Distributed, Autonomous Control of Space Habitats

David Kortenkamp and R. Peter Bonasso
NASA Johnson Space Center ER2
Houston TX 77058
kortenkamp@jsc.nasa.gov, r.p.bonasso1@jsc.nasa.gov

Devika Subramanian
Rice University
Department of Computer Science
Houston TX 77005
devika@cs.rice.edu

Abstract—Long-duration space missions require advanced life support (ALS) systems that can regenerate air, water and food. These ALS systems need complex control strategies that can maintain stable system performance and balance resources with small margins and minimal buffers. In this paper we will describe the ALS control task in detail and give some examples of previous control solutions. Then we will look at how machine learning techniques can help create a more adaptive ALS control system. We will examine reinforcement learning and genetic algorithms and their relationship to optimizing resource utilization in an ALS system. Finally, we will present an innovative multi-step genetic algorithm that generates control strategies that perform much better than traditional reinforcement learning or traditional genetic algorithms.

TABLE OF CONTENTS

1 INTRODUCTION

Advanced life support systems require complex control strategies that can maintain stable system performance and balance resources with small margins and minimal buffers. In closed-loop life support systems there are complex interactions between sub-systems such as air, water, food production, solids processing, and the crew. Recent research at NASA Johnson Space Center has led to significant insights into autonomous control of advanced life support (ALS) systems (see Section 2.1). In fact, routine control of an ALS system is well within the reach of current techniques. For example, the autonomous control system described in [Schreckenghost et al. 1998] operated around the clock for 73 straight days during a 90 day crewed test with minimal human intervention. However, control systems such as these rely on hand-coded planning operators, domain knowledge and reactive procedures. As long as the operations are routine, well-understood and properly coded, the control systems perform well. Trouble occurs when the control system encounters situations that have not been hand-coded or when control system operators want different behavior that is not embodied explicitly in the control program. In these cases, the system fails and (expensive) human programmers and domain experts must add the relevant information into the control system. This led to our recent focus on applying machine learning techniques to control of advanced life support systems to improve their robustness in the face of changes in dynamics or objective functions.

Any solution method for controlling a life-support system must address three significant complexities that the task is dynamic, non-stationary, and safety-sensitive.

- *Dynamic*: This means that it is not sufficient only to find a particular setting that optimizes an objective function. The issues of state dynamics and delayed reward have to be considered.

- *Non-stationary*: An ALS controller must account for the fact that the dynamics of the system change over time. This is due to the presence of adaptive biological organisms (that is, humans and plants), as well as to the gradual degradation of certain system components (e.g., filters that clog). Some of these change slowly, others change abruptly. The controller must adapt to these changes while the system is operating. This adaptation requires *learning from data* to track changes in system dynamics, and *active experimentation* with improvements to the controller during system operation in order to respond to those changes.

- *Safety-sensitive*: Because of the presence of humans in the system, all adaptations of the controllers must be conducted within guaranteed margins of safety. This restricts the applicable learning algorithms to those which can both incorporate engineer-specified control limits and model the domain uncertainty to avoid risky experiments. This also requires machine learning techniques that are *inspectable* and *instructable* by humans.

In addition to these complexities of the task, there are other significant scientific questions that need to be asked with respect to using machine learning to control advanced life support systems.

0-7803-6599-2/01/$10.00

1. What model of itself and its physical plant should an autonomous control system have? How detailed does that model have to be? Will a qualitative model suffice?

2. How can the autonomous control system become aware of the aspects of the physical plant state that are hidden with respect to its sensors or its knowledge?

3. What are the limits of safe experimentation in the real physical system? To what extent can models or simulation replace experimentation?

4. What is the role of human intelligence in learning for autonomous control? How can human expertise be brought to bear to guide the learning systems in their explorations? How can learning systems be inspectible and instructable?

This paper examines the role of machine learning in the control of a distributed, advanced life support system. In the next section we introduce advanced life support systems and the challenges inherent in controlling them. In Section 3 we will look at the general role of machine learning in control of advanced life support systems by introducing specific challenges and methodologies. In Section 4 we introduce a simplified simulation of the complete ALS control task and describe a number of experiments performed using that simulation. Section 5 describes an innovative genetic algorithm approach that has been developed and its results in controlling the simplified simulation. Section 6 looks at how we used the results from the genetic algorithm to reimplement a close-loop reinforcement learning system.

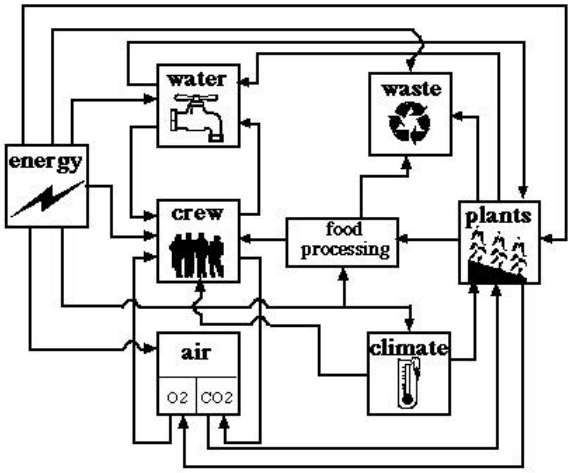


Figure 1 The interacting subsystems of an advanced life support system

2 ADVANCED LIFE SUPPORT SYSTEMS

When humans embark on long duration missions such as the establishment of permanent bases on the Lunar Surface or travel to Mars for exploration, they will continue to need food, water and air. For these long duration missions, it may not be economical or practical to resupply basic life support elements from Earth. We will need to develop systems that produce food, purify their water supply, regenerate oxygen and remove undesirable components of air. Such a system would be a tightly controlled and closed loop system in which the growth of crop plants would contribute to the life support functions. The natural function of plants would provide food and contribute to water purification, air revitalization and even the processing of waste materials. All systems would have to operate under the restrictions of minimizing volume, mass, energy and labor. Figure 1 shows a typical advanced life support configuration and the interaction between subsystems.

2.1 Previous and current control systems

Over the last five years there have been a number of advanced life support system tests. Some of these have had crews, others were integrated tests of individual subsystems and some were just simulations. We have been involved in autonomous control of these tests since the beginning.

Our first experiences were with interchamber monitoring and control (IMC) software for a crewed test. This software was used during the Phase III test of NASA's Lunar/Mars Life Support Test Program (LMLSTP). For this test, four crew members lived in a closed habitat for 91 days. Wheat for air recycling and food was grown in a separate closed chamber. Our control software managed the movement of gases between the two chambers and also managed an incinerator. For details on this control system see [Schreckenghost *et al*. 1998].

Our latest control software is for a water recovery system (WRS) being tested at NASA JSC (see Figure 2). Our control software manages all components of the system autonomously. The integrated system runs 24 hours a day and broadcasts information to user interfaces for remote operation from the office or home.

Figure 2 The WRS hardware

2.2 Conclusions

From these applications we have an indication that autonomous control of advanced life support systems is possible. However, experience with the IMC and WRS shows that even small changes in the underlying physical system or in the overall goals of the system require significant re-coding of control software. For example, routine recalibration of sensors requires stopping the autonomy software, changing by hand the control parameters and then restarting the autonomy software. The re-coding comes at significant expense. None of these systems have the ability to self-adapt to changes. The research described in this paper is aimed at providing just such self-adaptability to ALS control systems.

2.3 The future: BIO-Plex

The Bioregenerative Planetary Life Support System Test Complex (BIO-Plex) is the latest NASA JSC testbed for advanced life support systems. It will consist of a complex of five chambers at NASA JSC combining biological and physiochemical life support technologies to provide all the air and water, and most of the food (up to 90%) for a crew of four on a continuous basis and crews of up to eight during crew change-over [Tri 1999; Barta *et al* 1999]. Initial testing (120-day human test) is scheduled for 2003. A 240-day human test is scheduled for 2005 and a 425-day human test is scheduled to start in 2007. BIO-Plex is meant to simulate on the ground many aspects of a long duration planetary mission with minimal resupply. It will consist of several distinct subsystems including air, water, food, solid waste and climate control.

BIO-Plex will need an advanced control system to maintain life support. The goal of the control system is completely autonomous operation of BIO-Plex. Completely autonomous control means that there will be no ground support personnel monitoring the facility during operation. This does not mean that people will never interact with the autonomous control system. On the contrary, one of the biggest open research issues is how to allow a crew that depends on an autonomous control system for its life to safely and effectively interact with it to achieve complex goals.

Autonomous control of BIO-Plex will be a large and challenging AI research area. This paper focuses only on the application of machine learning techniques to the autonomous control system. BIO-Plex is an attractive testbed for machine learning because of the vast interconnectedness present in the system. Subsystems such as power, thermal, water, air, and food affect one another in both obvious and subtle ways. Small changes in one system can cause profound changes in other parts of the BIO-Plex. Optimizing for all of these variables is beyond the current state-of-the-art in machine learning. This problem is compounded by the fact that the system is constantly shifting -- filters clog, pumps break, biological water processors become more or less efficient, etc. -- so optimization cannot occur just once but must occur many, many times over the lifecycle of the system.

3 The role of machine learning in ALS control

We have begun investigating the role of machine learning on the control of ALS systems. We have identified some potential applications of machine learning techniques. In this section we examine these applications and raise some research challenges.

3.1 Detecting signatures

Certain sensor "signatures" require specific responses from the autonomous control system. These signatures are often hand-coded by the programmer. For example, the programmer might state that if the temperature is above 100 and the pressure is above 1000 then vent the tank. This is a trivial example and real-world examples will be more complex and involved. This means that hand-coded signatures may not accurately capture the event, especially if the environment or the physical plant are changing. A variety of machine learning techniques could be used to look at the history of the system and adjust the signatures automatically. A key research challenge in this area is the following:

- Can we learn to parse the real-world data stream into recognizable system modes or events?

3.2 Refining models

Many autonomous controllers contain a model of the system they are trying to control, either explicitly or implicitly. Typically, these models are hand-coded and do not change. However, in long-duration missions the underlying physical system may change dramatically due to damage or degradation. For example, a filter in a life support system may clog more frequently than expected. In each of these cases, machine learning techniques could be used to modify the internal model of the system based on real-world data. This is especially necessary if other machine learning techniques are using these models for optimization (as discussed in Section 3.6). A key research challenge in this area is the following:

- Can feedback from actual operation of the system be used automatically to refine our simulations and models?

3.3 Learning/optimizing control plans

The behavior of most autonomous systems can be characterized as a control plan that leads to some desired result. These plans can be generated by planners, be hand-coded by programmers, or "emerge" from the interactions of independent behaviors. In any case, control plans are at the

heart of an autonomous control system. Machine learning techniques can be used to optimize control plans, especially in the case of control plans that are generated by planners or are hand-coded. Optimization of control plans can occur either by experimenting with a simulation or by looking at data from previous executions of control plans in the real world. For example, taking a system from a wake operating mode to a sleep operating mode might involve establishing a series of setpoints. Learning could be used to identify the trajectory of setpoints that transitions the system most efficiently. Some research challenges in this area are:

- How can we provide safety guarantees when transferring a controller learned from simulations into the real system?
- How can the system learn not only control plans but also the contexts in which the control plans apply?
- What are the tradeoffs between having the system learn control plans from scratch as opposed to "tweaking" existing, working control plans?

3.4 Integration with autonomous control architectures

A large complex ALS system will have existing control procedures and possibly even an overarching control architecture. Very few of these incorporate learning or adaptation. Integration of learning into an existing autonomous control system raises a number of interesting research issues, such as:

- How good does the initial set of control behaviors or rules need to be for learning to be effective? Can we start *tabula rasa* or do we need very effective initial strategies?

- What are the differences between learning control information, learning procedural information, learning qualitative modeling information and learning planning information.

- What are the criteria whereby the autonomous control system turns on or off learning? Should learning ever be turned off?

- How does the autonomous control system decide when to use new learned actions?

3.5 Control system design methodology

The solution space for a control policy in an ALS system is enormous. We believe that machine learning techniques are a useful way to "probe" the solution space and give control system designers an idea as to its topology. Machine learning techniques could be used to determine the important control variables and the important (and sometimes hidden) interactions. 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. Using machine learning in this way helps overcome some validation and verification issues (since the actual control code is written by programmers, not the algorithms). Still, there are a number of research questions that need to be answered:

- How good of a simulation is needed for results from a machine learning algorithm that probes the solution space to be applicable to the physical system?

- How can the discoveries of machine learning algorithms be presented to control system designers such that they understand the topology of the solution space?

- Which machine learning techniques (i.e., reinforcement learning, memory-based learning, genetic algorithms, etc.) are most useful for probing the solution space? What are the strengths and weaknesses of these techniques with respect to guiding a control system designer?

3.6 Finding optimal resource allocations

Long-duration missions face severe resource constraints, especially with respect to life support consumables such as oxygen, water and food. Given adequate simulations, machine learning techniques such as reinforcement learning or genetic algorithms can search through combinations of control actions to discover a control policy that optimizes usage of a particular resource (or combination of resources). The resulting policy can be implemented by the autonomous control system to increase mission duration. The rest of this paper will focus on resource allocation within the context of a simple simulation. We will look at a variety of machine learning techniques. Other groups at NASA are looking at market-based allocation of resources for ALS systems [Crawford *et al* 2000].

4 EXPERIMENTS WITH A SIMPLIFIED SIMULATION

Evaluating machine learning techniques for a system as complicated as ALS is a daunting task. We decided that the best way to start was to create a simplified simulation of an advanced life support system. We wanted a simulation that was simple enough for us to understand what needed to be controlled, yet complicated enough to support quality machine learning research. Using this simulation we would explore a variety of machine learning techniques, including reinforcement learning and genetic algorithms. This section describes the simple simulation in detail and then discusses some of our early experiments.

4.1 The simulation

This section summarizes the cogent aspects of air, water and food processes. It is important to note that 1) they are all

deterministic, and 2) the processes take time to both degrade and to recover, i.e., a reduction or increase in energy to a process will not immediately reduce or increase its output. Also, the simulation makes no attempt to balance the mass flow.

Simulation time is measured in 'ticks', which are nominally one hour. A single tick will run each process once. That is, the crew will take in clean air, water and food and produce waste air, water and science. The air and water processes will take in waste air and water and produce clean air and water respectively. The plant module will take in energy and water and produce food and waste water.

The air module takes in exhalant air (i.e., "dirty") and uses energy to produce the same amount of clean air as long as the energy is at least one unit. If the energy is less than one unit for that tick, no clean air is produced. If energy is less than 0.25 units no clean air will be produced until three consecutive time units (ticks) of greater than 0.25 units of energy have passed.

The water module takes in wastewater and energy and produces clean water at a specified recovery rate, e.g., 97 %. Whenever the energy to the water processing drops below a certain level, the water processor needs to accumulate a specified amount of energy before it begins to produce potable again. In particular, if the energy drops below 0.5 units for more than five ticks, its capability to produce potable water is lost until E energy is accumulated. The amount of time this takes depends on how much energy is allocated to the water recovery module each tick.

Food production is a function of water and energy, thus in the integrated system, the water output from the water recovery module is split evenly between the crew and the food production systems. The food production curve is selected based on the ideal that for N crew, N units of daily food requirement should be produced in a "days" (24 ticks) time. Thus, the curve produces N units of food when water and energy are both greater than or equal to N, and drops off sharply when the product of the units drops below 1. Because grain crops are involved, if the energy drops below 0.25 units for more than 168 ticks or if the water drops below 0.25 units for more than 120 ticks, the crops all die and no further food production is possible.

The crew model takes in clean air, clean water and food and produces science and dirty air and waste water at varying rates depending on the specified activity level during a given tick. These rates are shown below (Wd = wastewater, Wc = clean water, Ad = exhalant air, Ac = clean air, F = food, and S = science):

Low Activity Level
Wd = .95Wc
Ad = .95Ac
S = (F + Wc + Ac)*0.3

Medium Activity Level
Wd = .85Wc
Ad = .85Ac
S = (F+Wc+Ac)*0.6

High Activity Level
Wd = .75Wc
Ad = .95Ac
S = (F+Wc+Ac)*0.9

At the higher activity levels, more science is produced but with varying loss of efficiencies for converting water and air. Of course, with enough loss of any of the inputs the crew will cease to produce science. Specifically the mission is over if Ac is less than 0.25 per crew member for more than three ticks, or Wc is less than 0.25 per crew member for more than 10 ticks or F is less than 0.25 per crew member per 24 ticks for more than 20 ticks.

There are also stores for food, water and air. We typically used 25 ticks worth of stores for each. For four people that is 200 units of water and 100 units of clean air and food respectively. At each tick the corresponding store can be used instead of the air, water or food process, thus saving energy used. The simulation starts the flow from the crew module, producing science, used air and waste water based on the starting activity level from an assumed input of N units of clean air, water and food (given gratis). From there the dirty air and water flow to each process where they are either bypassed if the store will be used, or energy is allocated and the process takes in the flow and yields its product. Ultimately, clean air, water and food reach the crew thus completing a tick. Clean water is equally divided between the crew and the plant crops.

The amount of energy available for the mission is set in an energy store. When this amount is fixed, the goal of the system is to allocate energy among the subsystems in an optimal fashion. Optimality is determined by whether the goal is the longest mission, the most cumulative science or some combination of the two. When the energy is used up and the stores depleted, the life support processes will wind down and the mission will eventually be terminated due to the lack of air, water or food for the crew. For the majority of the experiments described, the energy store was the sun and was replenished every time step.

4.2 Introduction to learning techniques

We experimented with two different approaches to learning an optimal control strategy for the system described in the previous subsection. Formally, the problem we have can be formulated as a Markov Decision Process (MDP) $M = (S, A, next, r)$, where S is the set of system states, A is the set of available control actions, $next : SxA \rightarrow S$ is the transition function of this

deterministic system and *r* is the local reward function $r : SxA \rightarrow \Re$. The goal of the learning system is to find a policy $\prod : S \rightarrow A$ such that $E(\sum_{t=0}^{\infty} r(S_t, \prod(S_t)))$ is maximized or $\sum_{t=0}^{\infty} r(S_t, \prod(s_t))$ is maximized since the system is deterministic.

One approach to this problem is to learn a value function $Q_{\prod}(s,a)$, which is the expected cumulative local reward if one starts from *s* with action *a* and follows policy $\prod$ thereafter. We can use dynamic programming and recursively formulate $Q_{\prod}(s,a)$ as:

$$Q_{\prod}(s,a) = \max_{a \in A}[r(s,a) + Q_{\prod}(next(s,a),a)]$$

with appropriate boundary conditions where Q values of terminal states are provided. This equation forms the basis of the Q learning algorithm [Watkins and Dayan, 1992], which starts with initial estimates (zero in our case) of $Q(s,a) \forall s,a$ and then updates it on-line upon each state transition (s,s',a,r) as follows:

$$Q(s,a) = (1-a)Q(s,a) + a[r + \max_{a' \in A} Q(s',a')]$$

where a is the learning rate and indicates how aggressively new estimates replace old ones.

A second approach to this problem is to directly learn a policy $\prod : S \rightarrow A$ without learning a value function. A popular way to do this is to use a genetic algorithm [Holland 1975]. Genetic algorithms characterize the solution as a bit string, where each bit represents an action or input into the system. A population of bit strings is initialized and each string is tested against the simulation. The best strings are preserved and operations such as mutation and cross-over performed. Then the bit strings are re-evaluated and the process starts over. The next two sections look at these approaches as applied to the life support simulation.

4.3 Reinforcement learning

The reinforcement learner was a standard on-policy zero-initialized Q learning system. The state space consisted of the energy allocated to each of water, air and food, the crew activity level and the whether each store was in use or not (yes or no). An action consisted of a designation of one of seven modules -- air, water, food activity level or one of the three stores -- and a change: increase by one, decrease by one or no change -- for an action space of 21. To make the Q-table more tractable for our initial investigations we allowed only energy allocations of 3, 4 or 5 units, which still resulted in a Q-table of over 9000 entries.

The local reward function depended on our goal. When we wanted to maximize the length of a mission the local reward was one if the mission lasted one more tick. If the goal was to maximize science the reward was the amount of science produced by the crew in that one tick. Whenever the mission ended due to lack of air, water or food, a negative reward of -10000 was given. We used epsilon-greedy sampling (10%) with a learning rate of 0.9 and no discounting. For most runs we used a benign starting state where all stores are used, energy allocation is N for each module and an activity level of 1. The simulation was run continuously during the learning in this fashion. If t was zero, the simulation was initialized, the stores and energy allocation corresponding to the action were set, and the simulation was run for one step. The learner took in the resulting state, computed a reward function, updated its Q table, and selected another action. If the result of the last action was an end of the mission, the next action would cause the simulation to be reinitialized before continuing. In effect, we used the model to simulate as many missions as necessary to reach convergence.

We ran several experiments of varying numbers of samples, but the Q learner results varied wildly from experiment to experiment. For example, it produced 185 units of science in 38 ticks before mission end using six thousand samples in one experiment, and 98 units of science in 23 ticks using 20000 samples in another experiment. The next two tables show our typically "best" results.

Samples	science	ticks
502	150.448308	13
1018	318.30795	20
1525	179.515967	30
2037	318.30795	20
2548	318.30795	20
3059	322.06935	20
3577	318.30795	20
4078	318.30795	20
4594	318.30795	20
5111	322.06935	20
5629	322.06935	20
6009	322.06935	20

Table 1 Science as a function of Q-learning samples (rewarding for science)

Samples	ticks	% Random
1011	28	90
2014	25	80
3035	28	70
4037	28	60
5070	28	50
6082	28	40
7107	28	30
8217	38	20
9276	38	20
10080	38	20

Table 2 Ticks as a function of Q-learning samples (rewarding for mission duration)

4.4 Genetic algorithms

Recall from Section 4.2 that a genetic algorithm searches for a policy by manipulating a bit string that represents actions. We used a bit-string consisting of 2 bits each (water, air and food) 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 crew activity level (1,2, & 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. We typically ran twenty experiments with starting populations of 100 each. To evaluate a string, the simulation was run using that string each tick until the mission ended, i.e., the crew was unable to produce science due to a lack of air, water or food. The value of the string was typically either the total ticks until mission end raised to a power or the accumulated science raised to a power. Depending on whether the function stressed mission duration or science the runs invariably resulted in a mission duration of 31 ticks with 148 units of total science or 26 ticks with 184 units of science. Two examples are shown below (additional experiments resulted in no better policy).

Exp	Science	Ticks
1	183.9	26
2	184	26
3	183.9	26
4	184	26
5	184	26
6	184	26
7	183.8	26
8	183.8	26
9	184	26
10	184	26

Table 3 11 Bit GA String Results (evaluation = total science cubed) in 10 experiments (Population size = 100)

Exp	Ticks	Science
1	31	147.7
2	31	147.5
3	31	49.2
4	31	49.2
5	31	49.2
6	31	51.9
7	31	52
8	31	147.5
9	31	140.9
10	31	147.5

Table 4 11 Bit GA String Results (evaluation = mission duration (i.e., ticks [5]) in 10 experiments (Population size = 100)

5 AN INNOVATIVE GENETIC ALGORITHM APPROACH

After implementing both the Q-learning system and the genetic algorithm system it was obvious that they both were achieving roughly the same results, with the Q-learner doing slightly better at maximizing mission duration, and almost twice as well maximizing science. Yet, a simple intuitive policy of using the stores first before using the life support processes could achieve a mission duration of 56 ticks and 324 units of science. These learners should have been doing better.

While the Q-learner was learning a state-action policy, the GA was learning one action to take regardless of the state of the system. So we hypothesized that learning what to do for the next n steps with the GA would result in a better policy. The idea was to make the strings *n* times 11 bits long and evaluate them by stepping the simulation for each 11-bit action. For example, for a three step GA, to evaluate a 33 bit string, one would invoke the actions indicated in the first 11 bits, step the simulation, then invoke the actions specified by the next 11 bits then step the simulation, and finally invoke the actions designated by the last 11 bits then step the simulation. The value for the whole string is again the same function as in GA1, e.g., the accumulated ticks or science raised to a power, but only *after the three steps*. When the mission ended, the simulation was simply reinitialized, which resulted in zeroing the accumulated ticks and science. The final most fit individual discovered in twenty experiments now becomes the policy to use for every n steps.

Despite the fact that the policy is essentially run open loop (i.e., no state is used), the results with GA2 were fairly spectacular compared to the results in Table 3 and Table 4. We summarize the results in the following graphs.

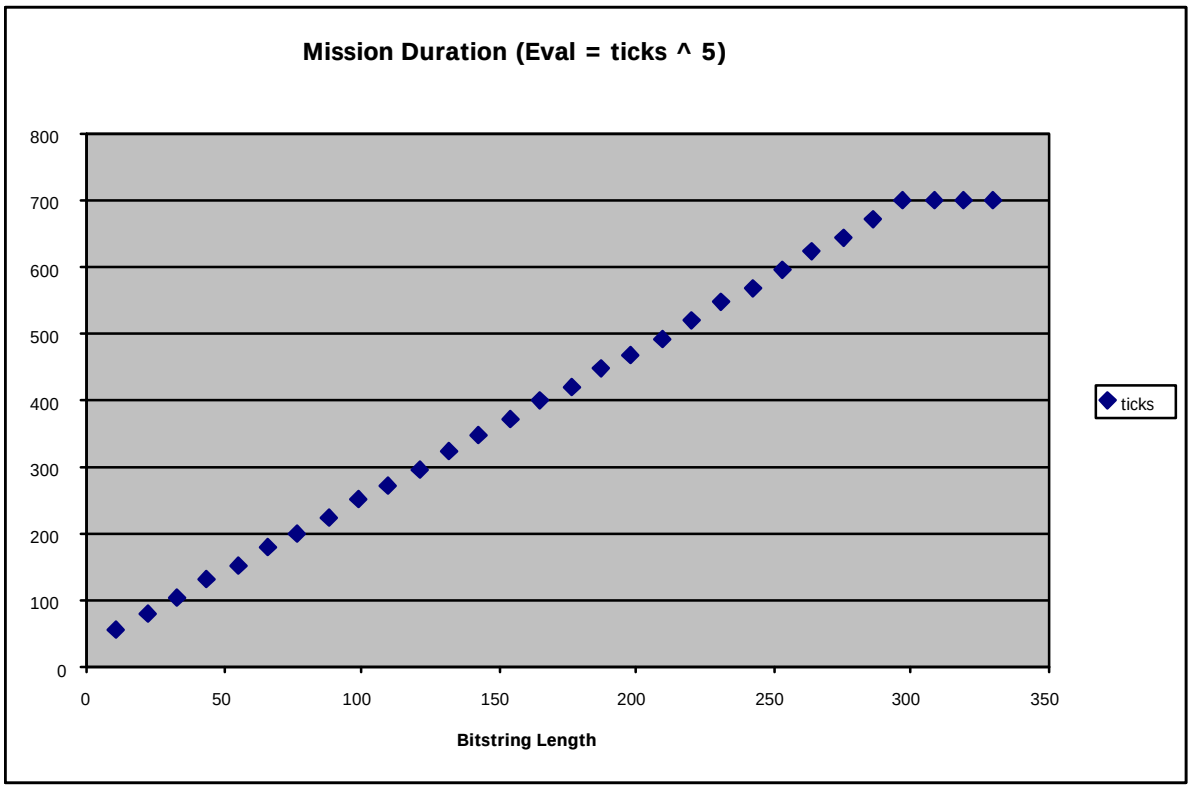


Figure 3 Multistep GA Ticks Results (Population size = 100)

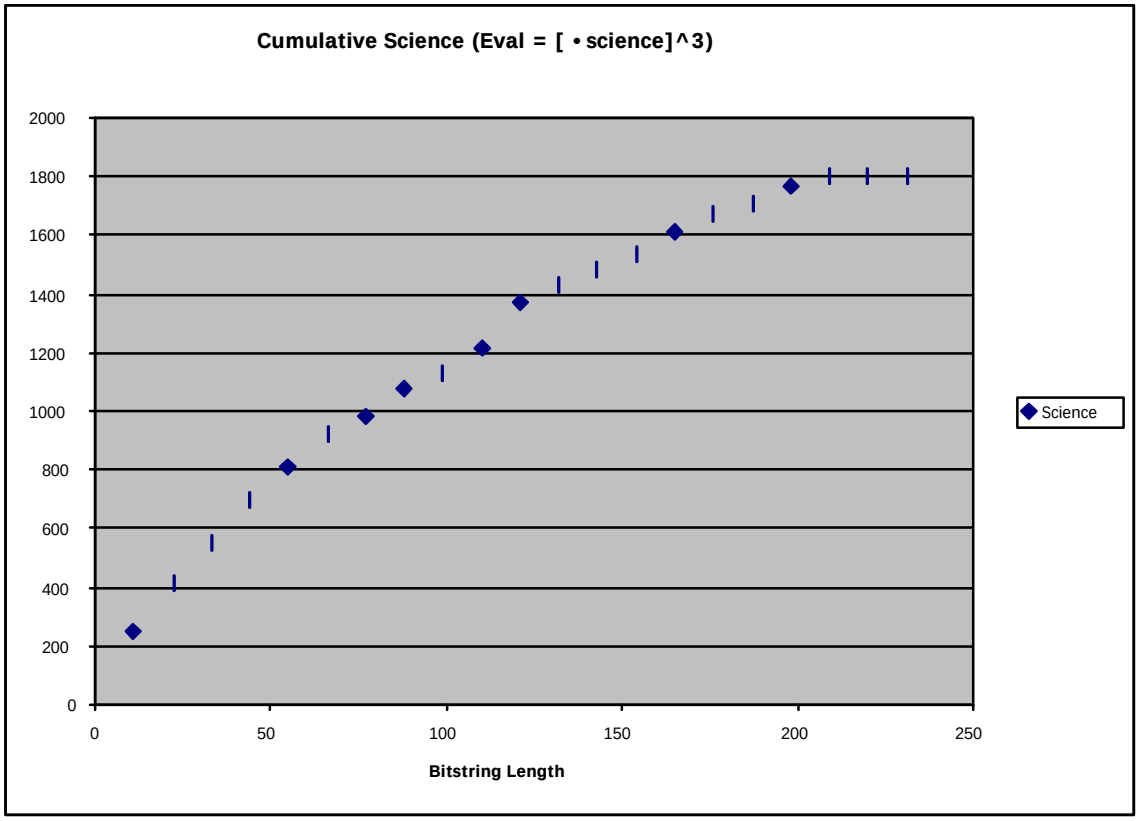


Figure 4 Multistep GA Science Results (Population size = 100)

Before continuing it should be pointed out that these curves are generated in the following manner. A bitstring length of 11 was selected and an initial evaluation target of 31^5 (in the case of a pro-ticks search) was asserted. The GA would be run for twenty experiments with each experiment ending when the policy represented by the best string was greater than the evaluation target. The best score of all the experiments would be set as the new evaluation target, the bitstring length would be increased by 11, and twenty more experiments would again be run. The process terminated when the next increase in bitstring length would only equal the evaluation target or if no experiment could equal or better the target after 3000 generations. So the above charts (and all subsequent) are extended one or two steps to indicate the evaluation target ceiling that was reached.

GA2 learned to achieve a mission length of 700 days, or to generate over 1800 units of science. What was it that GA2 was learning about this simple life support simulation? Figure 5 graphically depicts a kind of pulsing policy GA2 learned for maximizing mission duration.

The ticks policy optimized for mission duration lasted 702 ticks generating 1243 units of science. Figure 5, showing the first hundred ticks of the policy in effect. An indirect measure of what is happening in the simulation is the dirty air and wastewater output from the crew at each tick. Higher values indicate that the crew module saw more mass of clean air and clean water as input. Figure 5, showing the first hundred ticks of the policy in effect, shows that GA2 discovered a policy that allowed the air and water flow to reach very low levels before pulsing the system with first a water store use and then an air store use. This is continued until first the air, and then the water stores is exhausted at ticks 671 and 675 respectively (see Figure 6). It takes another 30 ticks for the air to fall to mission ending levels.

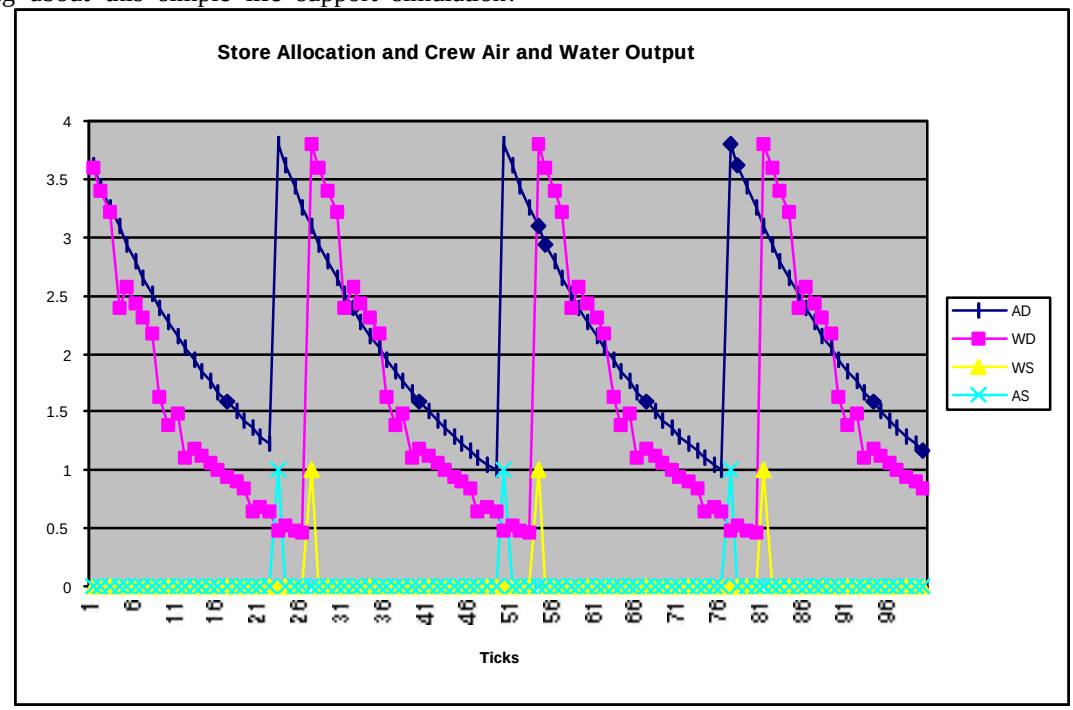


Figure 5 Comparison of Store Use With Dirty Air and Waste Water Output from Crew Module

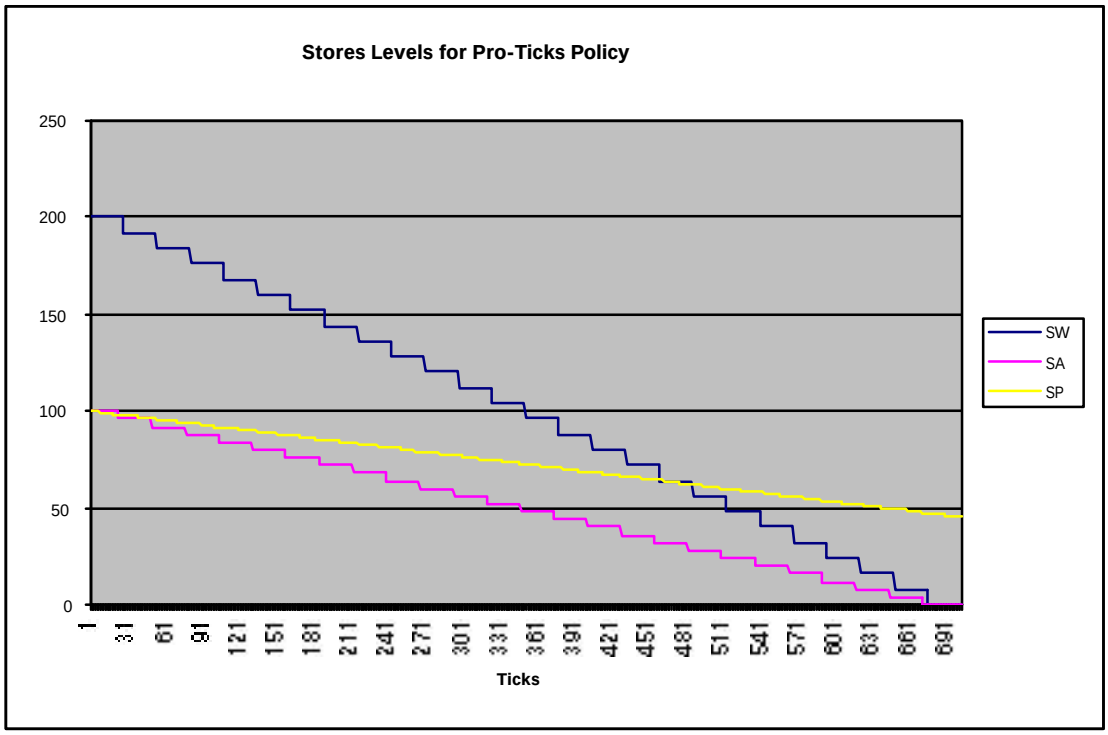


Figure 6 Stores Consumption

6 Revisiting Reinforcement Learning

Our first attempt at using reinforcement learning was not productive. We were trying to use Q-Learning to implement an open-loop controller. In an attempt to get the Q-learning system to perform better, we recast the state and action spaces to reflect those facets the multi-step GA had uncovered. In particular, our state space would reflect the air and water outputs of the crew, the store status and some measure of the time since an action was taken.

We constrained the problem so that the only decision the learner was required to make was whether to use each store or not. The energy allocation was constrained to be 3 units for each process, and the activity level was fixed at 3 (for a pro-science policy).

The Q model was as follows:

State Space:

Water & Air outputs from crew:

W: 0,1,2 (0 means < 33%, 1 means 33% <= x < 66%, and 2 means >= 66%), where 100% = 4.0
A: 0,1,2 (0 means < 33%, 1 means 33% <= x < 66%, and 2 means >= 66%), where 100% = 4.0

Stores Status:

WS: 0,1 (0 means < 50%, 1 means > 50%), where 100% = 200
AS: 0,1 (0 means < 50%, 1 means > 50%), where 100% = 100
PS: 0,1 (0 means < 50%, 1 means > 50%), where 100% = 100

Ticks since positive action:

SW: 0,1 (0 means less than 9, 1 means greater than 9)

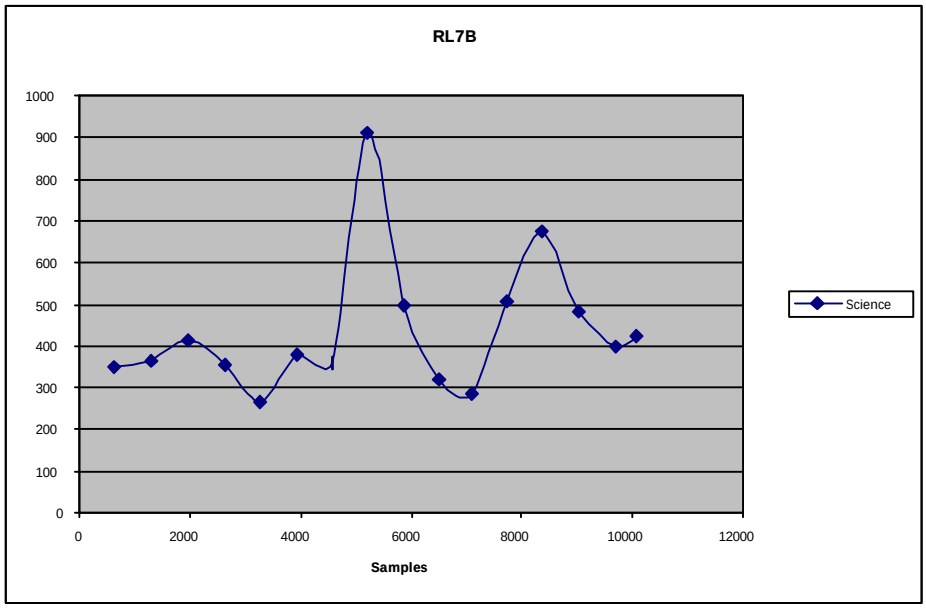
SA: 0,1 (0 means less than 9, 1 means greater than 9)

Action Space:

Use or not use the store for each of water air and plants, e.g., (1 0 1) means use the water and plant stores, not the air store)

Reward function: science^2 x water-store x WD x air-store x AD

The results of one of the runs is shown in Figure 7. While it is clear that the reinforcement learning algorithm is not converging, it is producing science at a level equal to the multi-step genetic algorithm. An examination of the control philosophy shows the same “pulsing” strategy employed by the multi-step genetic algorithm.



Samples	Science	Ticks
647	350.166058	67
1295	362.099302	67
1951	411.486381	68
2622	355.082955	61
3267	264.029052	45
3925	379.153828	68
4554	361.391647	61
5213	909.616541	176
5865	496.283236	92
6500	319.51492	61
7108	284.349158	49
7718	505.187961	92
8361	672.872989	123
9015	482.361459	87
9681	398.32097	74
10068	423.490573	77

Figure 7 Reinforcement learning results for a pro-science policy

7 Future Research Goals

This paper describes the very start of a long research effort into applying machine learning techniques to the lay out the research issues and to explore some possible approaches. We are currently working on a more complex simulation that will have stochastic processes, more natural cycles (e.g., the crew will sleep, the plants will go through normal growth cycles, etc.) and more interesting dynamics. As we slowly increase the complexity of the simulation we will explore how the different learning approaches scale. We also plan to distribute this simulation to the research community to encourage others to try their approaches. Our ultimate goal is to integrate adaptive control processes into the BIO-Plex control system for experimentation with human crew.

8 References

[Barta *et al* 1999] Dan Barta, et al., "The Biomass Production System for the Bioregenerative Planetary Life Support Systems Test Complex: Preliminary Designs and Considerations," SAE Paper 1999-01-2188, 29th ICES, 1999.

[Crawford *et al* 2000] Sekou Crawford, Christopher Pawlowski and Cory Finn, "Power Management in Regenerative Life Support Systems Using Market-Based Control," *30th International Conference on Environmental Systems*, 2000.

[Holland 1975] John Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor MI, 1975.

[Kortenkamp *et al* 2000] David Kortenkamp, Debra Schreckenghost and R. Peter Bonasso, "Real-time Autonomous Control of Space Habitats," *AAAI Spring Symposium on Real-Time Autonomous Control*, 2000.

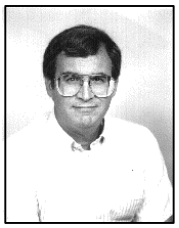
[Tri 1999] Tri, T. O.; "Bioregenerative Planetary Life Support Systems Test Complex (BIO-Plex): Test Mission Objectives and Facility Development," SAE Paper 1999-01-2186, 29th International Conference on Environmental Systems, 1999.

[Schreckenghost *et al*. 1998] Debra Schreckenghost, Daniel Ryan, Carroll Thronesbery and R. Peter Bonasso, "Intelligent Control of Life Support Systems for Space Habitats," *Proceedings of the Conference on Innovative Applications of Artificial Intelligence*, 1998.

[Watkins and Dayan 1992] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, 8, 279-292.

David Kortenkamp *is a senior research scientist at Metrica Inc., supporting NASA Johnson Space Center. He received his Ph.D. in computer science and engineering from the University of Michigan in 1993 and his B.S. in computer science from the University of Minnesota in 1988. Dr. Kortenkamp directs nearly $800,000 in annual research projects and is on a number of conference and workshop program committees. He is co-author of the book* Mobile Robots and Artificial Intelligence *and is associate editor of the MIT Press series on Intelligent Robotics and Autonomous Agents.*

R. Peter Bonasso *is a senior staff consultant for AI & Robotics at Metrica Inc, based at NASA Johnson Space Center. He currently supports the Automation, Robotics and Simulation Division investigations of intelligent monitoring and control using layered architectures. He is the co-developer of the 3 Tiered Robot Control Architecture, and has applied that architecture to the control of robotic and life support machines. He received his B.S. in engineering from the U.S. Military Academy at West Point in 1968 and master's degrees in operation research and computer utilization from Stanford in 1974.*



Devika Subramanian *is an Associate Professor of Computer Science at Rice University. She received her Ph.D. in computer science from Stanford University in 1989. She is currently on the IJCAI advisory board and was the program chair of AAAI 1999. She is on the editorial board of the Journal of AI Research. Dr. Subramanian is the author of numerous conference and journal articles in the area of machine learning and control optimization.*