The Interval Reduction Strategy for Monitoring Cupcake Problems

Paul R. Cohen, Marc S. Atkin, and Eric A. Hansen
Experimental Knowledge Systems Laboratory
Department of Computer Science, LGRC, Box 34610
University of Massachusetts, Amherst, MA 01003
{cohen,atkin,hansen}@cs.umass.edu
(413) 545-3638

Abstract

Monitoring is the process by which agents assess their environments. Most AI applications rely on periodic monitoring, but for a large class of problems this is inefficient. The *interval reduction* monitoring strategy is better. It also appears in humans and artificial agents when they are given the same set of monitoring problems. We implemented two genetic algorithms to evolve monitoring strategies and a dynamic programming algorithm to find an optimum strategy. We also developed a simple mathematical model of monitoring. We tested all these strategies in simulations, and we tested human strategies in a "video game." Interval reduction always emerged. Environmental factors such as error and monitoring costs had the same qualitative effects on the strategies, irrespective of their genesis. Interval reduction appears to be a general monitoring strategy.

1 Introduction

All embedded agents must monitor. Monitoring means seeing how plans are progressing, checking how much progress has been made, finding out what time it is, updating one's location, looking for obstacles, making sure that nothing has changed unexpectedly, and so on. Monitoring has been studied to some extent by AI researchers [9, 19, 21, 22, 25, 26], indeed, the earliest work on planning for the Shakey robot emphasized monitoring [5, 6], but AI research has little to say about monitoring *strategies*, about when and how often to monitor, and how these decisions depend on the dynamics of the environment [15]. Most systems monitor periodically, although this can be wasteful of effort.

Our interest in monitoring is only partly to find efficient strategies, however. We also suspect some monitoring strategies might be very general. If they are determined largely by environment dynamics, then we might observe the same strategies in agents as different as bumblebees, children, and simulated robots. While we cannot report anything about bumblebees, we will describe a monitoring strategy called *interval reduction* that shows up in children and adults, and was evolved by two genetic programming algorithms that produce programs to control simulated robots. We will also develop a simple statistical model of the strategy and describe the performance of a policy based on the model. All these strategies have the same general performance, but no two are identical, which prompts us to ask which is best and how close is it to optimal. To answer these questions, we develop an optimal monitoring strategy by stochastic dynamic programming. The fact that it, too, is an interval reduction strategy lends further support to our belief that interval reduction might be very general; that is, we expect it to evolve in other agents in similar environments.

2 The Cupcake Problem

In 1985, Ceci and Bronfenbrenner described a monitoring task for children that they called the *cupcake problem* [4]. Each child was instructed by an older sibling (who served as a confederate in the experiment and collected the data) as follows: "We are going to put some cupcakes in the oven and they will be done in thirty minutes. You can play PacMan while you wait, but you mustn't forget to take the cupcakes out of the oven. To help you remember, there's a clock on the wall." Cleverly, the clock was put behind the child, so the sibling could easily see when the child turned around to monitor the time. In this way, Ceci and Bronfenbrenner obtained latencies between monitoring events. For our purposes two results are notable: First, all the children monitored quite frequently for the first few minutes; Ceci and Bronfenbrenner interpret this as a period of "calibration," getting one's internal clock in synch with real time. Second, ten-year-old children monitored approximately periodically for the remainder of the trial, but fourteen-year-olds monitored infrequently after the initial calibration, and

This work was supported by ARPA/Rome Laboratory under contract #F30602-91-C-0076 and under an Augmentation Award for Science and Engineering Research Training.

increased the frequency of monitoring as the deadline approached. We call this an interval reduction strategy because the interval between monitoring events is reduced as the deadline approaches.

Cupcake problems require an agent to traverse some time or distance, at which point an event or destination is expected. If the agent quits before reaching this point or overshoots it, a penalty is incurred. Sometimes the penalties are asymmetric around the goal point; for example, a racing driver loses a race if he or she doesn't "push the envelope," but might die by pushing too far. In some cupcake problems, an agent can backtrack if it overshoots the goal; in others, particularly temporal problems, this is not an option. It is characteristic of cupcake problems that the agent cannot be sure of its location without monitoring. In spatial problems this can be due to sensor or movement errors, or to movement of the goal point itself; in temporal problems the agent's internal clock might be inaccurate. We have studied one- and two-dimensional cupcake problems. Most of the results in this paper concern the former.

3 A Strategy for One-dimensional Cupcake Problems

In a one-dimensional cupcake problem, an agent moves toward its goal along a line, and errors in the agent's estimate of its location accumulate. For instance, if you close your eyes and start walking toward a wall, you will experience uncertainty about how far you are from the wall. This is a one-dimensional problem in the sense that drift away from a line normal to the wall is negligible, and, in any case, the painful error accumulates along this line. Assume you begin with an accurate estimate of the distance to the wall. You close your eyes and begin walking. When should you look again?

To answer the question we need to model how errors in estimates of location accumulate. We assume a simple but quite flexible *binomial* model: Let G be the distance an agent must travel by taking steps of size $1 \pm d$. On each step the agent will travel $1 - d$ or $1 + d$ with equal probability. If the agent takes N steps it will travel $D(N)$ units of distance. The mean of $D(N)$ is N and the variance of $D(N)$ is Nd^2.[1]

If $d > 0$ and the agent aims for the goal (i.e., $N = G$) then it will overshoot the goal with probability .5. Perhaps this is satisfactory, but often the agent will want a smaller probability. Let $D(N)_{.01}$ be a distance that the agent will exceed no more than 1% of the time it moves N steps. Because the distribution of $D(N)$ is binomial with $p = .5$ we may approximate it with a normal distribution with mean N and variance Nd^2. If we can find a value above which 2% of the distribution of $D(N)$ lies, then we can expect the agent to exceed this value in 1% of its trials: The error model is symmetric, so every extremely large value of $D(N)$ is matched by an extremely small one. We know that 2% of a normal distribution lies more than 2.05 standard deviations above its mean. Thus, if an agent travels 10 steps, $D(10)$ will exceed $10 + 2.05(\sqrt{10d^2})$ no more than one percent of the time. In general,

$$D(N)_\alpha = N + z_{2\alpha}\sqrt{Nd^2}$$

where $z_{2\alpha}$ is the number of standard deviations above the mean of a normal distribution that cuts off the highest $100\alpha\%$ of the distribution.

To ensure that an agent does not exceed the desired distance, G, with greater than $\alpha/2$ probability, we set $D(N)_\alpha = G$ and solve the previous equation for N:

$$N = \frac{2G + d^2 z_\alpha^2 - (dz_\alpha\sqrt{4G + d^2 z_\alpha^2})}{2} \qquad (1)$$

We call this policy SIR for simple interval reduction. In all our experiments, we allowed the agent to quit when it came within one unit of its destination (otherwise Zeno would have the last laugh). For example, assume $d = .5$ and the agent wants a low probability, say .01 or less, of overshooting its destination $G = 100$ units away. SIR says the agent should travel

$$\frac{(2 \cdot 100 + .5^2 \cdot 2.05^2 - (.5 \cdot 2.05 \cdot \sqrt{4 \cdot 100 + .5^2 \cdot 2.05^2}))}{2}$$

$$= 90.26$$

units before monitoring again. This, then, is how far the agent *intends* to travel, but due to accumulating errors its actual location after 90.26 steps will be something other than 90.26. Let's say $D(90.26) = 93$. Now the agent must travel seven more units to its goal, and SIR says it should go 4.76 units before monitoring again. Note, however, that the total probability of overshooting the goal, denoted α_t, is now greater than .01 because the agent has taken this bet twice, once on each move. The probability that it will *not* overshoot during m moves is $(1 - \alpha)^2$, so $\alpha_t = (1 - (1 - \alpha))$, which is approximately $m\alpha$ for small m. Because an agent under the control of SIR rarely monitors more than three or four times (see below) it suffices to reduce α by a factor of five, say, to ensure a desired α_t.

4 An Optimal Policy for One-dimensional Cupcake Problems

SIR is suboptimal if there are costs for monitoring and penalties for overshooting (or falling short of) the goal.

[1] On each step the agent goes forward one unit and then forward or backwards by d units, so $D(N) = n + d(f - (n - f)) = 2df - n$, where f is the number of times in n steps the agent travelled forward by d. The variance of $2df - n$ is $4d^2\sigma_f^2$ and because f is binomial, with variance $n/4$, the variance of D(N) is Nd^2.

If monitoring costs are large and penalties small, then monitoring isn't worthwhile, but SIR will do it anyway. Unfortunately the tradeoff between monitoring costs and penalties can be played out at every place the agent might stop to monitor. In fact, deciding when to monitor is a *sequential decision problem* [24]. Finding an optimal *control policy* is also a sequential decision problem. Actions can have delayed effects that must be considered in choosing the action to take in each state, and a policy that chooses actions solely for their immediate effects may not be optimal over the long term. Monitoring problems have an additional aspect: a control action need not be taken every time a process is monitored (in cupcake problems the control action is to quit—take the cupcakes out of the oven, quit walking toward the wall). Deciding whether to act immediately or wait and monitor again with the option of acting later is a sequential decision problem because a sequence of later opportunities for acting must be considered in deciding what to do [13, 14].

Stochastic dynamic programming is a well-known optimization technique for solving sequential decision problems, but monitoring problems differ from conventional control problems in two respects: It isn't necessary to take a control action each time a process is monitored, and it isn't necessary to observe the process at each time step. The first difference is easily modeled by including a null action in the controller's action set. The second and more significant difference comes into play if monitoring incurs a cost and the controller itself must decide when to observe the state of the process.

To compute a monitoring policy as well as a control policy, the key idea is to distinguish the time steps of a sequential decision problem from its decision points (or "stages"). At a decision point, the controller observes the state of the process and makes a decision. The assumption in conventional dynamic programming is that a decision point takes place at each time-step. This assumption is relaxed when computing a monitoring policy. When the controller is responsible for deciding when to observe the state of the process, it can wait an arbitrary number of time-steps before monitoring, as determined by its monitoring policy. However, this means that the conventional payoff functions and state transition probabilities must be extended so they are defined for an arbitrary number of time steps instead of a single step. Multi-step state transition probabilities and a multi-step payoff function let the controller *project* the state of a process and the payoff it expects to receive an arbitrary number of time-steps into the future. They also add complexity to the dynamic programming search. Hansen [14] has shown that the curse of dimensionality is ameliorated if utility is assumed to be a monotonic function of monitoring interval, and he also suggests finding an acceptable but coarse-grained time interval. This suggestion was tested with good results in [23].

5 Genetic Algorithm Solutions to One- and Two-dimensional Cupcake Problems

Given parameterized functions describing when to monitor next, the MON system runs a genetic algorithm to determine the best parameter values—the ones that minimize the expected cost. For the cupcake problem, the function was $N = ct + b$. Given the time t remaining till the deadline, $N = ct + b$ computes when to monitor next. When $b = 0$ and $c > 0$, the resulting strategy is a form of interval reduction called *proportional reduction*; for instance, if $c = .8$ then the strategy would always go 80% of the remaining distance before monitoring again. We call c the proportional reduction constant. If $c = 0$ and $b > 0$ then the strategy is a form of periodic monitoring. MON also learned a maximum number of times to monitor. It would quit a trial when this number was exceeded or if it went past the goal. The cost function consisted of the cost of monitoring multiplied by the number of times the agent monitored, plus the squared distance to the goal upon trial termination. This cost was also the fitness measure used by the genetic algorithm. The rest of the genetic algorithm was very basic, too. It used roulette wheel selection, fixed mutation and crossing over rates, and a population size typically around 100 [1]. MON's strategies have been tested with results shown in Table 1.

Until now we have described rather abstract one-dimensional monitoring problems. LTB is a genetic programming algorithm that evolves monitoring strategies for simulated robots in two-dimensional worlds. The programs that control all the robots' activities, including monitoring, are expressed in a language of basic effector commands such as MONITOR, MOVE and TURNRIGHT, and some control structures such as loops and simple conditionals.

The robots in LTB monitor as they approach a particular position in the map, the *goal point*, which has an obstacle on it. Their aim is to get as close to this point as possible without hitting it. They have a sensor (activated by MONITOR) that returns the distance to the goal and a command TURNTOGOAL that points them in the right direction. Since overshooting the goal point is penalized highly, this problem is in fact an *asymmetric* cupcake problem.

The version of LTB that produced the data in Table 1 represents programs as linear lists of commands, instead of the tree structures common in the Genetic Programming field [16, 17]. Crossing over simply swaps two chunks of code between two individuals; mutation changes one command into another. Since the language is relatively simple, only a few constraints are needed to keep programs legal. Tournament selection with a tour-

```
Main program:
 NOP
 NOP
 TURNTOGOAL
 MOVEQUICK
 LOOP 5 time(s):
   MOVE
   MONITOR: object_distance
   NOP
   NOP
   NOP
   NOP
   NOP
   LOOP (object_distance)/10+1 times:
     LOOP 2 time(s):
       MOVE
       NOP
       MOVE
       NOP
*reached_goal* interrupt handler:
 DISABLE: reached_goal
 NOP
 LOOP (direction)/10+1 times:
   NOP
   NOP
*object_distance* interrupt handler:
 NOP
*hit_object* interrupt handler:
 NOP
 NOP
 NOP
 IF (object_distance) <= 75 THEN STOP
 MOVEQUICK
 MONITOR: reached_goal
```

Figure 1: A proportional reduction strategy generated by LTB

nament size of two was chosen as the selection scheme, the population size was set at 1000. Since genetic algorithms are not guaranteed to produce optimal results, the system was rerun 10 times on each training problem and the best program from these 10 runs was selected as the monitoring strategy for a case.

An example for a monitoring strategy evolved by LTB is given in Figure 1. The program is the best output of LTB for a test case corresponding to $d = 1.0$, $G = 150$. The top half the code is the main body of the program; the bottom half consists of the three *interrupt handlers* corresponding to the agent's sensors; they are not used, however[2]. The program itself is quite simple: After turning itself towards the goal via TURNTOGOAL, the agent goes into a finite loop. Within this loop, it will measure the distance to the goal (MONITOR: OBJECT_DISTANCE) and then loop over this distance. Since it is executing four MOVE commands for every ten distance units, it is moving 40% of the distance remaining after each monitoring action. The proportional reduction constant is therefore $c = .4$. The program will terminate after monitoring five times. Note that there is an extra MOVE instruction within the outermost loop, so the pure proportional reduction strategy will be distorted very slightly. All the other LTB programs in this study did implement proportional reduction, however.

[2]Interrupt handlers are the mechanism by which a robot can react directly to external events. Each program contained an interrupt handler for each kind of sensor the robot had. They executed automatically when the corresponding sensor value changed.

6 Adult Human Strategies for a One-dimensional Cupcake Problem

We implemented a "video game" cupcake problem for human subjects. On the display, the subject sees a line marked with ticks at regular intervals. When a trial begins, a ball is at the leftmost end of the line. The goal is to get the ball as close as possible to the end of the line, while simultaneously minimizing the number of moves required to do it and the penalty for falling short or overshooting the end. The subject moves the ball by pointing to a tick mark with the mouse. This is equivalent to selecting N in the SIR model or the dynamic programming or genetic algorithm policies. When the subject clicks on the line, the ball travels N steps of size $1 \pm d$. The subject is then assessed the cost of monitoring and is given the opportunity to quit the trial or move again toward the goal. A trial ends when the subject decides to move no closer to the goal, or if the ball overshoots the goal. On each trial the subject is told d, the error function parameter, and also the cost of monitoring. During training the subject is told that the penalty for stopping short of the goal or overshooting the goal is the squared distance to the goal. The cost of a trial is the penalty plus the number of monitoring events. No incentives (besides pride) are provided to encourage the subjects to minimize costs. We explain to the subjects that movement errors accumulate with distance, so if d is high, aiming for the end of the line can produce big errors and penalties. We also train the subjects on as many problems as they desire before presenting them with a set of test problems.

7 Comparing Strategies

We compared humans, MON parametric functions, optimal dynamic programming policies and SIR policies on a set of one-dimensional cupcake problems. LTB programs were tested on two-dimensional versions of very similar problems. The apparatus for humans was the "video game" described earlier. MON, SIR and the optimal policies were tested in a simulator, in which each trial went as follows:

Loop

1. Monitor to find distance to the goal
   - If beyond the goal, quit; else
   - If within one unit of the goal, quit; else
2. Decide N, the number of steps to move before monitoring again.
3. Take N steps of $1 \pm d$ units

G	mon.-	optimal		SIR		MON			LTB		human	
	cost	mon's	cost	mon's	cost	mon's	cost	%	mon's	cost	mon's	cost
20	.5	1.94	.99	2.75	1.97	1.0	4.08	*84*				
50	.5	1.92	1.0	2.76	1.89	1.43	1.2	*90*				
100	.5	2.14	1.18	3.05	2.11	1.55	1.05	*67*				
150	.5	2.66	1.37	3.1	2.27	1.67	1.53	*84*				
20	1.0	1.65	1.81	2.65	2.68	1.0	3.38	*58*	1.0	15.33	2.07	2.51
50	1.0	1.9	1.96	3.11	3.35	1.6	2.03	*87*				
100	1.0	2.22	2.25	3.11	3.52	1.46	2.7	*93*				
150	1.0	2.34	2.38	3.92	3.52	1.55	2.32	*84*	2.0	65.33	2.72	3.18
20	2.0	1.34	3.06	2.68	4.34	1.0	3.75	*60*				
50	2.0	1.68	3.72	2.93	4.88	1.0	7.57	*42*				
100	2.0	1.86	4.2	2.93	5.35	1.5	3.52	*90*				
150	2.0	1.82	4.12	3.08	5.89	1.69	4.56	*98*				
20	10.0	1.0	11.37	2.22	14.74	1.0	12.05	*21*			1.84	19.42
50	10.0	1.05	13.19	2.53	17.71	1.0	16.44	*52*				
100	10.0	1.14	15.55	2.66	19.89	1.0	24.0	*65*				
150	10.0	1.3	16.78	2.75	20.69	1.0	24.97	*51*			2.11	23.1
20	.5	3.34	1.7	3.17	3.97	2.74	3.69	*98*				
50	.5	3.67	1.93	3.66	4.17	2.74	4.38	*93*				
100	.5	4.92	2.57	3.84	7.26	3.64	4.3	*100*				
150	.5	4.4	2.27	3.71	8.16	3.35	5.51	*97*				
20	1.0	3.36	3.45	3.10	5.14	1.85	7.00	*80*	1.0	38.92	2.13	4.42
50	1.0	3.77	3.88	3.48	5.6	2.66	5.02	*88*				
100	1.0	4.5	4.54	3.67	5.71	2.28	8.43	*95*				
150	1.0	4.34	4.5	3.54	14.26	2.42	8.29	*92*	5.0	37.38	2.9	10.35
20	2.0	2.36	5.77	3.42	8.08	1.91	9.65	*96*				
50	2.0	3.15	6.88	3.33	9.26	2.72	8.75	*98*				
100	2.0	3.75	8.13	3.89	9.98	2.79	10.99	*99*				
150	2.0	3.56	8.53	3.68	16.28	2.47	9.89	*93*				
20	10.0	1.48	19.27	3.05	23.86	1.0	32.79	*66*			1.90	24.43
50	10.0	2.06	25.37	3.37	26.66	1.75	28.15	*92*				
100	10.0	2.59	30.45	3.75	34.46	1.64	29.26	*67*				
150	10.0	2.54	29.44	4.01	38.02	1.72	32.76	*82*			2.34	32.54

Table 1: The results of the cupcake experiments for all five domains; the top section is for $d = 0.333$, the bottom section for $d = 1.0$.

The three approaches differed only in how they selected N, with MON consulting its parametric equation $N = ct + b$, SIR consulting Equation 1, and the optimal approach consulting its policy. The apparatus for LTB was the two-dimensional simulated robot world, described earlier. Robots try to get as close to a point in the world as they can; bumping into it or moving past it is equivalent to running off the end of the line in the humans' video game or going beyond the goal in the simulator, above. All these conditions terminate the trial.

Trials differed in the following factors and values:

G, the initial distance to the goal. The four values of G were 20, 50, 100, 150. In the two-dimensional world, G was the length of a line between the robot's starting location and the goal object.

M, the cost of monitoring. We tried four monitoring costs: .5, 1, 2, and 10

d, the error function parameter. In pilot experiments we varied d from .2 to .5, but when we ran human subjects we discovered that these errors were too small to bother them. The current experiment is for $d = .333$ and $d = 1.0$.

The penalty function. We set the penalty to be the square of the distance between the goal and the location of the agent when it quits a trial. We tried a cubic function but it made little difference to monitoring strategies.

We did not run humans in all 32 conditions of this experiment, fearing that fatigue would affect their performance. Instead, we ran two levels of each of three factors: $M = 1, 10$, $d = .333, 1.0$, and $G = 20, 150$: Each of six subjects was tested on ten problems in each condition, for a total of 80 trials per subject. MON, SIR and the optimal policies were tested in 100 trials in each of the

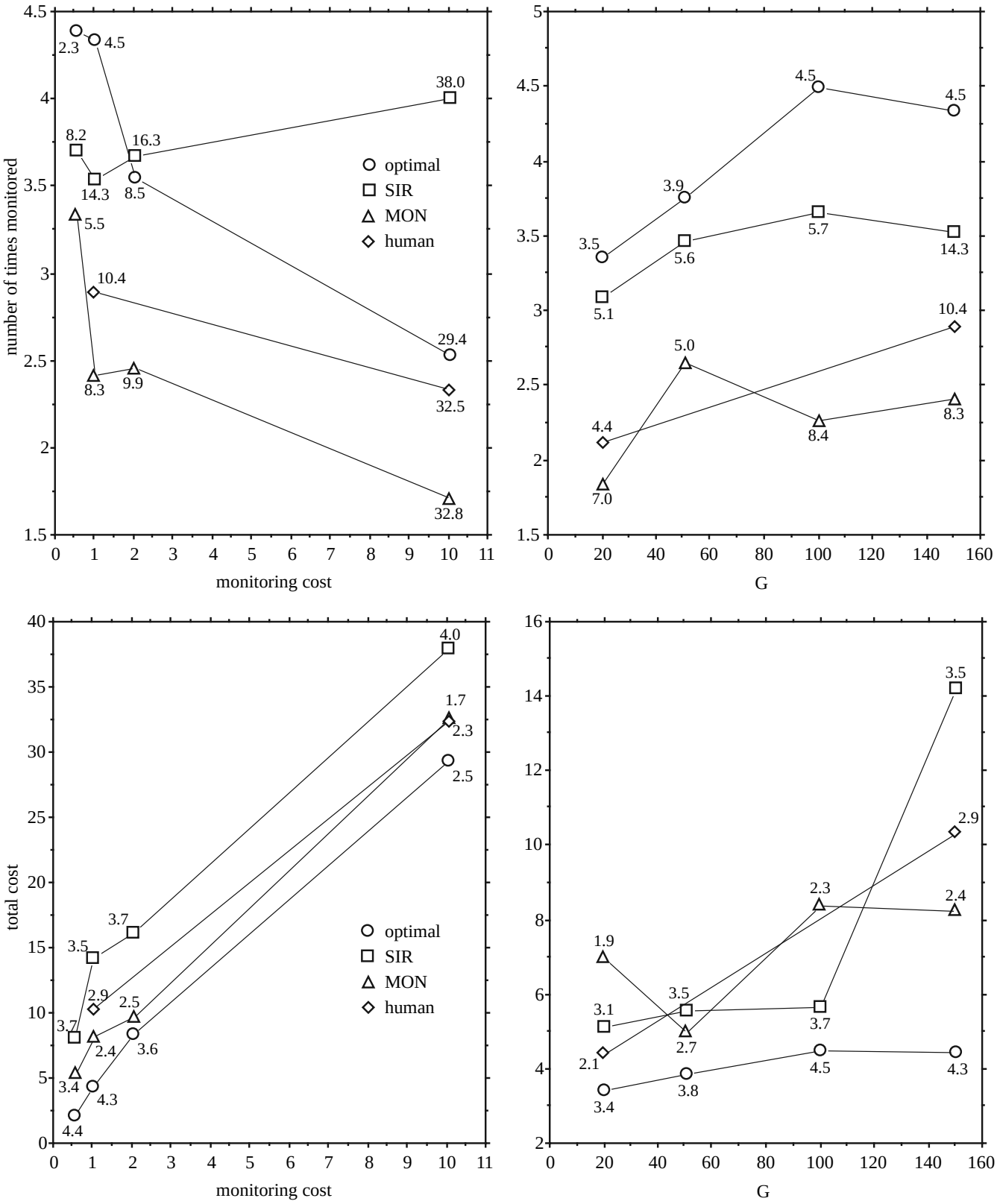

Figure 2: A selection of graphs for $d = 1.0$, illustrating data from Table 1. The graphs on the left are plotted for $G = 150$, the ones on the right for a monitoring cost of 1.0. The numbers at the nodes indicate total costs for the top graphs, and number of times monitored for the lower ones.

32 conditions. LTB was trained and tested in just four conditions. We have always had considerable difficulties getting LTB to consistently evolve monitoring strategies, even when the environment was designed in such a way that monitoring strategies were favored with higher fitness values [3].

The results of this experiment are shown in Table 1. Most data are two numbers: first, the mean number of monitoring events in a trial, and second, the mean *cost* of a trial. This cost is the number of monitoring events times the cost of monitoring, plus the penalty for falling short of the goal or overshooting it (the square of the final distance to the goal). For example, for $d = .333, G = 20, M = .5$, the optimal strategy monitors 1.94 times on average and incurrs an average penalty of .99. Data for MON include, in italics, the number of times in 100 trials that the fittest agent monitored at all during the trial. The summary statistics are based on this subset. Six trials of 448 with the human subjects were discarded because they had huge trial costs—incurred when the subjects became confused about whether a trial had ended—that skewed the means and variances.

Data from the lower half of Table 1, for $d = 1.0$, are plotted in Figure 2. The top two graphs show the average number of monitoring events plotted against G and M, respectively, and the numbers represent mean trial costs; the lower graphs show mean trial costs plotted against G and M, and the numbers represent mean monitoring incidence. Our principal observations follow:

All the agents used an interval reduction or proportional reduction strategy. The genetic algorithms, MON and LTB evolved other strategies, but they never performed as well as proportional reduction. In other words, the fitness of the proportional reduction agents was higher.

The interval reduction and proportional reduction strategies are very efficient compared to, say, periodic monitoring. Note that the optimal strategy never monitored more than five times and, averaged over all trials and conditions, it monitored 2.52 times per trial. It's easy to show that, except for very short distances, a periodic monitoring strategy will incur high penalties if it monitors as infrequently as an interval or proportional reduction strategy.

As expected, the incidence of monitoring generally decreased with M, the cost of monitoring. (The exception is SIR which doesn't take the cost of monitoring into account.) The incidence of monitoring generally increased with d, the parameter of the error function, and G, the initial distance to the goal. Note also that MON evolved a monitoring stategy least often when the cost of monitoring was high, but more often when error (D) was high. Agents monitor because they must—because error or initial distance is high.

Not surprisingly, the optimal dynamic programming strategies had the lowest trial costs in all conditions.

Individual human subjects had remarkably consistent average trial costs, yet some monitored often and others infrequently. A one-way analysis of variance showed no significant difference over subjects on mean trial cost (the means ranged from 13.62 to 18.2); but individuals differed significantly ($p < .0001$) on their mean numbers of monitoring episodes (ranging from 1.8 to 3.23 per trial). This suggests that some people kept trial costs down by monitoring relatively often and incurring small penalties, while others avoided monitoring costs and occasionally incurred large penalties.

Human trial costs were statistically indistinguishable from those of the optimal strategy when the error (d) and the monitoring cost (M) were both low. In the other six conditions, the optimal strategy outperformed humans (two-sample t tests, $p < .0001$), who tended to monitor too much when M was high and too little when M was low.

The SIR strategy paid dearly for monitoring too often. This is partly because it doesn't take M into account, partly because it treats monitoring decisions are independent. In trials with $G = 150, d = 1.0$, for example, SIR would find itself, say, six distance units shy of the goal and it would decide to move, say, four units. When the error was high it would sometimes not move at all, or might move only one unit. At this point a human will say, "I paid for a monitoring event and yet I moved no closer to my goal; I won't let that happen again. This time I will aim closer to the goal and I will just have to risk overshooting it." The optimal strategy will have compiled similar reasoning into its policy. But SIR will treat the next movement decision exactly as the previous one. We observed sequences of up to six useless or nearly useless monitoring events in a single trial.

The MON strategies don't perform well (compared with the optimal strategy) when initial distance to the goal G is low. They usually monitor just once in these conditions and they run up big penalties for overshooting the goal. It's surprising that better strategies don't evolve, given that MON is a simple parameter optimization algorithm.

The LTB strategies produce much more variable results than any others, in part because LTB robots are tested in a two-dimensional environment where they wander around instead of moving on a line toward the goal. Nevertheless, in this experiment and others, involving hundreds of trials with LTB and other genetic programming approaches to two-dimensional cupcake problems [2, 3] we have never observed a fitter strategy than interval reduction.

8 Discussion

Given the opportunity, adults and genetic algorithms will use interval reduction to solve cupcake problems. Dynamic programming tells us that interval reduction is optimal. So why is periodic monitoring, which is much less efficient, the norm?

One answer is that periodic monitoring always works, whereas interval reduction assumes that the agent monitors to find out where it is with respect to a goal. If the agent doesn't have a goal, or if the environment provides no information about it, then interval reduction won't help. Suppose your goal is to detect an event and you know the probability that it will happen in any time interval, but it happens without warning (i.e., you are monitoring a stationary Bernoulli process). If C is the cost per time unit of not detecting the event and H is the cost of monitoring for it, and p is its probability in each time unit, we can show easily that the optimal interval between monitoring is $\sqrt{2H/Cp}$. This periodic strategy is necessary because absolutely nothing is known about the event besides its probability of occurring, and nothing is learned about it by monitoring (other than whether it happened). However, if monitoring provides estimates about when or where future events will occur, then we believe periodic monitoring is inferior to interval reduction.[3]

Another answer goes like this: Periodic monitoring might be inefficient, but monitoring is free, so who cares? Monitoring is not free, of course, but in distributed architectures it can be offloaded to dedicated processors and so appears to be free. What happens when one of these processors detects something? It sends a message to other processors who must monitor their messages, or it sends an interrupt. In the first case monitoring obviously isn't free, and in the second it is bad design if the dedicated processor learns anything from its monitoring about the system's goals. Norman and Bobrow [20] show convincingly that distributed systems can avoid catastrophic failure if they adopt the *principle of continuously available output*, in effect sharing intermediate results among components. If monitoring provides information, it shouldn't be held privately by the dedicated monitoring processor, but if it is shared with other processors, they must attend to it, and monitoring is not free.

We have not found any convincing reasons to prefer periodic monitoring over interval reduction, except in situations where an agent has no goal or monitoring provides no information about progress toward the goal. Both "goal" and "progress" should be broadly construed, because the interval reduction strategy appears to be preferred in spatial and temporal domains; in one and two dimensions; when errors are due to sensor inaccuracy or movement inaccuracy; when penalty functions are symmetric or asymmetric, continuous or discrete; by agents as diverse as humans and simple, simulated robots, evolved by genetic programming.

[3] We have not been able to prove this as a general proposition, but we have proved it for special cases and we have the empirical evidence presented earlier.

Acknowledgements

This research is supported by ARPA/Rome Laboratory under contract #F30602-91-C-0076 and under an Augmentation Award for Science and Engineering Research Training. The US Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

References

[1] Atkin, M.S., 1991. Research Summary: Using a Genetic Algorithm to Monitor Cupcakes. *EKSL Memo #24*, Experimental Knowledge Systems Laboratory, University of Massachusetts, Amherst.

[2] Atkin, M. & Cohen, P.R., 1993. Genetic Programming to Learn an Agent's Monitoring Strategy. CMPSCI Technical Report 93-26. University of Massachusetts, Amherst.

[3] Atkin, M.S. & Cohen, P.R., 1994. Learning Monitoring Strategies: A Difficult Genetic Programming Application. *The First IEEE Conference on Evolutionary Computation.* Forthcoming.

[4] Ceci, S.J. & Bronfenbrenner, U., 1985. "Don't forget to take the cupcakes out of the oven": Prospective memory, strategic time-monitoring, and context. *Child Development*, Vol. 56. Pp. 152-164.

[5] Fikes, R., 1971. Monitored execution of robot plans produced by STRIPS. *Proc. IFIP Congress* 71, Ljubljana, Yugoslavia (August 23-28, 1971), pp. 189-194.

[6] Fikes, R., Hart, P. & Nilsson, N., 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3(4), pp. 251-188. Reprinted in: Allen, Hendler, and Tate (eds.), *Readings in Planning*, Morgan Kaufman, San Mateo, CA.

[7] Firby, R.J., 1987. An investigation into reactive planning in complex domains. *AAAI 87*, pp. 202-206.

[8] Georgeff, M. & Lansky, A., 1987. Reactive reasoning and planning. *AAAI 87*, pp. 677-682. Reprinted in: Allen, Hendler, and Tate (eds.), *Readings in Planning*, Morgan Kaufman, San Mateo, CA.

[9] Ghallab, M., Alami, R. & Chatila, R., 1988. Dealing with time in planning and execution monitoring. In:

Robotics Research: The Fourth International Symposium, ed. by Bolles and Roth, MIT Press, pp. 431-443.

[10] Goldberg, D.E., 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA.

[11] Goldberg, D.E. & Kalyanmoy, D., 1991. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms, in *Foundations of Genetic Algorithms* (Gregory J.E. Rawlins ed.), pp. 69-93. Morgan Kaufman, San Mateo, CA.

[12] Hanks, S., 1990. Practical temporal projection. *AAAI 90*, pp. 158-163.

[13] Hansen, E.A., 1992. Note on monitoring cupcakes. *EKSL Memo #22.* Experimental Knowledge Systems Laboratory, Computer Science Dept., University of Massachusetts, Amherst.

[14] Hansen, E.A., 1992. Learning A Decision Rule for Monitoring Tasks with Deadlines. CMPSCI Technical Report 92-80. University of Massachusetts, Amherst.

[15] Hansen, E.A., 1994. Monitoring the execution of robot plans: A survey. In preparation for *AI Magazine.*

[16] Koza, J.R., 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection and Genetics.* MIT Press, Cambridge, MA.

[17] Koza, J.R. & Rice, J.P., 1992. Automatic Programming of Robots using Genetic Programming. *AAAI 92*, Pp. 194-207

[18] McDermott, D., 1992. Robot Planning. *AI Magazine*, Summer 1992, pp. 55-79.

[19] Musliner, D., Durfee, E. & Shin, K., 1991. Execution monitoring and recovery planning with time. *Proceedings IEEE Seventh Conference on Artificial Intelligence Applications*, pp. 385-388.

[20] Norman, D.A & Bobrow, D.G. 1974. On data-limited and resource-limited processes. Xerox Technical Report CSL 74-2. Xerox Palo Alto Research Center.

[21] Schoppers, M., 1992. Representing the plan monitoring needs and resources of robotic systems. *Proceedings of the Third Annual Conf. on AI, Simulation and Planning in High Autonomy Systems*, pp. 182-187.

[22] Schoppers, M., 1992. Building plans to monitor and exploit open-loop and closed-loop dynamics. *Proceedings 1st International Conference on Artificial Intelligence Planning Systems*, ed. J. Hendler, pp. 204-213. Morgan Kaufman, San Mateo, CA.

[23] St. Amant, R., Kuwata, Y., & Cohen, P.R., 1993. Reactive Planning with Dynamic Programming Envelopes. CMPSCI Technical Report, University of Massachusetts

[24] Sutton, R.S., 1990. Integrated architectures for learning, planning, and reacting based on approximately dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning*, pp. 216-224.

[25] Tate, A., 1984. Planning and condition monitoring in a FMS. *Proceedings of the International Conference on Flexible Automation Systems.* Institute of Electrical Engineers, London, July 1984, pp. 62-69.

[26] Wilkins, D., 1985. Recovering from execution monitoring errors in SIPE. *Computational Intelligence* 1, pp. 33-45.