# Decision-Theoretic Control of Planetary Rovers

Shlomo Zilberstein[1], Richard Washington[2],
Daniel S. Bernstein[1], and Abdel-Illah Mouaddib[3]

[1] Univ. of Massachusetts, Dept. of Computer Science, Amherst, MA 01003, USA
[2] RIACS, NASA Ames Research Center, MS 269-3, Moffett Field, CA 94035, USA
[3] Laboratoire GREYC, Université de Caen, F14032 Caen Cedex, FRANCE

**Abstract.** Planetary rovers are small unmanned vehicles equipped with cameras and a variety of sensors used for scientific experiments. They must operate under tight constraints over such resources as operation time, power, storage capacity, and communication bandwidth. Moreover, the limited computational resources of the rover limit the complexity of on-line planning and scheduling. We describe two decision-theoretic approaches to maximize the productivity of planetary rovers: one based on adaptive planning and the other on hierarchical reinforcement learning. Both approaches map the problem into a Markov decision problem and attempt to solve a large part of the problem off-line, exploiting the structure of the plan and independence between plan components. We examine the advantages and limitations of these techniques and their scalability.

## 1 Towards Autonomous Planetary Rovers

The power of a mobile platform to perform science and explore the surface of distant planetary surfaces has long attracted the attention of the space exploration community. Unmanned rovers have been deployed on the Moon and on Mars, and they have been proposed for exploring other planets, moons, and small bodies such as asteroids and comets. The challenges and goals of planetary exploration pose unique constraints on the control of rovers, constraints that differentiate this domain from others that have traditionally been considered in mobile robotics. In addition, operation of a rover on a planetary surface differs significantly from operation of other distant spacecraft.

In this paper, we describe the problem of rover control and illustrate its unique aspects. We show how these characteristics have led us to consider utility as a fundamental concept underlying planetary exploration; this in turn directed our attention and effort to decision-theoretic approaches for planetary rover control. We will survey these approaches, particularly concentrating on two methods: one based on adaptive planning and the other on hierarchical reinforcement learning.

A planetary rover is first and foremost a science tool, carrying a suite of instruments to characterize a distant environment and to transmit information to Earth. These instruments may include cameras, spectrometers, manipulators, and sampling devices. Under some level of control from Earth-bound scientists and engineers, the rover deploys the instruments to gain information about the planetary surface. For example, in the Mars Smart Lander mission, currently planned for 2009, a rover will traverse a few kilometers between scientifically interesting sites. At each site, the rover will visit a number of targets (typically rocks) and deploy instruments on each one. In the current mission scenario, the targets and rover actions will be completely specified by scientists and rover engineers. The work presented in this paper would enable the rover to perform many of these steps autonomously.

The level of success of a rover mission is measured by the "science return," or amount of useful scientific data returned to the scientists on Earth. Although it is difficult to measure concretely, some attempts have been made to characterize it precisely for particular scenarios [30]. Criteria such as rover safety, navigation accuracy and speed, data compression ratios, and resource management contribute to science return. An important characteristic of using science return as a mission success criterion is that it is a quantity to be maximized, not a discrete goal to be achieved. This differs markedly from traditional applications of planning technology to mobile robotics. From the early days of planning, applications to robotics have typically concentrated on achieving discrete goals [18, 15, 28]. More recently, decision-theoretic planning has extended beyond all-or-none goals to handle overall reward [21, 29], offering a more suitable framework for planetary rover control.

Autonomous control of rovers on distant planets is necessary because the round-trip time for communication makes tele-operation infeasible. Many earth-based rovers, as well as lunar rovers to a certain extent, can be controlled via tele-operation, using advanced user interfaces to compensate for latency in communication links [11, 1]. For Martian or other distant planetary exploration, the latency increases beyond the limits of tele-operation. In addition, because of constraints on communication resources and cost, currently envisioned missions will limit communications to once or twice daily. Between these communication opportunities, the rover must operate autonomously.

An important and distinctive feature of planetary robotics, and a challenge for autonomous operations, is uncertainty. With planetary rovers, there is uncertainty about many aspects of sequence execution: exactly how long operations will take, how much power will be consumed, and how much data storage will be needed. Resources such as power and data storage are critical limits to rover operations; resource limits must be respected, but unused resources generally translate to wasted mission time and thus decreased productivity. Furthermore, there is uncertainty about environmental factors that influence such things as rate of battery charging or which scientific tasks are possible. In order to allow for both sources of uncertainty, a traditional spacecraft command plan is conservative: only limited operations are allowed within a single uplink, time and resource usage are based on worst-case estimates, and the plan contains fail-safe checks to avoid resource overruns. If an operation takes less time than expected, the rover waits until the time prescribed for the next operation. If an operation takes longer than expected, it may be terminated before completion; in some cases, all non-essential operations may be halted until a new command plan is received. These situations result in unnecessary delays and lost science opportunities.

An example is the Mars Smart Lander mission, where the rover will visit at most one target in a single uplink, and in fact the rover will only approach a target and place an instrument before waiting for the next command plan [22]. Although conservative, this is still an advance over previous rovers (Sojourner [23] or the 2003 Mars Exploration Rovers), which required multiple days to accomplish as much. The techniques

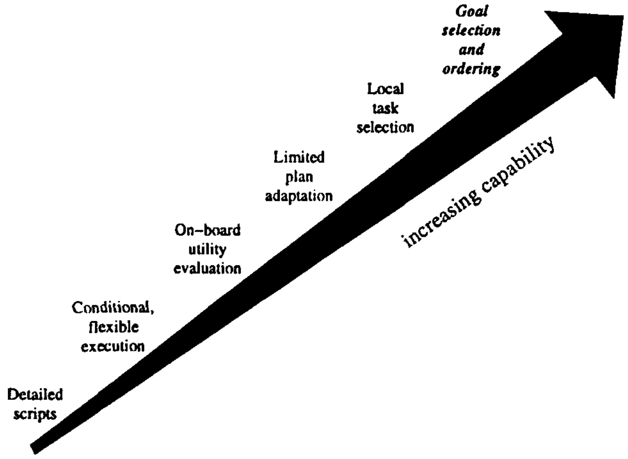
Goal selection and ordering
Local task selection
Limited plan adaptation
On-board utility evaluation
Conditional, flexible execution
Detailed scripts
increasing capability

**Fig. 1.** Increasing levels of capability for planetary rovers.

described in this paper provide the rover with the ability to select and balance tasks across multiple targets, allowing more ambitious exploration.

The highly uncertain operational environment distinguishes rover control from other spacecraft control. A deep space probe works in a harsh but stable environment, and its actions have relatively predictable effects, barring anomalies. Planning procedures designed for spacecraft [25, 17] do not explicitly handle all the types of uncertainty; applications of these technologies to the problem of rover control [14] rely on the presence of planners on board to replan when execution diverges from a single nominal plan.

The computational power of planetary rovers is also severely limited by the use of radiation-hardened, low-power processors and electronics. Increases in processor performance are more than made up for by the desire for increased on-board processing of images and science data, as well as improved navigation. In addition, the processor is a draw on the overall power budget. Thus control approaches that minimize on-board computation are preferable.

Constrained by action and environmental uncertainty, and limited computational resources, our objective is to increase the science productivity possible within a single uplink. To this end, we are pursuing a program of increasing capabilities, illustrated in Figure 1. Starting from the capabilities of the Sojourner rover, which used detailed, time-stamped scripts of low-level commands, we are moving toward autonomous goal selection and ordering. The latter is the main focus of this paper. Before presenting that work, we first review the steps along this spectrum of capabilities.

In all past and currently planned missions, the command plans for the rover are completely specified on the ground. In this case, additional flexibility in terms of time and state conditions, as well as contingent branches, may allow a wider range of behaviors than fixed-time sequences [6]. Additional capability can be realized by calculating utilities of plan branches with respect to the situation at execution time [7]. If we allow limited innovation on board, the rover can adapt its plan to the situation by skipping

**Fig. 2.** The K9 Rover.

steps or merging in plan fragments from a plan library constructed and verified on the ground [8].

The capabilities described to this point make use of plans that have been prespecified to full detail. To specify plans at a higher level of abstraction, such as desired science targets, the decomposition of the high-level tasks into detailed actions must be performed on board in a way that is sensitive to the execution context. Decision-theoretic planning and control methods can perform this dynamic choice to maximize science return.

If science targets are considered individually, the problem is a local problem of *task selection*. The control problem in this case is to decide which experiments to perform and when to move to another target [3]. The latter depends on the expected information to be gained from other targets and the difficulty of reaching them. The former depends on the available resources as well as characteristics of the target.

Alternatively, we may reason about a group of targets together; in planetary exploration this is often referred to as a *site*. By considering the activities within a site together, the overall science return can be improved compared to target-specific control policies. It is at this level of capability that we concentrate for the remainder of the paper.

The planning and execution techniques described in this paper allow the rover to re-prioritize and reorder scientific activities based on progress made, scientific observations, and the success or failure of past activities. The solution relies on off-line analysis of the problem and on pre-compilation of control policies. In addition, we have

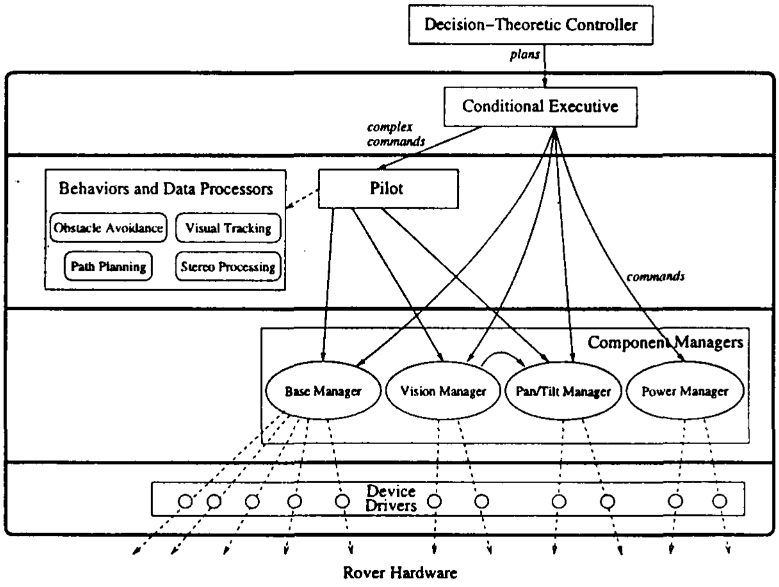

**Fig. 3.** Layers of rover software in the existing K9 rover software architecture.

used the independence between various mission tasks and goals to reduce the complexity of the control problem. The rest of the paper is organized as follows. Section 2 sketches the multiple layers of control and the way the decision-theoretic planning component interacts with the lower-levels. The section also provides a general introduction to decision-theoretic control and the two approaches we have developed. These approaches, adaptive planning and hierarchical reinforcement learning are detailed in sections 3 and 4. We conclude with a discussion of the merits of these two approaches and future work.

## 2 Layers of Control

The focus of this paper is on high-level, decision-theoretic control. However, the decision-theoretic component does not interact directly with the rover's actuators. It rests on a number of existing layers of control, which bridge the gap between decision-theoretic plans and the low-level control of the robotic mechanisms. In this section we describe the entire control architecture and the experimental platform for which it has been developed.

We are targeting our work for the NASA Ames "K9" rover prototype, pictured in Figure 2. The existing rover software architecture in place on the K9 rover consists of four distinct layers, as shown in Figure 3. Low-level device drivers communicate with hardware. Mid-level component controllers receive simple commands (such as direct movement, imaging, and instrument commands) and communicate with the device drivers to effectuate the commands. Abstract commands implement compound or complex actions (such as movement with obstacle avoidance, visual servoing to a target, and arm placement). A plan executive interprets command plans and calls both simple and abstract commands as specified in the plan. For more details on this architecture, see [9].

A high-level, decision-theoretic controller interacts with this architecture by proposing high-level actions, potentially more abstract than commands within the architecture. The high-level actions are then decomposed into small command plans; these command plans are provided to the rover plan executive, which in turn manages the execution and monitoring of the low-level commands within the command plans. Information about success of the high-level action and the resulting state of the system is returned to the decision-theoretic controller at the end of execution of each command plan.

The rover control problem, at the level that we are addressing, consists of a set of science-related goals. These science goals identify a set of targets, each of which has particular scientific interest. The rover has a set of instruments available, and thus a set of possible experiments, to gather relevant information about the targets. Given the set of targets and desired information, the rover's task is to choose activities that provide the maximum information possible about the targets within resource and time constraints and to return that information to the scientists.

## 3 Decision-Theoretic Control

The high-level control of the rover presents a sequential decision problem under uncertainty. At each point, the rover must select the next action based on the current state and the remaining plan. The state in this case includes both features characterizing the environment and features characterizing the rover itself, such as an indication of the remaining resources. Such problems can be modeled as a *Markov decision process* (MDP), assuming that each action transforms the current state into one of several possible outcome states with some fixed transition probability. This assumption is also referred to as the Markov assumption.

More formally, an MDP is defined by a finite set of *states*, $S$; a finite set of possible *actions*, $A$; and a *transition probability* function $Pr(s'|s,a)$ that indicates the probability that taking action $a \in A$ in state $s \in S$ results in a transition to state $s' \in S$. Each transition has an associated *reward*, $R(s,a)$, which can capture the cost of the action, the value of the outcome, or some combination of both. The objective is to maximize the reward over a finite- or infinite-horizon. In the later case, future reward is typically discounted by a factor of $\gamma^i$, where $i$ is the number of steps. Partially-observable MDPs (or POMDPs) generalize the MDP model by allowing the agent to have only partial information about the state. At the end of each action, the agent can make an observation $o \in \Omega$. A belief function over states can be maintained using Bayesian updating given the *observation probability* function, $Pr(o|s,a)$. (In general, both transition probabilities and observation probabilities may also depend on the outcome state.) We limit our discussion in this paper to MDPs, which provide adequate model for high-level rover control.

A solution to an MDP can be represented as a mapping from states to actions, $\pi : S \rightarrow A$, called a *policy*. Several dynamic programming algorithms (such as value iteration and policy iteration) have been developed for finding optimal control policies for MDPs [27, 31]. It has also been shown that MDPs can be solved using heuristic search by such algorithms as LAO* [19]. The advantage heuristic search has over dynamic programming is that, given an initial state, it can find an optimal solution without evaluating the entire state space. Dynamic programming, in contrast, evaluates the entire state space, finding a policy for every possible starting state. For problems with large state spaces, heuristic search offers substantial computational savings.

One important characteristic of the rover control problem is the explicit modeling of the amount of resources action use. Extensions of MDPs to handle duration of actions have been previously studied. For example, in Semi-Markov Decision processes (SMDPs) actions can have stochastic durations and state transitions are stochastic [27]. In Stochastic Time Dependent Networks (STDNs) actions can have stochastic durations, but state transitions are deterministic [34]. In Time-Dependent MDPs (TMDPs) actions can have stochastic, time-dependent durations and state transitions are stochastic [5]. The model we use in this paper have both stochastic state transitions and actions that consume varying levels of resources (not just time). In this sense, the model is a proper extension of the previous ones. While modeling the consumption of resources by actions is not difficult, it increases the state space dramatically. The number of states grow linearly with the number of resource units (which could be large) and exponentially with the number of resources. However, resources have additional characteristics that simplify their treatment: all the units of a resource are typically exchangeable, the number of units goes down as they are consumed (unless the resource is renewable), and the amount of resources used by an action typically depends only on the action itself. Therefore, treating resources as just any other component of the state is wasteful.

In the following two sections we examine two decision-theoretic techniques that take advantage of the unique characteristics of the rover control problem in order to simplify it. both approaches are based on modeling the rover set of activities as a loosley-coupled MDP. The scientific experiments the rover performs in each location are largely independent, but they share the same resources. The first approach develops a local policy for each activity that takes into account the remaining plan by computing a cost function over resources. By estimating quickly this cost function at run-time, we can avoid solving the entire MDP while producing near-optimal control policies. The second approach is based on a hierarchical reinforcement learning algorithm designed to take advantage of the natural decomposability offered by loosely-coupled MDPs. It maintains two different value functions: a low-level state-action value function defined over all state-action pairs and a high-level state value function defined only over "bottleneck" states that bridge the components of the MDP.

The two approaches share the ability to accelerate policy construction by exploiting the structure of the MDP, but they offer different advantages and disadvantages. The first approach exploits a model of the domain and allows for off-line policy construction and compact policy representation, both important issues in rover control. The second approach is model free and is particularly suitable for operation in poorly modeled environments or for adaptation of an existing policy to new environments. The next two sections describe the two approaches and examine their characteristics.

# 4 Adaptive Planning Approach

The adaptive planning approach is based on off-line analysis of each possible rover activity and construction of policies for each possible activity using dynamic programming. The key question is how to adapt pre-compiled policies at run-time to reflect the dynamic execution state of the plan. The dynamic information includes the remaining workload and the remaining resources, both of which can be captured by the notion of *opportunity cost.*

Each *plan* assigned to a rover is composed of a sequence of target *activities* represented as progressive processing task structures [24, 35]. An initial resource allocation is also specified. Resources are represented as vectors of discrete units. We assume here that the plan is totally ordered and that resources are not renewable. A generalization of the technique to acyclic graphs has been examined in [10].

## 4.1 The Rover Model

The rover can perform a certain set of predefined activities, each of which has an associated fixed task structure. The task structure is represented as a *progressive processing unit* (PRU), which is composed of a sequence of *steps* or *processing levels*, $(l_1, l_2, \ldots)$. Each step, $l_i$, is composed of a set of *alternative modules*, $\{m_i^1, m_i^2, \ldots\}$. Each module of a given step can perform the same logical function, but it has different computational characteristics defined by its *descriptor*. The module descriptor, $P_i^j((q', \Delta r)|q)$, of module $m_i^j$ is the probability distribution of output quality and resource consumption for a given input quality. Module descriptors are similar to *conditional performance profiles* of anytime algorithms.

When the rover completes an activity, it receives a reward that depends on the quality of the output and the specific activity. Each PRU has an associated *reward function*, $U(q)$, that measures the immediate reward for performing the activity with overall quality $q$. Rewards are cumulative over different activities.

Given a plan, a library of task structures that specify a PRU for each activity in the plan, the module descriptors of all the components of these PRUs, and corresponding reward functions for each activity, we want to select the best set of alternative modules to maximize the overall utility or scientific return of the rover.

## 4.2 Optimal Control of a Single Activity

We begin with the problem of meta-level control of a single progressive processing unit corresponding to a single activity. This problem can be formulated as a Markov decision process (MDP) with states representing the current state of the activity. The state includes the current level of the PRU, the quality produced so far, and the remaining resources. The rewards are defined by the utility of the solution. The possible actions are to *execute* one of the modules of the next processing level. The transition model is defined by the descriptor of the module selected for execution.

**State transition model.** The execution of a single progressive processing unit can be seen as an MDP with a finite set of states $\mathcal{S} = \{[l_i, q, r]\}$, where $i$ indicates the last executed level, $q$ is the quality produced by the last executed module, and $r$ is the remaining resources. When the system is in state $[l_i, q, r]$, one module of the $i$-th level has been executed. (The first level is $i = 1$; $i = 0$ is used to indicate the fact that no level has been executed.)

The initial state of the MDP is $[l_0, 0, r]$, where $r$ is the available resources for plan execution. (Additional resources may be reserved for rover operation once execution of the plan is complete.) The initial state indicates that the system is ready to start executing a module of the first level of the PRU. The terminal states are all the states of the form $[l_L, q, r]$, where $L$ is the last level of the PRU. In particular, the state $[l_L, 0, r]$ represents termination with no useful result and remaining resources $r$. Termination with $r = 0$ is also possible; if that happens during the execution of an intermediate level of the PRU, a last transition is made to the end of the PRU by skipping all the levels that cannot be performed.

In every nonterminal state, $[l_i, q, r]$, the possible actions, designated by $\mathbf{E}_{i+1}^j$, execute the $j$-th module of the next level. The outcome of action $\mathbf{E}_{i+1}^j$ is probabilistic. Resource consumption and quality uncertainties define the new state as follows.

$$Pr([l_{i+1}, q', r - \Delta r] \mid [l_i, q, r], \mathbf{E}_{i+1}^j) = P_{i+1}^j((q', \Delta r)|q) \qquad (1)$$

**Rewards and the value function.** Rewards are determined by the given reward function applied to the final outcome. Note that no rewards are associated with intermediate results, although this could be easily incorporated into the model. The value function (expected reward-to-go) over all states is defined as follows. The value of a terminal state is based on the utility of the results.

$$V([l_L, q, r]) = U(q) \qquad (2)$$

The value of a nonterminal state of the MDP is defined as follows.

$$V([l_i, q, r]) = \max_j \sum_{q', \Delta r} P_{i+1}^j((q', \Delta r)|q) \; V([l_{i+1}, q', r - \Delta r]) \qquad (3)$$

This concludes the definition of a finite-horizon MDP, or equivalently, a state-space search problem that can be represented by a decision tree or AND/OR graph. It can be solved using standard dynamic programming or using a search algorithm such as AO*.

Because the rover model satisfies the Markov property, it is easy to show that given an arbitrary PRU, an initial resource allocation and a reward function, the optimal policy for the corresponding MDP provides an optimal control strategy for the rover [36].

We note that the number of states of the MDP is bounded by the product of the number of levels, the maximum number of alternative modules per level, the number of discrete quality levels, and the number of possible resource vectors. While resources could vary over a wide range, the size of the control policy can be reduced by using coarse units. Therefore, unit choice introduces a tradeoff between the size of the policy and its effectiveness. An implementation of the policy construction algorithm for problems that involve one resource confirms the intuition that the optimal policy can be approximated with a coarse resource unit [36]. This observation leads to a significant reduction in policy size and construction time.

### 4.3 Optimal Control of Multiple Activities Using Opportunity Cost

Consider now the control of a complex plan composed of $n + 1$ PRUs. One obvious approach is to generalize the solution for a single PRU to sequences of PRUs. That is, one could construct a large MDP for the combined sequential decision problem including the entire set of $n + 1$ PRUs. Each state must include an indicator of the activity (or PRU) number, $i$, leading to a general state represented as $[i, l, q, r]$, $i \in \{0, 1, \ldots, n\}$. However, our objective is to eliminate the need to solve complex MDPs on-board by the rover. Transmitting to the rover a very large policy for the entire plan is also unacceptable. Instead, we examine a technique to factor the effect of the remaining plan on the current policy using the notion of opportunity cost.

We want to measure the effect of the remaining $n$ PRUs on the execution of the first one. This can be expressed in a way that preserves optimality while suggesting an efficient approach to meta-level control that does not requires run-time construction of the entire policy.

**Definition 1** *Let $V^*(i, r) = V([i, l_0, 0, r])$ for $i \leq n$, and $V^*(n + 1, r) = 0$. $V^*(i, r)$ denotes the expected value of the optimal policy for the last n – i PRUs with resources $r$.*

To compute the optimal policy for the $i$-th PRU, we can simply use the following modified reward function.

$$U_i'(q, r) = U_i(q) + V^*(i + 1, r) \qquad (4)$$

In other words, the reward for completing the $i$-th activity is the sum of the immediate reward and the reward-to-go for the remaining PRUs using the remaining resources. Therefore, the best policy for the first PRU can be calculated if we use the following reward function for final states:

$$U_0'(q, r) = U_0(q) + V^*(1, r) \qquad (5)$$

**Definition 2** *Let $OC(r, \Delta r) = V^*(1, r) - V^*(1, r - \Delta r)$ be the resource* **opportunity cost function.**

The opportunity cost measures the loss of expected value due to reduction of $\Delta r$ in resource availability when starting to execute the last $n$ PRUs.

**Definition 3** *Let the* **OC-policy** *for the first PRU be the policy computed with the following reward function:*

$$U_0'(q, r) = U_0(q) - OC(r_0, r_0 - r)$$

The OC-policy is the policy computed by deducting from the actual reward for the first task the opportunity cost of the resources it consumed.

**Theorem 1** *Controlling the first PRU using the OC-policy is globally optimal.*

**Proof:** From the definition of $OC(r, \Delta r)$ we get:

$$V^*(1, r_0 - \Delta r) = V^*(1, r_0) - OC(r_0, \Delta r) \tag{6}$$

To compute the optimal schedule we need to use the reward function defined in Equation 4 that can be rewritten as follows.

$$U_0'(q, r_0 - \Delta r) = U_0(q) + V^*(1, r_0) - OC(r_0, \Delta r) \tag{7}$$

Or, equivalently:

$$U_0'(q, r) = U_0(q) + V^*(1, r_0) - OC(r_0, r_0 - r) \tag{8}$$

But this reward function is the same as the one used to construct the OC-policy, except for the added constant $V^*(1, r_0)$. Because adding a constant to a reward function does not affect the policy, the optimality of the policy is preserved.

Theorem 1 suggests an optimal approach to control an arbitrary set of $n + 1$ activities by first using an OC-policy for the first PRU that takes into account the resource opportunity cost of the remaining $n$ activities. Then, the OC-policy for the second PRU is used taking into account the opportunity cost of the remaining $n - 1$ activities and so on.

### 4.4 Using Estimated Opportunity Cost and Precompiled Policies

How can we exploit the modularity introduced in the previous section to meet the objective of minimizing on-line planning? In particular, we want to avoid any complex procedure that involves computing the exact opportunity cost or re-constructing the corresponding OC-policies on-board. We also want to avoid constructing these policies at the control center because transmitting them to the rover is not feasible. Instead, a solution based on the following two principles has been developed [36].

1. A fast approximation scheme is derived off-line to estimate the opportunity cost of an arbitrary given plan; and
2. Pre-compiled policies are stored on-board to control each activity for different levels of opportunity cost.

We have examined several approaches to estimating the opportunity cost of one resource. Function approximation techniques seem to be suitable for learning the opportunity cost from samples of examples for which we can compute the exact cost off-line. In order to avoid computing a new policy (for a single PRU) each time the opportunity cost is revised, we can divide the space of opportunity cost into a small set of regions representing typical situations. For each region, an optimal policy is computed off-line and stored in a library. At run-time, the system must first estimate the opportunity cost and then use the most appropriate pre-compiled policy from the library. These policies remain valid as long as the overall task structure and the utility function are fixed.

# 5 Hierarchical Reinforcement-Learning Approach

Another approach to the rover control problem that exploits its MDP representation is based on hierarchical reinforcement learning [2]. There has been increased interest in recent years in classes of MDPs that are naturally decomposable and in developing special-purpose techniques for these classes [4]. The rover control problem can be modeled as a weakly-coupled MDP, which falls in this category. A weakly-coupled MDP is an MDP that has a natural decomposition into a set of subprocesses. The transition from one subprocess to another requires entry into one of a small set of bottleneck states. Because the subprocesses are only connected through a small set of states, they are "almost" independent. The common intuition is that weakly-coupled MDPs should require less computational effort to solve than arbitrary MDPs.

The algorithm that was investigated is a reinforcement-learning version of a previously studied planning algorithm for weakly-coupled MDPs [13]. The planning algorithm is model-based, whereas the learning algorithm requires only information from experience trajectories and knowledge about which states are the bottleneck states. This can be beneficial for problems where only a simulator or actual experience are available. The algorithm fits into the category of hierarchical reinforcement learning (e.g., [32]) because it learns simultaneously at the state level and at the subprocess level. We note that other researchers have proposed methods for solving weakly-coupled MDPs [16, 20, 26], but very little work has been done in a reinforcement learning context.

The hierarchical algorithm has been compared with Q-learning; it is shown to perform better initially, but it fails to converge to the optimal policy. Hence, the hierarchical approach could be beneficial in situations in which computation time is limited and a fast approximation of the optimal policy is needed. Surprisingly, a third algorithm which is given the optimal values for the bottleneck states at the start learns more slowly. We discuss this counterintuitive observation at the end of this section.

## 5.1 Solving MDPs Using Reinforcement Learning

Consider an MDP that contains a finite set $S$ of states, with $s_0$ being the start state. For each state $s \in S$, $A_s$ is a finite set of possible actions. $P$ is the table of transition probabilities, where $P(s'|s,a)$ is the probability of a transition to state $s'$ given that the agent performed action $a$ in state $s$. $R$ is the reward function, where $R(s,a)$ is the reward received by the agent given that it chose action $a$ in state $s$. In this section, we use the infinite-horizon discounted optimality criterion. Formally, the agent should maximize

$$E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))\right], \tag{9}$$

where $\gamma \in [0,1]$ is the discount factor.

Algorithms for MDPs often solve for *value functions*. For a policy $\pi$, the state value function, $V^\pi(s)$, gives the expected total reward starting from state $s$ and executing $\pi$. The state-action value function, $Q^\pi(s,a)$, gives the expected total reward starting from state $s$, executing action $a$, and executing $\pi$ from then on.

Reinforcement learning techniques are particularly useful when only a a simulator or real experience are available [31]. With these techniques, experience trajectories are used to learn a value function for a good policy. Actions taken on a trajectory are usually greedy with respect to the current value function, but *exploratory* actions must also be taken in order to discover better policies. One widely-used reinforcement learning algorithm is Q-learning [33], which updates the state-action value function after each transition from $s$ to $s'$ under action $a$ with the following rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right], \tag{10}$$

where $\alpha$ is the learning rate.

### 5.2 Reinforcement Learning for Weakly-Coupled MDPs

Consider an MDP with a state set $S$ that is partitioned into disjoint subsets $S_1, \ldots, S_m$. The *out-space* of a subset $S_i$, denoted $O(S_i)$, is defined to be the set of states not in $S_i$ that are reachable in one step from some state in $S_i$. The set of states $B = O(S_1) \cup \cdots \cup O(S_m)$ that belong to the out-space of at least one subset comprise the set of *bottleneck* states. If the set of bottleneck states is relatively small, we call the MDP *weakly-coupled.*

In [13], the authors describe an algorithm for weakly-coupled MDPs that can be described as a type of policy iteration. Initially, values for the bottleneck states are set arbitrarily. The low-level policy improvement phase involves solving each subproblem, treating the bottleneck state values as terminal rewards. The high-level policy evaluation phase consists of reevaluating the bottleneck states for these policies. Repeating these phases guarantees convergence to the optimal policy in a finite number of iterations.

The rules for backpropagating value information in the reinforcement learning algorithm are derived from the two phases mentioned above. Two benefits of this approach are that it does not require an explicit model and that learning can proceed simultaneously at the high level and at the low level.

Two different value functions must be maintained: a low-level state-action value function $Q$ defined over all state-action pairs and a high-level state value function $V_h$ defined only over bottleneck states. The low-level part of the learning is described as follows. Upon a transition to a non-bottleneck state, the standard Q-learning backup is applied. However, when a bottleneck state $s' \in B$ is encountered, the following backup rule is used:

$$Q(s,a) \leftarrow Q(s,a) + \alpha_l \left[R(s,a) + \gamma V_h(s') - Q(s,a)\right], \tag{11}$$

where $\alpha_l$ is a learning rate. For the purposes of learning, the bottleneck state is treated as a terminal state, and its value is the terminal reward. High-level backups occur only upon a transition to a bottleneck state. The backup rule is:

$$V_h(s) \leftarrow V_h(s) + \alpha_h[R + \gamma^k V_h(s') - V_h(s)], \tag{12}$$

where $k$ denotes the number of time steps elapsed between the two bottleneck states, $R$ is the cumulative discounted reward obtained over this time, and $\alpha_h$ is a learning rate.

It is possible to alternate between phases of low-level and high-level backups or to perform the backups simultaneously. Whether either approach converges to an optimal policy is an open problem. We chose the latter for our experiments because our preliminary work showed it to be more promising.

### 5.3 The Rover Model

The rover control problem fits nicely the weakly-coupled MDP framework. In this section we evaluate the approach using a simple scenario. In this scenario, a rover is to operate autonomously for a period of time. As in Section 4, the overall plan is composed of a sequence of activities, each of which includes a set of data gathering actions with respect to a certain target. Each activity has an associated priority and estimated difficulty of obtaining the data. The rover must make decisions about which activities to perform and when to move from one target to the next. The goal is to maximize the value of the collected data over a given time period.

The action set consists of taking a picture, performing a spectrometer experiment, and traversing to the next target in the sequence. Spectrometer experiments take more time and are less predictable than taking pictures, but they yield better data. The time to traverse between targets is a noisy function of the distance between them. The state features are the remaining time, the current target number (from which priority and estimated difficulty are implicitly determined), the number of pictures taken of the current target, and whether or not satisfactory spectrometer data has been obtained. Formally, $S = T \times I \times P \times E$, where $T = \{0 \text{ min}, 5 \text{ min}, \ldots, 300 \text{ min}\}$ is the set of time values; $I = \{1, 2, 3, 4, 5\}$ is the set of targets; $P = \{0, 1, 2\}$ is the set of values for pictures taken; and $E = \{0, 1\}$ is the set of values for the quality of the spectrometer data. The start state is $s_0 = \langle 300, 1, 0, 0 \rangle$. The sequence of targets used for our experiments is shown in Table 1.

**Table 1.** The sequence of targets for the rover to investigate

| Target | Priority | Estimated difficulty | Distance to next target |
|---|---|---|---|
| 1 | 8 | medium | 3 m |
| 2 | 5 | hard | 5 m |
| 3 | 3 | easy | 7 m |
| 4 | 2 | easy | 3 m |
| 5 | 9 | hard | N/A |

A nonzero reward can only be obtained upon departure from a target location and is a function of the priority of the target and the data obtained about the target. The task is episodic with $\gamma = 1$. An episode ends when the time component reaches zero or the rover finishes investigating the last target. The aim is to find a policy that maximizes the expected total reward across all targets investigated during an episode.

In order to see how this problem fits into the weakly-coupled MDP framework, consider the set of states resulting from a traversal between targets. In all of these states,

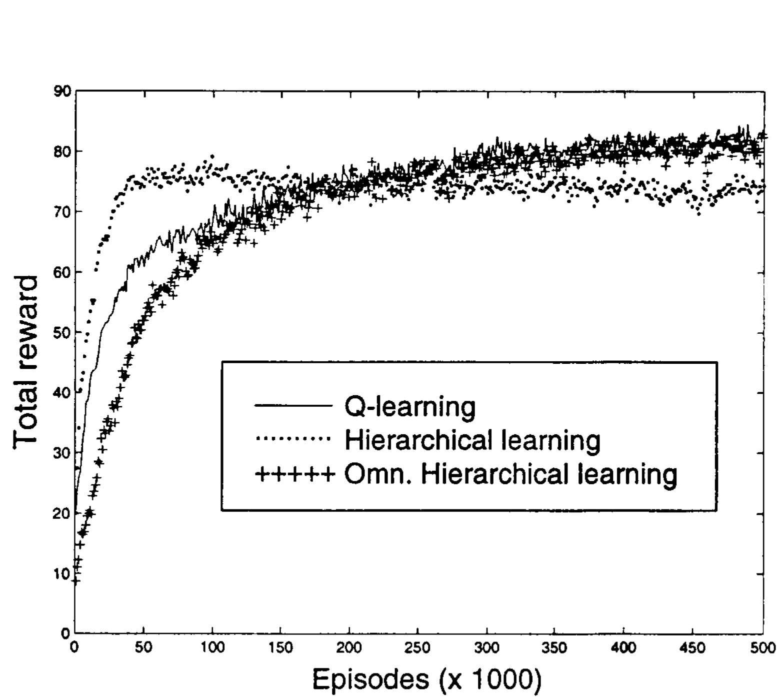

**Fig. 4.** Learning curves for Q-learning, hierarchical learning, and omniscient hierarchical learning

the picture and spectrometer components of the state are reset to zero. The set $B = T \times I \times \{0\} \times \{0\}$ is taken to be the set of bottleneck states, and it is over this set that we define the high-level value function. Note that the bottleneck states comprise only 300 of the problem's 1,800 states.

### 5.4 Experiments

The hierarchical algorithm has been tested against Q-learning using the above scenario. In addition, we tested an algorithm that we call the *omniscient* hierarchical learning algorithm. This algorithm is the same as the hierarchical algorithm, except that the values for the bottleneck states are fixed to optimal from the start, and only low-level backups are performed. By fixing the bottleneck values, the problem is completely decomposed from the start. Of course, this cannot be done in practice, but it is interesting for the purpose of comparison.

For the experiments, all values were initialized to zero, and we used $\epsilon$-greedy exploration with $\epsilon = 0.1$ [31]. For the results shown, all of the learning rates were set to 0.1 (we obtained qualitatively similar results with learning rates of 0.01, 0.05, and 0.2). Figure 4 shows the total reward per episode plotted against the number of episodes of learning. The points on the curves represent averages over periods of 1000 episodes.

A somewhat counterintuitive result is that the omniscient hierarchical algorithm performs worse than both the original hierarchical algorithm and Q-learning during the early stages. One factor contributing to this is the initialization of the state-action values to zero. During the early episodes of learning, the value of the "leave" action grows more quickly than the values for the other actions because it is the only one that leads directly to a highly-valued bottleneck state. Thus the agent frequently leaves a target without having gathered any data. This result demonstrates that decomposability does not always guarantee a more efficient solution.

The second result to note is that the hierarchical algorithm performs better than Q-learning initially, but then fails to converge to the optimal policy. It is intuitively plausible that the hierarchical algorithm should go faster, since it implicitly forms an abstract process involving bottleneck states and propagates value information over multiple time steps. It also makes sense that the algorithm does not converge once we consider that the high-level backups are *off policy*. This means that bottleneck states are evaluated for the policy that is being executed, and this policy always includes non-greedy exploratory actions. Algorithms such as Q-learning, on the other hand, learn about the policy that is greedy with respect to the value function regardless of which policy is actually being executed.

## 6 Discussion

We have described two decision-theoretic approaches to control planetary rovers. The two approaches share several characteristics: they both use an MDP representation of the control problem and they both exploit the fact that the plan components are only loosely-coupled. Both techniques use approximations; in previous experimentation with small scale problem instances, both produced near-optimal control. However, it is hard to predict how the optimality of the resulting control policies degrades with problem complexity and which one of the techniques will be more robust. This remains an open problem.

Both of the techniques are designed to minimize the complexity of on-line planning, and they both rely on pre-computing and storing control policies on-board. The size of these control policies could be substantial, but the required space is significantly smaller than the space needed for a complete policy for the entire plan. One advantage of the adaptive planning approach is that the control policies can be applied to an *arbitrary* plan, as long as the plan components are defined using known PRUs. This is not the case with the hierarchical reinforcement-learning technique. However, the latter has the advantage of not relying on knowing the exact model of the environment. Moreover, learning could be used on-board to refine a pre-calculated policy and adapt it to the real environment of operation.

A considerable amount of work remains to be done to examine the scalability of both solution techniques. In theory, if reinforcement learning is applied correctly, it can handle very large problems. However, in practice, this is a challenging problem. The scalability of the adaptive planning approach may be more predictable. We expect it to handle well larger plans in terms of the number of activities and their complexity. However, estimating the opportunity cost of multiple resources seems hard. The quality of these estimates degrades as the number of possible activities grows. Resource costs, unfortunately, are not independent, leading to exponential growth in the number of necessary pre-compiled policies as the number of resources increases.

Another important source of complexity comes from generalizing the topology of the plan, allowing cycles within an activity and partially-ordered activities in a plan. The utility of repeating an activity, such as taking pictures or collecting samples, is non-additive over the set of repetitions and may depend on the degree of success with previous attempts. This could lead to a significant increase in the number of state variables of the MDP. Constructing policies for MDPs with cycles is harder, but this has been addressed effectively by existing dynamic-programming and reinforcement-learning algorithms.

Introducing temporal and other constraints on activities is another important generalization that is the focus of current research. It would allow us to represent scientific or operational constraints on rover operations, such as illumination for imaging (or lack thereof for some spectral measurements), temperature for instrument performance, or pre-defined communication windows with Earth or orbiting relay satellites. Such constraints introduce interaction between plan components that we managed to avoid so far.

Yet another source of complication is partial observability of the quality of scientific data. If quality represents a simple aspect of the collected data, such as the resolution of an image, then we can assume that quality is fully observable. However, if we want to measure the actual quality of the scientific data, this can lead to additional tradeoffs in planning and execution. Estimating quality may be a non-trivial computational task that returns imperfect information. Integrating on-board data interpretation processes as part of the overall planning and control problem is another focus of current research efforts.

The results surveyed in this paper are part of a long term research program to make planetary rovers more autonomous and more productive. It is important to maintain in this effort a delicate balance between the various objectives and to aim for an appropriate level of autonomy for the task. Given the necessary interactions of the scientists with "their" rover on a planetary surface (to monitor the exploration and make ultimate determination of what is interesting), it does not seem necessary to give rovers a single goal, such as "find life" and leave it to its own devices. As rovers become more versatile and scientists produce more sophisticated science interpretation instruments and on-board analysis programs, we can imagine the level of autonomy increasing, requiring more sophisticated on-board planning and execution mechanisms. For the time being, the approaches we described match the level of autonomy needed to perform the mission efficiently and effectively.

## References

1. Bapna, D., Rollins, E., Murphy, J., Maimone, E., Whittaker, W., Wettergreen, D.: The Atacama Desert Trek: Outcomes. *IEEE International Conference on Robotics and Automation (ICRA-98)* (1998) 597–604
2. Bernstein, D.S., Zilberstein, S.: Reinforcement Learning for Weakly-Coupled MDPs and an Application to Planetary Rover Control. *European Conference on Planning* (2001)
3. Bernstein, D.S., Zilberstein, S., Washington, R., Bresina, J.L.: Planetary Rover Control as a Markov Decision Process. *Sixth International Symposium on Artificial Intelligence, Robotics, and Automation in Space* (2001)

4. Boutilier, C., Dean, T., Hanks, S.: Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research* **1** (1999) 1–93
5. Boyan, J.A., Littman, M.L.: Exact Solutions to Time-Dependent MDPs. *Advances in Neural Information Processing Systems* MIT Press, Cambridge, MA (2001)
6. Bresina, J.L., Golden, K., Smith, D.E., Washington, R.: Increased Flexibility and Robustness of Mars Rovers. *Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space* (1999)
7. Bresina, J.L., Washington, R.: Expected Utility Distributions for Flexible, Contingent Execution. *AAAI-2000 Workshop: Representation Issues for Real-World Planning Systems* (2000)
8. Bresina, J.L., Washington, R.: Robustness Via Run-Time Adaptation of Contingent Plans. *AAAI Spring Symposium on Robust Autonomy* (2001)
9. Bresina, J.L., Bualat, M., Fair, M., Washington, R., Wright, A.: The K9 On-Board Rover Architecture. *European Space Agency (ESA) Workshop on On-Board Autonomy* (2001)
10. Cardon, S., Mouaddib, A.-I., Zilberstein, S., Washington, R.: Adaptive Control of Acyclic Progressive Processing Task Structures. *Seventeenth International Joint Conference on Artificial Intelligence* (2001) 701–706
11. Christian, D., Wettergreen, D., Bualat, M., Schwehr, K., Tucker, D., Zbinden, E.: Field Experiments with the Ames Marsokhod Rover. *Field and Service Robotics Conference* (1997)
12. Dean, T., Boddy, M.: An Analysis of Time-Dependent Planning. *Seventh National Conference on Artificial Intelligence* (1988) 49–54
13. Dean, T., Lin, S.-H.: Decomposition Techniques for Planning in Stochastic Domains. *Fourteenth International Joint Conference on Artificial Intelligence* (1995) 1121–1127
14. Estlin, T., Gray, A., Mann, T., Rabideau, G., Castano, R., Chien, S., Mjolsness, E.: An Integrated System for Multi-Rover Scientific Exploration. *Sixteenth National Conference on Artificial Intelligence* (1999) 541–548
15. Fikes, R., Nilsson, N.: Strips: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* **2** (1971) 189–208
16. Foreister, J.-P., Varaiya, P.: Multilayer Control of Large Markov Chains. *IEEE Transactions on Automatic Control* **23**(2) (1978) 298–304
17. Fukunaga, A., Rabideau, G., Chien, S., Yan, D.: Toward an Application Framework for Automated Planning and Scheduling. *International Symposium on Artificial Intelligence, Robotics and Automation for Space* (1997)
18. Green, C.: Application of Theorem Proving to Problem Solving. *First International Joint Conference on Artificial Intelligence* (1969) 219–239
19. Hansen, E.A., Zilberstein, S.: LAO*: A Heuristic Search Algorithm that Finds Solutions with Loops. *Artificial Intelligence* **129**(1-2) (2001) 35–62
20. Hauskrecht, M., Meuleau, N., Kaelbling, L.P., Dean, T., Boutilier, C.: Hierarchical Solution of Markov Decision Processes Using Macro-Actions. *Fourteenth International Conference on Uncertainty in Artificial Intelligence* (1998)
21. Kaelbling, L., Littman, M., Cassandra, A.: Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence* **101**(1-2) (1998) 99–134
22. Limonadi, D.: Smart Lander Reference Surface Scenario and Reference Vehicle Description. Jet Propulsion Laboratory Interoffice Memorandum, November 2 (2001)
23. Mishkin, A.H., Morrison, J.C., Nguyen, T.T., Stone, H.W., Cooper, B.K., Wilcox, B.H.: Experiences with Operations and Autonomy of the Mars Pathfinder Microrover. *IEEE Aerospace Conference* (1998)
24. Mouaddib, A.-I., Zilberstein, S.: Optimal Scheduling of Dynamic Progressive Processing. *Thirteenth Biennial European Conference on Artificial Intelligence* (1998) 449–503
25. Muscettola, N., Nayak, P.P., Pell, B., Williams, B.C.: Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence* **103**(1-2) (1998) 5–47

26. Parr, R.: Flexible Decomposition Algorithms for Weakly-Coupled Markov Decision Problems. *Fourteenth International Conference on Uncertainty in Artificial Intelligence* (1998)
27. Puterman, M.L.: Markov Decision Processes– Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY (1994)
28. Sacerdoti, E.D.: Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence* 5(2) (1974) 115–135
29. Simmons, R., Koenig, S.: Probabilistic Robot Navigation in Partially Observable Environments. *Fourteenth International Joint Conference on Artificial Intelligence* (1995) 1080–1087
30. Stoker, C., Roush, T., Cabrol, N.: Personal communication (2001)
31. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
32. Sutton, R.S., Precup, D., Singh, S.: Between MDPs and Semi-MDPs: Learning, Planning, and Representing Knowledge at Multiple Temporal Scales. *Artificial Intelligence*, **112** (2000) 181–211
33. Watkins, C.: Learning from Delayed Rewards. PhD Thesis, Cambridge University, Cambridge, England (1989)
34. Wellman, M.P., Larson, K., Ford, M., Wurman, P.R.: Path Planning under Time-Dependent Uncertainty. *Eleventh Conference on Uncertainty in Artificial Intelligence* (1995) 532–539.
35. Zilberstein, S., Mouaddib, A.-I.: Reactive Control of Dynamic Progressive Processing. *Sixteenth International Joint Conference on Artificial Intelligence* (1999) 1268–1273
36. Zilberstein, S., Mouaddib, A.-I.: Adaptive Planning and Scheduling of On-Board Scientific Experiments. *European Space Agency (ESA) Workshop on On-Board Autonomy* (2001)