Monitoring in Embedded Agents

Marc S. Atkin and Paul R. Cohen

Computer Science Technical Report 95-66

Experimental Knowledge Systems Laboratory
Computer Science Department, Box 34610
Lederle Graduate Research Center
University of Massachusetts
Amherst, MA 01003-4610

Abstract

Finding good monitoring strategies is an important process in the design of any embedded agent. We describe the nature of the monitoring problem, point out what makes it difficult, and show that while periodic monitoring strategies are often the easiest to derive, they are not always the most appropriate. We demonstrate mathematically and empirically that for a wide class of problems, the so-called "cupcake problems", there exists a simple strategy, ***interval reduction***, that outperforms periodic monitoring. We also show how features of the environment may influence the choice of the optimal strategy. The paper concludes with some thoughts about a monitoring strategy taxonomy, and what its defining features might be.

This research was supported by ARPA/Rome Laboratory under contract F30602-91-C-0076.

1 Introduction

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. Every 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 represents a summary of the work we have done on discovering and categorizing monitoring strategies. We want to demonstrate that the prevalent scheme, monitoring after each effector action [6, 7, 8, 9, 15], is not always the most efficient. The emphasis will be on discussing what makes certain monitoring problems more difficult than others, and investigating the tradeoff between two monitoring strategies, periodic and interval reduction.

In robotic systems, monitoring happens on at least three levels. On the lowest level, physical sensors probe the environment. The information they provide will generally have to be processed before it is usable to the agent. Higher up, some sort of reactive controller might be frequently checking the pre-processed sensor data to decide whether conditions exist to which the agent must immediately respond. Higher still, a symbolic plan that is controlling the robot might require confirmation that an action's precondition holds. This condition might be quite complicated in terms of the raw sensor data; for example: "is there another agent in my vicinity?" It might be the result of a lot of computation and maybe multiple pollings of the environment on the part of the lower level sensory sub-systems. Each of these levels requires a monitoring strategy, but due to the different circumstances and costs involved, the strategies might be completely different. A distinction is frequently made between *sensing* and *monitoring* (e.g. [8, 9, 15]). 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.

The multi-level model of monitoring also explains why interrupts cannot be used to circumvent the need for monitoring. A monitoring problem cannot be avoided; all you can do is delegate it to a separate sensory system and instruct it to interrupt you when the event being monitored for occurs. The question of how this other sensory system should efficiently monitor for the event still has to be answered. And of course, introducing a new system for every possible event incurs a significant cost, be it in hardware for dedicated processors or execution time for interleaving several monitoring processes. There is also the problem of informedness. The process that monitors explicitly has potentially more information at its disposal—and can therefore do a better job at minimizing costs—than an interrupting process that only knows the interrupt criteria it was given when it was initiated and cannot take changing situations into account.

As we will use the term, a *monitoring strategy* is the scheme by which monitoring actions are scheduled. The strategy must balance the cost of monitoring against the cost of having inaccurate information about the environment. While in principle nothing more than an optimization problem, it is by no means trivial. Three factors make the problem difficult: First, the time between monitoring actions will depend on many factors, features of the environment and costs, some of which are not known or only known probabilistically. Second, these factors 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 placement of all the successive ones.

The rest of the paper will be structured as follows: We will show that optimal periodic monitoring strategies are easy to derive; however, they are not always optimal. Better strategies, such as *interval reduction*, do not lend themselves to such a simple analysis. We will then go on to describe how we discovered monitoring strategies with a genetic algorithm. This algorithm was also used to compare periodic and interval reduction in a certain class of scenarios. This comparison is augmented by the sketch of a proof that shows the asymptotic superiority of interval reduction. The paper concludes with some thoughts about the features that should be included in a taxonomy of monitoring strategies.

2 Periodic Monitoring

The simplest and most common monitoring strategy is periodic monitoring: monitor every r time units. It is easy to show that if you have no knowledge about how the event being monitored for is temporally distributed, and if monitoring tells you only whether the event has occurred or not, then periodic monitoring is the best strategy [3]. Periodic monitoring is also the best strategy if the event can occur at any given time with equal probability [3]. As an example, 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, and is therefore very costly. Call this cost 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

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

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 problem becomes slightly more complicated if the probability p of fire breaking out varies over time. If this distribution is known to the agent, it can still compute a good monitoring strategy, but it will no longer generally be periodic.[1] But consider the case where the agent knows only one fire will occur in a finite amount of time. When the agent monitors, the conditional distribution of the fire changes, allowing the agent to acquire *more* information about when the fire is liable to start by monitoring. If the fire has not yet started, then the agent knows the fire must occur in the future; if it has, the agent can stop monitoring. This type of problem is called a *sequential decision problem.* It is impossible to schedule one monitoring event optimally without taking the others into account. We believe this is a general rule: If monitoring changes the agent's knowledge about the distribution of an event, monitoring events are no longer independent and monitoring becomes a sequential decision problem.

3 Interval Reduction

In 1985, Ceci and Bronfenbrenner conducted an experiment with children from two different age groups [4]. 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

[1]The algorithm for this is quite straightforward: set up the function expressing combined cost of monitoring once plus the expected cost for not monitoring per unit time, starting at the current time. Monitor again at the local minimum of this function. This minimum can be determined numerically.

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, 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 time to monitor next is reduced after each monitoring event. We use "cupcake problem" to denote any scenario in which an agent must monitor for a deadline and accumulates an error while making progress towards this deadline. In the previous example, the error accumulated due to a human's inaccurate internal clock. However, the same strategy arises if one assumes the internal clock is accurate, but the information given to the agent is not. Like periodic monitoring, interval reduction is a very general strategy. We have shown empirically that it arises in a variety of agents, artificial and natural, in cupcake problems [5].

Interval reduction emerges if one simply tries to fix the probability of overshooting the goal during each epoch. 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.[2] 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 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:

$$t = D + \frac{m^2 z_{2\alpha}^2}{2} - m z_{2\alpha}\sqrt{D + \frac{m^2 z_{2\alpha}^2}{4}} \qquad (2)$$

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.

This approximate solution to the cupcake problem does not take the cost of monitoring or overshooting the deadline into account. If one does, the problem of finding the optimal solution turns out to be very difficult, because one cannot optimally place the next monitoring action without knowing whether or not there will be another chance to monitor again after that. If so, one might be more cautious this time around, for fear of overshooting the deadline; if not, one might take a larger step now. Again, we have a sequential decision problem. Eric Hansen has generated the optimal strategy for a given cupcake problem using a dynamic programming algorithm, and it is in fact an interval reduction strategy [11]. One computes the optimal placement of the last monitoring event and works backwards to the starting point. This algorithm has been extended to the general problem of monitoring plan execution [12].

The problem with a dynamic programming approach is that it gives you the monitoring strategy implicitly: at each time step, you know how long you should wait before monitoring again. You do not have a procedural representation of the strategy, such as given by equation 2, or even the general injunction to monitor periodically. Deriving the optimal strategy mathematically is not always an option due to the complexity of some of the optimization problems. We were therefore looking for an automatic, machine learning approach of generating *explicit* monitoring strategies for arbitrary problems. A genetic programming algorithm fit this billing.

[2] 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.

4 Periodic vs. Proportional Reduction

Motivated by the Ceci and Bronfenbrenner study, one of our main interests was the relative merits of interval reduction and periodic monitoring in the cupcake problem. In this section, we will present both empirical and analytical work (see [1] for a more complete description). The subclass of interval reduction strategies under investigation is perhaps the most simple, characterized by a linear interval reduction function: after monitoring, the agent waits a fixed proportion of the distance remaining to the deadline before monitoring again. We call this class of strategies *proportional reduction* strategies.

4.1 The Genetic Programming Algorithm

Genetic programming concerns itself with the evolution of *programs* via a genetic algorithm (e.g. [13, 14]; see [10] for an introduction to genetic algorithms). In our case, these programs represent monitoring strategies. But they are more than just that: they also tell the agent how to solve the task at hand. They are representations of *behavior.*

Our goal was to apply genetic programming to a wide variety of problems, and build a taxonomy from the resulting monitoring strategies based on general features of the task and environment [3]. For this purpose, we needed a programming language that was general and easily extendable. We did not want to limit ourselves to monitoring problems in the temporal domain, and therefore chose as our testbed a two-dimensional world with obstacles and other types of terrain, within which a simple simulated mobile robot, equipped with a few sensors, must operate. There were two types of constructs in our behavior language: those that actually trigger an effector action, and those that steer the control flow within the program. Examples for 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 for the latter category are LOOP commands and conditionals.

We developed several systems that learned monitoring strategies (see [2] for an overview). The one we discuss here, linear LTB, represented individuals as fixed-size lists of commands. Population size was 800, tournament selection with a tournament size of two was the selection scheme. A program's fitness score was based on the average of twelve different test cases. To account for the random and potentially sub-optimal performance of the genetic algorithm, we ran LTB ten times in each scenario. The best output of all these runs is taken to be the final output of the system for a given scenario.

To construct an instance of the cupcake problem in LTB's spatial world, we gave the robot the task of getting as close to a particular position in the map as possible, the *goal point*, without hitting it. The fitness function reflected this: it gave a penalty that was a quadratic function of the distance between the agent's final position on the map and the goal point. However, actually stepping on onto the goal field was 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 *asymmetric* cupcake problem.

The robot had an accurate movement effector (movement corresponds to waiting in the original cupcake problem), but an inaccurate distance sensor. The error was modeled by adding a normal 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 was an obstacle on the goal field (and nowhere else), this sensor now returns the distance to the goal. The variance of the noise was 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 could be varied

```
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 1: A proportional reduction strategy generated by linear LTB

to change the degree of inaccuracy in the sensor. In this scenario, the robot has two other sensors, both of which are automatically updated: reached_goal ("have I reached the goal field?") and hit_object ("have I hit an obstacle?").

An illustrative monitoring strategy evolved by LTB is given in figure 1. The top half of the code is the main body of the program; the bottom half consists of the three *interrupt handlers* corresponding to the agent's sensors; however, in this scenario, they are not used.[3] 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 variable "object_distance" after monitoring) and loops over its value. Four MOVE's are executed in this loop, each moving .1 distance units. 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, and 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 distance monitored.

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

4.2 Genetic Algorithm Results

We suspected that several features of the environment and the robot would influence the monitoring strategy, and it was the goal of the experiment described here to determine what that influence was. We had the following 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. The proportionality constant is the proportion of the robot's estimated distance to the goal that the robot actually moves before monitoring again.
3. Periodic monitoring becomes more likely as m increases (sonar data contains increasingly less useful information) and path length decreases (the advantage of proportional reduction, being able to traverse a distance with only a logarithmic number of monitors, is reduced).

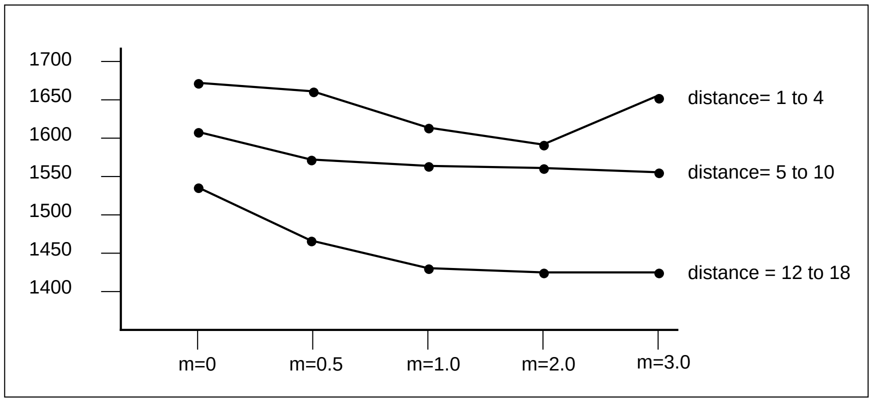

Figure 2: A plot of cell means for a two-way ANOVA, monitoring error m by distance. The dependent measure is the *fitness* of the best individual.

We ran the genetic algorithm on five levels of monitoring error m (0, 0.5, 1.0, 2.0, 3.0) and three ranges of path length (1-4, 5-10, and 12-18 fields). The path length had to be a range or the genetic algorithm will simply adapt to move exactly the required distance, without monitoring at all. 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*. Note that the second measure basically tells us how many times the agent monitored, as monitoring has a fixed cost.

Figure 2 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 $m = 3.0$, path length=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 ANOVA shows a clear effect of path length and monitoring error on fitness (for both main effects $p < 0.0001$), but no interaction effect, because both factors are independent variables.

We had expected to see monitoring cost increase as path length and especially monitoring error increases. This prediction is only partly verified. The expected pattern is distorted because when

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 1: The best individual's monitoring strategy

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 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 1 gives an overview of the best individual's monitoring strategy for each test case. As one can see, in cases $m = 3.0$, path length 5-10; and $m = 2.0$, path length 5-10, no monitoring strategy was found. Sometimes (case $m = 1.0$, path length 1-4 and path length 12-18), it will combine the strategy of moving past the obstacle with proportional reduction, ensuring that if proportional reduction fails, it will still not touch it.

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. It is hard to imagine how this could be advantageous.

An interval reduction strategy was found in nearly all situations. Usually this was not pure proportional reduction, but a slight variant (often an agent would move a small fixed distance within the monitoring loop or prior to it). 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-monitoring-error situations, but its fitness was slightly lower than that of proportional reduction. The one clear exception is the $m = 3.0$, range = 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.

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 1). 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 $m = 2.0$, path length = 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.

4.3 A Proof of Interval Reduction's Superiority

While it is certainly intuitive that proportional reduction should outperform periodic monitoring for the cupcake problem, a rigorous mathematical proof eluded us for a long time. A cupcake problem is characterized by three parameters, starting distance D, an error m, and monitoring cost *mon_cost*, and it is difficult to show that for *all* values of these parameters proportional reduction is superior, since the best strategy is a function of these variables. In fact, we assumed that for certain extreme values, such as short starting distances or large m, periodic monitoring would do better.

In this section, we will outline a proof of proportional reduction's *asymptotic* superiority over periodic monitoring in the cupcake problem. We have been able to show that for any cupcake problem with large enough D, a proportional reduction strategy exists that does better than the best periodic one. The proof is fairly involved, and cannot be presented in its entirety here (see [3]). Basically, the proof proceeds as follows: Derive a lower bound on the expected cost of the best periodic monitoring strategy for a cupcake problem. A periodic monitoring strategy is characterized solely by the parameter *period*, and involves monitoring every *period* time or distance units until the sensor reports that one is within *period/2* units of the deadline. After this cost bound has been established, derive an *upper* bound on the expected cost for the best proportional reduction strategy. We are trying to show that a proportional reduction strategy exists that will do better than the best periodic one. By establishing an upper bound on the cost of the former, we are only making the problem harder for ourselves. This proportional reduction strategy is dependent on two factors, the proportional reduction constant *prc*, and a threshold *thresh*. One terminates the trial once the sensor reports that one is within *thresh* time or distance units of the deadline.

It should be pointed out that the proof is based on the variant of the cupcake problem that assumes the agent's sensor is totally accurate, but its movement (or waiting) effector is not. This effector has a normal error with standard deviation $\sigma = m\sqrt{t}$, where t is the distance (or time) the agent planned to move/wait. Also, the quadratic distance penalty is symmetric, i.e., the same cost is charged for overshooting and undershooting the deadline.

For both strategies, the total expected cost is computed by estimating the number of times the strategy will monitor, z, and adding to $z\cdot$ *mon_cost* the agent's squared expected distance from the goal when the trial ends. In the periodic case, the lower bound on the total cost turns out to be a relatively simple function,

$$\text{cost}_{\text{periodic}} \geq \frac{D}{\text{period}}\text{mon\_cost} + \text{period} \cdot m^2 \qquad (3)$$

Minimizing for *period*, we get

$$\text{cost}_{\text{periodic}} \geq 2m\sqrt{D}\sqrt{\text{mon\_cost}} \qquad (4)$$

The upper bound on the cost for proportional reduction is a lot more complicated. It is computed by finding the bound on the cost for each epoch (monitor and move sequence), and summing over the number of times monitored. We get

$$\text{cost}_{\text{pr}} \leq \log_{\frac{1}{1-\text{prc}}} \frac{D}{\text{thresh}}(\text{mon\_cost} + 22 \cdot m^4 \frac{\text{prc}^2}{(1-\text{prc})^2}) + \text{thresh}^2 \tag{5}$$

In sum, the periodic cost is $\Omega(D^{\frac{1}{2}})$ and the proportional reduction cost is $O(\log(D))$. Proportional reduction will therefore beat periodic for a large enough D. The exact D value at which this occurs depends on m and *mon_cost*, and is typically in the range between 200 to 1000 distance units. For lower starting values, this proof does not show that proportional reduction is better. This does not mean, however, that the converse is actually true, and with tighter bounds on the costs, the minimal starting distance for which the proof holds can probably be reduced substantially.

5 A Taxonomy of Monitoring Scenarios and Strategies

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 experiment in section 4.2 was a small step in this direction: it analyzed the effects of two environmental factors, sensing error and starting distance, 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 dependent on each other? 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 have features like: 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 the above factors may influence the decision on what is the appropriate 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 change 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 [3]. This would imply that the factor subset cannot be very large.

An interesting way of looking at the difference between periodic and interval reduction is in terms of the amount 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 defined, 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. An example is monitoring for forest fires that occur with equal probability at any time. If the distribution of the event is known to the agent, it can precompute the optimal monitoring strategy.
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 for this is of course the cupcake problem. When to monitor again depends on the information returned by the sonar sensor, and also on the decision of whether or not to monitor *again* after the next movement cycle.

6 Summary

Monitoring is an important problem in agent design, one that is often overlooked. This paper has attempted to show that periodic monitoring is only the most simple of a whole range of different monitoring strategies. For many situations, in particular when the sensor returns more than just binary information, periodic monitoring is not necessarily the best strategy to use.

For virtually all instances of the cupcake problem, interval reduction appears to be superior to periodic. Using this example, we have shown how the environment can effect the strategy. We see this as the first step in constructing a monitoring strategy taxonomy. Many other factors will determine the choice of monitoring strategy, and we have listed some of the candidates. Determining which factors are relevant in a given scenario is still very much an open research question that we hope will be expanded on.

Acknowledgments

This research was supported by ARPA/Rome Laboratory under contract F30602-91-C-0076. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright notation hereon.

References

[1] 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*, pp. 36-41.

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

[3] Atkin, M. S., 1994. A Genetic Programming Approach to a Monitoring Strategy Taxonomy. Master's Thesis, University of Massachusetts at Amherst.

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

[5] 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.* MIT Press, Cambridge, MA.

[6] Doyle, R., Atkinson, D., & Doshi, R., 1986. Generating Perception Requests and Expectations to Verify the Execution of Plans. *AAAI-86*, pp. 81-88.

[7] Fikes, R., Hart, P., & Nilsson, N., 1972. Learning and Executing Generalized Robot Plans. *Artificial Intelligence* **3** (4), pp. 251-288.

[8] Firby, R. J., 1987. An Investigation into Reactive Planning in Complex Domains, *AAAI-87*, pp. 202-206.

[9] Georgeff, M. P., & Lansky, A. L., 1987. Reactive Reasoning and Planning, *AAAI-87*, pp. 677-682.

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

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

[12] Hansen, E. A., 1994. Cost-Effective Sensing During Plan Execution. *AAAI-94*, pp. 1029-1035.

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

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

[15] McDermott, D., 1978. Planning and Acting. *Cognitive Science* **2** (2), pp. 71-109.