Planetary Rover Control as a Markov Decision Process

Daniel S. Bernstein, Shlomo Zilberstein
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{bern,zilberstein}@cs.umass.edu

Richard Washington, John L. Bresina
NASA Ames Research Center, M/S 269-2
Moffett Field, CA 94035
{rwashington,jbresina}@arc.nasa.gov

Keywords planetary rover, decision theory, Markov decision process, reinforcement learning

Abstract

Planetary rovers must be effective in gathering scientific data despite uncertainty and limited resources. One step toward achieving this goal is to construct a high-level mathematical model of the problem faced by the rover and to use the model to develop a rover controller. We use the Markov decision process framework to develop a model of the rover control problem. We use Monte Carlo reinforcement learning techniques to obtain a policy from the model. The learned policy is compared to a class of heuristic policies and is found to perform better in simulation than any of the policies within that class. These preliminary results demonstrate the potential for using the Markov decision process framework along with reinforcement learning techniques to develop rover controllers.

1 Introduction

Planetary rovers have received increased attention in recent years and have been successfully deployed on actual space missions. A planetary rover is a small, unmanned vehicle that explores the surface of a planet, taking pictures and performing experiments. Among the many interesting problems that need to be solved in order to have a successful rover mission is a high-level decision-making problem. Frequently in the course of a mission, decisions have to be made regarding where to go and what to do at a given location, in order to maximize the amount of scientific return from the mission. One example of a decision that may occur is whether to take another picture at the current location or leave the current location and explore elsewhere.

This decision-making problem is difficult for a number of reasons. First, many of the decisions need to be made autonomously because of limited communication with Earth. The rover must choose a course of action based only on its local sensor readings and information about the objectives of the mission. Second, resource constraints need to be taken into account in the decision making. The time, battery capacity, and data storage that the rover has are all seriously limited and must be used wisely. Lastly, uncertainty needs to be taken into account. Actions in this domain have nondeterministic effects, and the controller for the rover needs to be closed-loop.

It is natural to try to formalize the high-level rover control problem as a decision-theoretic planning problem. With the decision-theoretic planning approach, we start with a model of the rover's environment that incorporates uncertainty in the form of probabilities. This model can be used to construct a plan that optimizes the expected value of a performance measure that we choose. In our case, the quantity to be optimized is a function of how effective the rover is at gathering scientific data. Once the plan is constructed, it can be integrated into the rover's execution architecture. During the mission, when a decision needs to be made, the plan is queried and returns a high-level action. This action is then compiled into a lower-level sequence of commands and executed. It is expected that training with a model will translate into intelligent decision making in real-time on the rover.

To be more specific, we formalize the problem as a discrete-time, continuous-state Markov decision process (MDP). Our model can be used to simulate the rover control problem, but it is not the type of model that can be used with classical MDP solution techniques, such as policy iteration. We could simplify the model further to be able to apply these techniques, but instead we use reinforcement learning techniques, which typically rely only on having a simulator that samples next-states and rewards given the current state. Although they are called learning algorithms, these techniques can be viewed as planning with a simulator-type model.

The application of reinforcement learning algorithms involves a significant amount of knowledge engineering, as there are several parameters to tune and design decisions to make. As such, we describe in this paper some of the decisions we made in the process of applying the techniques. After constructing our simulator and choosing the parameters for our reinforcement learning algorithm, we used the simulator to train our learning agent. The resulting policy performed better than any of a class of heuristic policies when evaluated on the simulator. Upon analyzing the learned strategy, we saw that it exploited subtleties in the model that we did not notice at first glance. These preliminary results suggest that the MDP framework and reinforcement learning techniques can be useful tools for designing controllers for planetary rovers.

We note that there has been other recent work on using decision-theoretic techniques to solve the rover control problem. Zilberstein and Mouaddib [8] discuss the application of the *progressive processing* framework to the problem. Bresina and Washington [3] study a way of incorporating uncertainty into the *Contingent Rover Language (CRL)* for rover control. These lines of work are all closely related and will likely inform each other in the future.

This paper is organized as follows. First, we formally describe the Markov decision process framework. Then we describe how the rover control problem can be formalized as an MDP. Next, we discuss ways of using knowledge to constrain the policy space. We describe a class of heuristic policies within this space and a way of using reinforcement learning techniques to find a policy in this space. Experimental results with both the heuristic policies and the learned policies are presented. Finally, we discuss the implications of the results and directions for future work.

2 Markov decision processes

Markov decision processes (MDPs) model an agent acting to maximize its long-term reward in a stochastic environment. We will consider discrete-time MDPs, where the process evolves at some discrete time scale $t = 0, 1, 2, 3, \ldots$. At each time step, the agent perceives the state of the environment, $s_t \in S$, and chooses an action, $a_t \in A_{s_t}$. One time step later, the environment produces a numerical reward, r_t, and a next state, s_{t+1}. The environment's dynamics are modeled by state-transition probabilities $P(s, a, s')$ and expected immediate rewards $R(s, a)$.

A *policy* for an MDP is a mapping from states to actions. Given an MDP, our aim is to find a policy that maximizes expected long-term reward. There are a number of different ways to define long-term reward, and thus a number of different optimality criteria. In our problem, the agent will be maximizing the total reward obtained over an episode of finite length, starting from the initial state s_0. (It is known that the episode will end, but the number of steps is not known a priori.) The end of an episode can be modeled as entry into an absorbing state that has an associated reward of zero. In this case, the agent's long-term reward is

$$\sum_{t=0}^{\infty} R_t(s_t, a_t),$$

where $R_t(s_t, a_t)$ is the reward received at time step t. This optimality criterion is commonly referred to as the undiscounted, infinite-horizon criterion because all rewards are given the same weight regardless of how far in the future they occur.

3 Rover problem formulation

It is a difficult task to come up with detailed descriptions of the problems that rovers will face on upcoming missions. We describe one scenario that we view as a simplified version of what missions in the near future will be like. Our model is not intended to be entirely realistic. It is simply a first step in developing complex and realistic simulators that can be used in the construction of real rover controllers. In taking an incremental approach to model construction, we can experiment with the simpler models and use the lessons learned from them to guide the development of the more complicated models.

Our scenario can be described as follows. There is a rover on Mars that communicates with Earth exactly once each Martian day. The rover sends information about its current state to Earth, and this information is incorporated into an MDP model. The MDP is solved, and the resulting policy is uplinked to the rover to direct the next day's activities. The resource levels at the time of communication determine the initial state s_0 of the MDP. We focus on one particular initial state (the one in which all the resource levels are maximized), so that we can easily understand our experimental results. However, we note that our solution technique should work equally well for any initial state.

During its day, the rover must gather data from various sites. There are several alternative ways to do this, and the rover's action choices dictate which is implemented. The action set, A, contains four actions: abandon the current site in favor of a new site, take a picture of the current site, perform a spectrometer experiment at the current site, and wait for the battery to recharge.

State Variable	**Domain**
Time Remaining	[0 hr, 12 hr]
Battery Capacity	[0 AH, 10 AH]
Data Storage	{0 M, 1 M, 2 M, 3 M, 4 M}
Difficulty	{easy, medium, hard}
Importance	$\{0, 1, \ldots, 9\}$
Data Gathered	$\{0, 1, 2\} \times \{0, 1\}$

Table 1: The domains of the state variables.

Having listed the actions, we now describe the set of states for which the rover must choose an action. Resource levels comprise three of the state features. These include the time remaining until the next communication, the battery capacity, and the storage available. Time and storage are nonrenewable resources. Battery capacity, on the other hand, can actually increase during the day. In our model, during the entire middle third of the day, the capacity level is at its maximum. This is a simplified model of the sun being in the right position to charge the battery.

In addition to the resource levels, information about the current site is included as part of the state. One piece of information is the *difficulty* of the current site. This is related to the probability that the rover will obtain satisfactory data from the site when it performs a spectrometer experiment. Another component is what we call the *importance* of the site. This is a measure of how valuable data from the site will be to the scientists. The importance component is a factor in the reward function for the model. The final component corresponds to what has already been accomplished at the current site. This feature can take on one of six different values. At any point, the rover has taken no pictures, one picture, or two pictures; also, it has either obtained satisfactory spectrometer data or not. The state space, S, is the set of all allowable assignments to the state variables. The domains for these variables are shown in Table 1. The first two are continuous, while the last four are discrete.

The state-transition function, $P(s, a, s')$, is described qualitatively as follows. Picture taking can consume time, storage, and battery capacity. As long as its resource levels are high enough, the rover will always be successful in taking a picture. Spectrometer experiments can consume time and battery capacity. In contrast to picture taking, there is always a chance that a spectrometer experiment will fail, regardless of the resource levels. Waiting consumes time but can lead to an increase in battery capacity. Finally, when the rover decides to go to a new site, the importance and difficulty components of the next site are chosen randomly from a uniform distribution. In addition, time and battery capacity can be consumed during the traversal to the new site. Descriptions of boundary conditions and the actual probability distributions corresponding to all of these state-transition rules are deferred to a longer version of the paper. As we mentioned before, $P(s, a, s')$ is not represented in tabular form (which would actually be impossible since the state space is continuous). We note that the parameters used in our simulator are rough estimates based on our present knowledge of the problem. We expect these estimates to improve over time as we obtain more information from field tests and missions.

To complete our presentation of the model, we must describe the reward function, $R(s, a)$. A reward of zero is obtained for all transitions except those that take the rover to a new site. When the rover does leave a site, its reward is the importance of the site it just left times a function of the number of pictures it was able to take and whether or not it obtained acceptable spectrometer data. The total reward for one episode is then the sum of the rewards for all the sites.

4 Solution approach

4.1 Restricting the policy space

Some knowledge engineering can be done to constrain the policy space for the agent. What we do is essentially put a layer in between the process and the agent that restricts the information available to the agent and the ways it can affect the process. Doing this has the advantage of making the problem simpler by reducing the size of the space that needs to be searched. Of course, a disadvantage is that potentially effective policies can be thrown out. This problem is addressed in the discussion section.

We start by describing the constraints on the action space of our agent. Recall that the available actions at a site are to leave the site, use the spectrometer, take a picture, or wait for the battery to recharge. We define a new, temporally-extended action called *investigate*. This macro-action consists of taking two pictures of the site and then performing spectrometer experiments until satisfactory data are obtained. In our new problem, the agent only has two choices: investigate the current site or leave it — there is nothing in between.

Next we describe the discretization of the state space, which restricts the information on which the agent can base its decisions. We use a very coarse discretization. The continuous set of time values is divided into six bins. The importance component

Threshold	Expected Reward	Std Dev
0	132.8	31.8
1	138.5	27.1
2	143.3	29.0
3	144.8	29.1
4	144.5	26.2
5	141.7	27.2
6	130.4	27.3
7	116.0	27.3
8	99.8	29.1
9	63.7	33.4

Table 2: Expected rewards for the threshold policies.

is not affected, and thus the agent can perceive ten different values for this component. All possible values for the rest of the features are clustered together. In other words, the agent can't distinguish between different values for those features, so they are effectively ignored in the learning and decision-making. Therefore, the agent can only distinguish between 60 different abstract states.

4.2 A heuristic policy

We experimented with a type of heuristic policy that we call a *threshold* policy, which is a subset of our constrained policy set. (Of course, the heuristic policy in no way requires the restrictions described above, but we describe things in this order for ease of exposition.) With this type of policy, the agent uses a threshold factor. If it comes upon a site with an importance value greater than or equal to the threshold, it investigates the site. Otherwise, it moves on to the next site.

We evaluated each of the ten possible threshold policies on our simulator. The results are shown in Table 2 (averaged over 500 episodes). It turns out that the optimal threshold value is 3. Intuitively, it makes sense that the optimal threshold is somewhere near the middle. If the threshold is set too low, then the rover doesn't discriminate enough and wastes its resources on unimportant sites. On the other hand, if the threshold is set too high, the rover wastes resources moving between sites and doesn't get enough useful work done.

4.3 Monte Carlo reinforcement learning

We now describe our approach to applying a reinforcement learning algorithm to the problem. The simulator provides our agent with the Markov state, but our coarse discretization of the state space makes it effectively partially observed. This factored into our decision about which reinforcement learning algorithm to use. Probably the most common reinforcement learning methods are temporal-difference learning algorithms, which "back up" the value of the next state to the current state. These algorithms enjoy nice theoretical properties when the state is assumed to be Markovian, but when the Markov state is only partially observed, the theoretical guarantees break down. In addition, empirical results point to Monte Carlo methods as a promising way of dealing with partial observability. We refer the reader to [2, 7] for more information on this subject.

Monte Carlo methods are simple reinforcement learning techniques that can be viewed as waiting until the end of an episode to do any credit assignment. With these methods, a state-action value function (or Q-function), $Q(s,a)$, is maintained. This is an estimate of the total expected reward obtained after starting in state s and executing action a. Implicit in the Q-function is a policy, namely the policy that at every state takes the action that is greedy with respect to the Q-function. The learning algorithm works as follows. The Q-values are initialized to some arbitrary values, and during the first episode the agent executes the corresponding policy, taking *exploratory* actions a fraction of the time determined by the parameter ϵ. Exploratory actions help the algorithm escape from local minima; in our case, taking exploratory actions simply means choosing the lower-valued action 100ϵ percent of the time. Consider the set of all state-action pairs that are encountered during the episode. For each pair (s,a) in this set, let $T(s,a)$ be the time step at which (s,a) first occurred. The Q-values are updated according to the following equation:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha \sum_{t=T(s,a)}^{\infty} R(s_t,a_t),$$

where α is a parameter called the *learning rate*. This is called *first-visit* Monte Carlo because the learning rule uses the total reward obtained after a state-action pair is first visited. Subsequent visits to the same state-action pair are ignored. With the new Q-values set, the process is repeated for subsequent episodes.

5 Experimental results

We used the Monte Carlo reinforcement learning technique described above with $\alpha = 0.01$ and $\epsilon = 0.3$. We performed a learning run consisting of 1,000,000 episodes. After the learning run, the resulting policy was evaluated on 500 episodes. The mean total reward was 148.4 with standard deviation 27.3. Compared to the results for the best threshold policy, this is an improvement that has a p-value of 0.023 in a standard two-sample t-test.

In order to understand this improved performance better, we looked carefully at the policies that were obtained from learning. The policies are similar to the threshold policy, except that they vary the threshold depending on the time remaining. This is an (admittedly simple) example of an automatic planner exploiting subtleties in the model that were not immediately apparent to the system designers. These preliminary results demonstrate that the MDP framework and reinforcement learning algorithms can be useful tools in designing controllers for planetary rovers. For complex problems such as planetary rover control, there is a limit to how well we can do with just heuristic policies.

6 Discussion

6.1 Finding better policies

There are a number of unanswered questions related to the model we have developed. One question is whether we can find policies that perform better in simulation than the ones we have studied thus far. Using state and action abstraction, we put severe constraints on the set of policies that were learnable by our agent. On the state abstraction side, we could make the discretization finer — allowing the agent to make "better informed" decisions. With regard to action abstraction, we could relax the all-or-none constraint that is currently imposed. This would allow the rover to do things such as leave a site after it has only taken one picture. Relaxing the constraints gives the agent more freedom to discover interesting and effective policies. The drawback of this, however, is that it can take a very long time to find even a decent policy in the enlarged policy space. Thus, we cannot be haphazard in this process. Perhaps a good way to deal with this tradeoff is to use *shaping*. With this approach, we start with a highly constrained problem and gradually relax the constraints over the course of learning.

Sometimes, MDPs have special structure that can be exploited in order to obtain good policies faster. Our model has interesting structure that we did not use in this work. In particular, the sites only depend on each other through the resources. In other words, when the rover decides to leave a site, the function that determines the next state only takes as input the time remaining, battery capacity, and data storage. This type of structure is reminiscent of work involving *weakly-coupled* MDPs [5, 6]. One way to exploit this structure is to have two value functions, a high-level function that only takes resource levels as input and is only updated during transitions between sites, and a low-level function that can take any of the state components as input and is updated at every time step. We have performed some informal experiments, and the results suggest that by combining the low-level and high-level value functions during learning, it is possible to get faster convergence than with a "flat" approach. We note that the high-level value function is closely related to what has been called the *opportunity cost* in previous work [8].

6.2 Generalizing the model

One of our main goals is to develop a realistic simulator from which we can derive policies that can be deployed on an actual rover. This will involve removing some of the simplifying assumptions on which our current simulator is based. One such assumption is that the parameters (e.g., importance, difficulty) of a new site are always drawn from a uniform distribution; they do not depend on the site that the rover has just left. In reality, the rover's next site is not just picked "at random." A plausible situation is one in which the rover has been assigned a fixed sequence of sites to explore, and it must make decisions at each site regarding how to perform activities and when to move on to the next site. Alternatively, the rover could be given a directed acyclic graph of sites it can visit. In this case, there are a number of possible ways to visit sites, and the rover is given the responsibility of choosing among them.

Another assumption we have made is that the state is fully observable to the agent. Without changing the dynamics of the simulator, we could just make some components of the state unobservable to the agent, resulting in a partially observable Markov decision process (POMDP). One component that perhaps should not be observed is the difficulty of obtaining data from a site. We should point out, however, that the results described in this paper would be the same even if the state were partially observable. This is because the learning algorithm ignored all of the state components except for time and importance.

Another way to make the model more realistic is to require that activities be performed during constrained time intervals. These time constraints arise because some experiments can only be performed under certain conditions which do not always hold. We would ideally like to add time constraints to the problem without greatly increasing its computational complexity.

In addition to making the model's high-level structure closer to what would be encountered on an actual mission, we would like to find accurate numerical parameters to describe various rover activities.

These activities include, but are not limited to, taking pictures (including panoramic images), performing various experiments, calibrating sensors, and performing failure detection. These parameters can be obtained by working with existing high-fidelity rover simulators in addition to the rovers themselves. In the course of experimenting with real rovers, we will get a better idea of which pieces of information must be incorporated into the activity models and which information can be ignored.

6.3 Coordinating multiple rovers

Another direction for future research is the extension of this work to multi-rover scenarios. Researchers have already pointed out the potential advantages of multi-rover missions over single-rover missions [4]. More rovers can lead to an increase in productivity, autonomy, and fault-tolerance. However, the coordination of multiple rovers introduces a new type of uncertainty and leads to a more complex decision-making problem. During a multi-rover mission, each rover must decide on a course of action based only on partial information about the states of the other rovers. In addition, decisions must be made regarding communication between rovers.

The multi-rover problem can be modeled as multiple agents controlling a Markov process in a distributed fashion. However, solving for a policy to control an MDP in a distributed manner is generally harder than solving for a centralized policy [1]. Thus, we need to develop heuristics that can yield acceptable solutions in a reasonable amount of time. One heuristic is to start by partitioning the set of sites that can be explored, assigning a subset to each rover, and solving the problem for each rover as an MDP. With this simple approach, however, we cannot have one rover take on the work of another rover if that rover encounters difficulties. Perhaps we could search for policies in which the rovers periodically communicate and reallocate the sites based on the current state. Of course, even this heuristic does not apply to multi-rover problems in which there are activities that require more than one agent to act together in a tightly-coupled manner. One of our goals is to find adequate ways to address this problem.

Acknowledgements

The authors thank Balaraman Ravindran and Ted Perkins for helpful conversations. This work was supported in part by the National Science Foundation under grants IRI-9624992 and INT-9612092, and by the National Aeronautics and Space Administration. Daniel Bernstein was supported by an NSF Graduate Research Fellowship and a NASA SSRP Fellowship.

References

[1] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov decision processes. In *Proceedings of the Sixteenth International Conference on Uncertainty in Artificial Intelligence*, 2000.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

[3] J. L. Bresina and R. Washington. Expected utility distributions for flexible, contingent execution. In *Proceedings of the AAAI-2000 Workshop: Representation Issues for Real-World Planning Systems*, 2000.

[4] T. Estlin, A. Gray, T. Mann, G. Rabideau, R. Castaño, S. Chien, and E. Mjolsness. An integrated system for mulit-rover scientific exploration. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 541–548, 1999.

[5] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 1998.

[6] R. Parr. Flexible decomposition algorithms for weakly coupled Markov decision problems. In *Proceedings of the Fourteenth International Conference on Uncertainty in Artificial Intelligence*, 1998.

[7] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[8] S. Zilberstein and A.-I. Mouaddib. Optimizing resource utilization in planetary rovers. In *Proceedings of the Second NASA International Workshop on Planning and Scheduling for Space*, 2000.