Monitoring Strategies for Embedded Agents: Experiments and Analysis

Marc S. Atkin and Paul R. Cohen
Experimental Knowledge Systems Laboratory
Department of Computer Science, LGRC, Box 34610
University of Massachusetts, Amherst, MA 01003
{atkin,cohen}@cs.umass.edu
(413) 545 3616

Abstract

Monitoring is an important activity for any embedded agent. To operate effectively, agents must gather information about their environment. The policy by which they do this is called a *monitoring strategy*. Our work has focussed on classifying different types of monitoring strategies, and understanding how strategies depend on features of the task and environment. We have discovered only a few general monitoring strategies, in particular *periodic* and *interval reduction*, and speculate that there are no more. The relative advantages and generality of each strategy will be discussed in detail. The wide applicability of interval reduction will be demonstrated both empirically and analytically. We conclude with number of general laws that state when a strategy is most appropriate.

Contents

1 Introduction: What are Monitoring Strategies and Why are We Interested in Them?

Monitoring is ubiquitous. Although often taken for granted, monitoring is in fact a necessary task for any agent, be it a human, a bumblebee or an AI planner. In any kind of realistic and therefore non-deterministic domain, an embedded agent must query its environment. If we are to understand how to design agents and discover general laws of agent behavior, we must understand the nature of monitoring. This paper presents our work on discovering and categorizing monitoring strategies.

A *monitoring strategy* is the scheme by which monitoring actions are scheduled. If monitoring were free—if it could be performed any time without any sort of penalty for the agent—the task of specifying the optimal monitoring strategy would be very easy: Monitor as frequently as sensors allow. In reality, however, monitoring will nearly always have some kind of cost, even if it is very small. It might be the time required to activate the sensor, the amount of energy that sensing requires, or the cost of processing sensed information. Deciding when the cost of not knowing the exact state of the environment outweighs the cost of monitoring can be a complicated optimization problem. Three factors make it difficult: First, the interval between monitoring actions will depend on features of the environment and costs, some of which are not known or only known probabilistically. Second, these features can be dynamic, changing over the course of a trial. Third, and perhaps most importantly, monitoring actions are not necessarily independent. The optimal placement of a monitoring action cannot always be computed without knowing the expected placement of all successive ones (Hansen, 1992b).

Monitoring, like any other behavior, depends on the interaction of three factors: the agent's architecture, task, and environment. The architecture describes how the agent is constructed, the task is what the agent is trying to do, and the environment is the set of external factors that act upon the agent (Cohen, Howe & Hart, 1990; Cohen, 1990). Our research goal is to discover and categorize monitoring strategies. Ultimately, we hope to lay down a taxonomy of monitoring strategies based on general features of agent architectures, tasks, and environments. Then, an agent designer would need only determine the relevant features of the agent's proposed architecture and the task and environment in which it is to operate, and look up appropriate monitoring strategies.

A distinction is frequently made between *sensing* and *monitoring* (e.g., Firby, 1987; Georgeff & Lansky, 1987; McDermott, 1978). Sensing refers to the typically low-level data acquisition mechanisms needed to keep a world model up-to-date; monitoring involves querying this world model. Monitoring, as we will use the term, means deciding when to ask for the information that sensors (or sensory subsystems) provide. Monitoring strategies will be indexed in a taxonomy by general architectural features, for example "the agent has no completely accurate way of keeping track of its progress towards the goal it is monitoring for." Through abstractions like these, we will be able to select appropriate strategies regardless of implementation.

We have explored the space of monitoring strategies empirically and analytically. In a series of exploratory experiments, we used a genetic algorithm to evolve appropriate monitoring strategies. We identified two very general strategies, *periodic monitoring* and *interval reduction*. We ran experiments with people and artificial agents to demonstrate that interval reduction always emerges in a class of tasks called *cupcake problems*. Analytically we have shown that periodic monitoring is optimal if the probability of an event occurring

is unknown, whereas interval reduction is asymptotically superior to periodic monitoring in cupcake problems.

2 Using Genetic Programming to Discover Monitoring Strategies

Generating monitoring strategies for a large number of tasks and environments requires a weak yet robust search method. The need to minimize designer bias while maintaining an automatic and powerful monitoring strategy discovery mechanism, led us to a technique known as *Genetic Programming*, the evolution of computer programs by genetic algorithms. We hoped that this method would find many different strategies for the different scenarios. Before discussing these efforts, we will describe how monitoring strategies are represented, and how the genetic algorithm manipulated them.

2.1 Behavior Representation

Monitoring is always a means to an end. As such the most general representation of a monitoring strategy will also encompass the task the agent is trying to accomplish—it will be a representation of the agent's behavior. We used a simple programming language, with loops and conditionals, as the behavior representation. Such a program is simultaneously the *operational* and *descriptive* manifestation of a monitoring strategy. This is important, because to construct a taxonomy of monitoring strategies, it will be necessary not only to observe what a strategy does, but to understand how it works.

2.1.1 A Simple Robot Control Language

For most of our experiments, our testbed was a mobile robot simulator in a two-dimensional world. The robot has basic abilities like turning and moving, and a set of sensors through which it can gather information about its environment. The robot's environment is divided into 800 square *fields*. Each has a unique terrain attribute, typically "grass" (no hindrance to movement) or "obstacle" (cannot be traversed). Even though the terrain was discretized, the robot moves continuously within the map.

Constructs in the behavior language are divided into two categories: those that actually trigger an action by the robot, and those that steer the control flow within the program. Examples of the former category are the MOVE command, which moves the robot forward by a small distance in the direction it's currently facing, or the MONITOR command, which makes the robot poll a specified sensor. Examples of the latter category are LOOP commands and conditionals. A summary of the behavior language is given in table 1.

The robot is equipped with sensors that depend on the *scenario* (the task and environment). A sensor can be as complex as the experimenter deems necessary. It can be defined to always have an up-to-date value or to require an explicit MONITOR action to have its value updated. Sensors, when polled, return a value between 0 and 255, which is stored in a special kind of variable called *environmental variable* or EVA. All knowledge the agent has of its environment is expressed in terms of EVA's. There is one EVA associated with each sensor. All EVA's are initially set to zero. Some program constructs perform actions contingent on variable values (e.g., the "LOOP *(var)+1*" and "COMPARE *const, var*"

Effector Commands	
MOVE	move forward $\frac{2}{10}$ of a field
MOVEQUICK	move forward $\frac{4}{10}$ of a field
TURNLEFT	rotate left 22.5 degrees
TURNRIGHT	rotate right 22.5 degrees
TURNTOGOAL	rotate robot in direction of goal
MONITOR *sensor*	poll specified sensor
WAIT	do nothing for one time unit
SHOOT	destroy first obstacle ahead of robot (range: 4 fields)
STOP	halt robot; end program execution

Program Control Commands	
NOP	do nothing
ENABLE *sensor*	enable interrupt handler corresponding to the specified sensor
DISABLE *sensor*	disable interrupt handler corresponding to the specified sensor
LOOP *constant*	loop a *constant* number of times
LOOP *(var)+1*	loop a variable *var* number of times
COMPARE *const, var*	compare the value of a constant to that of a variable; set flags
IF *flag* THEN ... ELSE	conditionally execute code based on state of flag *flag* (equal, unequal, larger, or smaller)
EXIT	exit from last loop

Table 1: Commands of the Robot Control Language

statements in table 1).

We envisioned putting our robotic agent into environments where it would have to react quickly to sensory events while performing another task that required monitoring. Say it has to search for food, but must interrupt this task and run away if it sees a predator. In a conventional programming language, such a behavior could be achieved only by repeatedly checking for the second event ("Do I see a predator?") while executing the first task ("Where's food?"), and, if the second event occurred, a piece of code, presumably a sub-program of some sort, would execute the "running away" behavior. The constructs necessary to implement this chain of events would be a MONITOR command to update the agent's information on the whereabouts of the predator, followed by an IF-statement which would call the "run" sub-program if necessary. This block of "monitor" and "if-statement" would have to be reproduced each time the monitoring strategy for the predator required an information update.

Because this block would seem to be quite common, we decided to augment the robot's architecture with a mechanism for responding directly and easily to external events. Associated with each sensor is a piece of code called an *interrupt handler*. When an EVA-value changes, normal program flow is interrupted, and the corresponding interrupt handler is executed in the manner of a sub-program. Interrupts can be disabled or enabled via explicit commands, making it possible for the robot to prioritize handler execution. At the start of a program's execution, all interrupts are disabled.

```
Main program:
 TURNTOGOAL
 ENABLE: reached_goal
 NOP
 NOP
 ENABLE: object_distance
 LOOP infinitely:
   LOOP 7 time(s):
     MOVE
     MOVE
   ENABLE: hit_object
   TURNTOGOAL
   MONITOR: object_distance
*reached_goal* interrupt handler:
 STOP
 STOP
 NOP
*hit_object* interrupt handler:
 WAIT
 STOP
 NOP
*object_distance* interrupt handler:
 TURNRIGHT
 MONITOR: object_distance
```

Figure 1: An example for a program that does obstacle avoidance.

2.1.2 An Illustrative Program

Interrupt handlers and loops are illustrated in Figure 1, which is the output of one of our genetic algorithm systems (called linear LTB). It is the best strategy found for a specific obstacle avoidance task. Figure 2 shows two execution traces for this scenario, which consists of a large rectangular obstacle in the center of the map, the robot's task being to move from a start point to a goal point without hitting the obstacle. The robot had three sensors, one that told it whether it had reached the goal, one that told it whether it had hit an obstacle, and one that could detect an obstacle ahead. The first two were provided so that the robot could complete its task; their values were automatically updated. The third sensor, the "sonar", was actively queried with a MONITOR command to update its value. The sonar was the only sensor that could be used to prevent a collision with an obstacle, so it was the one we wanted the monitoring strategy to focus its attention on.

The structure of the program is quite simple: After enabling the interrupt handlers corresponding to the "reached goal" and "object distance" sensors, and turning itself toward the goal with TURNTOGOAL (the goal position is assumed to be known), the program goes into an infinite loop. Within this loop, it periodically moves 14 times, which corresponds to 2.8 field widths, before monitoring for an obstacle ahead (via "MONITOR: object_distance", which sets the object_distance sensor to be the distance to the obstacle—or zero, if there is none). Since the "object distance" interrupt handler had been previously enabled, when an obstacle is detected, this interrupt handler will be called. In it, the robot turns right by 22.5 degrees (TURNRIGHT), and then monitors for the obstacle again. If the obstacle is still visible, the interrupt handler will be called recursively, turning the robot further. This

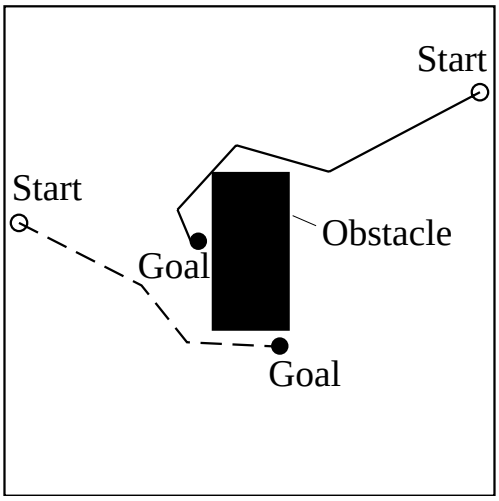

Figure 2: An execution trace generated by the program in figure 2 for two of the ten start-goal pairs. Hollow circles denote starting fields, full circles goal fields.

will continue until the obstacle is no longer visible, then the agent will move forward for 2.8 fields in its current direction before turning itself towards the goal again and rechecking for the presence of an obstacle. When the goal point is reached, the "reached goal" handler will be invoked, which ends the trial with the STOP command. Although the "hit object" handler is enabled, it never gets called. In figure 2, the changes in direction in the path of the robot indicate the points where monitoring took place and the obstacle was detected.

2.2 The Genetic Algorithm

A genetic algorithm is a machine learning scheme based on principles of natural evolution. A *population* of potential solutions (*individuals*) is maintained. At each generation, each individual is evaluated and assigned a *fitness value*, higher being better. This value determines the individual's chances of being selected into the next generation, because better solutions have a higher probability of surviving. New individuals are formed either by changing a part of the individual randomly (the so-called *mutation* operation) or by swapping a section of one individual with a section from another (the *crossing-over* operation) (Goldberg, 1989).

In our work, individuals are programs that control the robot, and it is these programs that the genetic algorithm is trying to improve. In the previous section, we saw what these programs look like and how they represent monitoring strategies. Now, we will use the linear LTB system to illustrate how the genetic algorithm works. For linear LTB, the population size was typically in the range of 500-1000 programs of the form we saw in section 2.1.2. Although each program is randomly initialized at the beginning of the algorithm, some constraints are placed on how commands are interpreted during program execution. For example, if a byte represents a sensor number, and the agent has three sensors, then the value of the byte will be taken modulo three, guaranteeing that it produces a legal value. Consequently, even at the beginning of the algorithm, some programs in the population will be able to do *something*, albeit poorly.

Figure 3 shows the major steps in one run of the algorithm. After initialization, the genetic algorithm goes into a repeated cycle of phases called *evaluation* (step 2.1) and

1. Randomly initialize population.
2. Do until fitness of best individual in population has not improved in 200 generations:
   - 2.1 Compute average fitness of every individual in population by measuring its performance in the simulator on several test cases.
   - 2.2 Copy top 10% of old population into new population.
   - 2.3 While there is still room in the new population:
     - 2.3.1 Randomly select two individuals from the old population. Call the better of them *father*.
     - 2.3.2 Randomly select two individuals from the old population. Call the better of them *mother*.
     - 2.3.3 Copy *father* and *mother* into the new population.
     - 2.3.4 With a low probability, cross-over two random pieces of code between *mother* and *father*.
     - 2.3.5 With a low probability, mutate a command in *mother* and/or *father*.
3. Output best individual in last generation

Figure 3: The basic algorithm of LTB

reproduction (steps 2.2 and 2.3). The combination of these two phases is called a *generation*. Within one generation, a modified population is created from the old one.

Individuals are evaluated by running them in the simulator on different test cases (for example, ten different start-goal points), and computing the average fitness over these trials. The three major components of the fitness measure are a quadratic reward for getting close to the goal, a cost for each effector action taken, and a cost for every time an obstacle was touched. In the reproduction phase, individuals are selected based on their fitness values. As the selection scheme, we chose tournament selection with a tournament size of two over roulette wheel selection, because of it has been shown to be less likely to cause premature convergence (Goldberg & Kalyanmoy, 1991). Tournament selection selects individuals based on their *rank* in the population. Fitter individuals have higher chances of being selected, and as the best in the population can expect to be selected several times, they multiply. This is an important feature of genetic algorithms, which operate under the assumption that combining good solutions randomly will sometimes result in even better solutions, so good solutions should be given more of chance to combine with others.

Selected individuals are sometimes changed by crossing-over and mutation. Crossing-over takes two random pieces of code of equal length from two individuals (the "parents") and swaps them (see figure 4). The mutation operator randomizes one command in the program, changing it into another. The *crossing over rate* is typically set so that roughly 1 in every 5 individuals is crossed over. Mutation changes roughly every 125th byte. Later versions of linear LTB dynamically changed mutation and crossing over rates in order to counter premature convergence.

Step 2.2 in figure 3 copies the best of the current population into the next without change. This procedure is called "elitism", and it is done so that a good solution already present in the population can never be destroyed accidentally. To alleviate the impact of the genetic algorithm's potentially sub-optimal output, we generally ran LTB ten times (or more) in each scenario. The best output of all these runs (called *iterations*) was taken as

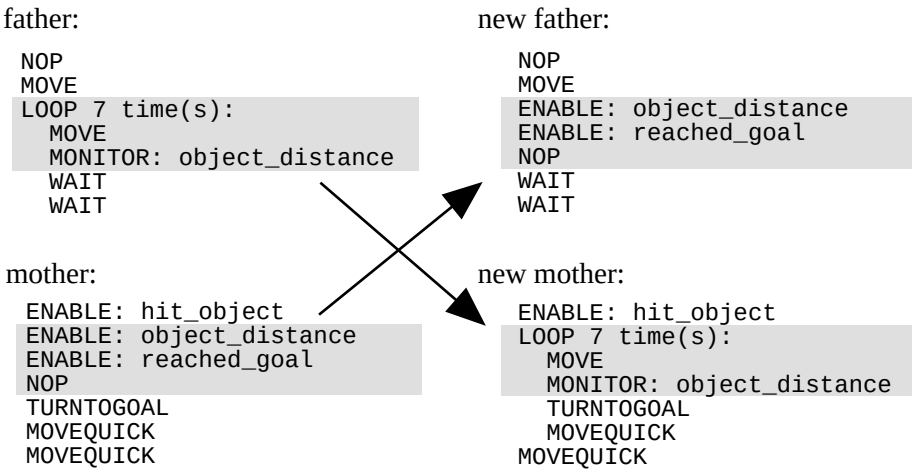

Figure 4: An illustration of crossover in linear LTB

the final output of the system for a given scenario.

3 Exploring the Space of Monitoring Strategies

We had hoped that our genetic programming systems would provide the monitoring strategies we needed for our taxonomy. This section describes the experiments we did to generate strategies, and the conclusions we drew from them.

3.1 Periodic vs. Proportional Reduction

3.1.1 Motivation

The first strategies we looked at, periodic and proportional reduction, were motivated by work found in the child development literature. Ceci and Bronfenbrenner (1985) conducted an experiment with children from two different age groups. They instructed the children to take cupcakes out of an oven in 30 minutes, and to spend the time until then playing a video game. Behind the child was a wall clock, which the child could check, but checking it (the act of monitoring) was a distraction from the game, and therefore had an associated cost. In the first few minutes of a trial, all children checked the clock frequently. Ceci and Bronfenbrenner interpreted this as a "calibration" phase for the children's internal clock. Later, however, the ten-year-olds monitored approximately periodically, whereas the fourteen-year-olds monitored more frequently as the deadline approached.

We have termed this second strategy *interval reduction*, since the interval between monitor events is reduced after each event. When the interval is reduced for a fixed proportion at each event, we call the strategy *proportional reduction*. We borrow the term "cupcake problem" to denote any scenario in which an agent must monitor for a deadline and in which the possible error between where the agent thinks it is and where it actually is grows with the interval between monitoring events. In Ceci and Bronfenbrenner's study, the error was due to humans' inaccurate internal clocks. However, the same strategy arises if one assumes the internal clock is accurate, but the information given to the agent is not.

It is interesting to note that both periodic and interval reduction strategies have long been known to behaviorial psychologists, although not under those names. There has been a lot of work generated on how behavior is affected by different reinforcement schedules (see, e.g., Schwartz, 1984, pp. 279-298; Mackintosh, 1974, pp. 164-182; a summary of the original work by Skinner can be found in Ferster & Skinner, 1957). In the *fixed interval* (FI) schedule, the subject receives reinforcement for its first response occurring at more than T time units since its previous reinforcement. The subject will typically wait a certain fraction of the time T before responding again, and then respond more and more frequently as T approaches. This is an interval reduction strategy, the so-called *fixed interval scallop*.

The FI schedule is analogous to the cupcake problem if one treats T as the deadline, and the subject's response as a monitoring action for the presence of the reward. Due to the subject's inaccurate internal clock, it cannot wait exactly T time units, which would be the optimal strategy. What distinguishes the cupcake scenario from the FI schedule is that in the latter, monitoring provides only binary information as to whether T has passed or not—it does not provide any information about the time remaining until T. As we shall see in Section 4, the optimal monitoring strategy for the cupcake problem involves a gradual decrease in the time between monitoring events. This is not, however, what is observed in response curves for the FI schedule. Typically, subjects start responding at a relatively high rate immediately after the first response. In fact, there was an extensive debate as to whether the response curves were actually scalloped (Lowe, Harzem & Bagshaw, 1978). Some believed that a period of no response, followed by a period of a high rate of response (*break-and-run* behavior) was a more accurate characterization (e.g., Schneider, 1969).

The break-and-run strategy could be described as waiting until you think you are fairly close to the deadline, and then monitoring periodically. It seems like an intuitive strategy given an inaccurate internal clock and only binary information. Again, we have one scenario, the FI reinforcement task, and two distinct strategies, periodic monitoring after a pause and interval reduction. An experiment performed by Lowe, Harzem, and Bagshaw (1978) highlights this distinction. He had two groups of human subjects. Each had access to two panels: Panel A provided reinforcement (points that could be converted to money), Panel B signaled whether reinforcement was available from Panel A. For those subjects in condition 1, Panel B implemented a binary clock; when pressed, it would indicate the availability of reinforcement. In condition 2, pressing Panel B briefly illuminated a digital clock which showed time elapsed since the last reinforcement. Lowe found that all subjects in condition 1 used a break-and-run strategy, and that the response rate was relatively independent of T. Those in condition 2, on the other hand, produced the classical scalloped response curve: a gradual decline of interresponse intervals as T approached.

Lowe et al.'s and Ceci and Bronfenbrenner's experiments give rise to the following questions: Given the specifications of a scenario, how does one choose an appropriate strategy? What are the tradeoffs between different strategies, such as periodic monitoring and interval reduction? When does one strategy outperform the other? We chose to investigate these questions in the cupcake scenario. The experiment described in the remainder of this section was designed to determine the conditions under which proportional reduction, as opposed to periodic monitoring, is advantageous. We developed an environment within linear LTB in which we could vary parameters and record how the evolved monitoring strategy was affected.

3.1.2 The Experimental Scenario

LTB's spatial world had to be modified so that it represented an instance of the cupcake problem. The robot is given the task of getting as close to a particular position in the map as possible, the *goal point*, without hitting it. The fitness function reflects this by imposing a penalty that is a quadratic function of the distance between the agent's final position on the map and the goal point. However, actually stepping on to the goal field is penalized highly, with a large constant fitness penalty (this was implemented by placing an obstacle on the goal field). Since overshooting the goal is treated differently than undershooting it, this problem is in fact an example of an *asymmetric* cupcake problem.

```
Main program:
 NOP
 NOP
 TURNLEFT
 NOP
 LOOP 4 time(s):
   LOOP 1 time(s):
     TURNTOGOAL
     NOP
     NOP
     NOP
     MONITOR: object_distance
     NOP
     DISABLE: reached_goal
     NOP
     NOP
     MOVE
     IF (object_distance) <= .5 THEN STOP
     NOP
     LOOP (object_distance)+1 times:
       LOOP 4 time(s):
         NOP
         MOVE
*reached_goal* interrupt handler:
*hit_object* interrupt handler:
 IF (object_distance) <= 4.6 THEN STOP
 TURNLEFT
 NOP
 NOP
*object_distance* interrupt handler:
 DISABLE: hit_object
 MOVE
 NOP
 WAIT
```

Figure 5: A proportional reduction strategy generated by linear LTB

The robot has an accurate movement effector but an inaccurate distance sensor. Movement corresponds to waiting in the original cupcake problem. The error is modeled by adding a gaussian noise term to the value returned by the "object_distance" sensor, the "sonar" that gives the agent the distance to an obstacle in front of it. Since there is an obstacle on the goal field (and nowhere else), this sensor now returns the distance to the goal. The variance of the noise is proportional to the distance remaining to the goal point (this models the fact that errors in the sensor will accumulate with the size of the distance measured) and a noise parameter, m, that can be varied to change the degree of inaccuracy

in the sensor.

An illustrative monitoring strategy evolved by LTB is given in figure 5. The program implements a slight variation on the proportional reduction strategy. Within the program's outer loop, "LOOP 4 times", the "MONITOR: object_distance" command is executed repeatedly. Nested within this loop is a second one, "LOOP (object_distance)+1 times", which takes the current distance to the goal (stored in the EVA "object_distance" after monitoring), rounds it up, and loops over its value. Four MOVE's are executed in this loop, each moving .1 distance units[1]. Therefore, the proportionality constant is .4: the robot will move .4 times its estimate of the distance remaining before monitoring again. The command "IF (object_distance) $\leq$ .5 THEN STOP" terminates the program when the obstacle has reached a critical distance, the TURNTOGOAL points the robot towards the goal. Note that this program is not pure proportional reduction, because there is one MOVE statement in the main loop that gets executed independently of the monitored distance.

Interrupts are not explicitly enabled in this program, so the interrupt handlers are never called. In fact, only the object_distance handler could contribute any functionality in this scenario. For instance, it might test whether the robot is close enough to the goal instead of doing so in the main loop. Occasionally, programs do actually use the object_distance handler in this way. This would mean that the distance is checked immediately after monitoring, since a change in the distance monitored would cause the interrupt handler to be called.

3.1.3 Results

We suspected that several features of the environment and the robot would influence the monitoring strategy. In particular, the amount of sonar noise should directly affect the *proportional reduction constant*, denoted *prc*, the proportion of the robot's estimated distance to the goal that the robot actually moves before monitoring again. A lot of noise should result in a more cautious program that moves shorter distances between monitoring. We hypothesized that when sonar data are extremely noisy, the proportional reduction strategy would show no advantage over periodic monitoring, as the data no longer contain any useful information.

Intuitively, proportional reduction should show more of an advantage as the distance between the robot's starting position and the goal grows, since the proportional reduction strategy can traverse a given distance in a logarithmic number of monitoring steps, whereas periodic monitoring needs a linear number.

To summarize our hypotheses:

1. If the monitoring error, m, is 0, a robot should monitor once only.
2. As m increases, the agent should be forced to take more cautious steps, i.e. the proportionality constant *prc* should decrease.
3. Periodic monitoring becomes more likely as m increases (sonar data contains increasingly less useful information) and path length decreases.

We ran the genetic algorithm on five levels of monitoring error m (0, 0.5, 1.0, 2.0, 3.0)

[1] A field was considered one distance unit. Different versions of LTB had different movement distances for the MOVE command. The information given in table 1 only applies to the most recent version of the system.

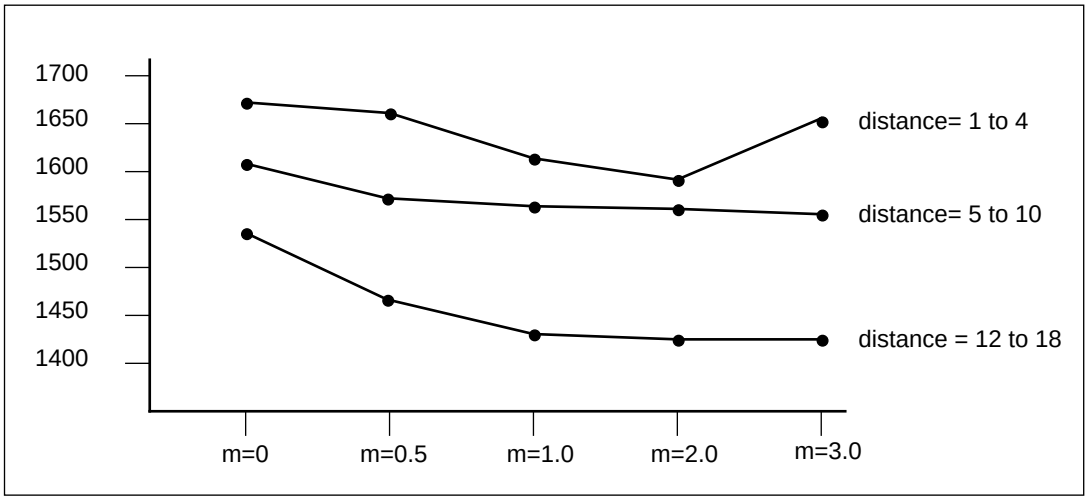

Figure 6: A plot of cell means for a two-way ANOVA, monitoring error m by distance. The dependent measure is the *fitness* of the best individual.

and three ranges of path length D (1-4, 5-10, and 12-18 fields)[2]. Each agent was trained on 12 start-goal pairs, and LTB was run ten times in each scenario. The population size was 800. The best overall program for a scenario was re-tested on 36 start-goal pairs in each specified length range. Four measures were averaged over the 36 trials to evaluate the program's performance and strategy: fitness, total cost of monitoring, percentage of distance traversed while using proportional reduction, and the proportional reduction constant *prc*. The first three measures were calculated by an evaluation function directly from the program's execution trace; the fourth was determined manually by looking at the program. Note that the total cost of monitoring tells us how many times the agent monitored, as monitoring has a fixed cost.

Figure 6 shows the average fitness values for the final program on the 36 test pairs. For the most part, fitness values decrease monotonically as m and path length increase, with the exception of the $m = 3.0, D = 1-4$ situation. Both these effects were predicted: Longer path lengths mean higher energy consumption, which lowers fitness, and higher values of m mean more elaborate and energy-consuming monitoring strategies. Note however that path length has a much greater effect than monitoring error. Apparently the genetic algorithm copes quite well with finding good monitoring strategies. An analysis of variance (ANOVA) shows a clear effect of both path length and monitoring error on fitness (for both main effects $p < 0.0001$), but no interaction effect. The exception in the monotonic decrease of fitness is actually due to the values for the m=1.0 and m=2.0 (range 1-4) cases being too low. In these cases, the robot hit the obstacle once or twice on the 36 test pairs, whereas it didn't on the 12 training pairs. Obviously the solution to this problem is to train on more pairs, but it is a trade-off between better results and run time of the algorithm.

We had expected to see monitoring cost increase as path length and especially monitoring error increases. This prediction was only partly verified. The expected pattern is distorted because when the monitoring error gets too high, the genetic algorithm does not come up with a monitoring strategy at all, but instead tries to find a different solution

[2]The path length must be a range, otherwise the genetic algorithm will simply adapt to move exactly the required distance, without monitoring at all

length	$m = 0.0$	$m = 0.5$	$m = 1.0$	$m = 2.0$	$m = 3.0$
1-4	PR^2 (**)	PR (.7)	PR^2 (*) (**)	PR (.5) (**)	check (*) (**)
5-10	PR^2	DPR	PR (.4)	– (*)	– (*)
12-18	DPR	PR (.7)	DPR (*) (**)	DPR	DPR

Key:
PR (c): proportional reduction strategy with proportional reduction constant
PR^2: moves a proportion of the squared distance remaining
DPR: disproportionate reduction: moves a proportion of distance remaining plus a constant distance.
check: stops if obstacle is visible after monitoring
–: no monitoring strategy
(*) deliberately moves past goal to one side
(**) strategy is only capable of monitoring once

Table 2: The best individual's monitoring strategy

such as deliberately moving past the goal point (to the left or right) for a distance that is approximately the average of the path lengths of its training set. Table 2 gives an overview of the best individual's monitoring strategy for each test case. As one can see, in cases $m = 3.0$, $D = 5 - 10$; and $m = 2.0$, $D = 5 - 10$, no monitoring strategy is found. Sometimes (case $m = 1.0$, $D = 1 - 4$ and $D = 12 - 18$), the agent will combine the strategy of moving past the obstacle with proportional reduction, ensuring that if proportional reduction fails, it will still not touch the obstacle.

Interestingly, no program monitors more than once when the path length is very small. We had expected the "monitor only once" strategy only in the absence of sensor noise, but apparently, over such short distances, the extra effort involved in coming extremely close to the goal is not worth the energy necessary to achieve it. Another surprise is that for longer path lengths and no sensor error, programs still monitor more than once (refuting our first hypothesis). It is hard to imagine how this could be advantageous.

Some form of proportional reduction was found in nearly all situations. In fact, in all but one of the cases where a monitoring strategy was found, proportional reduction was used. Our hypothesis that periodic monitoring would become dominant for high values of monitoring error was not confirmed. It is important to mention, however, that periodic monitoring was indeed discovered by the algorithm in several high-error situations, but its fitness was slightly lower than that of proportional reduction. The one clear exception is the $m = 3.0$, $D = 1 - 4$ case. Here, the robot moves a certain distance, then monitors for the obstacle. If the obstacle is visible ahead, it stops; otherwise it moves a fixed distance further, curving around the obstacle. This is notable because it is not a proportional reduction strategy, the sensor data are too inaccurate; the sensor is used only to detect the presence or absence of the obstacle.

The proportional reduction strategy does not usually manifest itself in its pure form. Frequently, the robot will move a short distance before beginning to monitor. This is not surprising: in many trials, the robot is guaranteed a minimum distance from the goal by the lower bound on the path length. Sometimes after monitoring, in addition to moving a distance proportional to the remaining distance, the robot moves a fixed distance as well. We call this "disproportionate reduction", as the robot is in effect moving a larger proportion of the distance remaining to the goal as it gets closer to the goal. In pure proportional reduction, this proportion remains constant throughout the strategy. The

program in Figure 5 implements this behavior. Note that the fixed additional distance was usually small with respect to the distance moved via proportional reduction. It is therefore very possible that the extra MOVE commands that cause it are simply a relic of the evolutionary process: They cause little harm and are therefore not removed in later generations.

In two of the cases where the agents only monitors once, and in one of the cases where $m = 0$, the robot moves a proportion of the squared distance remaining. All these cases can be interpreted as examples of over-adaptation: the number of MOVE's in the loop can be adjusted to get the the robot close to the goal given the particular range of path lengths. We do not view this proportional reduction variant as a distinct strategy.

Our second hypothesis was that the proportionality constant *prc* should decrease as m grows. Of course this constant is only really defined in cases of unmodified proportional reduction, which limits our available data points. But the few applicable cases do show the predicted effect (see Table 2). The range of the constant decreases from .7 in two of the $m = 0.5$ cases to .4 in the second $m = 1.0$ case. The constant of .5 in the $m = 2.0$, $D = 1 - 4$ case should not be weighed too strongly as the robot only monitors once here. Even so, it is still close to the previous .4 value.

The hypothesis that periodic monitoring will surpass proportional reduction as m and path length increase appears to be refuted. Periodic monitoring never beat proportional reduction, although it did come close in the $m = 2.0$ cases. We are led to believe that periodic monitoring will only be advantageous when virtually no distance information is given. This is the main result of this experiment: Some form of interval reduction seems to be appropriate for nearly all instances of the cupcake problem.

3.2 Additional Genetic Programming Experiments

The periodic vs. proportional reduction experiment with LTB was only the first in a series of experiments to explore the space of monitoring strategies. In order to accommodate more complex environments, we equipped the robot with five additional sensors: robot direction, goal direction, speed, terrain type, and obstacle density. The environments departed from the cupcake problem as well, to emphasize more the monitoring necessary for obstacle avoidance. We varied the number and size of the obstacles, and their density and distribution. We also had a scenario that involved a new terrain type which could be used as a predictor for obstacles. In one case we changed the architecture of the robot by forcing it to move forward at every step and only giving it the ability to change direction.

Unfortunately, our genetic algorithm was seemingly incapable of evolving monitoring strategies for these more complex scenarios. Instead, it would find good solutions to the problem that did not use monitoring, often exploiting unintended features of the scenarios. Monitoring strategies only make up a small fraction of the space of possible programs, and although they usually have higher fitness values, it was extremely difficult for the genetic algorithm to find them as the search space grew larger. We now believe that genetic programming may in general have problems scaling up to large search spaces (Atkin & Cohen, 1994). The likely cause of our problems is a phenomenon known as *premature convergence*. It occurs when a good program in the population comes to dominate over the less fit members and starts to fill a large proportion of the population with its genetic material, causing the genetic diversity which drives the algorithm out of local minima to be lost.

We modified our LTB system in several ways in order to alleviate the premature convergence problem. The major change was to introduce the idea of "behavior modules". Instead of representing programs as fixed sized lists of commands, we used trees, similar to the s-expressions used by many other researchers in the genetic programming field (Koza, 1992; Koza & Rice, 1992). This representation does not force programs to be a particular length, and it allows for meaningful behavioral subunits, "modules," to be represented as sub-trees. Swapping over now makes more semantic sense, since it can swap two modules, across individuals. To our great surprise, however, the percentage of time LTB found a monitoring strategy in our original obstacle avoidance scenario (Figure 2) dropped from about 50% to 30% after we introduced the tree representation—all other factors remaining the same. So the more flexible representation seemed to actually be hurting performance.

We attempted to reduce the size of the search space by making the agent's architecture more conducive to monitoring. The robot's control mechanism was redefined so that *all* generated behaviors are monitoring strategies—the task of the genetic algorithm is to learn *when* to monitor next. This approach is very reminiscent of one of our first learning algorithms, "MON." MON was a parameter optimization algorithm, and the agent it controlled operated in a simpler environment than the two-dimensional world described in earlier sections. MON learned strategies in the *time* domain, its strategies told the robot how long to wait before monitoring again. Given a parameterized set of functions describing when to monitor next (e.g. $c\log(t) + b$), MON ran a genetic algorithm to determine the best parameter values for c and b, i.e. the ones that minimized expected cost. The cost consisted of two parts: A cost for monitoring, and a cost determined by the task. We will discuss MON's strategies a little later, in section 4.

Our final system, LMOUSE ("Learning Monitoring Using Simulated Evolution"), learned explicit functions that computed when to monitor next, in the form of parse trees with mathematical operators at the internal nodes and sensor values or constants as the leaves. Unlike MON, LMOUSE is not restricted to a particular class of functions. Also, the environment remains two-dimensional and thus more complex than MON's, and the robot still has several sensors—not just one—some of which might not pertain to the task at all. And it must learn a monitoring function for all of them. Yet, although LMOUSE was obliged to learn monitoring strategies, it, too, did not learn any interesting ones when its search space became large.

3.3 A Conjecture

Despite these efforts—five different genetic algorithm methods, nearly a dozen tasks and environments, and thousands of trials—we did not discover a rich taxonomy of varied monitoring strategies. It is interesting to catalog the strategies we were *expecting* the algorithms to produce for the scenarios listed at the beginning of section 3.2. In most cases, periodic monitoring is the appropriate strategy. In some cases the optimal period can be precomputed, whereas in others it depends on the current context. Only one scenario, which involves traversing an environment where the obstacle density increased linearly, requires a non-periodic strategy. Plotted over time, as the robot moves from west to east towards the goal, intervals between monitoring events decrease, very reminiscent of the interval reduction strategy, even though there is no clear deadline.

Two things are noteworthy: First, periodic monitoring is in fact a good strategy for many situations, and second, no strategy other than periodic or interval reduction emerged

in our experiments. In fact, all the strategies that we have come across during the years we have been working on monitoring, including those generated by our genetic algorithms, have been either periodic, interval reduction, or combinations of these two. We now believe that there might be only a very small number of general monitoring strategies, and that their apparent complexity arises from the fact that their parameters can be complex functions of environment variables. It should also be emphasized that monitoring strategies are appropriate not only for reaching spatial or temporal goals, but can be used to reach any goal, concrete or abstract, as long as the information needed by the strategy (e.g. a measure of distance) can be sensed somehow.

Periodic monitoring is a well-known and often used strategy. Interval reduction, however, is not. If our conjecture is correct, and interval reduction is one of a very few distinct monitoring strategies, then it warrants further study. The rest of this paper will look at interval reduction in more detail. In particular, we will demonstrate its general applicability empirically, and present a series of approaches to analyzing both periodic and interval reduction strategies mathematically.

4 The Generality of Interval Reduction

The interval reduction strategy showed up in Ceci and Bronfenbrenner's study, the operant conditioning literature, and the experiment in section 3.1. This section will attempt to demonstrate interval reduction's general applicability by presenting further corroborating evidence. We conducted an extensive experiment involving a wide range of agents and agent environments (Cohen, Atkin & Hansen, 1994), including adult humans. There were five types of agents overall. Two were evolved by the MON and LTB systems we saw earlier. The other three will be described in the following sections. We wanted to know whether interval reduction was the strategy of choice in all these cases. Different environments were generated by varying three parameters: the starting distance, the sensor error, and also the monitoring cost. This experiment was run on a version of the cupcake problem more closely related to Ceci and Bronfenbrenner's original study. Instead of an inaccurate distance sensor, error was modeled by having a small fixed error $\pm m$ associated with each WAIT or MOVE action. This results in nearly exactly the same cumulative error distribution as in the case of the inaccurate sensor.

4.1 SIR

Our first agent used used a decision rule based on the probability of overshooting the goal during each epoch. This is what we call the *SIR strategy* for "simple interval reduction". Let's assume when the agent plans to wait t time steps, it actually waits $w(t)$, where w is a random variable normally distributed around t with standard deviation $\sigma = \sqrt{t}m$; m is a parameter describing how inaccurate the internal clock is.[3] The probability that the agent will wait more than

$$w(t)_\alpha = t + z_{2\alpha}\sigma$$

time units is exactly α, 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

[3] One obtains this normal distribution if one assumes that on each time step of the agent's internal clock, the agent actually waits $1 \pm m$ time steps, each with equal probability.

the agent does not exceed the desired deadline, D, with greater than α probability, we set $w(t)_\alpha = D$ and solve the previous equation for t:

$$t = D + \frac{m^2 z_{2\alpha}^2}{2} - m z_{2\alpha} \sqrt{D + \frac{m^2 z_{2\alpha}^2}{4}} \tag{1}$$

This equation tells us to wait a certain variable fraction of the current distance remaining to the deadline, D, after each monitoring event. This is an interval reduction strategy.

4.2 Dynamic Programming

SIR is suboptimal if there are costs for monitoring and penalties for overshooting (or falling short of) the goal. 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* (Sutton, 1990). 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 (Hansen, 1992a; Hansen, 1992b).

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. This problem was solved by Hansen (1994), working in our lab.

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 (1992b) 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 work has also been extended to monitoring plan execution (Hansen, 1994).

Using these algorithms, the optimal strategy for a given cupcake problem has been shown to be an interval reduction strategy. It is not proportional reduction, however, since the time to wait when one still far away from the deadline is relatively larger than when one is close.

4.3 Adult Humans

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 a line. The goal is to get the ball as close as possible to the rightmost 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. When the subject clicks on the line, the ball travels the selected number of steps N of size $1 \pm m$. 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 when the ball overshoots the goal. On each trial the subject is told m, 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. We explain to the subjects that movement errors accumulate with distance, so if m 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.

4.4 Comparing Strategies

We compared humans, MON's parametric functions, optimal dynamic programming policies and SIR policies on a set of temporal (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 dynamic programming policies were tested in a simulator, in which each trial went as follows:

Loop

1. Monitor to find distance to the goal
   - If the agent is currently beyond the goal, quit; else
   - If the agent is within one unit of the goal, quit; else
2. Consult the control policy to decide N, the number of steps to move before monitoring again.
3. Take N steps of $1 \pm m$ units

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 would try to get as close to a point in the world as they can; bumping into it or moving past it was 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:

D, the initial distance to the goal. The four values of D were 20, 50, 100, 150. In the two-dimensional world, D 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

m, the error function parameter. In pilot experiments we varied m 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 $m = .333$ and $m = 1.0$.

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: $M = 1, 10$, $m = .333, 1.0$, and $D = 20, 150$: 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 32 conditions. LTB was trained and tested in just four conditions.

Some of the results of this experiment are shown in figure 7. The top two graphs show the average number of monitoring events, and the numbers represent mean trial costs; the bottom two graphs show the mean cost of a trial, here the numbers represent mean monitoring incidence. The 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). On the left, the graphs have M on the horizontal axis; on the right, D. 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—which skewed the means and variances. 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 m, the parameter of the error function, and D, the initial distance to the goal. Note also that MON evolved a monitoring strategy least often when the cost of monitoring was high, but more often when error (m) was high. Agents monitor because they must—because error or initial distance is high.

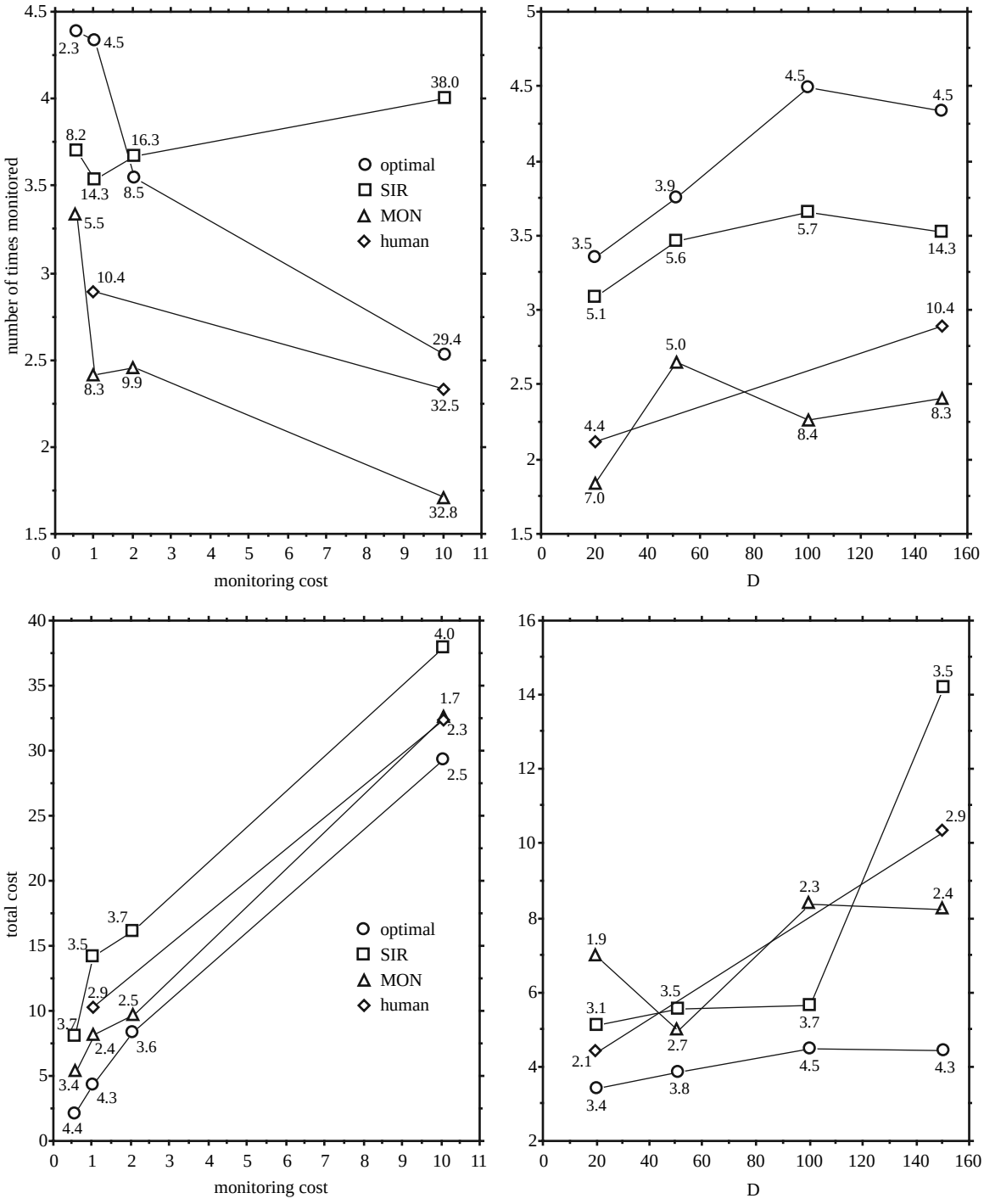

Figure 7: The results of the cupcake experiments for $m = 1.0$, in four of the five domains. The graphs on the left are plotted for $D = 150$, the ones on the right for a monitoring cost of 1.0. The numbers at the nodes indicate total costs for the top graphs, and number of times monitored for the lower ones.

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 (m) 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 $D = 150, m = 1.0$, 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 D 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 (Atkin & Cohen, 1993; Atkin & Cohen, 1994) we have never observed a fitter strategy than interval reduction.

4.5 Interval Reduction is Appropriate for Cupcake Problems

Ceci and Bronfenbrenner's study, together with the data provided in this section, gives strong cumulative evidence for the generality of interval reduction. Only when an agent has no goal, or monitoring provides no information about progress towards this goal, should periodic monitoring be chosen. Not only is interval reduction the appropriate strategy, but it is consistently learned and used by both natural and artificial agents. The class of cupcake problems is very large: both "goal" and "progress" should be broadly construed. Interval reduction is 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.

We also managed to prove mathematically that the proportional reduction strategy

(which is a special case of interval reduction) outperforms periodic monitoring for *any* cupcake problem provided the starting distance is large enough (see section 5.2). Together with all the experimental evidence presented in this section, we are encouraged to formulate a general "law of interval reduction". This law constitutes one piece of our monitoring strategy taxonomy:

> A Sufficient Condition for the Applicability of Interval Reduction:
>
> *Given a task that requires monitoring for a deadline in space or time,*
> *any environment,*
> *and a sensor that measures the distance to the deadline, but makes larger errors as the distance to the deadline grows,*
> $\rightarrow$ *use an* interval reduction *strategy.*

5 Analyzing Monitoring Strategies

In this section, we augment the empirical work presented thus far with some mathematical results. First we will look at periodic monitoring, which is more easily approached mathematically, and then at the relationship between periodic monitoring and proportional reduction.

5.1 Periodic Monitoring

5.1.1 Periodic Monitoring as a Simple Optimization Problem

Often, people are not concerned with cost of monitoring or consider it negligible, and simply monitor as frequently as the sensor will allow or wherever it happens to be convenient in the program code. But even if one *does* take costs into account, periodic monitoring is often the best strategy to use. Computing the optimal period is quite straightforward. Consider fires breaking out in a forest. Let the probability that a fire breaks out on any given day be p; a fire incurs a cost F for every day it burns undetected. Assume that checking for fires requires sending a helicopter out to fly over the forest, which costs H. What monitoring period r will minimize the combined expected cost of fire burning and monitoring? More accurately, what r will minimize the expected combined cost *per unit time*, since a trial can go on for an indefinite amount of time?

If one waits r days before monitoring the first time, the combined cost per unit time interval is

$$\begin{aligned} & \frac{1}{r}(H + F\sum_{i=1}^{r} p(r-i)) \\ = \quad & \frac{1}{r}(H + Fp\frac{r^2+r}{2}) \end{aligned} \tag{2}$$

Optimizing for r, one gets the period $\sqrt{2H/Fp}$. Since the monitoring epochs (an epoch is one instance of sending out the helicopter) are independent, this is the optimal period not just for the first epoch, but for the whole trial. The proof in the next section generalizes this example to any situation where an event has a uniform probability distribution (probability p does not change over time). The proof shows that periodic monitoring

is the optimal strategy for this scenario, and it also tells us how to compute the optimal period:

A Sufficient Condition for the Applicability of Periodic Monitoring:

Given a task that requires monitoring for an event in time or space,
and this event can occur at any time (or at any position) with equal probability,
and the agent has a sensor that can detect whether the event has occurred,
$\rightarrow$ *use a* periodic monitoring *strategy with a period t' that minimizes the equation* $\frac{C(t)}{t}$*, where C(t) is the total cost for not monitoring for t time units plus the cost for monitoring once.*

While not a formal statement, it will appeal to the reader's intuition that when the agent has no a priori information about an event's distribution, and no sensor by which to gather that information, the agent can do no better than monitor periodically. Due to its limited knowledge, the agent must in essence pretend the event can occur at any time with equal probability:

A Sufficient Condition for the Applicability of Periodic Monitoring:

Given a task that requires monitoring for an event in time or space,
no knowledge of the distribution of this event in the environment,
and a sensor that returns only binary information as to whether the event has occurred or not,
$\rightarrow$ *use a* periodic monitoring *strategy.*

5.1.2 Periodic Monitoring is Optimal if the Probability of an Event Remains Constant Over Time

Lemma: *If the probability of a process P starting is constant over a time interval T, then monitoring with a constant period t' is the optimal monitoring strategy.*

If C(t) denotes the total cost incurred in a monitoring interval of length t (a monitoring interval being the time between two monitoring instances), then t' is a t-value for which $\frac{C(t)}{t}$ is minimal.

Proof:

Define C(t) as above, i.e., C(t) contains the expected process associated cost for not monitoring for t time units plus the cost of monitoring itself.

Let D be the time P starts. D is equally distributed over T by assumption. This means that the expected value for C(t) for a fixed t is constant over the whole of T. This means, given a monitoring interval t, the expected cost C(t) for this value is constant, independent of where the interval is within T.

Thus, C(t) is a well-defined value. It is therefore possible to minimize (optimize) $\frac{C(t)}{t}$, the cost per unit time in one monitoring interval, over all values of t. Call a t with a minimal $\frac{C(t)}{t}$-value t'.

Now it remains to be shown that monitoring with a period of t' is better than any other kind of monitoring strategy.

Consider an arbitrary other monitoring strategy. Say it monitors n times, t_i being the interval length between the $(i-1)$th and ith monitoring instance.
Then $\sum_{i=1}^{n} C(t_i)$ expresses the total expected cost of monitoring for this strategy. We will show that this sum is always greater than $\frac{T}{t'}C(t')$, which is the total (expected) cost for monitoring with period t':

$$\begin{aligned}
\sum_{i=1}^{n} C(t_i) &= \sum_{i=1}^{n} t_i \frac{C(t_i)}{t_i} \\
&> \sum_{i=1}^{n} t_i \frac{C(t')}{t'}, \qquad \text{because } t' \text{ is the value for which} \frac{C(t)}{t} \text{ is minimal} \\
&= \frac{C(t')}{t'} \sum_{i=1}^{n} t_i \\
&= \frac{C(t')}{t'} T \\
&= \frac{T}{t'} C(t')
\end{aligned}$$

q.e.d.

5.2 A Proof of Proportional Reduction's Superiority over Periodic Monitoring in the Cupcake Problem

The experiment in section 3.1 gave us empirical evidence for proportional reduction's superiority over periodic monitoring in the cupcake problem. This section will supply some analytical evidence as well. Our proof is based on a cupcake problem with fixed monitoring cost (i.e., a constant cost is charged every time the agent monitors) and a quadratic distance penalty function. The agent has a sensor that returns the distance to the deadline. This sensor is completely accurate, whereas the agent's internal clock is not. When the agent plans to wait t time units (or move t distance units), it will actually wait t' units, where t' is a random variable which has a normal distribution with mean t and standard deviation $\sigma = m\sqrt{t}$. The parameter m is a constant expressing how inaccurate the internal clock is.

The proof approach is the following: Set up expected cost functions for both strategies. Find a general lower bound for the periodic strategy. Then find a proportional reduction strategy that has an upper bound *smaller* than the lower bound set by the periodic strategy. If this can be done, it shows that there exists a proportional reduction strategy that beats periodic.

In the following section, we will formulate the problem and derive upper and lower bounds on the expected cost for the periodic and proportional reduction strategies, respectively. Section 5.2.4 will then compare the two strategies based on these cost functions.

5.2.1 The Asymptotic Superiority of Proportional Reduction

Let us now define the cupcake problem and the two strategies:

Definition 1: *A* cupcake problem *is a tuple* (D, m, mon_cost)*, where* D *is the starting distance to the deadline,* m *describes the inaccuracy of the agent's internal clock,*

and mon_cost *is the cost of monitoring once. The agent's task is to minimize the cost of trial, the cost being* z·mon_cost + distance_penalty, *where* z *is the number of times the agent monitored. The* distance penalty *is quadratic in time units from the deadline when the trial ends. A trial ends when the agent decides to stop or when upon monitoring, the agent finds that it has overshot the deadline. The agent must determine how long to wait until monitoring again. For this purpose, it has an internal clock that, when given a time* t, *will wait* t', t' *being a random variable distributed normally under* $\varphi_{t,m\sqrt{t}}$[4]

The cupcake problem can also be formulated for the spatial domain. The definition is completely analogous.

Definition 2: *A* periodic strategy *is one that involves checking periodically for the deadline. If the deadline is within* period/2 *units, the agent stops. If it is not, it waits* period *units and then monitors again. The following algorithm illustrates the strategy:*

```
monitor for deadline
while I am not within period/2 of the deadline:
    wait period units
    monitor for deadline
```

The periodic strategy is characterized solely by the parameter period.

Definition 3: *A* proportional reduction strategy *involves waiting a fixed proportion of the distance remaining at each monitoring step before monitoring again. This proportion is called the* proportional reduction constant *(prc). If the agents sees that it is within* thresh *units of the deadline, it stops. The following algorithm illustrates the strategy:*

```
monitor for deadline
while I am not within thresh of the deadline:
    wait prc * distance remaining units
    monitor for deadline
```

The proportional reduction strategy is characterized by the parameters prc *and* thresh.

Definition 4: *An* optimal *strategy is one that, given a cupcake problem* (D, m, mon_cost), *minimizes the expected cost for solving this problem over all the strategy's parameters.*

This section will show that the following theorem holds:

Theorem 1: *For any given cupcake problem* (D, m, mon_cost) *with* $D \geq D_{min}(m, mon\_cost)$, *there exists a proportional reduction strategy that does better than the optimal periodic one.*

5.2.2 A Lower Bound on the Expected Cost for Periodic Monitoring

In this section, we are are trying to make a statement about the optimal periodic strategy, namely, finding a lower bound on its expected cost. It will therefore be necessary to find the value for *period* that minimizes this cost. Since we are interested in the *lower* bound, assumptions that lower the expected cost are permissible. To simplify things, we will assume the following:

[4] $\varphi_{t,\sigma}$ is the normal distribution with mean t and standard deviation σ.

Before it executes its last wait step, the agent is exactly period *units from the deadline.*

This assumption eliminates all the problems having to do with agent not being able to hit the deadline exactly because the starting distance D is not divisible by *period*. It lowers the expected distance penalty, because if the agent is any other distance than *period* away from the deadline before it waits for the last time, the mean of the agent's final position will not be D, and the expected squared distance can only increase.
The number of times we expect the agent to monitor, z, can be expressed very easily[5]:

$$z = \frac{D}{\text{period}}$$

Given the above assumptions, distance error can only arise from the fact that on the last monitoring step, the agent does not hit the deadline exactly. The expected distance penalty can be computed as follows. Note that the penalty is exactly the variance of the normal distribution.

$$\int_{-\infty}^{\infty} (x - D)^2 \varphi_{D, m\sqrt{\text{period}}}(x) = \text{period} \cdot m^2$$

So the total expected cost is:

$$\begin{aligned} \text{cost}_{\text{periodic}} &\geq z \cdot \text{mon\_cost} + \text{penalty} \\ &= \frac{D}{\text{period}} \text{mon\_cost} + \text{period} \cdot m^2 \end{aligned} \tag{3}$$

What is the minimum value of expression (3)? Taking the first derivative with respect to *period* and setting it to zero results in the following minimum (it is actually a minimum, which can be verified with the second derivative):

$$\text{period} = \frac{\sqrt{D \text{mon\_cost}}}{m}$$

Plugging this value for *period* into (3):

$$\begin{aligned} \text{cost}_{\text{periodic}} &\geq \frac{\text{mon\_cost} D m}{\sqrt{D}\sqrt{\text{mon\_cost}}} + m^2 \frac{\sqrt{D \text{mon\_cost}}}{m} \\ &= 2m\sqrt{D}\sqrt{\text{mon\_cost}} \end{aligned} \tag{4}$$

This is a lower bound on the total expected cost for the optimal periodic monitoring strategy applied to a cupcake problem $(D, m, \text{mon\_cost})$.

5.2.3 An Upper Bound on the Expected Cost for Proportional Reduction

Calculating the expected cost for proportional reduction is not as straightforward. Since we are attempting to prove that proportional reduction is better than periodic, in this section we will be interested in determining a *upper bound* on the expected cost.

[5] z is actually only an *estimator* for the number of times we expect to monitor in the periodic strategy. Since we are dealing with normal distributions, which are additive, it seems likely that $D/period$ is actually the average number of times monitored. However, a rigorous proof of this assumption is not included here.

If we denote the proportional reduction constant with *prc* ($0 < prc < 1$), and we use *thresh* to denote the distance from the deadline within which the agent will end the trial, the number of times the agent will expect to monitor can be expressed by the following equation[6]:

$$z = \log_{\frac{1}{1-\mathrm{prc}}} \frac{D}{\mathrm{thresh}}$$

On each "monitor and wait" sequence (*epoch*), the agent could potentially overshoot the deadline. The probability of doing so will depend on *prc* (lower values reduce the chances of overshooting) and m (higher values increase the chance). But deriving the overshoot probability is not enough, we must also determine the expected penalty that is incurred. The following equation expresses the expected penalty on epoch i ($0 \leq i < z$). Denote by $D_i = D(1 - \mathrm{prc})^i$ the average distance remaining on epoch i.

$$\mathrm{distance\_penalty}_{\mathrm{epoch}} = \int_{D_i}^{\infty} \varphi_{\mathrm{prc}\cdot D_i,\sigma}(x)(x - D_i)^2 dx, \tag{5}$$

where $\sigma = m\sqrt{\mathrm{prc} \cdot D_i}$. The integral of the normal distribution cannot in general be expressed in closed form. But as long as the x-value is greater than 1, one can use e^{-x} as an over-estimator for the expression e^{-x^2} in the normal distribution function. So using this approximation, equation (5) becomes

$$(5) \leq \int_{D_i}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{x-\mathrm{prc}D_i}{\sqrt{2}\sigma}} (x - D_i)^2 dx \qquad (\text{under the constraint} \frac{D_i - \mathrm{prc}D_i}{\sqrt{2}\sigma} \geq 1)$$

Substituting $y = \frac{x-\mathrm{prc}D_i}{\sqrt{2}\sigma}; x = y\sqrt{2}\sigma + \mathrm{prc}D_i; \frac{dx}{dy} = \sqrt{2}\sigma$:

$$\begin{aligned} &= \int_{\frac{D_i-\mathrm{prc}D_i}{\sqrt{2}\sigma}}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-y}((\sqrt{2}\sigma y + \mathrm{prc}D_i) - D_i)^2 \cdot \sqrt{2}\sigma dy \\ &= \int_{\frac{D_i(1-\mathrm{prc})}{\sqrt{2}\sigma}}^{\infty} \frac{1}{\sqrt{\pi}} e^{-y}(\sqrt{2}\sigma y + D_i(\mathrm{prc} - 1))^2 dy \end{aligned}$$

Define $\lambda_i = \frac{D_i-\mathrm{prc}D_i}{\sqrt{2}\sigma} = \frac{\sqrt{\frac{D_i}{\mathrm{prc}}}(1-\mathrm{prc})}{\sqrt{2}m}$. Then the above equation becomes

$$\begin{aligned} &\int_{\lambda_i}^{\infty} \frac{1}{\sqrt{\pi}} e^{-y}(\sqrt{2}\sigma y + D_i(\mathrm{prc} - 1))^2 dy \\ = &\int_{\lambda_i}^{\infty} \frac{1}{\sqrt{\pi}} 2\sigma^2 y^2 e^{-y} dy + \int_{\lambda_i}^{\infty} \frac{1}{\sqrt{\pi}} 2\sqrt{2}\sigma D_i(\mathrm{prc} - 1) y e^{-y} dy + \int_{\lambda_i}^{\infty} \frac{1}{\sqrt{\pi}} D_i^2(\mathrm{prc} - 1)^2 e^{-y} dy \\ = &\underbrace{\frac{2}{\sqrt{\pi}}\sigma^2 \int_{\lambda_i}^{\infty} e^{-y} y^2 dy}_{I} + \underbrace{2\sqrt{\frac{2}{\pi}}\sigma(\mathrm{prc} - 1)D_i \int_{\lambda_i}^{\infty} e^{-y} y dy}_{II} + \underbrace{\frac{1}{\sqrt{\pi}}(\mathrm{prc} - 1)^2 D_i^2 \int_{\lambda_i}^{\infty} e^{-y} dy}_{III} \end{aligned}$$

The integrals in terms (I), (II), and (III) can be solved with partial integration ($\int y^2 e^{-y} dy = -y^2 e^{-y} - y e^{-y} - e^{-y}, \int y e^{-y} dy = -y e^{-y} - e^{-y}, \int e^{-y} dy = -e^{-y}$). It then follows (recall

[6] Again, it should be mentioned that this expression is only an estimator for z. It can be shown that the expression given is close to the average number times monitored.

$\lambda_i = \frac{\sqrt{D_i}(1-\text{prc})}{\sqrt{\text{prc}}\sqrt{2}m}$):

$$\begin{aligned}
(I) &= \frac{2}{\sqrt{\pi}}\sigma^2(\lambda_i^2 e^{-\lambda_i} + \lambda_i e^{-\lambda_i} + e^{-\lambda_i}) \\
&= \frac{2}{\sqrt{\pi}} D_i \text{prc} m^2 e^{-\lambda_i}(\lambda_i^2 + \lambda_i + 1) \\
&\leq \frac{2}{\sqrt{\pi}} D_i \text{prc} m^2 e^{-\lambda_i} 3\lambda_i^2 \qquad (\text{if } \lambda_i \geq 1) \\
&= \frac{3}{\sqrt{\pi}} D_i^2 e^{-\lambda_i}(1-\text{prc})^2 \\
(II) &= 2\frac{\sqrt{2}}{\sqrt{\pi}}\sigma(\text{prc}-1)D_i(\lambda_i e^{-\lambda_i} + e^{-\lambda_i}) \\
&= 2\frac{\sqrt{2}}{\sqrt{\pi}} D_i^{\frac{3}{2}} \sqrt{\text{prc}}(\text{prc}-1) m e^{-\lambda_i}(\lambda_i + 1) \\
&\leq 2\frac{\sqrt{2}}{\sqrt{\pi}} D_i^{\frac{3}{2}} \sqrt{\text{prc}}(\text{prc}-1) m e^{-\lambda_i}\lambda_i \qquad (\text{note that (II)} \leq 0 \text{ since (prc-1)} \leq 0) \\
&= -\frac{2}{\sqrt{\pi}} D_i^2 (\text{prc}-1)^2 e^{-\lambda_i} \\
(III) &= \frac{1}{\sqrt{\pi}}(\text{prc}-1)^2 D_i^2 e^{-\lambda_i} \\
(I)+(II)+(III) &\leq D_i^2 e^{-\lambda_i}(\text{prc}-1)^2\left(\frac{3}{\sqrt{\pi}} - \frac{2}{\sqrt{\pi}} + \frac{1}{\sqrt{\pi}}\right) \\
&= \frac{2}{\sqrt{\pi}} D_i^2 e^{-\lambda_i}(\text{prc}-1)^2 \qquad (6)
\end{aligned}$$

Equation (6) gives an upper bound on the distance penalty on one epoch. The total distance penalty is the sum over all epochs i, $0 \leq i < z$, where $z = \log_{\frac{1}{1-\text{prc}}} \frac{D}{\text{thresh}}$ is the number of times monitored.

$$\text{distance\_penalty} \leq \sum_{i=0}^{z-1} \frac{2}{\sqrt{\pi}} D_i^2 (\text{prc}-1)^2 e^{-\frac{\sqrt{D_i}(1-\text{prc})}{\sqrt{\text{prc}}\sqrt{2}m}} \qquad (7)$$

If one determines the maximum of the expression within the sum over all possible values of D_i (again by taking the first derivative with respect to D_i and finding the roots), one will find that it has its maximum at $D_i = 32\text{prc}m^2(1-\text{prc})^2$. Plugging this into (7) gives us

$$\begin{aligned}
(7) &\leq \frac{2}{\sqrt{\pi}} \sum_{i=0}^{z-1} (32 \cdot \text{prc} \cdot m^2 \frac{1}{(1-\text{prc})^2})^2 e^{-4}(1-\text{prc})^2 \\
&= z\frac{2}{\sqrt{\pi}} (32 \cdot \text{prc} \cdot m^2 \frac{1}{(1-\text{prc})^2})^2 e^{-4}(1-\text{prc})^2 \\
&\leq z \cdot 22 \cdot m^4 \frac{\text{prc}^2}{(1-\text{prc})^2} \qquad (8)
\end{aligned}$$

Now we can write an upper bound for the expected cost for the proportional reduction strategy. Note that equation (8) only expresses the penalty when the agent overshoots the deadline sometime during a trial. But if it doesn't, $thresh^2$ gives an easy upper bound on the distance penalty.

$$\text{cost}_{\text{pr}} \leq \log_{\frac{1}{1-\text{prc}}} \frac{D}{\text{thresh}} (\text{mon\_cost} + 22 \cdot m^4 \frac{\text{prc}^2}{(1-\text{prc})^2}) + \text{thresh}^2 \qquad (9)$$

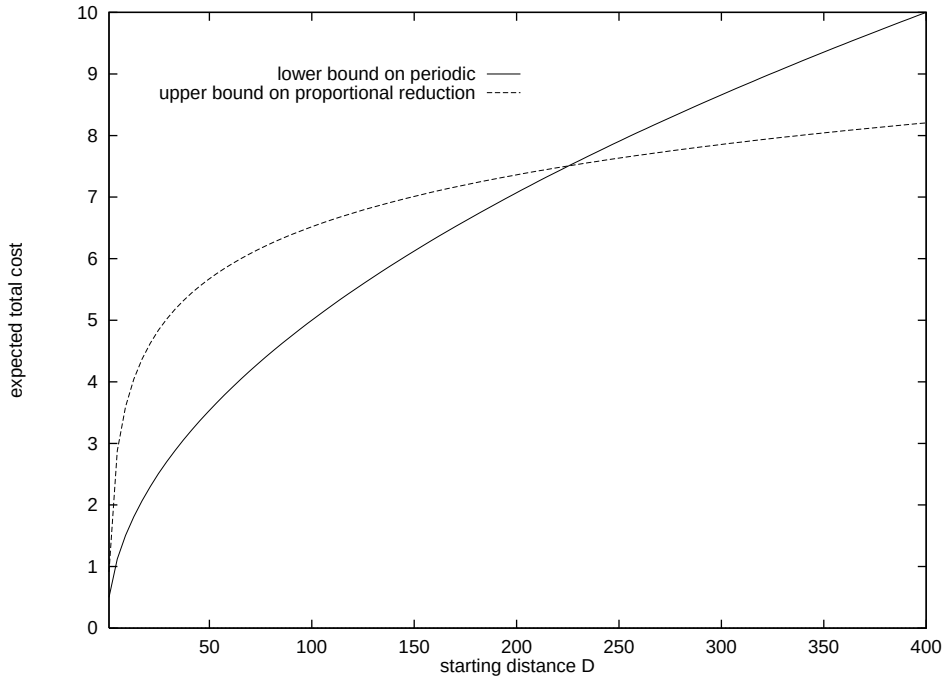

Figure 8: A plot of the bounds on the cost for periodic and proportional reduction with increasing starting distance; $mon\_cost = 1$, $m = 0.25$, $thresh = 0.8$, $prc = .69$.

In the above approximations, λ_i was assumed to be greater than 1. What does this mean?

$$\lambda_i \geq 1 \quad \Leftrightarrow \quad \frac{\sqrt{D_i}(1-\text{prc})}{\sqrt{\text{prc}}\sqrt{2}m} \geq 1 \quad \Leftrightarrow$$

$$D_i \geq \frac{2\text{prc} \cdot m^2}{(1-\text{prc})^2} \quad \Leftrightarrow \quad \text{prc} \leq \left(\frac{m}{\sqrt{2D_i}} - \sqrt{1 + \frac{m}{\sqrt{2D_i}}} \right)^2 \tag{10}$$

Increasing *prc* will also increase the lower bound on D_i. In effect, this limits how small we can make *thresh*, since the above estimations will not be valid below a certain threshold for the distance remaining. Or, alternatively, we can use a setting of *thresh* to determine what the maximum allowable value of *prc* will be.

To summarize: Equations (4) and (9) give us the lower and upper bounds on periodic monitoring and proportional reduction, respectively. The periodic cost is $\Omega(D^{\frac{1}{2}})$ and the proportional reduction cost is $O(\log(D))$, so clearly, proportional reductions beats periodic for a large enough D. Therefore, theorem 1 has been shown to be true.

5.2.4 When is Proportional Reduction Better?

For practical applications, knowing the exact point, $D_{\min}$, at which the cost curves for periodic monitoring and proportional reduction cross, would be preferable. This point will depend on the particular settings of all the other parameters. We will now look at how to choose the still unset parameters in the cost equation of the proportional reduction strategy in such a way that the cost is minimized.

The monitoring cost is not a parameter that can be set by the strategy, so let us assume it is one. High values of *prc* are generally beneficial, since it reduces the size of

the base of the log-function. This tendency is balanced, however, by the expression $\frac{prc^2}{(1-prc)^2}$ in the expected cost function (9). Also, because of constraint (10), *prc* cannot be made arbitrarily large without also increasing *thresh*. The noise parameter m also limits how large *prc* can be made. Figure 8 shows the two cost curves for $m = 0.25$. Numerically, we determined that *thresh* $= 0.8$ and *prc* $= 0.69$ minimizes the point at which the lowest possible periodic cost exceeds the highest possible proportional reduction cost. This point lies at a D-value of approximately 225. Table 3 shows the best numerically determined *thresh* and *prc* values for increasing values of m. The column marked D_{min} shows the point on the x-axis at which proportional reduction provably surpasses periodic.

m	thresh	prc	prc_{max}	D_{min}
0.1	0.6	0.91	0.91	314
0.25	0.8	0.69	0.80	225
0.5	1.3	0.41	0.70	386
0.75	1.8	0.25	0.62	717
1.0	2.3	0.165	0.55	1226
2.0	4.4	0.05	0.38	5365

Table 3: The best values for *thresh* and *prc* for increasing values of m, the noise parameter. D_{min} denotes the point at which the two lines cross, prc_{max} denotes the maximum allowable value for *prc*, given the constraints.

The results of the table are intuitive. As the error m of the internal clock grows, so will the risk the proportional reduction strategy runs of overshooting the deadline by a large amount. The strategy compensates for this risk by decreasing *prc*—taking more cautious steps. The main advantage of proportional reduction is that it can come close to the deadline with a logarithmic number of steps. This advantage cannot show itself when the starting distance is small. We had originally assumed that larger monitoring costs would benefit the proportional reduction strategy. According to the bound equations, however, this is not the case: the cost of periodic monitoring grows with the square root of *mon_cost*, whereas the proportional reduction cost grows linearly.

5.2.5 Summary

Proportional reduction will outperform periodic monitoring if one makes the starting distance large enough. The starting distance at which this occurs grows as the inaccuracy of the internal clock increases. We have not yet been able to show that proportional reduction is always superior, and it may in fact not be. Making the strategies' cost bounds tighter (in particular the bound on the sum of the individual distance penalties for each epoch in the proportional reduction strategy) would reduce the size of D_{min}, and shed more light on the relationship between proportional reduction and periodic monitoring for small starting distances. Intuitively, proportional reduction should perform well in those cases, too. It would also be desirable, although difficult, to have a closed form relationship between m and the minimum starting distance.

Note that for these small starting distances, the costs of both strategies are not that far apart. Figure 8 is a typical plot: the bounded cost of proportional reduction never exceeds the bounded cost of periodic monitoring by more than a factor of 2.5. If the upper

bound on proportional reduction could be improved slightly, it would make a relatively large impact on the size of the minimum starting distance. And if one could show that periodic monitoring is in fact superior when one is very close to the goal, it would make for an interesting hybrid strategy: Use proportional reduction to get close to the goal, and then periodic for the final few steps.

6 What is the Structure of a Monitoring Strategy Taxonomy?

As mentioned in the introduction, we believe three factors determine an agent's behavior: the task it is given, the environment it is operating in, and the way it is built (its architecture). To understand monitoring, it will be necessary to understand the influence of these three factors. The experiments in sections 3.1, 3.2 and 4 were a small step in this direction: they analyzed the effects of the environmental factors sensing error, starting distance, and monitoring cost on the cupcake problem.

On a more abstract level, though, what will be the features upon which a monitoring strategy taxonomy is built? Here are a few candidates regarding task and environment: What shape is the penalty function? Is it symmetric, does it change over time? What is the type of task, what are the associated costs? Do several goals have to be achieved in parallel? Are they independent? Does monitoring for one goal provide information for another? Are monitoring acts independent? Is deciding when to stop monitoring part of the problem, or are trials of indefinite extent? How noisy and unpredictable is the environment? Is it a spatial or temporal domain? Does it allow actions to be reversed? (In the temporal cupcake problem, one cannot increase the time to the deadline, in the spatial version one can increase the distance by going backwards or turning around.) Do false actions have fatal consequences? Are other agents present?

With respect to the agent's architecture, we must consider what types of sensors are available. Can they be used independently? What are their costs, their error functions? Is the error constant or a function of other environmental parameters? Can the effectors influence what is being sensed, or are they independent? What degree of control does the agent have over its environment? What is the agent's reaction time to external events? What is its computing power? Any of these factors may influence the choice of monitoring strategy. The hard problem is determining what subset of factors are important in a given scenario. We suspect that in the cupcake problem, only a few factors influence the choice of best monitoring strategy, the most important one being the amount of information returned by the sensor. Most other factors we looked at, including things like the shape of the penalty function or the monitoring cost, do not alter the superiority of interval reduction. During our experiments with different scenarios, we have come to believe that there may only be a relatively small number of basic monitoring strategies. This would imply that although many factors can influence the strategy, few are relevant in a given scenario. Otherwise, we would be seeing a much a larger number of monitoring strategies.

An interesting way of looking at the difference between periodic and interval reduction is in terms of the quantity of information the sensor provides. If the agent in the cupcake problem had no information other than whether it had overshot the deadline yet or not, it would be forced to use periodic monitoring. Based on this idea, we propose three *complexity classes* for categorizing monitoring problems. "Complexity" is not formally de-

fined, but intuitively it is a combined measure of how much effort is required to solve these problems optimally.

1. *precomputable*: These are problems where the complete sequence of monitoring can be planned in advance, where the time between monitoring events does not depend on any information that must be sensed from the environment. An example is the forest fire scenario where an event has the same probability of occurring at any time. If this fact is known to the agent, it can precompute the optimal period according to the 2nd sufficient condition for periodic monitoring in section 5.1.1. If the event distribution has a more complex form, the optimal sequence of events can also be precomputed, although we do not yet have a guaranteed optimal algorithm for this.
2. *dynamically dependent on environment*: Some strategies will depend on the current state of the environment. An example of this is monitoring for fires under different weather conditions: Monitoring will always be periodic, but the period is dependent on the current weather, as this influences the chances of a fire starting. Because of this variable, the sequence of monitoring actions cannot be precomputed. It might often be possible, however, to express the optimal period in closed form, with the weather being simply one of the variables in the equation.
3. *sequential decision problems*: These are the most difficult problems. Not only is the monitoring rate dependent on the environment, but also on all future monitoring actions. An example of this is of course the cupcake problem. When to monitor depends on the information returned by the sonar sensor, and also on the decision whether or not to monitor *again* after the next movement cycle. If the agent does not monitor again, it will be better for it to move further now; if it knows it will, it can be more cautious on this movement step, and rely on future monitoring events to get it closer to the deadline. This dependence of monitoring actions is what makes the cupcake problem hard to solve optimally. Even though we can compute a optimal solution with dynamic programming, we have not been able to find a closed form for the optimal interval reduction function.

7 Related Work

Many fields touch on monitoring problems to at least a certain extent. There is a considerable literature on reinforcement schedules in behavioral psychology; on modeling human monitoring behavior during process control; and a smaller literature on AI planning and plan execution monitoring. Unfortunately, ethology, the study of animal behavior, does not deal with monitoring strategies on the same abstract level that we do. We will discuss each of these areas in turn.

The relevant behavioral psychology paradigm has already been discussed in Section 3.1.1. Let us address some additional points here. It is interesting to note that animals and humans do not typically display the same behavior on the fixed interval reinforcement task. Often humans will respond either at a very low rate (once or twice at the end of the reinforcement interval T) or at a high constant rate throughout. Lowe, Beasty, and Bentall (1983) have shown that preverbal children, however, display the classical scalloped response curve, and argue that language ability is responsible for the differences in behavior across species. But why do adult humans behave this way? Looking at the reinforcement task as a monitoring problem may shed some light. The human low rate response is in fact

close to the optimal strategy: monitoring with period T. The high rate response behavior could conceivably be explained by assuming humans do not associate a cost with responding, whereas animals do. If this is the case, a monitoring strategy with a high period is the best strategy; it guarantees reinforcement as soon as it becomes available. Humans subjects *know* that they are in some sort of experiment—animals don't. One could argue that humans therefore have nothing better to do than respond, effectively making the cost of a response zero, unless some sort of cost is explicitly charged.

Another parallel between our work and operant conditioning is the investigation of the postreinforcement pause. In a fixed interval schedule, subjects do not usually respond for a certain time since the last reinforcement. How does this pause depend on the interval length T? Initial studies assumed a linear relationship (e.g., Schneider, 1969), but more recently, the evidence appears to indicate a sub-linear relationship of the form $p = cT^x$, where x is a constant smaller than one, c is a small positive number (usually ranging from about 0.5 to 2.5), and p is the pause duration (e.g., Lowe, Harzem & Spencer, 1979; Wearden, 1985). This fits well with the optimal monitoring strategy found via dynamic programming, which has a very similar form.

Human visual sampling behavior has been studied extensively in the areas of process control—in which a human supervises a potentially unstable process such as a nuclear power plant—and pilot's and air traffic controller's instrument viewing patterns (see Moray, 1986, and Senders, 1983, for reviews). It should be emphasized that the aim of these studies is to model and predict *human* performance on monitoring tasks, not to determine the theoretically optimal strategy in any given situation.

Many models of human monitoring behavior have been proposed. Most fall into one of two categories: those based on information theory, and those based on expected costs. What follows is a selection of the models that appear to be of particular relevance to the work presented in this paper.

The earliest work on quantitative models of monitoring behavior was done by Senders and his co-workers. They studied eye movements between instrument displays (Senders et al., 1966; Senders, 1983). The underlying assumption is that each instrument is a zero mean gaussian process with limited information bandwidth, i.e. the rate of change is bounded. It is also assumed that a human observer will fixate on an instrument often enough to extract all the information from it. The Nyquist theorem, which states that a signal with bandwidth W must be sampled at a minimal rate of $2W$, is assumed to control the monitoring frequency. While not perfect, this model did capture the qualitative behavior of test subjects. It predicts a periodic strategy.

Senders extended his models to account for a signal observation that is close to the *critical limit* for an instrument. He assumed the agent will monitor again when the likelihood that a threshold is exceeded is maximum. This approach resembles at least superficially the one we took to derive the SIR strategy. It was also taken by Moray (1986, pp. 40-14–40-16) in developing his model of radar operators. It should however be pointed out that all these models assume band-limited gaussian signals, and that the task of the subject is to extract information from these signals. This appears to be a different kind of monitoring task than, say, monitoring for an "event" such as a forest fire breaking out. Such events are instantaneous and cannot be predicted; they have unlimited bandwidth.

Carbonell proposed a queueing model for an agent that has to monitor several displays (Carbonell, 1966; Carbonell, Ward & Senders, 1968). Instruments were modeled

more realistically by moving away from the gaussian distribution assumption. Instead, errors accumulated until a control action is taken. Looking at one display means possibly delaying the observation of another, allowing it to potentially cross a critical threshold. Observations are scheduled by considering the cost of not looking at the other instruments during this time period. This model performed very well predicting actual pilot behavior during an airport approach.

Both Senders' and Carbonell's models are based on information theory. The models we will discuss now, based on expected costs, have much more in common with our own. Kvalseth (1977) performed an experiment conceptually very similar to the one in Section 4, although it wasn't done in the cupcake problem domain, but instead involved monitoring a sequence of numbers and indicating when they exceeded a threshold. He too investigated the effect of monitoring cost and errors on the sampling period, and found, as we did, that agents monitor more frequently as the process becomes less predictable (higher errors) and that it declines as the cost of monitoring grows. He also found, as we did, that human performance on these tasks is slightly suboptimal. Moray (1986, p. 40-17) attributes this suboptimality to a lack of practice on part of the subjects.

Sheridan (1970) proposed a method for computing the next time to monitor. This method can be seen as an extension of the proof in Section 5.1.2. Sheridan defines a cost term consisting of the difference of the value of the last observation (we used accumulated expected cost since the last observation) and the average monitoring cost per unit time. One monitors again when this function reaches its maximum. If the properties of the monitored process do not change, this method results in a periodic strategy.

In the artificial intelligence domain, the work most closely related to ours comes from the area of monitoring plan execution, which deals with the question of how to determine whether the execution of a plan is proceeding according to expectations. The need for monitoring arises from non-determinism of the environment—no planner can completely accurately predict future events or the effects of proposed actions. However, the issues addressed in these systems revolve mainly around *how* to combine monitoring with planning, and not on what the resulting strategies look like. Some systems already assume the agent has a basic sensing policy (e.g., Chrisman & Simmons, 1991). Monitoring costs are also not usually taken into account.

Typically, in AI planning systems, monitoring plays the role of verifying plan execution. This involves checking the unfolding execution of a plan, so as to replan if something goes wrong, or adjust the plan opportunistically. One of the first systems that used monitoring for both these purposes was the autonomous mobile robot "Shakey" (Raphael, 1976), in particular its execution monitoring component "PLANEX" (Fikes, Hart & Nilsson, 1972). Systems that don't construct explicit plans, such as reactive (Agre & Chapman, 1987; Kaebling, 1987; Schoppers, 1987) or run-time planners (Firby, 1987; Georgeff & Lansky, 1987; McDermott, 1978), do not need to check plan execution, but they will still have to monitor to assess their current state.

There appears to be a growing consensus that no clear distinction can or should be made between planning and execution (Agre & Chapman, 1990; Dean, 1987). This implies that the process of finding monitoring strategies cannot be separated from the planning process. In other words, the planner is responsible for generating an appropriate monitoring strategy to go with its plan; there is no point in having a general purpose theory that finds a monitoring strategy for an arbitrary plan, since the generator of the monitoring strategy

requires the same reasoning capabilities as the planner (McDermott, 1992). This does not contradict the notion of a monitoring strategy taxonomy, since the taxonomy incorporates the task of the agent, as well. As we have seen, LTB simultaneously learned the "plan" for solving the task and the monitoring strategy it employed.

As previously mentioned, many planners make the distinction between monitoring and sensing. Typically, this is accomplished by making the planner's primitive actions responsible for doing the actual sensing, and having the planner rely on a high-level world model. While this approach has the advantage of delegating the problem of sensor integration (combining possible conflicting data over a period of time) to the procedural subsystems that execute the actions, it also means that the planner has no direct sensor control. Doyle, Atkinson, and Doshi (1986) include explicit sensor commands in the plan language, which are used to verify the plan's execution. However, these *verification operators* are only inserted *after* the plan has been generated, thus separating action planning from the process of finding a monitoring strategy. Ambros-Ingerson and Steel's (1988) system truly integrates these two tasks.

The reliance on primitive, uninterruptable actions also forces even those planners that can monitor explicitly to place the monitoring commands *between* these actions. In particular, this means that the length of time it takes to execute a primitive action defines an implicit minimum monitoring period. Apart from keeping the planner's primitive actions very basic, the only way around this problem is to allow some sort of concurrent monitoring procedures (Hendler & Sanborn, 1987; Simmons, 1990). Even without multiple processors, concurrency can be achieved by allowing external events to interrupt the current action being executed.

Limits on an agent's sensory system are both physical and computational: An agent does not have a sensor for everything that could be measured in the environment, and it does not have infinite computing power, so it cannot necessarily use all the available information. Together with other costs levied on the act of monitoring, this forces the agent to monitor only some features of the environment, or monitor the environment only at particular times. While most of our work has focused on the latter approach, and attempts to formalize general strategies for given scenarios (Atkin, 1991; Atkin & Cohen, 1993; Atkin, 1993; Cohen, Atkin & Hansen, 1994; Hansen, 1992a; Hansen, 1994), the former approach is by no means trivial. It encompasses the whole area of active perception in robotics and computer vision (see for example Bajcsy, 1988).

Some methods reminiscent of Senders' sampling models have been used in AI planning. One can estimate the amount of uncertainty introduced by each effector command that is executed, and monitor again when this uncertainty exceeds a threshold. This mechanism was implemented in the original Shakey robot (Raphael, 1976) and other robotic systems (Latombe, Lazanas & Shekhar, 1991).

Another approach is to model the rate of change in the environment, and to use this as the basis for the monitoring period. This is the design philosophy behind reactive systems (e.g., Agre & Chapman, 1987), which are supposed to monitor the environment fast enough to be able to act appropriately in a timely fashion. Monitoring at a high rate for all environmental conditions might be an easy solution, but is certainly not optimal in terms of monitoring costs.

Ethology studies the behavior of animals. Should it not also consider monitoring strategies? Apparently not in the same abstract sense as we do. Take for example the area

of optimal foraging, which has been analyzed in a rigorous mathematical fashion (see Pyke, Pulliam & Charnov, 1977, and Pyke, 1984, for a review).[7] Here the question being answered is how an organism can minimize the energy it expends catching prey while maximizing its food intake. A result might be an equation governing the maximum distance an organism should move to catch something given a certain likelihood of prey appearing. However, the question we would be interested in is how the agent *checks* its environment. How does the information the organism gathers affect this strategy? This type of question is not usually under investigation, not even when predator-prey interaction is the focus of research.

In sum, what has the literature told us about a monitoring strategy taxonomy? Unfortunately, not all that much. There are many examples of different monitoring strategies in different situations, many rigorous mathematical models of behavior in specific scenarios, and some insights into what factors of the task and environment are important, but most work is not at the right level of abstraction for our purposes. For a taxonomy, we need not only a *description* of the decision rule that specifies when to monitor next, but a *classification* of the type of strategy this rule will produce, and an *optimality criteria* for strategies in this scenario. In many cases, it is hard to say just by looking at a rule or a response rate graph what the strategy will be, but that information is necessary.

8 Summary

The contribution of this work has been the identification of two very general monitoring strategies, periodic monitoring and interval reduction, and the demonstration of their wide applicability. As a consequence of this work, we have come to believe that there may be no further basic strategies. Moreover, we have been able to show experimentally and mathematically that interval reduction is the superior strategy for a class of problems that require monitoring for a deadline. This is an important result in and of itself, but it also drives home the point that periodic monitoring is *not* always the best strategy to use. The reason it is so prevalent a strategy might either be due to the fact that the people using it regard the non-optimality of their monitoring strategy as inconsequential, or that monitoring is assumed to be free. This work has demonstrated that alternative strategies need not be very complicated and can sometimes outperform periodic monitoring by a substantial margin.

Using the cupcake problem as an example, we have shown how the environment can effect the strategy. We believe that the method of identifying the potential features of task, architecture, and environment that are relevant to a certain problem, varying them, and then comparing the resulting behaviors of the agent, is a general method for finding rules of agent design. We have listed many factors that may influence the choice of monitoring strategy. Determining the relevance and influence of factors in a given scenario is still very much an open research question that we hope will be expanded on.

We have proposed three sufficient conditions on when to use periodic monitoring and interval reduction. The conditions in these rules are very general, and refer to very abstract properties of environment, task, and architecture. It seems to us that a larger set of such rules could effectively realize the goals we had hoped the taxonomy would fulfill: enabling an agent designer to work more efficiently. In order to increase their utility, more

[7] Foraging strategies comparable with those found in nature have even been evolved by genetic programming (Koza, Rice & Roughgarden, 1992).

work needs to be done on describing the relationship between features of the scenario and parameters of the strategy, instead of just stating which strategy to use.

Acknowledgements

We would like to thank Rod Grupen and Ramesh Sitaraman for fruitful discussions, and Peter Todd for pointing out a wealth of related work. Our research was supported by ARPA/Rome Laboratory contracts #F30602-91-C-0076 and #F49620-89-C-00113, an Intelligent Real-Time Problem-Solving Initiative contract AFOSR-91-0067, and by NTT Data Communications Systems Corporation. The US Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the Advanced Research Projects Agency, Rome Laboratory of the U.S. Government.

References

Agre, P. E., & Chapman, D., 1987. Pengi: An implementation of a Theory of Activity. *Proceedings of the Sixth National Conference on Artificial Intelligence*, 268-272.

Agre, P. E., & Chapman, D., 1990. What are plans for? In: P. Maes (Ed.), *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*. MIT Press, Cambridge, MA.

Ambros-Ingerson, J. A., and Steel, S., 1988. Integrating Planning, Execution, and Monitoring, *Proceedings of the Seventh National Conference on Artificial Intelligence*, 83-88.

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

Atkin, M. & Cohen, P. R., 1993. Genetic Programming to Learn an Agent's Monitoring Strategy, *Proceedings of the AAAI 93 Workshop on Learning Action Models*, 36-41.

Atkin, M. S, 1993. Using Genetic Algorithms to Learn Monitoring Strategies. Unpublished.

Atkin, M. S. & Cohen, P. R., 1994. Learning Monitoring Strategies: A Difficult Genetic Programming Application. *Proceedings on the First IEEE Conference on Evolutionary Computation*, 328-332a.

Bajscy, R., 1988. Active Perception. *Proceedings of the IEEE* **76** (8), 996-1005.

Carbonell, J. R., 1966. A queuing model model for many-instrument visual sampling. *IEEE Transactions on Human Factors in electronics*, **HFE-7**, 157-164.

Carbonell, J. R, Ward, J. L., and Senders, J. W., 1968. A Queueing Model of Visual Sampling; Experimental Validation. *IEEE Transactions on Man-Machine Systems*, **MMS-9**, 3, 82-87.

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*, **56**, 152-164.

Chrisman, L. & Simmons, R., 1991. Sensible Planning: Focusing Perceptual Attention. *Proceedings of the Ninth National Conference on Artificial Intelligence*, 756-761.

Cohen, P. R., Howe, A. E., & Hart, D. M., 1990. Intelligent Real-time Problem Solving: Issues and Examples. Computer Science Technical Report 90-20. University of Massachusetts, Amherst.

Cohen, P. R., 1990. Modeling How Interactions Between Agents' Architectures and Environments Produce Behaviors, for the Purpose of Design and Analysis. *Proceedings of the AAAI Workshop on Planning.*

Cohen, P. R., Atkin, M. S., and Hansen, E. A., 1994. The Interval Reduction Strategy for Monitoring Cupcake problems, *Proceedings of the Third International Conference on the Simulation of Adaptive Behavior*, 82-90.

Dean, T., 1987. Planning, Execution, and Control. *Proceedings of the DARPA Knowledge-Based Planning Workshop*, 29-1–29-10.

Doyle, R., Atkinson, D., & Doshi, R., 1986. Generating Perception Requests and Expectations to Verify the Execution of Plans. *Proceedings of the Fifth National Conference on Artificial Intelligence*, 81-88.

Ferster, C. B. and Skinner, B. F., 1957. *Schedules of Reinforcement.* Appleton-Century-Crofts, New York, NY.

Fikes, R., Hart, P., & Nilsson, N., 1972. Learning and Executing Generalized Robot Plans. *Artificial Intelligence* **3** (4), 251-288. Reprinted in: Allen, J. F., Hendler, J., and Tate, A. (Eds.), *Readings in Planning*, Morgan Kaufman, San Mateo, CA.

Firby, R. J., 1987. An Investigation into Reactive Planning in Complex Domains, *Proceedings of the Sixth National Conference on Artificial Intelligence*, 202-206.

Georgeff, M. P., & Lansky, A. L., 1987. Reactive Reasoning and Planning, *Proceedings of the Sixth National Conference on Artificial Intelligence*, 677-682.

Goldberg, D. E., 1989. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley, Reading, MA.

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.). Morgan Kaufman, San Mateo, CA.

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

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

Hansen, E. A., 1994. Cost-Effective Sensing During Plan Execution. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1029-1035.

Hendler, J., & Sanborn, J., 1987. A Model of Reaction for Planning in Dynamic Environments. *Proceedings of the DARPA Knowledge-Based Planning Workshop*, 24-1–24-10.

Kaebling, L., 1987, An Architecture for Intelligent Reactive Systems. In Georgeff and Lansky (Eds.), *Reasoning About Actions and Plans.* Reprinted in: Allen, J. F., Hendler, J., and Tate, A. (Eds.), *Readings in Planning.*

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

Koza, J. R. & Rice, J. P., 1992. Automatic Programming of Robots using Genetic Programming. *Proceedings of the Tenth National Conference on Artificial Intelligence*, 194-207.

Koza, J. R., Rice, J. P., Roughgarden, J., 1992. Evolution of Food-Foraging Strategies for the Carribean *Anolis* Lizard Using Genetic Programming. *Adaptive Behavior*, **1**, 171-199.

Kvalseth, T., 1977. The effect of cost on the sampling behavior of human instrument monitors. In Sheridan, T. B., and Johannsen, G. (eds.), *Monitoring behavior and supervisory control*, Plenum, New York, NY.

Latombe, J.-C., Lazanas, A., and Shekhar, S., 1991. Robot Motion Planning with Uncertainty in Control and Sensing. *Artificial Intelligence*, **52** (1), 1-47.

Lowe, C. F., Harzem, P., and Bagshaw, M., 1978. Species Differences in Temporal Control of Behavior II: Human Performance. *Journal of the Experimental Analysis of Behavior*, **29** (3), 351-361.

Lowe, C. F., Harzem, P., and Spencer, P. T., 1979. Temporal Control of Behavior and the Power Law. *Journal of the Experimental Analysis of Behavior*, **31** (3), 333-343.

Lowe, C. F., Beasty, A., and Bentall, R. P., 1983. The Role of Verbal Behavior in Human Learning: Infant performance on Fixed-Interval Schedules. *Journal of the Experimental Analysis of Behavior*, **39** (1), 157-164.

Mackintosh, N. J., 1974. *The Psychology of Animal Learning.* Academic Press, London.

McDermott, D., 1978. Planning and Acting. *Cognitive Science*, **2** (2), 71-109. Reprinted in: Allen, J. F., Hendler, J., and Tate, A. (Eds.), *Readings in Planning*, Morgan Kaufman, San Mateo, CA.

McDermott, D., 1992. Robot Planning. *AI Magazine*, Volume 13, No. 2 (Summer 1992), 55-79.

Moray, N., 1986. Monitoring Behavior and Supervisory Control. In Boff, Kaufman, and Thomas (eds.), *Handbook of Perception and Human Performance*, Vol. II, chapter 40. Wiley, New York, NY.

Pyke, G. H., Pulliam, H. R., and Charnov, E. L., 1977. Optimal foraging: a selective review of theory and tests. *Q. Rev. Biol.*, **52**, 137-154.

Pyke, G. H., 1984: Optimal foraging: A critical review. *Annual Review of Ecology and Systematics*, **15**, 523-575.

Raphael, B., 1976. *The Thinking Computer: Mind Inside Matter.* W. H. Freeman and Company, San Francisco., 153-159, 169, 275-281, 287-288.

Schneider, B. A., 1969. A Two-State Analysis of Fixed-Interval Responding in the Pigeon. *Journal of the Experimental Analysis of Behavior*, **12** (5), 677-687.

Schoppers, M., 1987. Universal Plans for Reactive Robots in Unpredictable Domains. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1039-1046.

Schwartz, B., 1984. *Psychology and Learning Behavior*, 2nd edition. W. W. Norton & Company, New York, NY.

Senders, J. W., Elkind, J. I, Grignetti, M. C., and Smallwood, R., 1966. An Investigation of the Visual Sampling Behavior of Human Observers. NASA CR-434, National Aeronautics and Space Administration, Washington, DC.

Senders, J. W., 1983. Visual Scanning Processes. University of Tilburg Press.

Sheridan, T. B., 1970. How often the supervisor should sample. *IEEE Transactions on Systems, Science, and Cybernetics*, **SSC-6**, 2, 140-145.

Simmons, R. G., 1990. An Architecture for Coordinating Planning, Sensing, and Action. *Proceedings of DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 292-297.

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*, 216-224.

Wearden, J. H., 1985. The Power Law and Weber's Law in Fixed-Interval Postreinforcement Pausing: A Scalar Timing Model. *The Quarterly Journal of Experimental Psychology*, **37B** (3), 191-211.