have a state space of size $4N^2$.

9.2 Computation Efficiency of the Hierarchical Construction Approach

- The overall computation complexity is

$$\Theta(M^2(\frac{N}{M})^{2k} + M^{2k}).$$

  The first term concerns with solving the M^2 local Markov decision processes over the local regions, each of whose size is $\Theta(\frac{N^2}{M^2})$. The second term concerns with solving the abstract decision process of size M^2.

- When $M = \Theta(N^{1/2})$, the first term and the second term of the overall computation complexity are roughly balanced, which yields a $\Theta(N^{k+1})$ time complexity.

- Compared with the $\Theta(N^{2k})$ complexity of directly applying standard techniques, the hierarchical construction method is always much more efficient. However, there is no guarantee about the quality of the derived policy.

9.3 Computation Efficiency of the Iterative Approximation Approach

- The computation complexity of each iteration of the iterative approximation method is

$$\Theta(M^2(\frac{N}{M})^{2k}) + (MN)^3).$$

  The first term concerns with solving the M^2 local Markov decision processes over the kernels of the local regions, each of whose size is $\Theta(\frac{N^2}{M^2})$. The second term concerns with deriving the local policy over U, the union of the peripheries, by solving $\Theta(MN)$ linear equations that correspond to the $\Theta(MN)$ states in U.

- When $M = \Theta(N^{(2k-3)/(2k+1)})$, the first term and the second term of the overall computation complexity are roughly balanced, which yields a $\Theta(N^{(12k-6)/(2k+1)})$ time complexity on each iteration. More precisely, we have $\Theta(N^{18/5})$ time complexity when $k = 2$, and $\Theta(N^{30/7})$ time complexity when $k = 3$.

- Compared with the $\Theta(N^{2k})$ overall complexity in directly applying standard techniques, the hierarchical construction approach is more efficient if the computation converges to an optimal (or near optimal) solution within (i) $O(N^{2/5})$ number of iterations when $k = 2$ and (ii) $O(N^{12/7})$ number of iterations when $k = 3$.

10 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 [13] and hierarchies of state-space operators [22] [12]. Closely related is the work on decomposing discrete event systems modeled as (deterministic) finite state machines [23] [2].

In the area of reinforcement learning, there is work on deterministic action models and continuous state spaces [18] and stochastic models and discrete state spaces [11]. The hierarchical policy construction method described in Section 4 provides an alternative formulation of Kaelbling's hierarchical learning algorithm [11] and suggests how Moore and Atkeson's parti-game algorithm might be extended to handle discrete state spaces.

The analysis in this paper of the paper borrows heavily from the work in operations research and combinatorial optimization for representing Markov decision processes as linear programs [7] [8] and decomposing large systems generally [4] [16] and Markov decision processes specifically [15]. The approach described in [5] 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.

11 Conclusion

The benefit of decomposition techniques is that we are able to deal with subproblems of smaller size, the tradeoff is that often 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 a problem instance of very large size, decomposition techniques are valuable even if standard polynomial-time algorithms exist. It is always more efficient to apply a standard technique to solve smaller subproblems than to solve the original problem directly (unless the the standard technique provides a linear-time algorithm). The only question is whether we can efficiently combine the solutions to the subproblems into a solution to the original problem.

There exist standard techniques that can solve Markov decision processes in time polynomial in the size of the given state space. However, given a factored state-space representation for a Markov decision process, the underlying state space is exponential in the number of state variables mentioned in a problem instance. This exponential state space makes it impractical to apply the standard techniques to many AI planning problems that have factored state-space representations.

Frequently, a domain expert may exploit the factored state-space represen-

tation and partition the state space into regions, each of which has a abstract description of reduced dimensions. In this paper, we develop 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 $\bar{\beta}$ parameter that provides an abstract summary of the interactions among regions.

In Sections 4 and 5, we develop two approaches to combining the solutions to subproblems: a hierarchical construction approach and an iterative approximation approach. The hierarchical construction approach provides a quick solution with an intuitive interpretation. The iterative approximation approach, iteratively decomposes a problem instance into different subproblems. On each iteration, we combine the solutions to the subproblems to obtain an improved solution and determine whether the current solution is optimal or within some specified tolerance. In particular, we relate the iterative approximation framework to the research on the use of Dantzig-Wolfe decomposition in solving Markov decision processes as linear programs, which can be interpreted as an iterative method. This iterative method is guaranteed to converge to the optimum in a finite number of iterations. For practical purposes, a few number of iterations may be sufficient for a solution of good quality.

Acknowledgments

We thank Craig Boutilier, Moises Goldszmidt, Steve Hanks, and David Smith for comments on the content and presentation of the ideas described in this paper.

References

[1] Bellman, Richard, *Adaptive Control Processes*, (Princeton University Press, Princeton, New Jersey, 1961).

[2] Caines, Peter E. and Wang, S., COCOLOG: A Conditional Observer and Controller Logic for Finite Machines, *Proceedings of the 29th IEEE Conference on Decision and Control*, Hawaii, 1990.

[3] Chvatal, Vasek, *Linear Programming*, (W. H. Freeman and Company, 1980).

[4] Dantzig, George and Wolfe, Philip, Decomposition Principle for Dynamic Programs, *Operations Research*, **8**(1) (1960) 101–111.

[5] Dean, Thomas, Kaelbling, Leslie, Kirman, Jak, and Nicholson, Ann, Planning With Deadlines in Stochastic Domains, *Proceedings AAAI-93, Washington, D.C.*, AAAI, 1993, 574–579.

[6] Dean, Thomas and Kanazawa, Keiji, A Model for Reasoning About Persistence and Causation, *Computational Intelligence*, **5**(3) (1989) 142–150.

[7] D'Epenoux, F., Sur un probléme de production et de stockage dans l'aleatoire, *Management Science*, **10** (1963) 98–108.

[8] Derman, Cyrus, *Finite State Markovian Decision Processes*, (Cambridge University Press, New York, 1970).

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

[10] Howard, Ronald A., *Dynamic Programming and Markov Processes*, (MIT Press, Cambridge, Massachusetts, 1960).

[11] Kaelbling, Leslie Pack, Hierarchical Learning in Stochastic Domains: A Preliminary Report, *Proceedings Tenth International Conference on Machine Learning*, 1993.

[12] Knoblock, Craig A., Search Reduction in Hierarchical Problem Solving, *Proceedings AAAI-91, Anaheim, California*, AAAI, 1991, 686–691.

[13] Korf, Richard, Macro-Operators: A Weak Method for Learning, *Artificial Intelligence*, **26** (1985) 35–77.

[14] Kushner, H. J. and Kleinman, A. J., Mathematical Programming and the control of Markov Chains, *International Journal on Control*, **13**(5) (1971) 801–820.

[15] Kushner, Harold J. and Chen, Ching-Hui, Decomposition of systems governed by Markov chains, *IEEE Transactions on Automatic Control*, **AC-19**(5) (1974) 501–507.

[16] Lasdon, Leon S., *Optimization Theory for Large Systems*, (Macmillan Company, 1970).

[17] Lin, Shieu-Hong and Dean, Thomas, Exploiting Locality in Temporal Reasoning, Sandewall, E. and Backstrom, C., (Eds.), *Current Trends in AI Planning*, Amsterdam, IOS Press, 1994.

[18] Moore, Andrew W. and Atkeson, Christopher G., The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State Spaces, To appear in Machine Learning, 1995.

[19] M.S. Bazaraa, H. D. Sherali, J. J. Jarvis, *Linear Programming and Network Flows*, (John Wiley & Sons, New York, 1990).

[20] Papadimitriou, Christos H. and Tsitsiklis, John N., The Complexity of Markov Chain Decision Processes, *Mathematics of Operations Research*, **12**(3) (1987) 441–450.

[21] Puterman, Martin L., *Markov Decision Processes*, (John Wiley & Sons, New York, 1994).

[22] Sacerdoti, Earl, Planning in a Hierarchy of Abstraction Spaces, *Artificial Intelligence*, **7** (1974) 231–272.

[23] Zhong, H. and Wonham, W. M., On the Consistency of Hierarchical Supervision in Discrete-Event Systems, *IEEE Transactions on Automatic Control*, **35**(10) (1990) 1125–1134.