Scaling Up Decision Theoretic Planning to Planetary Rover Problems

Nicolas Meuleau[*] and **Richard Dearden**[†] and **Rich Washington**[†]
NASA Ames Research Center, Mail Stop 269-3
Moffet Field, CA 94035-1000
{nmeuleau, dearden, richw}@email.arc.nasa.gov

Abstract

Because of communication limits, planetary rovers must operate autonomously during consequent durations. The ability to plan under uncertainty is one of the main components of autonomy. Previous approaches to planning under uncertainty in NASA applications are not able to address the challenges of future missions, because of several apparent limits. On another side, decision theory provides a solid principled framework for reasoning about uncertainty and rewards. Unfortunately, there are several obstacles to a direct application of decision-theoretic techniques to the rover domain. This paper focuses on the issues of structure and concurrency, and continuous state variables. We describes two techniques currently under development that address specifically these issues and allow scaling-up decision theoretic solution techniques to planetary rover planning problems involving a small number of goals.

Introduction

There are many problems inherent in direct human control of remote devices such as planet exploratory rovers and satellites: (i) Communication takes significant time. For instance, the minimum delay for communicating with a rover on the surface of Mars is roughly 20 minutes, making teleoperation very hazardous; (ii) There may be obstacles blocking the communication between Earth and the device. For instance, the rover may be on the wrong side of the planet; (iii) The Deep Space Network is the only way to communicate with these devices, and it is highly oversubscribed. Therefore, remote rovers and satellites must be able to operate autonomously over substantial periods of time. The Mars Exploration Rovers (MER), for example, are designed to communicate with the ground only twice per Martian day and must operate autonomously the rest of the time.

Moreover, the surfaces of planets are very uncertain environments. In the case of Mars, there is uncertainty about the terrain, the meteorological conditions, and the state of the rover itself (position, battery charge, solar panels, component wear, etc.), resulting in a great deal of uncertainty in the duration, energy consumption, and outcome of the rover's actions (Bresina *et al.* 2002). This may have a serious impact on missions. It has been estimated that the 1997 Mars Pathfinder rover spent between 40% and 75% of its time doing nothing because plans did not execute as expected.

The need for autonomy and robustness in the face of uncertainty will grow as rovers become more capable and as missions explore more distant planets. The MER rovers require an average of 3 days to visit a single rock. However, with recent progress in areas such as automatic instrument placement (Pedersen *et al.* 2003), multiple rock visits in a single communication cycle will be possible in future missions.[1] When this happens, it is probable that the expectations of space scientists will increase dramatically and that rovers will end up highly oversubscribed. Moreover, planning for Europa and Titan exploration will require reasoning over much longer time frames, in more uncertain environments. Simple unconditional plans as used by MER will probably have a very low probability of success in such a context, so that the robot would spend almost all its time waiting for new orders from mission control. Moreover, planning for telescopes such as SOFIA or SIRTF also demonstrates a need for planning under uncertainty for large sets of tasks.

Planning systems that have been developed for planetary rovers and other NASA applications typically use a deterministic model of the environment and action effects (Muscettola *et al.* 1998; Jónsson *et al.* 2000; Estlin *et al.* 2002). Given a pre-specified set of goals, they produce a deterministic sequence of actions that achieves the goals under nominal conditions. They do not model the uncertainty in the domain, but instead rely on replanning to handle unexpected events. This straightforward approach presents several drawbacks:

The tradeoff of the value and risk of goals: Greedily achieving the highest priority goal can be a very poor strategy. For instance, a rover might try single-mindedly to get to a goal that is in fact unreachable, while neglecting other goals that are slightly less valuable but much easier to reach. A fine trade-off between the value of a goal and the likelihood of achieving it is necessary to act opportunistically.

The choice of branch time/point: Waiting for failure to change plans can also be very inefficient, since failure

[*] QSS Group Inc.
[†] Research Institute for Advanced Computer Science.

[1] This capability will be demonstrated during field tests at NASA Ames Research Center in Fall 2004.

time can be too late to respond intelligently to the new situation. A system must consider changing its plans when it can predict a possible failure. For instance, a rover should change its route when it perceives a major obstacle instead of waiting until it actually reaches the obstacle.

The problem of set-up actions: The prototypical example of this problem is identifying the benefit of putting a spare tire in the trunk before going on a car trip. Taking the spare tire has no benefit on the nominal plan; however, it will prove very useful if we have an unexpected flat tire. In the context of Mars rovers, this problem appears when choosing waypoints and preparing instruments. One may prefer a waypoint not directly on the path to the nominal goal because alternative goals are more easily reached from this waypoint.

These weaknesses may not have a big impact on current Missions such as MER. However, they will become critical as the missions' complexity, duration, and intervals between communication episodes grow.[2]

Decision theory (DT) is a principled framework for reasoning about uncertainty, rewards, and costs (Blythe 1999; Boutillier, Dean, & Hanks 1999). DT avoids the three pitfalls of re-planning and JIC approaches: (i) it makes optimal tradeoffs between the value of goals and plans, and the risk associated with them; (ii) it selects optimal branch points;[3] (iii) it captures the necessity of performing set-up actions each time there is benefit in doing so.

Theb NASA Ames Research Center (ARC) *Limited Contingency Planning* (LCP) project—demonstrated in several rover field tests (Pedersen *et al.* 2003)—attempted to address the previous three shortcomings (Dearden *et al.* 2003). It proposed several heuristics for selecting branch points and goals using decision theoretic tools. The LCP project did not formalize the whole problem of planning for Mars rovers in the DT framework, but it identified three challenges to a direct use of optimal algorithms such as dynamic programming (DP):

1. Structure and concurrency: Models of Mars rovers are expressed in high-level propositional representations that allow concurrent execution of several actions. The complexity of the planning problem growth exponentially with the size of these models.
2. The presence of multiple continuous state variables, such as time, energy, position, temperature, and storage available. These variables make the search space (uncountably) infinite;
3. A large number of possible goals: the complexity of the problem grows exponentially with the number of goals;

These difficulties prohibit the use of existing approaches for finding optimal or near-optimal solutions to problems involving uncertainty faced at NASA.

In this paper, we quickly survey two research directions currently under development at NASA ARC to tackle the first two issues above, and make the decision-theoretic techniques scalable to planetary rovers problems involving a small number of goals. These results will be published in more details in forthcoming papers.

Structure and Concurency

The first obstacle toward applying a full decision theoretic approach to problems such as Mars rovers is their structured and concurrent nature. These problems can be adequately modeled using the complex representation of classical AI planning including propositional, first-order, relational and object-oriented representations (Jónsson *et al.* 2000; Bresina *et al.* 2002). The main issue in using DP with structured representations is scalability: DP manipulates fully grounded Markov states, which are conjunctions of fluents, propositions or predicates. Thus the number of states grows exponentially with the number of fluents, propositions or predicates, and so does the complexity of DP. This fact is well known and several approaches have been developed to exploit representations of structure in decision theoretic models (Boutillier, Dean, & Hanks 1999; Boutilier, Dearden, & Goldszmidt 2000; Hoey *et al.* 1999). Most of them rely on the idea of (approximate) *model minimization* (or *state aggregation*) (Dean & Givan 1997): they avoid redundant computation by manipulating sets of states that produce similar effects, compactly represented in the domain language, instead of individual states.

In addition, rover and spacecraft models often allow concurrent execution of several activities. For instance, the rover can warm up an instrument while driving to a location. A number of classical AI planning approaches support concurrent actions. Most of them assume in simple model of concurrency where several actions can be performed simultaneously, but all actions have the same duration of one unit of time. More complex models allow concurrent execution of actions with different duration. The EUROPA planner developed at NASA ARC and currently used for rover planning is a constraint-based planner allowing complex temporal constraints and concurrency between activities (Jónsson *et al.* 2000). However, it assumes a deterministic model of the environment. Although our long term goal is to be able to introduce non-determinism in the rich models that allow concurrent actions with different durations, the research described here focuses on the simple models where all actions have the same durations.

[2] Just-In-Case (JIC) scheduling (Drummond, Bresina, & Swanson 1994) was used for automatic telescope scheduling. Although it is not an approach based on re-planning, it falls in the same paradigm: first it uses a deterministic, myopic model to generate an initial plan; then it adds branches in the most likely failure points. This is equivalent to waiting for failure before changing plans. The success of JIC scheduling may be explained by some characteristics of the domain of observation scheduling (notably, the fact that the failure to complete a planned observation may easily be compensated by adding a new observation to the schedule). However, for the reasons exposed above, this heuristic does not perform as well in the more complex domain of rover activity planning.

[3] DT is most often used to search for an optimal policy, that is, a universal plan with one "branch" for each possible situation that could be encountered at execution. However, it is also possible to use a decision theoretic approach to find optimal plans of other types, such as conformant plans (Hyafil & Bacchus 2003) and k-contingency plans (Meuleau & Smith 2003).

The most influential algorithm in the classical AI planning paradigm for concurrent planning is probably Blum and Furst's GraphPlan (GP) (Blum & Furst 1997). This algorithm obtained tremendous success and inspired many extension. It works in two stages:

Plan graph construction: The (incremental) plan graph (Smith & Weld 1998) is a bi-partite graph whose nodes are either fluents that can possibly become true at some time; or actions that can possibly be performed at some point. In short, it represents all the fluents and actions that are "reachable" at some point in time, given the initial conditions and the domain model. Causality links between fluents and actions are represented by the edges of the graph: each fluent is linked to every action that consumes it (that is, for which it is a precondition), and each action is linked to every fluent that it produces. The presence of two fluents in the plan graph indicates that they are both reachable, but it does not mean that they are both reachable at the same time (or even in the same run). The plan graph also features binary exclusion constraints ("mutexes") between pairs of actions and fluents. A mutex between two fluents indicates that they cannot both be true at the same time, and a mutex between two actions shows that they cannot be performed concurrently. Because they are only binary relations, mutexes represent only a partial reachability analysis. For instance, three fluents may be unreachable at the same time while there is no (binary) mutex relation between any two of them. In counterpart, the plan graph construction is polynomial in the size of the problem, while a complete reachability analysis would be exponential (because it requires enumerating all possible plans).

Goal regression: The next stage is planning itself: given a goal as a set of fluents (sub-goals), we *regress* this goal through the plan graph. That is, we enumerate all the possible ways to evolve from this set of sub-goals to the initial conditions by applying actions—including concurrent actions—backwards from the sub-goals until a feasible solution has been found. Although this search is potentially exponential, the mutex information allows substantial pruning. For instance, we can cease to regress a set of sub-goals as soon as we recognize a mutex between two of them. That saves the time that goal regression would take to discover there is no way to satisfy this set of goals. The search stops at the first feasible plan found.

This combination of a partial reachability analysis guiding a complete goal regression has proved very efficient and GraphPlan remains one of the most efficient planning algorithms for concurrent planning.

In a classical DT approach, we do not want to find *a* feasible plan, but *the best* feasible plan (in terms of expected utility), so the search must continue until all plans have been considered. Most algorithms use Bellman's optimality principle to avoid enumerating the complete policy-space.[4] As showed in (Boutillier, Brafman, & Geib 1998), partial reachability information as computed by GP may also be used to prune the DP search space in a fully sequential planning framework (without concurrency). Pushing this idea further, we have developed a PlanGraph Dynamic Programming (PGDP) algorithm that uses a stochastic STRIPS (Boutillier, Dean, & Hanks 1999) representation of the domain and combines three principles to accelerate the search for an optimal plan: (i) Bellman's optimality principle, as in any instance of DP; (ii) model minimization to accelerate DP; (iii) a partial reachability analysis, in the form of mutex relations between pairs of fluents, which is characteristic of the GP algorithm. Moreover, it can handle a simple model of concurrency. That is, it can output a plan with concurrent activities if this is an optimal plan.

PGDP follows the same scheme as the original GP algorithm. The first stage, plan-graph construction, is carried out in exactly in the same way. The second stage, goal regression, is replaced by a process of back-propagation of utility tables. This process plays a similar role, enumerating all plans that can lead to the goal, but does not stop at the first feasible solution found and continues until all plans in all reachable states have (implicitly) been considered. The basic entities manipulated are utility tables, defined by:

- A representation of the expected reward that can be obtained under some policy. In simple cases, this is just a real number. In a domain featuring continuous variables, it can be a piecewise constant or linear function of the continuous variables, as described in the next section.
- A condition, that is a list of fluents that must hold to obtain the utility encoded. To accelerate dominance tests (see below), we maintain pointers in both directions between utility tables and the fluents in their conditions: each fluent node in the plan graph contains a list of pointers to the tables having this fluent as condition, and each table points to the fluent nodes in its condition list.

A table indicates that it is possible to get the reward encoded in its value function if all the fluents in its condition are true.

PGDP is initialized by creating a table for each goal of the planning problem. These tables are then backed up in an asynchronous manner, until a steady state is obtained. The basic operation of table back-up uses two principles:

Logical inference: one or several sub-goals in the condition list of the table are regressed through an action or a set of actions. A new table is created where the regressed sub-goals have been replaced by the action preconditions. Mutual exclusion information is used at this level to filter out all tables with mutex fluents in their condition.

DP back-up: the utility encoded in the newly created table is obtained by composing the immediate effect (resource consumption, reward) of the action, and the long term utility encoded in the parent table, as in a classical DP back-up.

The algorithm also features a mechanism of table merging: When a table is backed up, the newly created table is compared with existing tables to ensure that dominated tables are discarded and only a minimum set is kept. This operation is accelerated by using the system of pointers between fluent-nodes of the plangraph and utility tables. This step

[4] One exception is the so-called *policy-search* approach.

Problem	# locations	# paths	# goals	# plan graph nodes	# plan graph levels	plan graph construction	utility tables back-prop.
Sep03	5	4	3	80	10	0.06s	0.05s
Sep03 mod 1	5	6	3	84	10	0.07s	0.13s
Sep03 mod 2	5	7	3	86	10	0.08s	0.31s
random3-1	7	10	3	120	9	0.16s	0.40s
random3-2	8	12	3	132	9	0.19s	0.44s
random3-3	6	8	3	104	10	0.14s	0.30s
random5-1	17	36	5	200	10	0.92s	20.72s
random5-2	26	74	5	312	12	3.56s	41.62s
random5-3	26	64	5	292	11	2.90s	37.52s

Table 1: PGDP simulation results. Sep03 is the problem used during the LCP field tests of Fall 2003. Other problem instances where randomly generated.

is fundamental: it is where Bellman's optimality principle is used to prune the search. Each time that we discard a dominated table, we abandon complete regions of the search space that we know do not contain an optimal policy.

PGDP has been implemented and tested using a real model of the K9 rover (the model used for LCP field tests of Fall 2003).[5] Table 1 presents preliminary complexity results obtained with different problem instances in this domain. It shows that PGDP produces optimal solutions to problems involving three scientific objectives in less than one second, and five objectives in less than one minute. Moreover, the smallest of these problems cannot be solved in fewer than a few hours if we disable either the partial reachability analysis or the ability to use Bellman's optimality principle. This indicates that the algorithm is taking advantage of both principles. PGDP is also used as a heuristic to recommend goals, branch points and branch conditions to the incremental contingency planner developed in the LCP project (Dearden *et al.* 2003).

Continuous Variables

A characteristic of many of NASA application domains is the existence of continuous state variables such as time, battery levels, location, and available memory. Most of these represent resources that constrain the planning problem. Moreover, most of the uncertainty in the domain results from the effect of actions on these variables. In the Mars rover domain, the biggest sources of uncertainty are the duration and energy consumption of actions and the storage space that pictures will require after compression. In contrast, the control framework is not completely continuous because decisions are made at discrete decision steps. Formally, the problem is that of a discrete-step decision model, such as an MDP, with several continuous state variables. The continuous variables make the state space continuously infinite and prevent a direct use of classical solution techniques.

Figure 1 shows the optimal value from the initial state of a typical Mars rover problem as a function of two continuous variables: the time and energy remaining (Bresina *et al.* 2002). The shape of this value function is characteristic of the rover domain, as well as other domains featuring a finite set of goals with positive utility and resource constraints. Such a value function features a set of humps and plateaus, each of them representing a region of the state space where a particular goal (or set of goals) can be reached. The sharpness of a hump or of the edges of a plateau reflects the uncertainty attached to the plan leading to this goal. Moreover, constraints on the minimal level of resource required to start some actions (Bresina *et al.* 2002) introduce abrupt cuts in the regions. This results in a landscape with vast regions where the expected reward is nearly constant. They correspond to regions of the state space where the optimal policy is the same, and the probability distribution on future trajectories induced by this policy is nearly constant.

Our current research aims at developing algorithms able to exploit such structure by grouping together states belonging to the same plateau, while reserving a fine discretization for the regions of the state space where it is the most useful (such as the curved hump where there is more time and energy available). The algorithms we are developing are largely inspired by previous work on time-dependent MDPs (TDMDPs) (Boyan & Littman 2000), that features a single continuous variable representing time, to the multi-dimensional case. They implement the same basic idea as most structured DP algorithm, that is, model minimization. Here, it is based on a form of *lazy discretization* whose principle is the following: instead of naively imposing an arbitrary discretization of state variables and then deducing a discretization of action effects from it, we do the inverse. That is, we start by building a discrete model of action effects on continuous variables, possibly using the same grid size (in each dimension) as in the naive approach. In the planetary rover domain, this consists of discretizing the resource consumption of actions (which can easily handle dependencies between different resources). Then, assuming that immediate rewards are piecewise constant functions of the continuous variables, a minimal discretization of the state space is computed at the same time as DP is performed

[5] In this preliminary implementation, we assume deterministic action consumptions and attach to each utility table a piecewise constant value function, as explained in the next section.

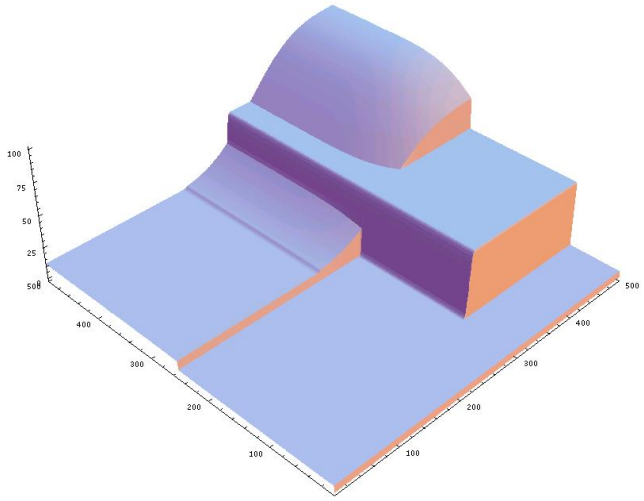

Figure 1: Value function in (Bresina *et al.* 2002).

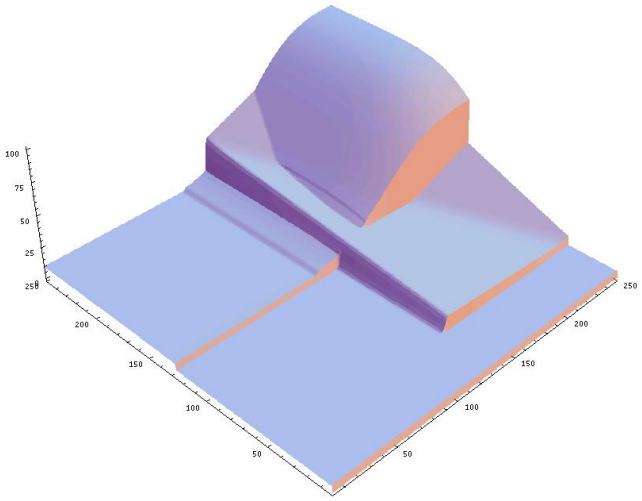

Figure 2: Value function of a variant of (Bresina *et al.* 2002) with piecewise linear rewards.

(that is, backward from the planning horizon to the initial time). The value function at each step is represented by a piecewise constant function of the continuous variables, and the set of pieces over which it is defined is kept minimal to render only the significant differences between states, given the discrete model of action effects. States matching the same piece of value function: (i) have the same optimal plan/policy, (ii) generate the same probability distribution on future history, in terms of actions performed, rewards received, and pieces of value functions traversed under this optimal policy (assuming the discrete model of action effects). Given a fixed discretization step in each dimension, lazy discretization attains exactly the same accuracy as naive discretization, but it avoids all redundant computation.

This approach has been tested on prototype rover problems (Feng *et al.* 2004). Our implementation uses kd-tress (Friedman, Bentley, & Finkel 1977) to store piecewise constant value functions defined over rectangular partitions, a mechanism for merging adjacent pieces with same value, and we are currently adding smart operators to limit the number of pieces created at each DP back-up. Figure 1 was obtained in the order of a few minutes using this technique with the highest level of discretization (of action outcomes). It required in the order of one day of computation to solve the same problem using a Monte Carlo approach and a naive discretization, and the quality of the solution was lesser (see the figure in (Bresina *et al.* 2002)).

Following (Boyan & Littman 2000), we further increased the model by allowing piecewise linear reward functions. For instance, to take into account the illumination of a rock, the value of a picture of it could vary (piecewise) linearly with the time of the day. Value functions are represented as a rectangular partition of the state space with *a set of* linear functions attached to each (rectangular) piece. Techniques from POMDP theory are used to perform Bellman back-ups, resulting in a form of *partitioned incremental pruning* algorithm (Kaelbling, Littman, & Cassandra 1998; Cassandra, Littman, & Zhang 1997). See (Feng *et al.* 2004) for details. Figure 2 provides an example of value function obtained with this representation.

These experimental results were obtained using toy problems, and not the real model used for testing PGDP. As explained above, the real problems have a complex structure involving both its discrete and continuous state variables. To address these problems, our future work will consist of integrating the techniques presented in this section with the PGDP algorithm presented in the previous section. In a preliminary implementation, a piecewise constant representation of the value function defined over the whole continuous variables space may be attached to each utility table, instead of the single scalar value. However, it is likely that any of these value functions share structure. In the extreme case, two discrete states may have identical value functions, in which case we would like to combine them. In other cases only a subset of the continuous state may match, and we may be able to interleave splits on discrete state with splits on continuous state in the kd-tree to capture the structure in mixed models efficiently.

Conclusions

The techniques described in this paper make it possible to compute optimal solutions to real instances of Mars rovers problems involving a small number of goals. The solutions proposed here fail when the number of goals to be attained increases, because the underlying MDP is exponential in the

number of goals. Therefore, there is no other alternative than heuristic and approximate approaches to solve problems involving a few tens of goals. We are currently exploring heuristic approaches using goal-based hierarchical models as a solution to the large number of possible goals that a real mission scenario may contain.

Acknowledgments

This work was supported by the NASA Intelligent Systems Program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of NASA.

References

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:281–300.

Blythe, J. 1999. Decision-theoretic planning. *AI Magazine* 20(2):37–54.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121:49–107.

Boutillier, C.; Brafman, R.; and Geib, C. 1998. Structured reachability analysis for Markov decision processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. 24–32.

Boutillier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: structural assumptions and computational leverage. *Journal of AI Research* 11:1–94.

Boyan, J., and Littman, M. 2000. Exact solutions to time-dependent MDPs. In *Advances in Neural Information Processing Systems 13*. 1–7.

Bresina, J.; Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*.

Cassandra, A.; Littman, M.; and Zhang, N. 1997. Incremental Pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, 54–61.

Dean, T., and Givan, R. 1997. Model minimization in markov decision processes. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 106–111.

Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D.; and Washington, R. 2003. Incremental contingency planning. In *ICAPS'03: Proceedings of the Workshop on Planning under Uncertainty and Incomplete Information*, 415–428.

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-In-Case scheduling. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1098–1104.

Estlin, T.; Fisher, F.; Gaines, D.; Chouinard, C.; Schaffer, S.; and Nesnasg, I. 2002. Continuous planning and execution for an autonomous rover. In *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*.

Feng, Z.; Dearden, R.; Meuleau, N.; and Washington, R. 2004. Dynamic programming for structured continuous Markov decision problems. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*. To appear.

Friedman, J.; Bentley, J.; and Finkel, R. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software* 3(3):209–226.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*.

Hyafil, N., and Bacchus, F. 2003. Conformant probabilistic planning via CSPs. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*, 205–214.

Jónsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; B.; and Smith. 2000. Planning in interplanetary space: theory and practice. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 117–186.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:99–134.

Meuleau, N., and Smith, D. 2003. Optimal limited contingency planning. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, 417–426.

Muscettola, N.; Nayak, P.; Pell, B.; and Williams, B. 1998. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence* 103(1–2):5–47.

Pedersen, L.; Bualat, M.; Lees, D.; Smith, D.; and Washington, R. 2003. Integrated demonstration of instrument placement, robust execution and contingent planning. In *Proc. of the 7th Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*.

Smith, D., and Weld, D. 1998. Conformant graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 889–896.