

Parallel MDPs

L. Enrique Sucar
ITESM Cuernavaca
Reforma 182-A
Temixco, Mor., Mexico
esucar@itesm.mx

Abstract

We propose a framework for solving complex decision problems based on a partition in simpler problems that can be solved independently, and then combined to obtain an optimal, global solution. Each aspect of the problem is represented as an MDP and solved independently. At any state of the problem, each MDP sends its value for each possible action (its Q value) and an “arbiter” selects the action with the greatest combined value. In contrast to previous approaches for hierarchical MDPs, in our approach all the MDPs work in *parallel*, so we obtain a reactive system based on a decision theoretical framework. We present an algorithm for solving parallel MDPs and prove it obtains the global optimum, assuming an additive value. We illustrate our approach with a simulated robot navigation problem.

1 Introduction

Our work is motivated by planning under uncertainty in robotics. Consider a mobile robot that has to perform a complex task in an uncertain environment. To accomplish its goal, the robot has to do several subtasks simultaneously, such as finding the shortest route to certain location, and at the same time, avoid obstacles and maintain its location in the map. It might also need to recognize objects in the environment and interact with people. A popular approach to solve this problem in robotics is based on Brooks *subsumption architecture* (Brooks, 1986), in which several processes can sense and act in *parallel*. The conflicts that could arise between the different *behaviors* are usually solved by a fixed priority structure. However, this way of task coordination has several drawbacks: (i) as the number of subtasks increases, defining the priority structure becomes very difficult, (ii) the priority is fixed, and can not change depending on the current situation. We consider an alternative approach based on decision-theoretic planning, in which the priority of the subtasks can be decided dynamically such that the *best*

action can be taken at each decision point.

Markov Decision Processes (MDPs) (Bellman, 1957; Puterman, 1994) have developed as a standard method for representing uncertainty in decision-theoretic planning. They are simple for domain experts to specify, or can be learned from data. They are the subject of much current research, and have many well studied properties including exact and approximate solution and learning techniques. However, if we represent the robot task coordination problem as a single MDP, we have to consider all possible combinations of all the possible simultaneous actions. This implies an explosion in the action—state space and thus an important increase in complexity for solving the MDP. It also becomes much more difficult to specify or learn the model. Given that each subtask is usually implemented as a separate software module, it is more natural to try to view each subtask as a different MDP and then in some way combine their policies to obtain the optimal global policy.

In this paper we propose an approach for solving several subtasks represented as MDPs and combine the results to obtain the optimal

global policy, that we call *Parallel MDPs*. Each aspect of the problem is represented as an MDP and solved independently. At any state of the problem, each MDP sends its value for each possible action (its Q value) and an “arbiter” selects the action with the greatest combined value. So we have the advantages of both, reactive and decision-theoretic planning: a reactive system based on a decision theoretical framework. We present an algorithm for solving parallel MDPs and prove it obtains the global optimum, assuming an additive value. We illustrate our approach with a simulated robot navigation problem.

The paper is organized as follows. In the next section we present a formal definition of MDPs, the standard techniques for their solution and factored and abstract MDP representations. We review related work on hierarchical and loosely coupled MDPs. We then give a formal definition of parallel MDPs, and an algorithm for their solution. We present preliminary results on robot navigation in a simulated environment. We conclude with a summary and directions for future work.

2 Markov Decision Processes

A Markov Decision Processes (MDP) (Puterman, 1994) models a sequential decision problem, in which a system evolves in time and is controlled by an agent. The system dynamics is governed by a probabilistic transition function that maps states and actions to states. At each time, the agent receives a reward that depends on the current state and the applied actions. Thus, the main problem is to find a control strategy or *policy* that maximizes the expected reward over time.

Formally, an MDP is a tuple $M = \langle S, A, \Phi, R \rangle$, where S is a finite set of states $\{1, \dots, n\}$. A is a finite set of actions. $\Phi : A \times S \rightarrow \Pi(S)$ is the state transition function specified as a probability distribution. The probability of reaching state s' by performing action a in state s is written $\Phi(a, s, s')$. $R : S \times A \rightarrow \mathfrak{R}$ is the reward function. $R(s, a)$ is the reward that the system receives if it takes action a in state

s .

A policy for an MDP is a mapping $\pi : S \rightarrow A$ that selects an action for each state. Given a policy, we can define its finite-horizon value function $V_n^\pi : S \rightarrow \mathfrak{R}$, where $V_n^\pi(s)$ is the expected value of applying the policy π for n steps starting in state s . The value function is defined inductively with $V_0^\pi(s) = R(s, \pi(s))$ and $V_m^\pi(s) = R(s, \pi(s)) + \sum_{s' \in S} \Phi(\pi(s), s, s') V_{m-1}^\pi(s')$. Over an infinite horizon, a discounted model is used to have a bounded expected value, where the parameter $0 \leq \gamma < 1$ is the *discount factor*, used to discount future rewards at a geometric rate. Thus, if $V^\pi(s)$ is the discounted expected value in state s following policy π forever, we must have $V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} \Phi(\pi(s), s, s') V_{m-1}^\pi(s')$, which yields a set of linear equations in the values of $V^\pi(\cdot)$.

A solution to an MDP is a policy that maximizes its expected value. For the discounted infinite-horizon case with any given discount factor $\gamma \in [0, 1)$, there is a policy V^* that is optimal regardless of the starting state that satisfies the *Bellman* equation (Bellman, 1957):

$$V^*(s) = \max_a \{R(s, a) + \gamma \sum_{s' \in S} \Phi(a, s, s') V^*(s')\} \quad (1)$$

Two popular methods for solving this equation and finding an optimal policy for an MDP are: (a) value iteration and (2) policy iteration (Puterman, 1994).

In policy iteration, the current policy is repeatedly improved by finding some action in each state that has a higher value than the action chosen by the current policy for the state. The policy is initially chosen at random, and the process terminates when no improvement can be found. This process converges to an optimal policy (Puterman, 1994).

In value iteration, optimal policies are produced for successively longer finite horizons until they converge. It is relatively simple to find an optimal policy over n steps $\pi_n^*(\cdot)$, with value function $V_n^*(\cdot)$ using the recurrence relation:

$$\pi_n^*(s) = \arg \max_a \{R(s, a) + \gamma \sum_{s' \in S} \Phi(a, s, s') V_{n-1}^*(s')\}, \quad (2)$$

with starting condition $V_0^*(.) = 0 \forall s \in S$, where V_m^* is derived from the policy π_m^* as described earlier.

An alternative formulation of an MDP can be given in terms of the action-value function or *Q function*. This function, $Q^\pi(s, a)$ gives the expected cumulative reward of performing action a in state s and following policy π thereafter. The optimal action value function $Q^*(s, a)$, gives the expected cumulative reward of performing action a in state s and following an optimal policy thereafter. Note that given $Q^*(s, a)$, we can obtain $V^*(s)$ by maximizing over the actions.

The main drawback of the MDP approach is that the solution complexity is polynomial on size of the state-action space, and this can be *very large* for most applications. There are two main approaches to deal with complexity: state space abstraction and problem decomposition.

2.1 Factored and Abstract MDPs

Traditional MDP solution techniques have the drawback that they require an explicit state space, limiting their applicability to real-world problems. Factored representations address this drawback via compactly specifying the state-space in factored form. In a factored MDP, the set of states is described via a set of random variables $X = \{X_1, \dots, X_n\}$, where each X_i takes on values in some finite domain $Dom(X_i)$. The framework of dynamic Bayesian networks (DBN) (Dean and Kanasawa, 1989) gives us the tools to describe the transition model function concisely. For each action, a two-stage DBN specifies the transition model. An even more compact representation can be obtained by representing the transition tables and value functions as decision trees (Boutilier et al., 1999) or algebraic decision diagrams (Hoey et al., 1999).

A further reduction in complexity can be obtained by state abstraction and aggregation techniques (Boutilier et al., 1999). Dean and Givan (Dean and Givan, 1997) describe an algorithm that partitions the state space into a set of blocks such that the each block is *stable*; that is, it preserves the same transition probabilities as the original model. Although this al-

gorithm produces an exact partition, this could still be too complex. In many applications an approximate model could be sufficient to construct near-optimal policies. Other approaches consider problem decomposition, in which an MDP is partitioned in several problems that are solved independently and then pieced together (Boutilier et al., 1999), called hierarchical MDPs.

2.2 Hierarchical MDPs

Hierarchical MDPs accelerate the solution of complex problems by defining different subtasks that correspond to intermediate goals, solving for each subgoal, and then combining these sub-processes to solve the overall problem. Hierarchical MDP approaches include MAXQ (Dietterich, 2000) and HAM (Parr and Russell, 1997), among others. Most of these approaches assume that the domain hierarchy is given; a notable exception that learns the decomposition is HEXQ (Hengst, 2002). Although our approach also considers a decomposition of the problem, it is a different one. Hierarchical MDPs provide a *sequential* decomposition, in which different subgoals are solved in sequence to reach the final goal. That is, at the execution phase, only one task is *active* at a given time. In Parallel MDPs, the subtasks are concurrent, so these are executed in parallel to solve the global task.

The previous work most closely related to our approach are loosely coupled MDPs (Meuleau et al., 1998). They consider several independent subprocesses whose reward and transition functions are independent of each other, but these are coupled due to common resource constraints. Their solution is divided in two phases. In the first, off-line phase, value functions are calculated for the individual subtasks. In the second, on-line phase, the value functions are used to calculate the next action for each process. To choose the actions, they use an iterative procedure based on a heuristic allocation of resources to each task. There are important differences in the types of problems we are solving and also in the solutions. We are interested in problems in which the subtasks have a common goal, but each consider a different aspect

of the problem. Each MDP has the same action set, and only one of these actions can be executed at any time; while in loosely coupled MDPs each subtask executes an action at each state. So our solution selects a single action for each state, and this is computed directly.

3 Parallel MDPs

Parallel MDPs are a set of discrete time Markov decision processes that are executed in *parallel*. At each time period an action is selected from each MDP, and an *arbiter* selects an action among the ones proposed by each process. These processes share a common state and action space, but Following we define a parallel MDP and give an algorithm for its solution.

Definition 1: A parallel MDP is a set of K Markov decision processes, P_1, P_2, \dots, P_k , such that each process, P_i is an MDP. All the MDPs share the same state space, action set and transition function, that is $S_1 = S_2 = \dots = S_k$; $A_1 = A_2 = \dots = A_k$; $\Phi_1(a, s, s') = \Phi_2(a, s, s') = \dots = \Phi_k(a, s, s')$. Each MDP has a different reward function, R_1, R_2, \dots, R_k . The total reward, RT , is the sum of the individual rewards: $RT = R_1 + R_2 + \dots + R_k$

For example, assume we have a simulated robot in a “grid world”, as depicted in figure 1. The robot has to go from the its actual position to the goal, and at the same time avoid obstacles. So we can define two MDPs: (i) a *Navigation MDP*, for going to the goal, (ii) an *obstacle avoidance MDP*, for avoiding obstacles. They both have the same state space (each cell in the grid) and actions (move up, down, right or left); but a different reward. The Navigation MDP gets a positive reward when it arrives to the goal, while the Obstacle Avoidance MDP gets a negative reward when it collides with an obstacle. So this case corresponds to a Parallel MDP. In the results section we present how this example is solved with our approach.

Next we will give an algorithm to obtain the optimal policy for a Parallel MDP, but first we present a theorem that gives the theoretical bases for the algorithm.

Theorem 1: Given a Parallel MDP, and as-

suming additive reward and value functions, the optimal value V_I^* and optimal policy π_I^* are:

$$V_I^*(s) = \max_a \sum_{i \in K} Q_i^*(s, a) \quad (3)$$

$$\pi_I^*(s) = \operatorname{argmax}_a \sum_{i \in K} Q_i^*(s, a) \quad (4)$$

Where $Q_i^*(s, a)$ is the optimal Q value obtained by solving each individual MDP.

Proof: The solution to the parallel MDP is given by the *Bellman* equation (Bellman, 1957):

$$V^*(s) = \max_a \{RT(s, a) + \gamma \sum_{s' \in S} \Phi(a, s, s') V^*(s')\} \quad (5)$$

Where the total reward RT is given by:

$$\begin{aligned} RT &= R_1(s, a) + R_2(s, a) + \dots + R_k(s, a) \quad (6) \\ &= \sum_{i=1}^K R_i(s, a) \end{aligned}$$

Given the additive value assumption, the total value can also be written in terms of the individual MDP values:

$$\begin{aligned} V^*(s') &= V_1^*(s') + V_2^*(s') + \dots + V_k^*(s') \quad (7) \\ &= \sum_{i=1}^K V_i^*(s') \end{aligned}$$

Substituting 6 and 7 in 5 we obtain:

$$\begin{aligned} V^*(s) &= \max_a \{ \sum_{i=1}^K R_i(s, a) \\ &+ \gamma \sum_{s' \in S} \Phi(a, s, s') \sum_{i=1}^K V_i^*(s') \} \quad (8) \end{aligned}$$

which can be written as:

$$V^*(s) = \max_a \{ \sum_{i=1}^K [R_i(s, a) + \gamma \sum_{s' \in S} \Phi(a, s, s') V_i^*(s')] \} \quad (9)$$

By definition:

$$Q_i^*(s, a) = R_i(s, a) + \gamma \sum_{s' \in S} \Phi(a, s, s') V_i^*(s') \quad (10)$$

so 9 can be written as:

$$V^*(s) = \max_a \{ \sum_{i=1}^K [Q_i^*(s, a)] \} \quad (11)$$

And by the definition of the Q function, $Q^*(s, a) = \sum_{i=1}^K [Q_i^*(s, a)]$, so:

$$\pi^*(s) = \operatorname{argmax}_a \{ \sum_{i=1}^K [Q_i^*(s, a)] \} \quad (12)$$

Q.E.D.

By using Theorem 1, we know give an algorithm for solving a Parallel MDP:

1. Solve each individual MDP by using value iteration (Puterman, 1994) and obtain the optimal Q_i values per action–state:

$$Q_i^*(s, a) = \{r_i(s, a) + \gamma \sum_{s' \in S} \Phi(a, s, s') V_i^*(s')\}$$

2. Compute the global optimal Q :

$$Q^*(s, a) = \sum_{i=1}^K Q_i^*(s, a)$$

3. Obtain the optimal policy:

$$\pi^*(s) = \operatorname{argmax}_a \{\sum_{i=1}^K Q_i^*(s, a)\}$$

We can think that, at any state of the problem, each MDP sends its value for each possible action (its Q value) and the “arbiter” selects the action with the greatest combined value.

Using a flat state representation, there is not reduction in complexity by using a parallel MDP, with respect to solving the compound problem (including all the subtasks) as a single MDP. However, there are several important advantages:

- In a similar way as hierarchical MDPs (Dietterich, 2000), parallel MDPs facilitate state abstraction. For instance, in the robot grid example, Navigation only needs to consider its position with respect to the goal; while Obstacle Avoidance might just include a local map with the distance to obstacles. By using these abstractions, an important reduction in the number of states can be achieved.
- The subtask decomposition is also helpful for learning. Each subtask Q function can be learned individually using reinforcement learning (Sutton and Barto, 1998), and then combined using our algorithm. In the grid world example, the robot can independently learn to go to the goal and to avoid obstacles, which is *easier* than learning both tasks simultaneously.
- From a practical perspective, in many areas such as robotics, there are different software programs for solving different aspects

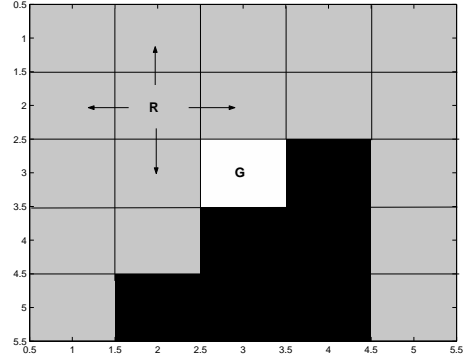


Figure 1: Grid world. This simulated environment consists of cells that can be free (gray), obstacle (black) and a goal (white). The robot (R) can move from each cell to its 4 neighbors as shown.

of the problem, such as path planning, obstacle avoidance, localization, etc. So it is better to consider each module as a task and combine their policies, instead of designing a single MDP for the complete task.

Next we illustrate our approach with an example of a simulated robot in the grid world.

4 Preliminary experiments

To test our approach, we consider a simulated robot in the grid world (see figure 1). The state is given by the coordinates (X, Y) of robot’s location in the grid. At each state the robot can move to its 4 neighbor locations, so there are 4 possible actions: *up*, *right*, *down*, *left*. The uncertainty in the actions is given by assuming that the robot has an 80% chance of going to the desired location, and 10% to each of the two adjacent locations. So this defines the transition function. In this experiments we consider two tasks: navigation and obstacle avoidance. The reward for navigation is a fixed amount for reaching the goal state and zero otherwise. The obstacle avoidance subtask receives a negative reward for going into a cell with an obstacle, and zero otherwise.

We tested with different grid sizes, goal positions and obstacle distributions, and compared the resulting policy of: (a) a parallel MDP with two subtasks, navigation and obstacle avoid-

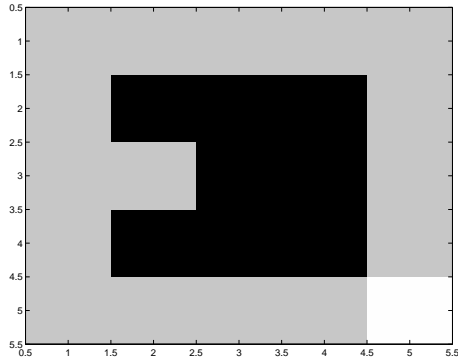


Figure 2: Test case used in the experiments. The black cells are obstacles and the white cell is the goal (in this case at 5, 5).

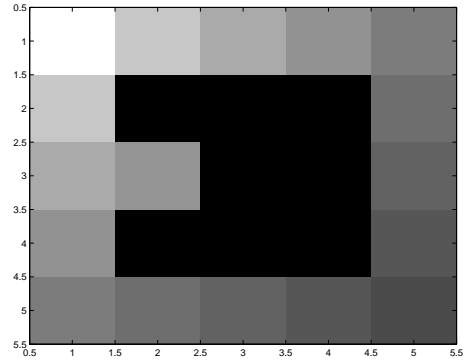


Figure 4: Value function for the goal at 1, 1. For each free cell, we show the value (as a gray level).

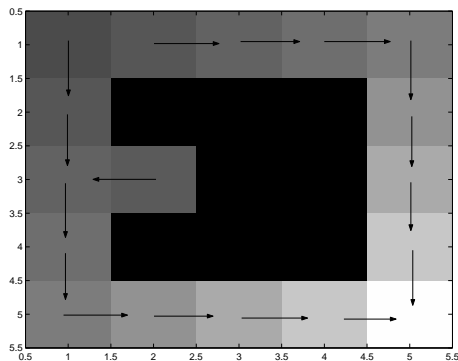


Figure 3: Value function and policy for the goal at 5, 5. For each free cell, we show the value (as a gray level) and the optimal action (arrow).

ance, and (b) a single MDP that considers both aspects. To illustrate the results we choose a small grid so the optimal value function and policy can be shown graphically. Figure 2 shows this test case. We show the results with the parallel MDP for two different position of the goal: (5, 5) and (1, 1). Figure 3 depicts the value and policy functions for the goal at (5, 5). The policy is represented in terms of the gray level of each cell, lighter represents a higher value. The policy is shown as arrows that correspond to the optimal action per state (cell). We can see that both, the value and the policy are optimal for this case.

Figure 4 shows the value function for the goal position at (1, 1). Again, this corresponds to the optimal value function. In these two exper-

iments, we solve the navigator task two times, one for each goal, but the obstacle avoidance only once.

In general, the results are the same for the parallel MDPs and for the single MDP. There are some cases in which there are small differences. These occur when an obstacle is at the border of the grid, so the parallel MDPs might try to go *out* of the grid to avoid the obstacles. Given that we are not representing explicitly the border around the grid, we think that these cases are due to this border effect, although they might require further investigation.

5 Conclusions and future work

We have presented a framework for solving complex planning problems by dividing them into several subtasks represented as MDPs. We obtain the optimal policy for each subtask, and then combine the results to obtain the optimal global policy. At any state of the problem, each MDP sends its value for each possible action (its Q value) and an “arbiter” selects the action with the greatest combined value. So we have the advantages of both, reactive and decision-theoretic planning: a reactive system based on a decision theoretical framework. We present an algorithm for solving parallel MDPs and prove it obtains the global optimum, assuming an additive value. Initial experiments with a simulated robot in the grid world show good results.

We are working on extending parallel MDPs

in several ways. We want to incorporate abstract states using a relation representation. We are also considering learning the Q values using reinforcement learning. An interesting extension is to include actions that can be done at the same time; for instance, a robot that is navigating and interacting with people. Finally, we want to use this framework for task coordination in a real robot performing a task that involves navigation and robot–human interaction.

References

- R.E. Bellman. 1957. *Dynamic Programming*. Princeton U. Press, Princeton, N.J.
- C. Boutilier, T. Dean, and S. Hanks. 1999. Decision-theoretic planning: structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94.
- Rodney Brooks. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, April.
- T. Dean and R. Givan. 1997. Model minimization in markov decision processes. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 106–111. AAAI.
- T. Dean and K. Kanasawa. 1989. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150.
- T. G. Dietterich. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.
- B. Hengst. 2002. Discovering hierarchy in reinforcement learning with hexq. In *Proceedings of the National Conf. on Artificial Intelligence*. AAAI.
- J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. 1999. Spudd: Stochastic planning using decision diagrams. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence, UAI-99*, pages 279–288.
- Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Pashkin, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. 1998. Solving very large weakly coupled Markov decision processes. In *Proc. AAAI*.
- R. Parr and S. Russell. 1997. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems (NIPS)*, volume 9. MIT Press.
- M.L. Puterman. 1994. *Markov Decision Processes*. Wiley, New York.
- R. Sutton and A. G. Barto. 1998. *Introduction to reinforcement learning*. MIT Press, Cambridge, MA.