Using Machine Learning to Investigate Control of Advanced Life Support Systems

R. Peter Bonasso and **David Kortenkamp**
NASA Johnson Space Center ER2
Houston TX 77058
r.p.bonasso@jsc.nasa.gov

Devika Subramanian
Rice University
Houston TX
devika@cs.rice.edu

Abstract

Long duration space missions require advanced life support (ALS) systems which can regenerate air, water and food. ALS systems are characterized by complex interactions between various tightly coupled subsystems. Since analytical models of complete system behavior are not available, control systems for the ALS cannot be constructed by traditional optimization techniques. We investigate the use of machine learning methods to construct good controllers for the ALS by exploring trajectories through the state space of the system. We describe a simplified version of the ALS task and the challenges it presents to modern machine learning techniques for acquiring robust control strategies. We then demonstrate how a direct search using genetic algorithms for a "short" open loop plan yields insight into the topology of the system state space. This insight is used to construct a hand-crafted, closed-loop controller which is near-optimal. This information can also be used to design the state and action space for a value function search technique (Q learning) for acquiring an optimal control policy. We present the beginnings of a methodology for using machine learning techniques as "probes" in understanding a complex system. This methodology has implications both for control system designers and for machine learning researchers.

1 Introduction

An advanced life support (ALS) system regenerates much of its water, air and food. Thus, an ALS system is critical for long-duration space missions because it minimizes both the stores of consumables and the need for resupply. Since ALS systems have small buffers and low margins, optimal control is essential to their success. However, tight interconnections between ALS components and the lack of complete models of system-level behavior make hand-crafted optimal control difficult, even for simple systems. So we have begun to explore how machine learning techniques can be used to produce optimal control policies for such systems.

Our research group has designed autonomous control systems for several ALS tests [Schreckenghost *et al.*, 1998; Malin *et al.*, 2000] but none of these were completely integrated systems. Because there is not yet an integrated ALS test system (one will come on-line at NASA in the next few years [Tri, 1999]) we started our research with a simple ALS simulation, whose details are given in Section 2. The simulation has modules for the crew, water and air recycling, and crops. Although we developed the simulation ourselves and have knowledge of how each module behaves in isolation, we were unable to predict how all of the modules interact. Thus, we had no *a priori* knowledge about the optimal control policy for the ALS simulation.

Since we have no *a priori* knowledge of the important control variables and key interactions in the ALS system, we treat the simulation as a "black box" as in [Kearns, Mansour, & Ng, 2000], and formulate our problem as that of determining the optimal control policy by appropriate sampling of state trajectories of the system. This paper describes our results from using learning algorithms for two different control approaches: open and closed loop control. We demonstrate how the direct search for a short open-loop plan using genetic algorithms yields insight into the topology of the system state space. This insight is used to construct a hand-crafted, closed-loop controller which is near-optimal.

As such, we present the beginnings of a methodology for using machine learning techniques as "probes" in understanding a complex system. This methodology has implications both for control system designers and for machine learning researchers.

2 A simple ALS simulation

A life support system for a crew on a planetary surface includes the processes of air revitalization, water recovery, food production via grain crops and the biological processes of crew in their habitat as they carry out their missions. The crew consumes clean air, water and food and produces waste water, exhalant air and science, a general term for the crew's productive output. All of the processes except the crew require energy.

Based on a number of past efforts with controlling life support systems [Crawford, Pawlowski, & Finn, 2000] as well as discussions with NASA's BIO-Plex engineers we developed a deterministic simulation (see Figure 1) of an abstraction of

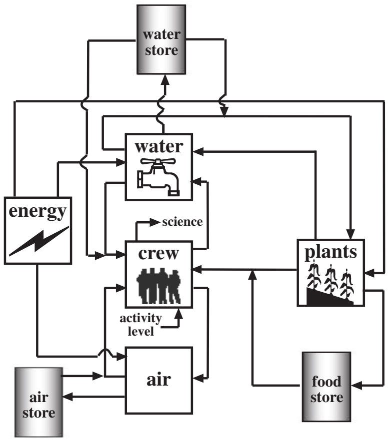

Figure 1: A simple ALS simulation and initial values.

these four processes.[1] The simulation includes stores for energy, clean air, clean water and food. The control choices for the simulation are the allocation of energy per simulation unit time to the various processes or modules, and whether to draw from the various stores at any given time step. A last control choice is the activity level of the crew: higher levels consume more air, water and food but produce more science.

Simulation time is measured in 'ticks', which are nominally one hour. The ALS system state space is the product of the state spaces of the four modules (internal states, inputs and outputs and states of the associated stores), and system level variables: the total energy left and the total simulation time elapsed. The ALS action space is the product of the action choices for each of the modules: energy allocations to each module, whether or not to use a corresponding store, and the crew activity level.

In a single tick each module makes a state transition based on the relevant component of the system action vector. The state transitions of the four modules (along with updates of global state variables) yields a one step ALS system transition. The crew module takes in clean air, clean water and food and produces waste air, waste water and science. The air recovery process takes in energy and waste air from the crew and produces clean air. The water recovery process takes in energy and waste water from the crew and plant crops and produces clean water. The crop module takes in energy and a portion of the clean water and produces food for the crew and waste water. If a store is to be used on a given tick, no energy is consumed for that module during that tick. Since all processes are degenerative (i.e., not mass-preserving), the simulation will eventually terminate due to air, water or food losses. Observable output from the simulation includes the amount of waste air and waste water from the crew module at each tick, the amount of science produced by the crew at each tick, the cumulative science and the total elapsed time since the simulation was initialized.

The simplified ALS system is a hybrid system with its state and action spaces composed of a mixture of continuous and discrete variables. Each module of the ALS is a finite state machine since each represents a physical process that has inertia: it takes time to get to steady state operation and it takes time to degrade in the absence of energy. The water and air recycling processes can enter periods of indefinite hibernation in the absence of energy for prolonged periods of time, while the crew and crop module shut down terminally under conditions hostile for life. The ALS system is thus composed of four interacting finite state machines. While we have detailed models for each of the four processes, there is no explicit model of the entire system. The interactions between them are sufficiently complex that we could not derive them analytically. We need to computationally explore the space of behaviors of the entire system to determine good control policies. The shape of the optimal control policy for such a system is a function of the desired objective; in this paper we consider maximizing a linearly weighted function of mission duration and science output from the crew.

[1]Contact the authors if you would like access to the simulation for your own research projects.

3 Learning closed loop policies using reinforcement learning

The simplified ALS system is a classical application for machine learning techniques that learn control policies for complex embedded systems. If the ALS system is fully observable, then we can treat it as a Markov decision process and learn an optimal value function V^* for the system using reinforcement learning [Sutton & Barto, 1998]. The value function V^* maps states to the optimal expected sum of rewards from that state, and it indirectly specifies an optimal control policy π^* mapping states to action. Unfortunately, the ALS learning problem is not a straightforward application of reinforcement learning. There are several difficulties.

- The continuous portion of the ALS state space is infinite, and the discrete part is also large. It is unclear how to discretize the continuous portion without loss of the Markovian property.
- The internal states of the modules may or may not be observable, making the ALS system a partially observable Markov decision process instead of a fully observable one.
- The continuous portion of the ALS action space is infinite, the discrete part (crew activity level and store use) is small. One modeling option is to discretize the continuous part of the action space; however this is difficult to do in the absence of knowledge of properties of the state space.
- The local rewards in the system (e.g., the science output of crew per tick) are not very good surrogates for the global objective function (i.e., there is a long delay between the choice of a particular action and its final effect).

These are fairly difficult conditions under which to learn value functions. The traditional approach to applying reinforcement learning in such circumstances is to (1) manually construct smaller state and action spaces using knowledge of the dynamics of the system [Crites & Barto, 1996;

Tesauro, 1995], (2) to manually construct better progress measures [Tesauro, 1995], and (3) to learn approximate value functions [Tesauro, 1995; Crites & Barto, 1996]. These techniques require knowledge of system dynamics that is good enough to identify key state space variables and functional forms for the approximate value functions.

As a prelude to understanding aspects of the ALS system state that are important to action choice, we looked at using reinforcement learning to acquire the best policy possible using extremely limited state information. We compared it against a baseline policy that uses the same limited state information. The baseline policy can be stated simply as *use all the stores up before depending on the air, water and food production processes*. Since we have a deterministic simulation, it appeared to us that this policy of using up reserves first would allow us to maximize science output during the initial stages when the regenerative processes were "warming up", and that it would stretch our energy reserves out furthest since using the stores requires no energy. With an initial crew of four, 100 initial stored units each of air and food, 200 units of stored water and 1000 units of energy and an allocation of four units of energy to each process, the baseline policy produces a mission duration (activity level = 1) of 56 ticks (and 95 units of science) or a cumulative science (activity level = 3) of 246 units (in 44 ticks of mission duration). The air and water stores are depleted in 24 ticks in both cases and there are over 700 units of energy left. To conserve space, we will concentrate on the objective of maximizing the cumulative science for the remainder of the paper.

Now we ask the question: how well does an optimal controller, acquired by a value function based reinforcement leaner with state information that includes stores, compare to the simple controller sketched above? We use Q learning [Watkins & Dyan, 1992], a temporal difference (TD) reinforcement learning algorithm. The Q learning algorithm learns the $Q^* : S \times A \rightarrow \Re$ function which maps each state-action pair in the product of the state and action space to a real number which is the expected sum of rewards for taking that action in that state. Since using the full state-action space of the ALS system is impractical, we design a state-action space that preserves information about the stores and which keeps the size of the space down to about 10^4 to obviate the need for function approximation.

If we use the store levels as part of the state information: we are faced with a state space size of $100*100*200*1000 = 2 \times 10^9$ assuming a discretization of 1 on the stored quantities[2]. Further, store levels are not directly visible from the simulation output. So we use the previous action vector with information about whether or not stores were used in the last cycle as an indication of whether stores are available or not. This is not an entirely accurate signal, but in conjunction with energy allocated to the different modules, it correlates highly with actual use of stores. So, the baseline policy is representable in our state space choice of the previous action vector for the system. The size of our state space is the product of 8 (whether or not each of 3 stores were used), 3 (the three crew activity levels), and 27 (energy allocated to the air, water and crop modules considering only the values 3, 4 and 5 for energy allocations per tick)[3]. If we used an action space with the same cardinality, we would have about 420,000 state-action pairs, far above our goal of about 10,000 state-action pairs. Therefore, we reduce the action space into two components: a quantity to be changed (energy allocation to air, water and crops, activity level, air, water, and food stores), and a change to the quantity (no change, increase or decrease by one). This conceptualization brings the action space size to 21. Now our Q table size is about 13600. Finally, we used the following local reward function:

$$r(s, a, s') \quad = \quad \begin{cases} -10000 & \text{if } s' \text{ is terminal} \\ science(s) & \text{otherwise} \end{cases}$$

The Q learning algorithm used is a standard zero-initialized algorithm with 0.1 epsilon-greedy sampling, a learning rate of 0.9, and no discounting. The primary result we obtained was an extraordinary sensitivity of the value of the final policy to the starting state of the simulation. For example, the final learned policy produced 185 units of science in 38 ticks before mission end in one experiment, and 98 units of science in 23 ticks in another experiment. To reduce the variance on the quality of the final result, we decided to start from a benign nominal starting state in subsequent experiments. While this change reduced policy value variance, we noticed policy instability: convergence to wildly different policies with similar values during different runs. We were unable to obtain the use stores first baseline policy, perhaps because it was not optimal for the given objective function of maximizing cumulative science output. However, it should be noted that we obtained 322 units of science in 20 ticks, while the baseline policy produced less science (246 units) over a longer time period (44 ticks).

It was clear to us that we needed a richer representation of the state space to be able to produce better science output. Since the addition of any additional state variable increased state space sizes significantly, we decided to attack the problem by a direct search in policy space, rather than with a value-function based approach. With the Markov assumption, a policy is a mapping from states to actions. A search for such a mapping over the full state and action space of the ALS also turns out to be prohibitively expensive without appropriate abstractions for the state and action spaces.

At this point we noticed that the optimal policies found by the Q-learner with the simplified state representation yielded short mission durations ($\leq$ 20 ticks). Together with the fact that results are extremely sensitive to starting state, we guessed that the system state space had a "safe core" and that the nominal start state was part of that core. Further, any action sequence longer than 20 steps took us away from that safe core into one of numerous terminal states where one or more of air, water, food and energy was completely depleted. We reconceptualized the goal of the control policy to be that of keeping the entire system within the safe or non-terminal

[2] We cannot choose a bigger discretization on these quantities without knowledge of the system

[3] This choice was not arbitrary. The processes nominally need 4 units of energy per tick and we looked at values one above and one below.

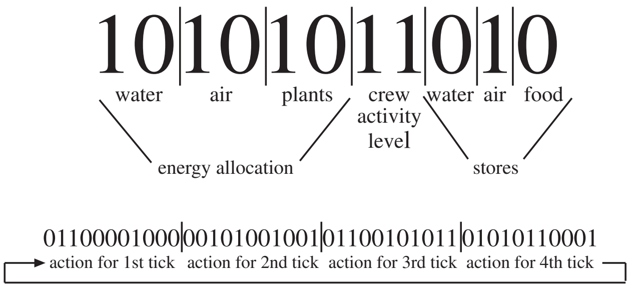

Figure 2: At top is the original genetic algorithm bit string representing actions to be taken. At bottom is a multi-step bit string of length 4.

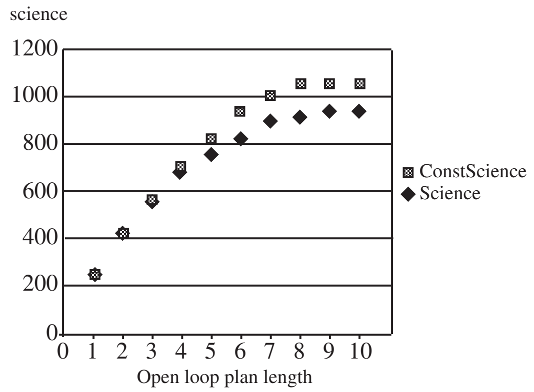

Figure 3: Quality of open loop plan as a function of plan length.

core at all times, while maximizing science output. We reformulated our control problem as follows: what is the shortest action sequence (i.e., an open loop plan) that keeps the system away from the non-terminal states and which maximizes science output?

Our hypothesis is that when an embedded system has the specific structure outlined above, an effective control technique for it is to execute a short sequence of control actions over and over till the end of the mission. The sequence of control actions is an open loop plan — it does not test for state conditions during its execution, it is therefore not state sensitive. This means that we do not need to make the Markov assumption, and full observability of the system is no longer a requirement. Further, it obviates the need for making abstractions of states. It works directly with the raw action set to find an open loop plan that functions best with respect to the externally specified objective function.

4 Searching for open loop plans using genetic algorithms

We settled on using a genetic algorithm to search the space of open loop plans of length less than 20. We used the implementation GAL written by William Spears of the Navy Center for Applied Intelligence [Spears, 2000]. The system action vector was encoded as a bit string as follows: 2 bits each for the energy allocations (0, 1 and 2 indicate 3, 4 and 5 energy units respectively, and 3 = 5 as well), 2 bits for the activity level (1,2, and 3, and 0 = 1 as well), and 1 bit for each of the stores (1 = store to be used, 0 = store not to be used) for a total of 11 bits (see Figure 2). To construct plans of length n, we repeated the string n times (see Figure 2). The GA parameters used were: two point crossover maintained at 60%, a very low mutation rate, selection based on proportional fitness, and population size of 100. The evaluation function was the cube of the cumulative science output[4]. To evaluate a string, the simulation was run using that string for n ticks (where n is the length of the plan), and repeating it every n ticks until the mission ended. We typically ran twenty experiments and chose the best string among the strings with the highest value from each experiment. The best single step open loop plan obtained by this technique produced 184 units of science in 26 ticks; far below the values of the baseline policy and the policies produced by the Q learner. We continued our experiments with open loop plans of length n starting at two, and used the value of the best individual from those experiments as the value to beat for $n + 1$, continuing to increase n, the number of actions in the sequence, until the next increase resulted in no improvement.

As shown in Figure 3 the results were fairly spectacular. The diamonds in Figure 3 show a plot of the best cumulative science achieved versus increasing plan size. The last two data points are meant to show that succeeding experiments could do no better, i.e., they either matched the previous value or had to be halted because succeeding generations in the experiment could not match the previous best value. This plot shows that the extended GA achieved 938 units of science (in a 120 tick mission duration) with a nine step (99 bit string) open loop plan. An examination of the plan showed that it tended to keep the activity level at three, and used the lowest energy level – three units – for each process. With this insight, we repeated the experiments but forced the activity level to be 3 and the energy allocation to 3 throughout. The plot of squares in Figure 3 shows that the GA then found the best open plan to be of length 8 rather than length 9, and also found a higher science solution (1049 units as opposed to 938).

5 Analysis

At this point we were anxious to understand what the open loop plans learned by the GA were doing, and what it revealed about the topology of the state space. To gain this understanding, we ran the eight step policy and captured the crew output and the energy and store decisions. As discussed above, from this data the GA had determined that none of the processes really needed more than three units of energy per tick and that a high activity level was key to high cumulative science. These two points are fairly obvious and our use-stores-first policy had already incorporated these parameters. So clearly, the open loop plan was doing more because it produced significantly higher levels of science. Figure 4 shows the results of running the eight step open-loop plan

[4]This choice was made to be able to better differentiate the quality of open loop plans generated.

for the first 100 ticks of the simulation. Plotted against ticks are the crew output of waste air (air-dirty or AD), waste water (water-dirty or WD) and science as well as the store allocation decisions (WS and AS for water and air stores respectively). What is clearly evident is that the plan lets the output of the air and water processes decay for a certain period of time before resuscitating them by using the stores. This pulsing action spreads the stores out over the course of the mission, thus increasing the number of ticks in which to accumulate science, while maximizing the average science per tick. Additionally, the plan used the plant stores on every tick.

The decay curves of the air and water are a direct result of the decay laws of the air and water processes in the simulation, assuming no use of stores. The air available to the crew at step n is given by: $air_n = ncrew \times (aclf)^n$, where $ncrew = 4$ (number of crew), and $aclf$ is the air loss across the crew process and is set by the activity level. For example $aclf = 0.95$ for activity level 3. For this activity level, the air at step 8 before a store use is 2.65 as shown in Figure 4.

The water available to the crew at tick n is also a power function. It is half the total water coming from the water recovery process, since we allocate that output evenly between the crew and the crops. The dirty water input to water recovery on step n is the sum of the crew dirty water output and the plant dirty water output from the previous tick. The crew dirty water output is in turn a function of the clean water to the crew from the previous tick times the water crew loss factor ($wclf$). Such a recursive computation yields a sum of powers in n of the crew loss factor. $WTC_n = (C_{0,n}WD_{plants,0} + C_{1,n}wclf + C_{2,n}wclf^2 + ... + C_{n,n}wclf^n)(R/2)^n$, where $WTC_0 = WD_{plants,0} = C_{0,1} = ncrew$, $C_{1,1} = WTC_0$, $R = 0.97$ (water recovery factor), and $C_{i,n} = C_{i-1,n-1} + C_{i,n-1}$. At $n = 8$, this recursive formula predicts the water from the crew will be 1.07 as shown in Figure 4.

The use of the plants stores on every tick conserves energy. Based on the food requirements of the crew, the food stores would last over five hundred ticks, far outlasting the air and water stores. The net result of the 8-step open loop plan's periodic use of stores was to keep the science output above four units at all times. We believe that the open loop plan found by the GA suggests the following key properties of the state space of the ALS.

- A very small fraction of the entire state space is "safe." That is, it is only from a small fraction of entire state space that there are action sequences that guarantee not ending up in a terminal state irrecoverably. This is one of the explanations for the extreme sensitivity to starting conditions for our learning algorithms.
- The "diameter" of the safe core of the state space is small. By "diameter," we mean the average number of random actions that will take the system from a safe start state to an irrecoverable, i.e., non-safe state. Evidence for this comes from the Q learner that initially finds policies that last for fewer than 20 ticks.
- To keep the system in the safe states while minimizing resource consumption and maximizing science, a good (perhaps optimal?) parametric strategy is one that is *periodic*, i.e., one which cycles the system through the safe core of states. It runs consumable resources in the individual subsystems down to critical levels and then restores them in a single pulse. This keeps the number of "refills" down (since there is a cost to refilling) and keeps each subsystem under safe operating conditions. An apt analogy is the strategy of keeping a balloon afloat by periodic upward thrusts to counter the effects of gravity.

Our analysis suggests that the important aspects of the control strategy to be learned are: the period of the dynamical system, the timing of the use of the various stores, and the critical levels of each resource to be maintained at all times. This is the key contribution of the simpler search in open loop plan space. It suggests the new variables that should form the state space for the search of a closed loop plan either by value function or direct policy search methods. It should be pointed out that none of the current learning methods have automated ways of constructing such new variables for expressing control policies.

6 From open loop back to closed loop control

Based on these findings we hand-crafted a closed loop controller as follows. At each tick, we allocate three units of energy to each process and use the plant stores. If the science output for the tick is below 4.0, we use the air and water stores. If the science output is at or above 4.0, and the air output from the crew is at or above 2.65, but the water output from the crew is below 1.0, we use the water stores. If the science output is at or above 4.0, and the water output from the crew is at or above 1.0, but the air output from the crew is below 2.65, we use the air stores. Otherwise, we do not use the air or water stores. The parametric structure of this policy as well as the constants were revealed by the analysis of the eight step open loop plan presented in the previous section.

The resulting policy produced 1051 units of science (2 units more than the open loop plan), over a mission duration of 198 ticks (the same mission duration for the best open loop plan for maximizing mission duration.) Our closed loop controller performs better than the open loop plan toward the very end of the mission duration where the air and water crew outputs vary irregularly rather than with the eight tick periodicity, and thus the closed loop policy is able to change its store use pattern to match the situation. Thus, guided by the GA we were able to achieve a little more science and match the mission duration of the best open loop GA generated plan, with our hand-crafted closed loop controller.

We are currently engaged in redesigning the state space of a Q learner to take the parametric structure of the policy into account, so we can, with certainty, find the optimal policy in the space with this structure.

7 Towards a methodology for understanding complex systems

The solution space for a closed loop control policy in an ALS system is enormous, as are the solution spaces for many complex, interacting systems. We believe that machine learning techniques are a useful way to "probe" the solution space and

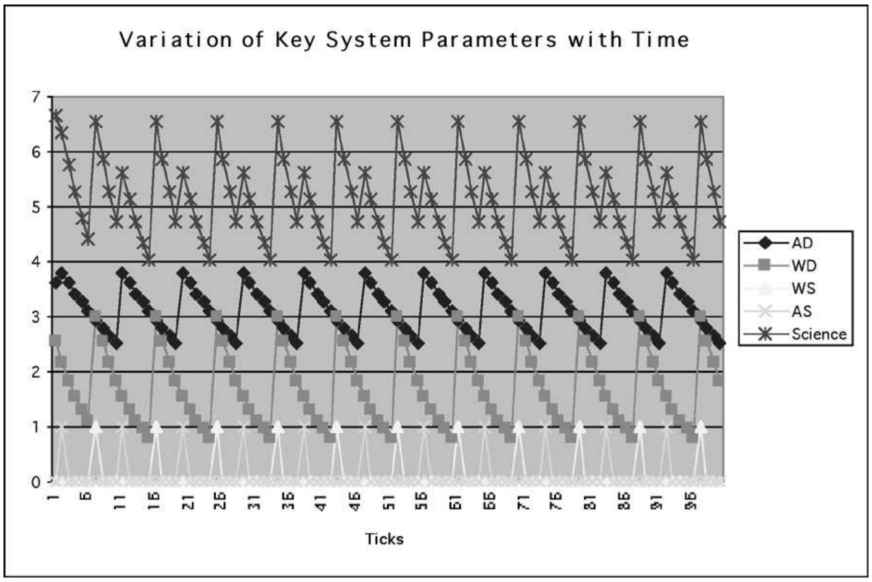

Figure 4: Analysis of open loop 8-step plan.

give control system designers an idea as to its topology. In this way, machine learning algorithms become not just a tool for adjusting the on-line control system, but also a tool for helping programmers design an *a priori* control policy. There are some important research questions that need to be answered with respect to using ML techniques as "probes" of the solution space, including: 1) For what class of dynamical systems is open loop plan analysis helpful for guiding state space design for closed loop policy search? 2) How do we design reinforcement learning algorithms that can design the right state space features by automating the meta-analysis we demonstrated in this paper? 3) Which machine learning techniques (e.g., reinforcement learning, memory-based learning, genetic algorithms, etc.) are the most useful for probing the solution space? 4) How can the discoveries of machine learning algorithms be presented to control system designers so that they can easily understand the topology of the search space and the important control variables and interactions? The performance analysis of the open loop eight step plan was key to uncovering the structure of the state space. How can we automate such performance analyses? and 5) How detailed must the simulation be for results from a machine learning algorithm that probes the solution space to be applicable to the real physical system? Our analysis was performed on a simplified version of the ALS task, which itself proved daunting for our machine learning algorithms. We need to devise robust sensitivity analyses to understand the aspects of the simulation that drive our results.

References

[Crawford, Pawlowski, & Finn, 2000] Crawford, S.; Pawlowski, C.; and Finn, C. 2000. Power management in regenerative life support systems using market-based control. In *Proceedings of the 30th International Conference on Environmental Systems*.

[Crites & Barto, 1996] Crites, R. H., and Barto, A. G. 1996. Improving elevator performance using reinforcement learning. In *NIPS 1995*, 1017–1023.

[Kearns, Mansour, & Ng, 2000] Kearns, M.; Mansour, Y.; and Ng, A. Y. 2000. Sparse sampling methods for planning and learning in large and partially observable markov decision processes. In *International Conference on Machine Learning*.

[Malin *et al.*, 2000] Malin, J.; Nieten, J.; Schreckenghost, D.; MacMahon, M.; Graham, J.; Thronesbery, C.; Bonasso, R. P.; Kowing, J.; and Fleming, L. 2000. Multiagent diagnosis and control of an air revitalization system for life support in space. In *Proceedings of the IEEE Aerospace Conference*.

[Schreckenghost *et al.*, 1998] Schreckenghost, D.; Ryan, D.; Thronesbery, C.; Bonasso, R. P.; and Poirot, D. 1998. Intelligent control of life support systems for space habitats. In *Proceedings of the Conference on Innovative Applications of Artificial Intelligence*.

[Spears, 2000] Spears, W. M. 2000. Gal: Genetic algorithm in lisp (http://www.aic.nrl.navy.mil/ spears/freeware.html).

[Sutton & Barto, 1998] Sutton, R., and Barto, A. 1998. *Reinforcement learning: an introduction*. MIT Press.

[Tesauro, 1995] Tesauro, G. 1995. Temporal difference learning and td-gammon. *Communications of the ACM* 38:58–68.

[Tri, 1999] Tri, T. O. 1999. Bioregenerative planetary life support systems test complex (BIO-Plex): Test mission objectives and facility development. In *Proceedings of the 29th International Conference on Environmental Systems*.

[Watkins & Dyan, 1992] Watkins, C. J., and Dyan, P. 1992. Q-learning. *Machine Learning* 8(3):279–292.