Distributed Control Methods

Brian Tung*

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024

Leonard Kleinrock

Computer Science Deparment
University of California, Los Angeles
Los Angeles, CA 90024

Abstract

The distributed system is becoming increasingly popular, and this produces the need for more sophisticated distributed control techniques. In this paper, we present a method for distributed control using simple finite state automata. Each of the distributed entities is "controlled" by its associated automaton, in the sense that the entity examines the state of the automaton to determine its behavior. The result of the collective behavior of all of the entities is fed back to the automata, which change their state as a result of this feedback. We give a new method of analysis which derives the steady state behavior of this system as a whole, by decomposing it into two parts: describing and solving an imbedded auxiliary Markov chain, and analyzing the behavior of the system within each of the states of this auxiliary chain.

Key Words: *distributed algorithms, finite state automata, Markov chain, queueing theory, state aggregation*

1 Introduction

With the advent of powerful workstations, we have migrated from the centralized paradigm popular in the era of the large mainframe to a distributed paradigm that takes advantage of many cooperating processors available at a lower total cost. With this change, many new and exciting research problems have arisen in distributed systems. Typically, these problems, such as distributed communication and robot coordination, require the cooperation of several entities in performing a single task with little or no medium for control communication.

These problems have a common theme. In each, we wish a collection of entities to cooperate on a task which is most easily controlled centrally (that is, from "outside" the system). We would like the entities to perform this task without outside control, or in some cases, even without outside presence. We desire a self-contained control mechanism, capable of producing cooperation in the entities, with only a simple command from outside. In this paper, we develop such an efficient control scheme with the use of simple automata associated with each entity. These automata independently guide the entities, and take into account feedback that captures the composite effect of all the entities' actions.

Let us introduce this scheme with a simple game, called the Goore Game by Tsetlin, who describes it in [8]. Imagine that we have many players, none of whom are aware of the others, and a referee. Every hour, the referee asks each player to vote yes or no, then counts up the yes and no answers. A reward probability $r = r(f)$ is generated as a function of the fraction f of the players who voted yes. We assume that $0 \leq r(f) \leq 1$. A typical function is shown in Figure 1. Each player, regardless of how he voted, is

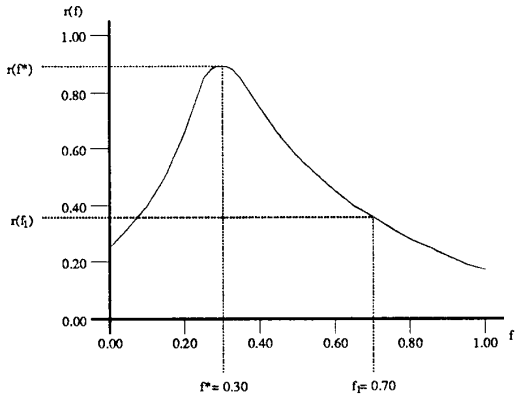

Figure 1: Typical Reward Function.

then independently rewarded (with probability r) or penalized (with probability $1-r$). For instance, let us suppose that at some point, the fraction of players voting yes was f_1. Then, the reward probability would be $r(f_1)$. Each player is then rewarded with probability $r(f_1)$. Note that the maximum of the example function occurs at $f^* = 0.3$. We can show the following: no matter how many players there are, we can construct automata such that exactly f^* of them (in this case, 0.3) vote yes—after enough trials—with a probability arbitrarily close to one. This property holds no matter what characteristics the function has—whether or not it is discontinuous, multimodal, etc. Note further that

*This work was supported by the Defense Advanced Research Projects Agency under grant MDA-972-91-J-1011, Advanced Networking and Distributed Systems.

0-8186-3900-8/93 $3.00 © 1993 IEEE

the individual automata know neither the fraction f nor the reward function $r(f)$.

Moreover, each player plays solely in a greedy fashion, each time voting the way that seems to give that player the best payoff. This is somewhat unexpected. Greed affects outcomes in an unpredictable manner. An example of greed leading to significantly suboptimal outcomes is the famous prisoner's dilemma [3]. In this scenario, two entities (the prisoners) greedily optimize their own behavior, but together they produce (for them) a globally suboptimal result. This effect is common in greedy solutions. However, we will see that the method used here does not have this property, because the players do not attempt to predict the behavior of the other players. Instead, each player performs by trial and error, and simply preferentially repeats those actions which produce the best result for that player.

Most of the control and coordination tasks in distributed systems cannot be taken care of in a straightforward manner, because the distributed systems have no leader, or anything of the sort—in fact, that's what makes them distributed! Even if they did, it would be hard to get a list of assigned tasks to all the members efficiently. Consider, for instance, the problem of communication on an Ethernet. It would be convenient if some machine could be given the task of asking every other machine if they had anything to say, and then drawing up a list of machines to transmit in order. However, there is no such machine in a distributed system. One could be elected somehow, *then* the lists could be generated and distributed. Unfortunately, the medium to be used for electing a leader and distributing the transmission lists is the communication channel itself! The very resource being used to do the allocation is also the resource that is being allocated. It would be helpful if the machines could organize themselves without explicitly communicating the lists to each other. The method outlined here allows them to do that.

In this paper, we examine the principles involved in stochastically "guiding" *one* automaton. We give a method that allows us to approximate the performance of these automata as a whole, without going into the exhaustive detail about their individual behavior that would render an analysis intractable.

2 Single Automaton Behavior

The automaton design we considerrelies on the same paradigm described for the Goore Game; that is, automata perform by trial and error in an attempt to maximize some reward probability. This is usually most applicable in the instance of many automata, but we first examine the single automaton case. This will form the basis of our examination of the many automata case in the next section.

Consider a single finite state automaton which is capable of two actions (outputs) A_0 or A_1. Suppose that every second, the current output is examined by an external agent, and based on that action, a reward probability is determined. If the output is A_0, the reward probability is $r = r_0$, and if the output is A_1, the reward probability is $r = r_1$. With probability r, the automaton is then rewarded; with probability $1-r$, it is penalized. The automaton may then change its state as a result of its reward (or penalty). The cycle is continually repeated: the automaton chooses either A_0 or A_1, the corresponding reward probability is determined, and the automaton is rewarded or penalized, etc. The automaton only knows that it takes some action, which in some way affects whether it receives a reward or a penalty. What sort of design can be postulated for an automaton that performs better than one that chooses A_0 or A_1 randomly with probability 1/2 each time?

One possible design is as follows. (This automaton is called $\mathcal{L}_{2,2}$ in [8].) Let the automaton have two states, 1 and -1. If the current state is -1, the automaton chooses A_0; if it is 1, it chooses A_1. If a reward results, the automaton stays in the same state; if a penalty results, it moves to the other state. Suppose that r_0 and r_1 are 0.4 and 0.8, respectively. Below, we show that this automaton, over the long run, will choose A_1 three times as often as A_0, regardless of which state it starts in. Note that this results in an average reward probability of 0.7, which exceeds the random choice whose average reward probability would be 0.6.

The equilibrium behavior of the automaton can be modeled as a Markov chain, where the external reward is transformed into an internal transition probability. Define π_i $(i = -1, 1)$ to be the steady state probability of finding the automaton in state i. Then, we get

$$\pi_1(1 - r_1) = \pi_{-1}(1 - r_0) \tag{1}$$

which in this example yields

$$0.2\pi_1 = 0.6\pi_{-1}$$

Since these are the only two states, we can also write that

$$\pi_1 + \pi_{-1} = 1$$

which gives us $\pi_1 = 0.75$ and $\pi_{-1} = 1 - 0.75 = 0.25$.

If the automaton has more than two states, the limiting proportion of time that the automaton chooses A_1 (in this example) increases and approaches unity asymptotically (which would yield an average reward probability of 0.8). Suppose that we have $2n$ states, $\{i, -i \mid 1 \leq i \leq n\}$. If the current state is negative, the automaton chooses A_0; if it is positive, it chooses A_1. If a reward results, the automaton stays in states n or $-n$ if it is in either of those states; otherwise, it moves from state i to $i + 1$ if i is positive, or from i to $i - 1$ if i is negative. If a penalty results, the automaton moves from state 1 to -1 or *vice versa*, if it is in one of those states; otherwise, it moves from state i to $i - 1$ if i is positive, or from i to $i + 1$ if i is negative. In general, the automaton moves away from the center if it wins, and toward the center if it loses. This behavior is summarized in Figure 2.

The equilibrium behavior of this automaton can also be modeled as a Markov chain. The resulting balance equations give us the following.

$$\pi_i = \pi_1 \left(\frac{r_1}{1 - r_1}\right)^{i-1} = \pi_1 \varphi_1^{i-1}, \quad i > 1$$

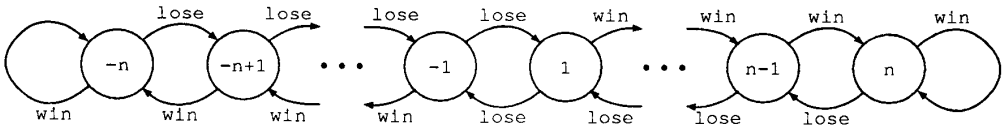

Figure 2: Automaton Design.

$$\pi_{-i} = \pi_{-1}\left(\frac{r_0}{1-r_0}\right)^{i-1} = \pi_{-1}\varphi_0^{i-1}, \quad i > 1$$

where $\varphi_0 = r_0/(1-r_0)$ and $\varphi_1 = r_1/(1-r_1)$. This in turn yields

$$\sum_{i=1}^{n} \pi_i = \pi_1 \cdot \frac{1-\varphi_1^n}{1-\varphi_1} \tag{2}$$

$$\sum_{i=1}^{n} \pi_{-i} = \pi_{-1} \cdot \frac{1-\varphi_0^n}{1-\varphi_0} \tag{3}$$

$$\pi_1(1-r_1) = \pi_{-1}(1-r_0) \tag{4}$$

Knowing again that the probabilities sum to 1, we can write

$$\sum_{i=1}^{n} \pi_i + \sum_{i=1}^{n} \pi_{-i} = 1$$

$$\pi_1 \cdot \frac{1-\varphi_1^n}{1-\varphi_1} + \pi_{-1} \cdot \frac{1-\varphi_0^n}{1-\varphi_0} = 1$$

$$\pi_1 \left(\frac{1-\varphi_1^n}{1-\varphi_1} + \frac{1-r_1}{1-r_0} \cdot \frac{1-\varphi_0^n}{1-\varphi_0}\right) = 1$$

$$\pi_1 = \left(\frac{1-\varphi_1^n}{1-\varphi_1} + \frac{1-r_1}{1-r_0} \cdot \frac{1-\varphi_0^n}{1-\varphi_0}\right)^{-1} \tag{5}$$

Substituting the values $r_1 = 0.8$ and $r_0 = 0.4$ in the above equations yields that the equilibrium probability of choosing A_1 is

$$\Pr(A_1) = \sum_{i=1}^{n} \pi_i = \frac{4^n - 1}{2 - 3(2/3)^n + 4^n} \tag{6}$$

This probability goes to 1 as $n \to \infty$. In fact, for any r_0, r_1, such that $r_1 > r_0$, equations 2 and 5 together show that the probability of choosing A_1 goes to 1 as $n \to \infty$. Similarly, if $r_0 < r_1$, the probability of choosing A_1 goes to 0 as $n \to \infty$. Simply put, as the memory size gets larger, the automaton chooses the best option with increasing certainty.

In this example, there is only one automaton attempting to behave optimally. If, instead, the reward probability is a function of the *aggregate* behavior of many automata, is it possible to design the automata such that similarly expedient behavior results, even if none of the automata may communicate directly with each other? This is what we examine in the next section.

3 Multiple Automata Behavior

In this section, we consider what happens when we have many automata, interacting only through the reward function. Specifically, the automata are rewarded based on the *fraction* of automata performing a certain action, and not on the *particular* automata performing that action. We wish to find the proportion of time that k of N automata perform a certain action. Hopefully, if we design the automata properly, they will collectively behave in a way such that they spend a large fraction of time near the maximum reward point.

Consider a population of N automata, $\{\alpha_1, \alpha_2, \ldots, \alpha_N\}$, each capable of exactly one of two actions, A_0 or A_1, at discrete moments in time ($t = 0, 1, 2, \ldots$). We call this population a *system* of automata. For all m, $1 \leq m \leq N$, let the output of automaton α_m at time t be represented by $A_{u_m(t)}$. Also, let $a(t)$ represent the number of automata with output A_1 and $f(t)$ the fraction of automata with output A_1 at time t. That is,

$$a(t) = \sum_{m=1}^{N} u_m(t) \tag{7}$$

and

$$f(t) = \frac{1}{N} a(t) \tag{8}$$

For each moment t, we compute a reward probability $r = r(f)$ whose value depends solely on the fraction $f = f(t)$ ($f = 0, 1/N, 2/N, \ldots, 1$). Each automaton then independently receives a stimulus $x_m(t)$, which is a binary valued (reward) random variable. It is either a reward (with probability r), or a penalty (with probability $1-r$). We assume that the automata know nothing of the reward function r or even of the existence of other automata.

Clearly, there exists a k^* such that $r(k^*/N) \geq r(k/N)$ for all k. Assume that k^* is unique. Let us find the limiting proportion of time that $a(t) = k^*$. First, define $\Phi(k)$ to be the limiting proportion of time that k automata have output A_1; that is,

$$\Phi(k) = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \zeta(a(t), k) \tag{9}$$

where the indicator $\zeta(x, y)$ is defined by $\zeta(x, y) = 1$ if $x = y$, and 0 otherwise. We then ask: is it possible to design (finite state) automata in such a way that $\Phi(k^*)$ can be made arbitrarily close to 1? The

answer is yes, although the behavior of the population is rather complex. The problem of the behavior in this context, essentially the Goore Game, has been examined by Tsetlin [8], but he only describes the construction and behavior of the automata, and does not develop a general method of analysis.

The automaton we use is the one defined in the previous section; the state diagram is displayed in Figure 2. The automaton is characterized by the memory size n—this size will be assumed to be the same for all automata in the population. For all m and t, let $s_m(t)$ be the state of automaton α_m at time t. We map states to outputs in a straightforward way. If $s_m(t) < 0$, then $u_m(t) = 0$; otherwise, $u_m(t) = 1$. The automaton is said to be *linear* [8]; that is, state transitions occur only between adjacent states, except for the self loops at the ends of the state space (n and $-n$). We use the mapping δ, where $\delta(s_m(t), x_m(t)) = s_m(t+1)$, to indicate that $s_m(t+1)$ is the state that results for automaton α_m when it is in state $s_m(t)$ and is subject to a stimulus (reward) $x_m(t)$.

We call this scheme the Goore Scheme. The motivation is to encourage behavior that produces a positive reward and to discourage behavior that produces a negative reward, that is, a penalty. We will show that for any population size N, it is always possible, with an appropriately large memory size n, to make $\Phi(k^*)$ arbitrarily close to 1.

Before we do so, however, let us consider the behavior of the system as the memory size of the automata and the number of automata increase without bound. Borovikov and Bryzgalov [1] show that when $n = 1$, that is, when the automata each have two states, the behavior is not optimal in the long run; in fact, with probability one, $f(t)$ approaches 1/2 in the limit as $N, t \to \infty$. This is undesirable since it does not depend on the nature of the reward function; the reward function might even have a minimum at $f = 1/2$! Their demonstration of this result uses transforms, and like any such demonstration, their analysis cannot easily be related back to the original physical situation. Therefore, we show this result intuitively as follows.

Lemma 1 *Suppose that the memory size for each automaton is $n = 1$. Suppose further that there exists some number $\Delta r > 0$ such that $\Delta r \leq r(f) \leq 1 - \Delta r$ for all f. Let $f_0 = \lim_{t\to\infty} f(t)$. Then $\lim_{N\to\infty} f_0 = 1/2$.*

Sketch of Proof As N increases without bound, it suffices to describe the entire set of automata with a single fraction $f(t)$. Associated with this fraction is a reward probability $r(f(t))$. From this probability we can derive the fraction $f(t+1)$.

$$f(t+1) = r(f(t))f(t) + (1 - r(f(t)))(1 - f(t)) \quad (10)$$

Suppose that $f(t) > 1/2$. Using the law of large numbers, we may assume that as N goes to infinity, the fraction of automata rewarded goes to *exactly* $r(f(t))$. The fraction $f(t+1)$ consists of the portion of $f(t)$ that was rewarded and the portion of $1 - f(t)$ that was penalized. Since $f(t) > 1/2$, we know that $f(t) > 1 - f(t)$, and therefore,

$$\begin{aligned} f(t+1) &= r(f(t))f(t) + (1 - \Delta r)(1 - f(t)) \\ &\quad - (r(f(t)) - \Delta r)(1 - f(t)) \\ &\geq r(f(t))f(t) + (1 - \Delta r)(1 - f(t)) \\ &\quad - (r(f(t)) - \Delta r)f(t) \\ &= \Delta r f(t) + (1 - \Delta r)(1 - f(t)) \end{aligned}$$

or

$$f(t+1) \geq 1 - f(t) + (2\Delta r f(t) - \Delta r)$$

Again, since $1 - f(t) < f(t)$, we also get (by a similar argument)

$$f(t+1) \leq (1 - \Delta r)f(t) + \Delta r(1 - f(t))$$

or

$$f(t+1) \leq f(t) - (2\Delta r f(t) - \Delta r)$$

In summary, from the condition on $r(f)$, we see that

$$\begin{aligned} 1 - f(t) + (2\Delta r f(t) - \Delta r) &\leq f(t+1) \\ &\leq f(t) - (2\Delta r f(t) - \Delta r) \end{aligned}$$

Therefore, we can conclude

$$\frac{|f(t+1) - 1/2|}{|f(t) - 1/2|} \leq 1 - \frac{\Delta r(2f(t) - 1)}{(f(t) - 1/2)} = 1 - 2\Delta r \quad (11)$$

On the other hand, suppose that $f(t) < 1/2$. Then, by the condition on $r(f)$, we can say that

$$\begin{aligned} f(t) + (\Delta r - 2\Delta r f(t)) &\leq f(t+1) \\ &\leq 1 - f(t) - (\Delta r - 2\Delta r f(t)) \end{aligned}$$

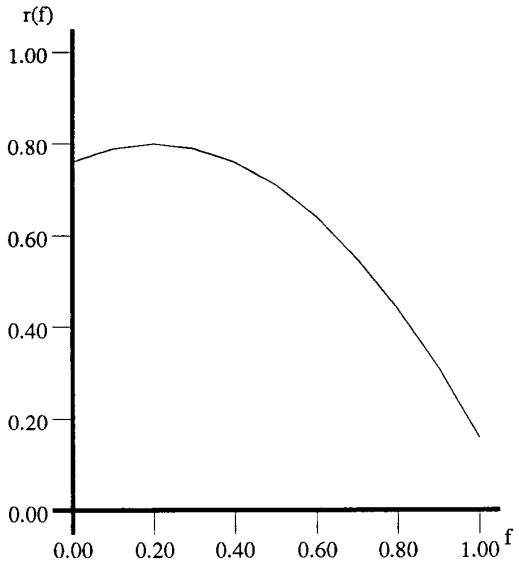

Figure 3: Example 1 Reward Function.

by analogy to the above analysis. Again, we can conclude

$$\frac{|f(t+1)-1/2|}{|f(t)-1/2|} \leq 1-\frac{\Delta r(2f(t)-1)}{(f(t)-1/2)} = 1-2\Delta r \quad (12)$$

Suppose finally that $f(t) = 1/2$. Then,

$$f(t+1) = (1/2)(r(1/2)+1-r(1/2)) = 1/2 \quad (13)$$

No matter what value $f(t)$ takes, $f(t+1)$ is closer to $1/2$ by at least a factor of $1-2\Delta r$. Therefore, $\lim_{N\to\infty} f_0 = 1/2$ (and the convergence factor is $2\Delta r$). □

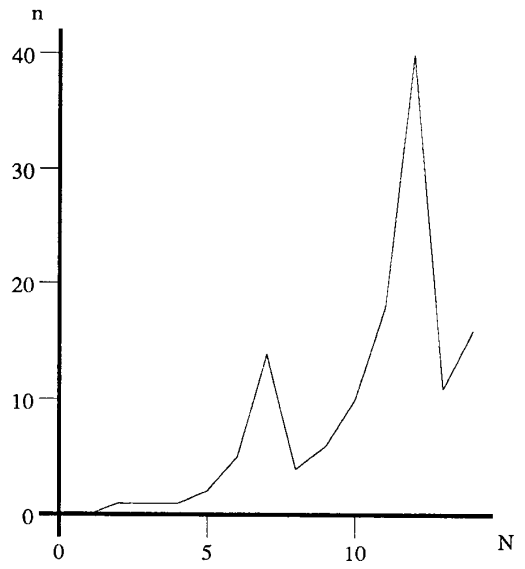

Figure 4: Increasing Memory Requirement To Maintain $\Phi(k^*) \geq 0.3$.

The corresponding exact analysis for $n > 1$ is extremely complex. However, based on simulations, and on the conclusions from the approximate analysis below in Section 3.1, we propose the following conjectures.

Conjecture 1 *For any value of N, there exists a value n_0 such that for all $n \geq n_0$,*

$$\lim_{t\to\infty} |f(t) - f^*| < \epsilon$$

This is to be distinguished from the similar

Conjecture 2 *For any value of n, there exists a value N_0 such that for all $N \geq N_0$,*

$$\lim_{t\to\infty} |f(t) - 1/2| < \epsilon$$

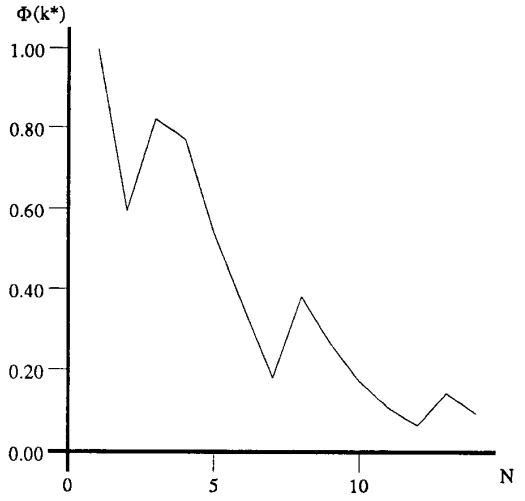

Figure 5: Decreasing Performance For Memory Size $n = 5$.

Conjecture 1 states that given any particular system with N automata, we can always set the memory size n high enough so that system operation is as close to optimal as desired. Conjecture 2 states conversely that given any memory size n for the automata, there is always some population size beyond which the system operates non-optimally; in fact, $f(t)$ again approaches $1/2$ asymptotically. Using the reward function $r(f) = -f^2 + 0.4f + 0.76$, shown in Figure 3, these two conjectures are illustrated through simulation in Figures 4 and 5. This function has a maximum at $f^* = 0.2$, so $k^* = 0.2N$, rounded off to the nearest integer. Suppose we would like $\Phi(k^*) \geq 0.3$. Figure 4 shows the increasing minimum memory size needed to maintain this level of performance. This illustrates Conjecture 1. Figure 5 shows $\Phi(k^*)$ decreasing as N increases, with memory size $n = 5$. This illustrates Conjecture 2. (The discontinuities in the graphs are due to the discrete jumps in the values of k^*; as noted above, k^* must be an integer. Thus, for instance, for N between 3 and 7, $k^* = 1$, but for N between 8 and 12, $k^* = 2$.)

3.1 State Aggregation

Our computer simulations illustrate these conjectures, but they do not explain why they seem to hold. Therefore, let us analyze this system as a Markov chain. The state space of the chain is an N-tuple whose m^{th} element, s_m, represents the state of α_m; the state is denoted by $\vec{s} = (s_1, s_2, \ldots, s_N)$. There are thus $(2n)^N$ states in the Markov chain. Let $\phi(\vec{s})$ for a Markov chain state $\vec{s}$ represent the number of positive elements in $\vec{s}$. We can then write the transition probabilities as follows.

$$q(\vec{s}, \vec{s'}) = \prod_{m=1}^{N} \Pr(x_m) \tag{14}$$

where s'_m for all m is the result of the mapping $\delta(s, x)$ defined above; that is, $s'_m = \delta(s_m, x_m)$, where

$$\Pr(\text{reward}) = r(\phi(\vec{s})/N)$$

and

$$\Pr(\text{penalty}) = 1 - r(\phi(\vec{s})/N)$$

Define $P(\vec{s})$ to be the equilibrium state probability for the state $\vec{s}$. Then, we can write the detailed balance equations.

$$P(\vec{s}) = \sum_{\vec{s'}} P(\vec{s'}) q(\vec{s'}, \vec{s}) \tag{15}$$

Knowing that the $P(\vec{s})$ sum to 1, we can solve for $P(\vec{s})$. Then,

$$\Pr(k \text{ automata are on}) = \sum_{\phi(\vec{s})=k} P(\vec{s}) \tag{16}$$

Unfortunately, solving a Markov chain with $(2n)^N$ states is far from trivial, and the solution would only give us a set of probabilities, with no description of the dynamics of the system. Therefore, we choose to simplify (and thus approximate) the analysis by aggregating sets of states of the Markov chain. We implicitly assume that the behavior of the system is more or less the same in each of the states that make up any particular aggregate state.

Assume for the moment that $r(f) > 1/2$ for all f, so that at any time, most of the automata are in the extreme states, that is, near n or $-n$. This means that $f(t)$ is relatively stable; since the automata are at or near the end most of the time, state transitions between -1 and 1 (let us call these "trigger transitions," since they "trigger" changes in $a(t)$, and hence, $f(t)$) are relatively rare, and we can assume with little loss of precision that at most one trigger transition can take place at a time. For that reason, we call this the *well-behaved* case. (By contrast, we define *ill-behaved* systems to be those with $r(f) < 1/2$ for some f, and these are more difficult to analyze, so their treatment has been deferred. Qualitatively, however, the optimal behavior of these systems is still the same.) This characteristic will become important when we estimate the lengths of the intervals between trigger transitions. Suppose that out of the N automata, k are currently on the positive side of the state space. Note that with the exception of the sign of its current state, the dynamic behavior of an automaton is the same on either side of the state space. In other words, there is no way to distinguish between an automaton and its "mirror image." All other factors being equal, any one of the k automata on the positive side is just as likely to make the first trigger transition as any one of the $N - k$ on the negative side. Therefore, the probability that the next trigger transition goes from 1 to -1 (from positive to negative) is k/N, and the probability that it goes from -1 to 1 (from negative to positive) is $(N - k)/N$. The former corresponds to a decrease in $a(t)$ by one, and the latter to an increase in $a(t)$ by one. This suggests constructing the Markov chain in Figure 6, where the states represent the various values that $a(t)$ can take, rather than the various states of any particular automata. To avoid confusion, we call the former *system states*, and the latter *automaton states.*

This chain does not represent the sequence of system states at each discrete moment in time, but is rather an *imbedded Markov chain* which represents the sequence of system states at the instants just after trigger transitions. We can solve this chain for the equilibrium system state probabilities $\Pi(k)$, and in the following lemma, we show that it has the solution

$$\Pi(k) = 2^{-N} \binom{N}{k} \tag{17}$$

Lemma 2 *Suppose that in any system state, any automaton is just as likely as any other to make a trigger transition. Then the above transition probabilities are valid, and*

$$\Pi(k) = 2^{-N} \binom{N}{k}$$

is the solution to the imbedded Markov chain.

Sketch of Proof Using the assumption in the statement of the lemma, we can write the following balance equation.

$$\Pi(k) = \Pi(k-1)\left(\frac{N-k+1}{N}\right) + \Pi(k+1)\left(\frac{k+1}{N}\right) \tag{18}$$

In addition, we require that

$$\sum_{k=0}^{N} \Pi(k) = 1$$

It is then a simple matter of algebra to show that the solution to this system of equations is

$$\Pi(k) = 2^{-N} \binom{N}{k}$$

□

$\Pi(k)$ represents the visit ratios to the various system states, normalized to sum to unity. It has a maximum at $k = N/2$, so the system makes the most visits to that system state. $\Phi(k)$, however, depends not only on $\Pi(k)$, but also on the average time spent in that system state per visit. (For more on this general method, see Appendix B.) We now make an estimate of this average time.

Given any memory size n, we define the persistence time $\tau_n(k)$ to be the average time that the population

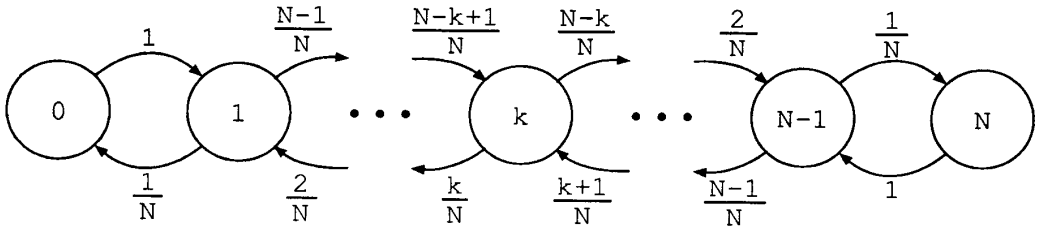

Figure 6: System State Diagram.

stays in system state k. The proportion of time that the population spends in system state k is then

$$\Phi(k) = \frac{\Pi(k)\tau_n(k)}{\sum_{k'=0}^{N} \Pi(k')\tau_n(k')} \tag{19}$$

Our object now is to estimate the persistence time $\tau_n(k)$.

Let $\bar{t}^n_{i,j}(r)$ be the average amount of time it takes for an automaton under reward probability r to move from automaton state i to automaton state j, with $1 \leq j < i \leq n$. (We can restrict our discussion to the positive side because the behavior on the negative end is identical, by symmetry, as discussed above.) When $i = n$, that is, at the end of the automaton state space,

$$\begin{aligned} \bar{t}^n_{n,n-1}(r) &= (1-r) + 2r(1-r) + 3r^2(1-r) + \cdots \\ &= 1 + r + r^2 + \cdots \\ &= \frac{1}{1-r} \end{aligned} \tag{20}$$

If, on the other hand, the automaton is currently at state i where $1 \leq i < n$, then we reason as follows. It will take the automaton at least one time unit to get to state $i-1$. After the first time unit, either it has actually moved down to state $i-1$ (with probability $1-r$), or it has moved up to state $i+1$ (with probability r), in which case it must first move back to state i before it can move to state $i-1$. This gives us the recurrence equation

$$\bar{t}^n_{i,i-1}(r) = 1 + r(\bar{t}^n_{i+1,i}(r) + \bar{t}^n_{i,i-1}(r)), \quad 1 \leq i < n$$

which can be rewritten as

$$(1-r)\bar{t}^n_{i,i-1}(r) = 1 + r\bar{t}^n_{i+1,i}(r), \quad 1 \leq i < n \tag{21}$$

This recurrence equation can be solved by the usual z-transform techniques [4] (see Appendix A) to yield

$$\bar{t}^n_{i,i-1}(r) = \frac{1}{1-2r}\left(1 - \left(\frac{r}{1-r}\right)^{n-i+1}\right) \tag{22}$$

for $1 < i \leq n$, and

$$\bar{t}^n_{1,-1}(r) = \frac{1}{1-2r}\left(1 - \left(\frac{r}{1-r}\right)^{n}\right) \tag{23}$$

Immediately after a trigger transition, at least one of the automata—in particular, the one that made the trigger transition—must be in either state 1 or -1. Therefore, for sufficiently large n, the time this particular automaton takes to make a trigger transition *back* to the other side approximates the time before a trigger transition by *any* automaton. (When n is too small, the likelihood of multiple trigger transitions become high, and this invalidates the estimates of the persistence times.) This is true even though this automaton may not be the same as the one that made the last trigger transition. We can see this in two boundary cases for well-behaved systems. When r is close to $1/2$, there are many automata distributed evenly across the automaton state space, and any of these is as likely as any other to make a trigger transition. When r is close to 1, on the other hand, the likelihood is great that the automaton that last made a trigger transition will soon be in one of the end states, and again, any automaton is as likely as any other to make a trigger transition. We therefore claim that $\bar{t}^n_{1,-1}(r)$ is a good approximation to $\tau_n(k)$, and denote our estimate by

$$\bar{\tau}_n(k) = \bar{t}^n_{1,-1}(r(k/N)) \tag{24}$$

so the proportion of time spent in system state k is approximately

$$\Phi(k) \doteq \frac{\Pi(k)\bar{\tau}_n(k)}{\sum_{k'=0}^{N} \Pi(k')\bar{\tau}_n(k')} \tag{25}$$

We have now established the following approximation.

Approximation 1 *Assume that any automaton is just as likely as any other to make a trigger transition. Also assume that the persistence times are approximately proportional to the average travel time $\bar{\tau}_n(k)$. Then an approximation to the limiting probability that the system is in state k is*

$$\Phi(k) \doteq \frac{\binom{N}{k}\frac{1}{1-2r(k/N)}\left(1 - \left(\frac{r(k/N)}{1-r(k/N)}\right)^n\right)}{\sum_{k'=0}^{N}\binom{N}{k'}\frac{1}{1-2r(k'/N)}\left(1 - \left(\frac{r(k'/N)}{1-r(k'/N)}\right)^n\right)} \tag{26}$$

This approximation gives some insight into the Conjectures above. We see that the equilibrium system state probabilities are simply weighted binomial

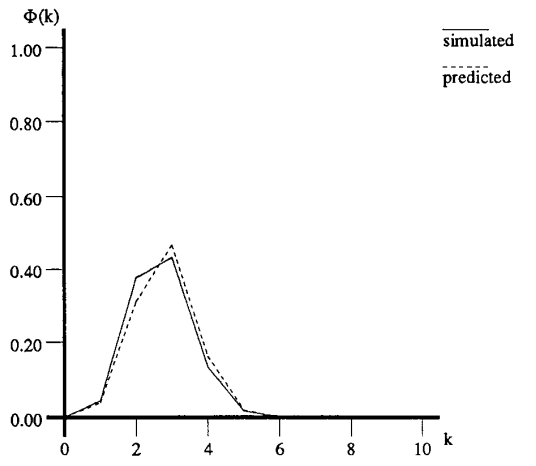

Figure 7: Steady State Probability Distribution for Example 1.

coefficients, where each weight is the persistence time associated with that system state. Suppose we have a reward function whose peak is at some f^* not equal to $1/2$. If we hold the population size N constant, and increase the memory size n, the persistence time for $k^* = f^*N$ will become larger and larger. Eventually, the system will spend most of its *time* at k^*, even though it makes more *visits* to the system state $k = N/2$, where the binomial coefficient is the greatest. This justifies Conjecture 1. If, instead, we hold the memory size n constant, and increase the population size N, the visit ratios to the system states in the vicinity of $k = N/2$ will grow larger and larger. Eventually, they will become so large that the greater persistence time for $k^* = f^*N$ is not enough to overcome the number of visits to $k = N/2$, and the system will spend most of its time around $k = N/2$. This justifies Conjecture 2.

We give two examples of systems of automata to show how well the approximation performs. In Example 1, there are $N = 10$ automata, each with a memory size of $n = 10$. The reward function is $r(f) = -f^2 + 0.4f + 0.76$, shown above in Figure 3. Recall that this function has a peak at $f^* = 0.2$, and that the optimal value of k^* is then 2. This example gives a flavor for the distribution of the fraction of time spent in various system states with a relatively small memory size, even when the reward function is relatively smooth. Figure 7 shows the estimated and simulated proportion of time spent in the various system states for these parameters. Note especially that even though some of the states have a reward probability less than $1/2$, the approximation is still accurate.

In Figure 8, we show the calculated steady state probabilities for Example 1 for various memory sizes n. Note that the system operates better and better as the memory size increases; in particular, the peak of the curve moves from $k = N/2$ for small values of n toward the optimal value $k = 2$ for large values of n.

In Example 2, there are $N = 10$ automata, each with a memory size of $n = 3$. The reward function is $r(f) = 0.9$ if $f = 1/N$ (0.1 in this case), and $r(f) = 0.6$ otherwise. Figure 9 shows the estimated and actual proportion of time spent in the various system states. Note that the probabilities are simply normalized binomial coefficients, except at $k = 1$ (that is, at $f = f^* = 0.1$), where it is much higher.

4 Other Considerations

So far we have only considered how much time, on average, the system spends in the optimal (or any other) system state. In any application, however, another parameter of interest is how long the system takes to *get* to the optimal condition. Since the system does not operate in equilibrium, we cannot speak of an exponential (or the like) convergence. Instead, we define the *walk period* ω^{-1} to be the average time between successive visits to the optimal system state, not counting the time actually spent in that optimal state. From the above discussion, it is clear that

$$\omega^{-1} = \frac{1}{\Pi(k^*)} \sum_{k \neq k^*} \Pi(k) \tau_n(r(k/N))$$

which we can write more concisely as

$$\omega^{-1} = \frac{1 - \Phi(k^*)}{\Phi(k^*)} \tau_n(r(k^*/N)) \qquad (27)$$

where $\Phi(k^*)$ is estimated from the above result. For a typical application, we usually want to maximize $\Phi(k^*)$ and to minimize ω^{-1}. Heuristics to do this can be found in [2] and [5].

Because of the possibly large walk periods, we might consider why we should use the Goore Game as a paradigm in these problems at all. If the reward function is known, why not use the value of f^* as a sort of probabilistic coin flip to determine the action of each automaton, since there is no walk period at all? The reason is that the Goore Scheme not only increases the likelihood that the percentage of automata performing a certain action is optimal when measured over a long period of time, it also increases that likelihood for each instant in time. For instance, suppose we were to take the above suggestion, and give each automaton a probability of f^* of performing action A_1. Then the probability that out of N automata, exactly k^* of them perform action A_1 is approximately

$$\begin{aligned} \Phi(k^*) &= \binom{N}{k^*} f^{*k^*} (1 - f^*)^{N-k^*} \\ &= \binom{N}{k^*} f^{*Nf^*} (1 - f^*)^{N(1-f^*)} \end{aligned}$$

which can be shown to be less than or equal to $1/2$ for $N \geq 2$. But as we have seen above, with a large enough memory size n, the probability that exactly

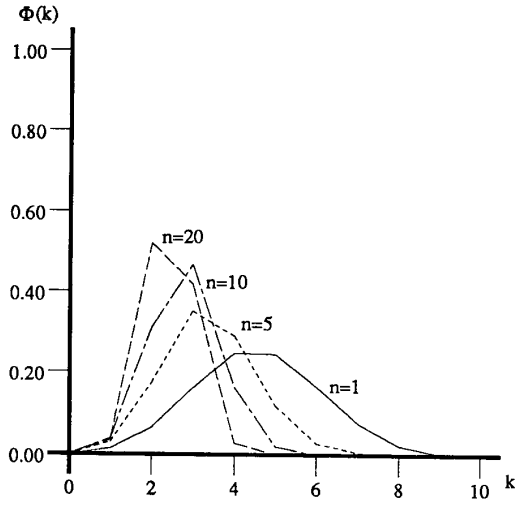

Figure 8: The Effect of Various Memory Sizes on Distribution.

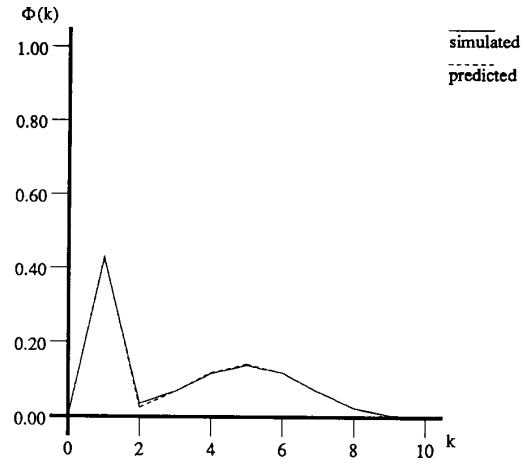

Figure 9: Steady State Probability Distribution for Example 2.

k^* automata perform action A_1 can be made as close to one as desired (given that one is willing to accept a correspondingly high premium on the walk period). Moreover, if the reward function changes, then the automata will adapt themselves automatically to adapt to the new function. This is not true when the automata are hard-wired to respond to a particular reward function. Most real life problems tend to create varying situations and this scheme allows the participants to react dynamically to these variations.

5 Summary

We have examined the problem of how to design automata so that they may work together cooperatively to achieve a common goal. We have taken a large class of systems, namely, the well-behaved systems, and derived a simple, quickly evaluated formula for the equilibrium system state probability distribution, which is approximate, but close enough for most purposes. We have described some other characteristics of these systems which also impact on the behavior of the systems, and given formulas for computing these parameters given the steady state probability distribution.

In future work, we plan to investigate the application of this scheme to the solution of real world problems. We intend to characterize the space of problems that are solvable by this method, and that are furthermore difficult to solve by other methods. We also propose to give specific solutions to various standard problems, and to detail necessary modifications to this scheme. We expect that there will be a wide range of tasks for which this technique is applicable.

A Solving for Persistence Time

In Section 3.1, we claimed that the solution to the recurrence equations 20 and 21 could be derived using z-transform techniques [4] to yield equations 22 and 23. In this appendix, we carry out this derivation. For any given n and r, we define

$$u_j = \bar{t}^n_{n-j,n-j-1}(r)$$

Then, from equations 20 and 21, we can write

$$u_0 = \frac{1}{1-r}$$

$$(1-r)u_{j+1} = 1 + ru_j$$

We define

$$U(z) = \sum_{j=0}^{\infty} u_j z^j$$

intending to discard any values of u_j for $j \geq n$. We multiply the recurrence equation above by z^j and sum from $j = 1 \rightarrow \infty$, and we get

$$\sum_{j=0}^{\infty}(1-r)u_{j+1}z^j = \sum_{j=0}^{\infty} z^j + ru_j z^j$$

$$\frac{1-r}{z}(U(z) - u_0) = \frac{1}{1-z} + rU(z)$$

$$\frac{1-r}{z}U(z) - \frac{1}{z} = \frac{1}{1-z} + rU(z)$$

$$\frac{1-r-rz}{z}U(z) = \frac{1}{z(1-z)}$$

$$U(z) = \frac{1}{(1-z)(1-r-rz)} = \frac{\frac{1}{1-2r}}{1-z} - \frac{\frac{r}{(1-r)(1-2r)}}{1-\frac{r}{1-r}z}$$

which we invert to get

$$u_j = \frac{1}{1-2r}\left(1 - \left(\frac{r}{1-r}\right)^{j+1}\right)$$

Using the substitution $j = n - i$, we get the equation claimed in the main text.

B System Decomposition

In the analysis presented in this paper, we decompose the system behavior into two parts: visit ratios and persistence time. Volkonskiy [9] makes use of this general method, but only for the simple case where the reward function is of the form $r(f) = r_0$ for $f \leq f_c$, $r(f) = r_1 < r_0$ for $f > f_c$, where $f_c < 1/2$ is some critical value. He also requires $r_1 > 1/2$. He then shows that for optimality, it is required that $\lim_{N\to\infty}(n/N) > \chi$, where

$$\chi = \frac{1 - H(f_c)}{\lg(\varphi_0/\varphi_1)}$$

and

$$H(f_c) = f_c \lg(1/f_c) + (1-f_c)\lg(1/(1-f_c))$$

where $\varphi_i = r_i/(1-r_i)$. If $\lim_{N\to\infty}(n/N) < \chi$, then the automata spend most of their time in nonoptimal system states. Pittel [7] examines nearly the same problem. The only change is in the automata; he assumes that in a trigger transition, the automaton has an equal chance (that is, 1/2) of taking the other action, or staying with the same action. Using a different method of analysis, he comes to the same conclusion. He adds that "from the set of best decisions [which are not unique, since the reward function is piecewise constant], the automata choose one that allows them the most rapid detection of any disadvantages resulting from any deviations from that [choice]."

References

[1] V. A. Borovikov and V. I. Bryzgalov. A simple symmetric game between many automata. *Avtomat. Telemekh.*, Vol. 26(No. 4), 1965.

[2] A. Giessler, J. Hanle, A. Konig, and E. Pade. Free buffer allocation—an investigation by simulation. *Computer Networks*, Vol. 1(No. 3):191–204, July 1978.

[3] Douglas R. Hofstadter. Metamagical themas. *Scientific American*, May 1983.

[4] Leonard Kleinrock. *Queueing Systems, Volume 1: Theory.* John Wiley and Sons, 1975.

[5] Leonard Kleinrock. On flow control in computer networks. In *International Conference on Communications*, June 1978.

[6] Leonard Kleinrock. On distributed systems performance. In *ITC Specialist Seminar: Computer Networks and ISDN Systems*, pages 209–216, 1990.

[7] B. G. Pittel. The asymptotic properties of a version of the Goore game. *Probl. Peredachi Inform.*, Vol. 1(No. 3), 1965.

[8] M. L. Tsetlin. *Finite Automata and Modeling the Simplest Forms of Behavior.* PhD thesis, V. A. Steklov Mathematical Institute, 1964.

[9] V. A. Volkonskiy. Asymptotic properties of the behavior of simple automata in a game. *Probl. Peredachi Inform.*, Vol. 1(No. 2), 1965.