Using Finite State Automata to Produce Self-Optimization and Self-Control

Brian Tung and Leonard Kleinrock

Abstract—A simple game provides a framework within which agents can spontaneously self-organize. In this paper, we present this game, and develop basic theory underlying a robust method for distributed coordination based on this game. This method makes use of finite state automata–one associated with each agent–which guide the agents. We give a new, general method of analysis of these systems, which previously had been studied only in limited cases. We also provide a physical example, which should hint at the type of problems resolvable using this method.

Index Terms—Approximation, decomposition, finite state automata, optimization, random walk, robotics, state aggregation.

1 Introduction: The Gur Game

In many of the problems in distributed systems, we wish a collection of agents to cooperate on a task which is most easily controlled centrally (that is, from "outside" the system). In other words, we desire a mechanism capable of producing cooperation in the agents, with only a simple command from outside. In this paper, we develop such a mechanism using finite state automata associated with each agent. These automata independently "guide" the agents, while taking into account feedback that captures the composite effect of all the agents' actions. Ramadge and Wonham [5], [9] give a similar treatment by means of discrete event decision systems (DEDS), but the conditions differ, as will be detailed below.

We introduce this scheme with a simple game, called the Gur Game by Tsetlin [6]. Imagine that we have N players, none of whom are aware of the others, and a referee. Each turn, 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 $0 \le r(f) \le 1$ for all f, $0 \le f \le 1$. A typical function is shown in Fig. 1. Each player, regardless of how he voted, is then independently rewarded with probability r, or penalized with probability $1 - r$. In general, the individual players know neither the fraction f nor the reward function $r(f)$. Let us suppose that at one turn, the fraction of players voting yes is f_1. Then, the reward probability is $r_1 = r(f_1)$. Each player is then rewarded with probability r_1, or penalized with probability $1 - r_1$.

The maximum of the reward function in Fig. 1 occurs at $f^* = 0.3$. We can show the following: no matter how large the population size N, we can construct finite state automata such that exactly f^* of them vote yes—after enough trials—with a probability arbitrarily close to one, as long as $r(f^*) > 0.5$. This property holds no matter what other characteristics the function has—whether or not it is discontinuous, multi-modal, etc. 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. Typically, greed leads to significantly suboptimal outcomes; an example of this is the prisoner's dilemma [2]. However, we will see that the method used here does not have this property, because each player effectively evaluates the success of an action *jointly* with the actions of the other players, rather than *conditioning* it on those actions.

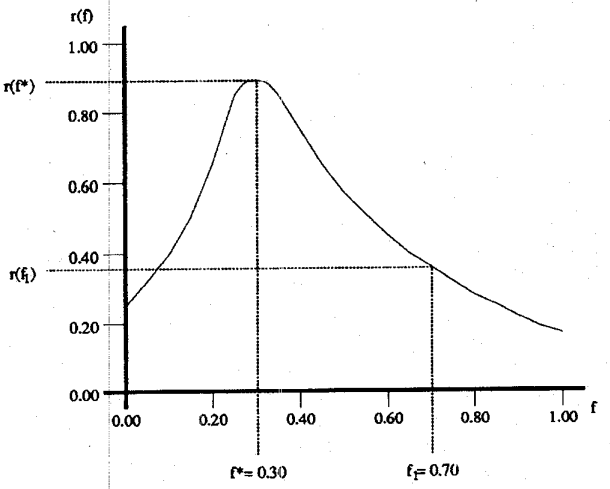


Fig. 1. Typical reward function.

Strictly speaking, the Gur Game concerns optimization and not distributed control. However, control often reduces to maximizing a desirable quantity; often, one does not care about the mode of control, as long as the results are desirable. For instance, when performing load balancing, it is not necessary to demand explicitly that a job be transferred from this machine to that one; that the performance is improved is all that matters. The distributed aspect seems

• *L. Kleinrock is with the Computer Science Department, University of Los Angeles, Los Angeles, CA 90024-1596. E-mail: lk@cs.ucla.edu.*
• *B. Tung is with the Information Sciences Institute, University of So. California, Los Angeles, CA. E-mail: Brian@isi.edu.*

Manuscript received Jan. 13, 1993; revised Oct. 30, 1994.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number D95087.

prevented by the requirement of a referee, but we shall see that the referee can in fact be replaced by a globally observable quantity, such as (in some cases) utilization.

Many of the tasks in distributed systems must be handled without explicit coordination, because the systems, being distributed, have no leader. Consider, for instance, the problem of communication on a shared channel. If one machine were given the task of polling the other machines, then it could make a list of machines to transmit in order. However, in a distributed system, there is no such machine a priori, and the only medium 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.

Therefore, it is necessary for the machines to organize themselves without explicitly communicating control information. (Notice that we do not prohibit the machines from making deductions about each other's state, based on the state of the channel.) The method outlined here allows them to do that. (In [4], we find mentioned the inherent cost in having the agents physically separated, due to the lack of *global knowledge*. In our approach, that cost is not being sidestepped; it is merely being distributed among the agents.)

The remainder of this paper is organized as follows. In Section 2, we examine the principles involved in using an automaton to guide *one* agent. In Section 3, we extend these principles to deal with *many* automata. We give a method that allows us to approximate the performance of these automata as a whole, without going into exhaustive detail about their individual behavior. In Section 4, we give an example of a physical system that can be designed with the aid of these automata. Finally, Section 5 summarizes our accomplishments and previews future research.

2 Single Automaton Behavior

The automaton design we consider assumes the paradigm described for the Gur Game; that is, automata perform by trial and error in an attempt to maximize some reward probability. We first examine the single automaton case. This will form the basis of our examination of the multiple automata case in the next section.

Consider a single finite state automaton which is capable of two outputs, A_0 or A_1. Suppose that at each discrete moment in time ($t = 0, 1, 2, \ldots$), the current output is monitored, and based on that output, a reward probability is determined. If the output is A_0, the reward probability is $r = r_0$; if it is A_1, the reward probability is $r = r_1$. (We assume we are given $r_0, r_1 \in [0, 1]$.) Then, with probability r, the automaton is rewarded; with probability $1 - r$, it is penalized. This cycle is repeated: the automaton chooses either A_0 or A_1, the corresponding reward probability is determined, and the automaton is rewarded or penalized. The automaton knows only that its output in some way affects whether it receives a reward or a penalty. We wish to devise an automaton that performs "well" (that is, it receives a greater proportion of rewards), where performance is measured relative to that of a "null" automaton, which simply chooses A_0 or A_1 randomly with probability 1/2.

Tsetlin [6] gives the following design, which he calls $L_{2,2}$. 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. The goal of this design is to encourage behavior that produces a reward, and discourage behavior that produces a penalty.

The steady state behavior of the automaton can be modeled as a Markov chain, where the external reward probabilities are represented by internal transition probabilities. Define π_i ($i = -1, 1$) to be the steady state probability of finding the automaton in state i. With the usual methods of analysis, we find that this automaton, over the long run, chooses A_1 three times as often as A_0. This results in an average reward probability of 0.7, compared to the null automaton, whose average reward probability is 0.6.

To improve performance, we can give the automaton more than two states. Suppose that it has $2n$ states,

$$S_n = \{i, -i \mid 1 \le i \le n\}.$$

This automaton is then said to have a *memory size* of n. (Tsetlin [6] calls this automaton $L_{2n,2}$.) If the current state is negative, the automaton chooses A_0; if it is positive, it chooses A_1.

If the automaton receives a reward, it 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 it receives a penalty, it 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. Roughly, the automaton moves away from the center if it wins, and toward the center if it loses. As before, this design encourages behavior that produces a reward, and discourages behavior that produces a penalty. This design is summarized in Fig. 2 below.

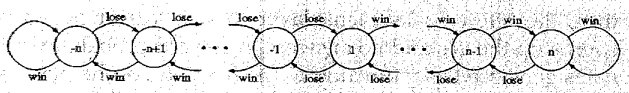


Fig. 2. Automaton design.

The behavior of this automaton can also be modeled using a Markov chain, with the balance equations

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

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

$$\pi_i \left(1 - r_1\right) = \pi_{-1}\left(1 - r_0\right)$$

where $\varphi_0 = r_0/(1 - r_0)$ and $\varphi_1 = r_1/(1 - r_1)$. Knowing that the probabilities sum to 1, we see that

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

Substituting the values $r_1 = 0.8$ and $r_0 = 0.4$ in the above equations, we find that the equilibrium probability of choosing A_1 in this example is

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

This expression yields 3/4 when $n = 1$, confirming our previous analysis. As $n \to \infty$, on the other hand, this probability goes to 1. In fact, our derivation shows that for any r_0, r_1, such that $r_1 > r_0$ and $r_1 > 1/2$, the probability of choosing A_1 goes to 1 as $n \to \infty$. In other words, as the memory size gets larger, the automaton chooses the best option with increasing certainty.

Here, there is only a single automaton, acting in isolation. This is basically the scenario developed by Ramadge and Wonham in their articles on DEDS [5], [9]. However, while they do account for the possibility of nondeterminacy in the feedback mechanism, there is no explicit mention of a reward function, nor is there an attempt to optimize behavior in that context. Furthermore, even though it is possible to use the "supervisory" automata in these papers to guide many separate agents, there is no essential difference in the treatment of this case; that is, it is assumed that there is no cooperation between the agents. In the next section, we examine whether it is possible to design automata that perform well *together* when the reward probability is a function of the *aggregate* behavior of many automata, even if none of the automata may communicate directly with each other.

3 Multiple Automata Behavior

We now consider a population of N automata that share only a common reward function. The automata are rewarded based on the *fraction* of automata producing a certain output (say, A_1), and not on those *particular* automata. We desire that the automata behave in such a way that they maximize their collective reward.

Consider a population of N automata $\{\alpha_1, \alpha_2, \ldots, \alpha_N\}$, each capable of two outputs, 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, $1 \le m \le N$, let the output of automaton α_m at time t be represented by $u_m(t)$. 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.

We assume the existence of a reward function $r(f)$, defined by $r[0, 1] \to [0, 1]$. For each moment t, we compute a reward probability $r = r(f)$, whose value depends solely on the fraction $f = f(t)(f = 0, 1/N, 2/N, \ldots, 1)$. Even though f can only take on one of $N + 1$ values, the function $r(f)$ is defined as accepting a continuous domain of values, both for historical reasons [6], and because it allows for changes in the population. (We do not analyze the case of a changing population in this paper, however.) Each automaton then independently receives a stimulus $x_m(t)$, which is a binary valued random variable. It is either a reward (with probability r), or a penalty (with probability $1 - r$). We do not assume that the automata know the reward function $r(f)$.

Clearly, there exists at least one k^* such that $r(k^*/N) \ge r(k/N)$ for all k. Assume that k^* is unique. Define $\Phi(k)$ to be the limiting proportion of time that k automata have output A_1; that is,

$$\Phi(k) = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \zeta(a(t), k) \tag{3}$$

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 $\Phi(k^*)$ is arbitrarily close to 1? (If k^* is not unique, then we simply sum over all optimal k.) The answer is yes, as noted by Tsetlin [6], although he only describes the construction and behavior of the automata, and does not develop a general method of analysis.

The automaton we examine is the one defined in the previous section; the state diagram is displayed in Fig. 2. We assume that all automata have the same memory size n. 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 $u_m(t) = A_0$; otherwise, $u_m(t) = A_1$. The automaton is said to be *linear* [6]; that is, state transitions occur only between adjacent states, except for the self-transitions at states n and $-n$. We define the mapping $s' = \delta(s, x)$ to indicate that an automaton moves to state s' when it starts in state s and receives a stimulus x.

The behavior of this system can be viewed as that of a random walk on the space S_n^N. Refer to Fig. 3 where for the sake of simplicity we consider the simple case $N = 2$, $n = 3$. Note that f can then take only three different values—0, 1/2, and 1.

Consider first one automaton in isolation. Its behavior resembles that of a random walk on the space S_3. However, the transition probabilities are not well-defined, because they depend on the state of the other automaton. The state of an individual automaton does not contain sufficient information to determine the reward probability.

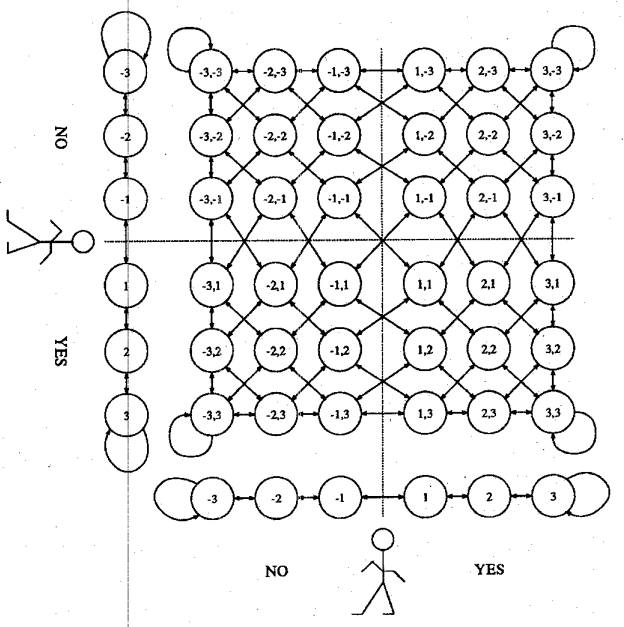


Fig. 3. Illustration of random walk.

Now consider the two automata in conjunction. If we define the state of the population as a whole to be an ordered

pair consisting of the states of the automata, then the behavior of the population *can* be viewed as a true random walk (on the space $S_3^2 = S_3 \times S_3$). The state now contains enough information to determine f, and in turn, $r(f)$. Hence, all the transition probabilities are well-defined.

We make the following important observation. In the simple case $N = 2$, we can divide the state space S_n^2 into four quadrants. In any given quadrant, all states have the same transition probabilities (with the exception of the border states). If the reward probability $r < 1/2$, there is a bias toward the "origin"—the point where the four quadrants meet. Conversely, if the reward probability $r < 1/2$, there is a bias away from the origin. The further r is from $1/2$, the stronger this bias. A similar assertion can be made for the case of general N: we divide the state space S_n^N into 2^N quadrants. This random walk view should illustrate that the system does not converge or get "stuck" in local optima. Instead, the population is constantly "in motion," responding to the reward function. With this observation in mind, we will be able to appreciate more intuitively the significance of our analysis below.

3.1 Previous Work: Asymptotic Behavior

First, however, let us first describe 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 have two states—the behavior is not optimal; 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 reward function at all; the reward function might even have a minimum at $f = 1/2$. Their demonstration of this result uses transforms; we instead show this result intuitively as follows.

LEMMA 1. *Suppose that the memory size for each automaton is $n = 1$. Suppose further that $r(f)$ is continuous, and that there exists some number $\Delta r > 0$ such that $\Delta r \le r(f) \le 1 - \Delta r$ for all f. Also, assume that all automata start in state 1. Let $f_\infty(t) = \lim_{N\to\infty} f(t)$. Then $\lim_{t\to\infty} f_\infty(t) = 1/2$.*

SKETCH OF PROOF. Let $f_0 = f_\infty(t_0)$ for any $t_0 \ge 0$. By the continuity of $r(f)$, we know that the reward probability in the neighborhood of f_0 is similarly in the neighborhood of $r_0 = r(f_0)$. Now, $f_1 = f_\infty(t_0 + 1)$ consists of that part of f_0 that was rewarded, plus that part of $1 - f_0$ that was penalized. That quantity is

$$f_1 = r_0 f_0 + (1 - r_0)(1 - f_0) \tag{4}$$

which is clearly a weighted average of f_0 and $1 - f_0$. By the bounds on $r(f)$, we know that this new average must be closer to $1/2$ by a factor of at least $2\Delta r$. □

While this proof is only applicable to the case of a continuous reward function and a particular initial condition, a more complex one exists for the general case. We will not attempt that here; however, the claim should be plausible. The corresponding exact analysis for $n > 1$ is too complex to carry out exactly. However, based on simulations, and on the conclusions from the approximate analysis below in Section 3.2, we propose the following conjectures.

CONJECTURE 1. *For any value of N,*

$$\lim_{n\to\infty} \Phi(k^*) = 1$$

In other words, given any system with N automata, we can always set the memory size n high enough so that the system spends as great a proportion of time as desired in optimal configurations.

CONJECTURE 2. *For any value of n, and any fraction $\epsilon > 0$,*

$$\lim_{N\to\infty} \Psi_N(\epsilon) = 1$$

where $\psi_N(\epsilon)$ is the steady state probability that f is within ϵ of $1/2$; that is,

$$\psi_N(\epsilon) = \sum_{\left|\frac{k}{N} - \frac{1}{2}\right| \le \epsilon} \Phi(k)$$

In other words, given any memory size n, there is always some population size beyond which $f(t)$ is nearly always around $1/2$.

EXAMPLE 1. To illustrate these conjectures, we conducted simulations using the reward function $r(f) = -f^2 + 0.4f + 0.76$ (See Fig. 4) This function has a maximum at $f^* = 0.2$, so $k^* = 0.2N$, rounded to the nearest integer.

From Conjecture 1, we should expect that for any population size N, we can set the memory size n in order to maintain $\Phi(k^*)$ above any given level, say 0.3. Fig. 5 shows that this is indeed true for this example. It also suggests that as N grows, we must increase n as well. The discontinuities in this graph are due to the discrete nature of k^*. For instance, k^* jumps from 1 when $N = 7$ to 2 when $N = 8$.

From Conjecture 2, we should expect that if the memory size n remains constant, a growing population N means the system will spend more and more of its time near $f = 1/2$. In Fig. 6, we set $n = 5$, and $\epsilon = 0.1$. Note that as N increases, the fraction of time spent within ϵ of $f = 1/2$ does in fact approach 1. (In this graph, we examine only the fractions for $N = 5, 10, 15, \ldots$).

We also conducted simulations on bimodal and multimodal reward functions with a variety of populations and memory sizes. The time scale varied, but the general behaviors outlined in the conjectures held true.

3.2 State Aggregation

Our simulations illustrate these conjectures, but they do not explain them. We, therefore, analyze system as a Markov chain. The state space of the chain is the set of N-tuples $\vec{s} = (s_1, s_2, \ldots, S_N)$, where s_m represents the state of α_m. There are thus $(2n)^N$ states in all. Let $\phi(\vec{s})$ represent the number of positive numbers in $\vec{s}$. We can then write the transition probabilities

$$q(\vec{s}, \vec{s}') = \prod_{m=1}^{N} \Pr(x_m) \tag{5}$$

where $s'_m = \delta(s_m, x_m)$ for all m, $\Pr(\text{reward}) = r[\phi(\vec{s})/N]$, and $\Pr(\text{penalty}) = 1 - r[\phi(\vec{s})/N]$.

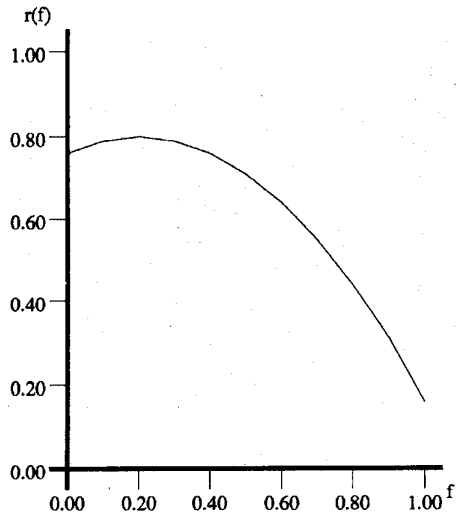


Fig. 4. Example 1 reward function.

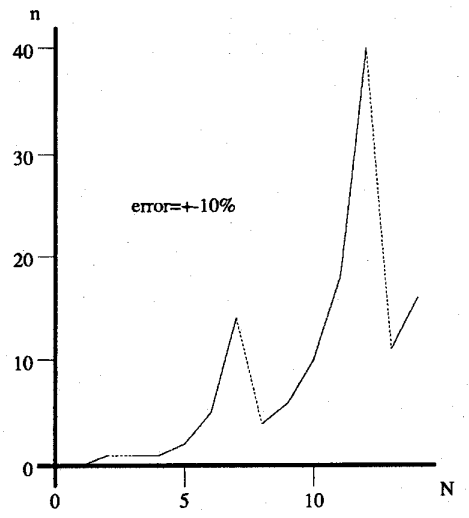


Fig. 5. Memory size required to maintain $\Phi(k^*) > 0.3$.

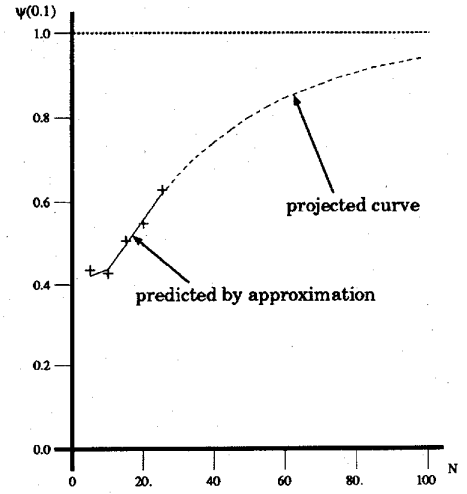


Fig. 6. Increasing $\psi(0.1)$ as a function of population size.

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

$$P(\vec{s}) = \sum_{\vec{s}'} P(\vec{s}')\, q(\vec{s}'), \vec{s}) \tag{6}$$

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

$$\Phi(k) = \sum_{\phi(\vec{s})=k} P(\vec{s}) \tag{7}$$

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 the states.

Assume for the moment that n is large and that $r(f) > 1/2$ for all f. Then, at any time, most of the automata are in the extreme states—that is, near n or $-n$. Therefore, $f(t)$ 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 uncommon, 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. (The remainder of the systems, where $r(f) < 1/2$ for some f, are called *ill-behaved*. We do not give a separate analysis of them here, because the analysis is protracted and because they behave differently—even though the resulting distributions are similar.)

If a system is well-behaved, its behavior consists of *plateaus* during which $a(t)$ is constant, punctuated by trigger transitions. Therefore, this behavior can be decomposed into

1) the probability distribution of $a(t)$ during a plateau
2) the average length of a plateau.

The quantity $\Phi(k)$ is simply a normalized product of these two factors. Volkonskiy [8] makes use of this general method for the simple case where the reward function is of the form $r(f) = r_0$ for $f \le f_c$, $r(f) = r_1 < r_0$ for $f > f_c$, where $f_c < 1/2$ is some critical value. He also requires $r_1 > 1/2$. We now consider these two factors in turn.

3.2.1 Probability Distribution of $a(t)$ During a Plateau

Suppose that of the N automata, k are in a positive state. With the exception of sign, the dynamic behavior of an automaton is the same on either side of the state space: there is no way to distinguish between an automaton and its "mirror image." Roughly, then, any of the k automata on the positive side is as likely to make the first trigger transition as any 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 approximately k/N, and the probability that it goes from -1 to 1 (from negative to positive) is approximately $(N - k)/N$. 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 approximate Markov chain in Fig. 7, where the new states represent the values that $a(t)$ can take, rather than the states of the automata. To avoid confusion, we call the former *system states*, and the latter *automaton states*. (The approximation lies in the as-

sumption that each automata is equally likely to make the next trigger transition. This is not true: the automata in the extreme states are less likely to do so. Empirically, however, this does not have a significant impact on the accuracy of our approximate formula.)

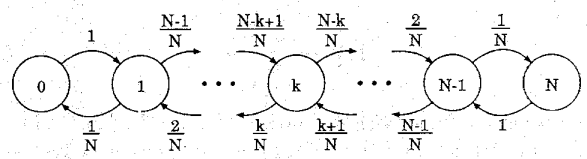


Fig. 7. System state diagram.

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 the trigger transitions. We can solve for the steady state probabilities $\Pi(k)$ of this imbedded chain.

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.

$$\Pi(k) = \Pi(k-1)\left(\frac{N-k+1}{N}\right) + \Pi(k+1)\left(\frac{k+1}{N}\right) \quad (8)$$

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 is

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

$\Pi(k)$ 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.

3.2.2 Average Length of a Plateau

Given a memory size n, we define the *persistence time* $\tau_n(k)$ to be the average time that the system spends in system state k. It is too difficult to determine $\tau_n(k)$ exactly; instead, we make an estimate $\bar{\tau}_n(k) \doteq \tau_n(k)$.

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 $-1 \le j < i \le n$. (We can restrict our discussion to the positive side because the behavior on the negative end is identical, by symmetry, as discussed above.) There is a self-transition at state n, so

$$\bar{t}^n_{n,n-1}(r) = \frac{1}{1-r} \quad (9)$$

Suppose on the other hand that the automaton is currently at state i where $1 \le i < n$. After one time unit, either it has 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

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

This recurrence equation can be solved by the usual z-transform techniques [3] to yield

$$\bar{t}^n_{i,i-1}(r) = \frac{1}{2r-1}\left[\left(\frac{r}{1-r}\right)^{n-i+1} - 1\right] \quad (10)$$

for $1 < i \le n$, and

$$\bar{t}^n_{1,-1}(r) = \frac{1}{2r-1}\left[\left(\frac{r}{1-r}\right)^{n} - 1\right] \quad (11)$$

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, the time this automaton would take to make a trigger transition back to the other side approximates the time between trigger transitions by any single automata. Since each of the N automata move independently, the expected time between trigger transitions is approximately

$$\tau_n(k) \doteq \bar{\tau}_n(k) = \bar{t}^n_{1,-1}(r(k/N))/N \quad (12)$$

Note that as r increases, so does the corresponding persistence time.

3.2.3 The Decomposition Result

Recall that the proportion of time that the population spends in system state k is the normalized product of the visit ratios and persistence times. That is,

$$\Phi(k) = \frac{\Pi(k)\,\tau_n(k)}{\sum_{k'=0}^{N} \Pi(k')\,\tau_n(k')} \quad (13)$$

We now have expressions for these two quantities, which allows us to establish 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 $\bar{\tau}_n(k)$. Then the limiting probability that the system is in state k is approximately

$$\Phi(k) \doteq \frac{\Pi(k)\,\bar{\tau}_n(k)}{\sum_{k'=0}^{N} \Pi(k')\,\bar{\tau}_n(k')} \quad (14)$$

This approximation gives some insight into the above conjectures. We see that the equilibrium system state probabilities are weighted binomial 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 in relation to all other system states. Eventually, the system will spend most of the *time* at k^*, even though it makes more *visits* to the system state $k = N/2$, where the binomial coefficient is the greatest. (However, there is a cost to raising n: since all persistence times are higher, the system responds slower to changes in the reward function. Attention should be paid to this property when response time is critical.)

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 in relation to all other system states. 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 those central states, and the system will spend most of its time around $k = N/2$.

EXAMPLE 2. Consider the reward function given in Fig. 8. Note that this function has two local maxima; that is, it is bimodal. In Fig. 9, we set $N = 10$ and $n = 5$. Note that the approximation is quite accurate, even though $r(f) < 1/2$ for some f.

In Fig. 10, we keep $N = 10$, and calculate the distribution for various values of n. The system performs better and better as the memory size increases; note in particular how the central peak splits to cover the two maxima of $r(f)$. □

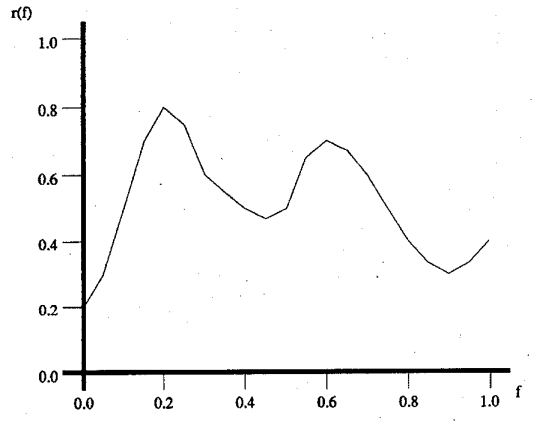


Fig. 8. Example 2 reward function.

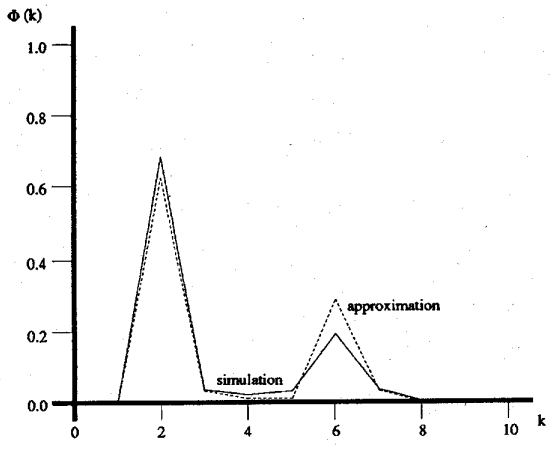


Fig. 9. Steady state probability distribution for Example 2.

In most cases, a bimodal or multimodal reward function does not adversely effect either the accuracy of the approximation or the steady state behavior of the system. Recall that the behavior of the system can be described as a random walk in which more time is spent in good system (aggregated) states than in bad ones. It is not the case, therefore, that a local optima can "trap" the system.

For the same reason, there is no dynamic difference between the transient behavior of the system and its steady state behavior. It is only that when considering transient behavior, we are usually more interested in the time it takes for the statistics to match the eventual steady state distribution.

To make a simple analogy, consider a man performing a random walk on a circle. If his movement is symmetric, the steady state distribution is uniform. Now, in the sense that one cannot tell by observing the man for a moment whether he just started his walk or has been walking for a long time, the transient behavior is identical to the steady state behavior.

However, it will take some time for the statistics of any single instance of this man's path to become anywhere near uniformly distributed. How long this takes is a parameter of interest, and depends on a number of factors, such as the size of the man's steps and the rate at which he takes those steps. Those factors correspond to the population size and the reward function in the Gur Game. The larger the population size, the shorter the steps, so to speak, and the more steps that must be taken to reach an optimal system state. Similarly, the "higher" the reward function, the longer the plateaus between trigger transitions, and hence the longer it takes to move from one system state to another. This can be seen as a quasi-transient behavior, and it is possible to calculate the average amount of time to reach the optimal system state, but that is beyond the scope of this paper.

It is important to remember that just as the system is not trapped into local optima, it is not trapped into the global

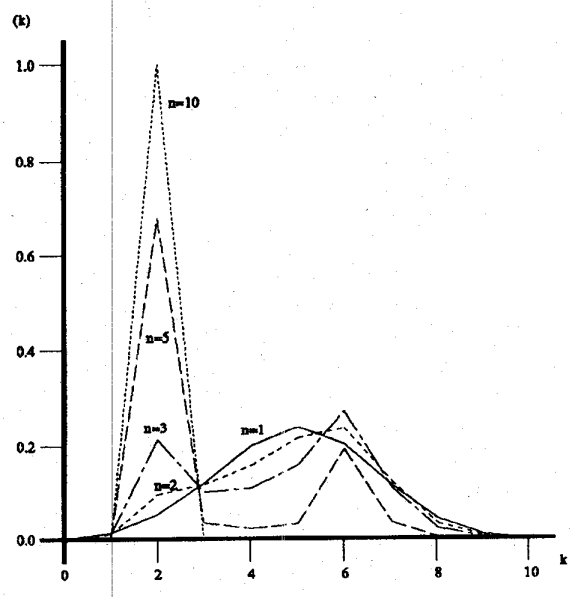


Fig. 10. Effect of memory size on distribution.

optimum, either. For this method to be applicable it must be acceptable to the system designer for optimality to be an average behavior rather than a constant one. If one requires the system to lock into a behavior that will always be optimal, then this method is not appropriate.

However, this feature has the effect that the system is able to respond to a changing population and reward function. The system will still perform as indicated, spending comparatively more time at the new maxima. This *robustness* is an important positive feature of this scheme; for instance, it tolerates faults such as malfunctioning agents or changes in the performance criteria.

EXAMPLE 3. In this example, we highlight the decomposition form of our formula. Consider the reward function

$$r(f) = \begin{cases} 0.9 & \text{if } = 0.1 \\ 0.6 & \text{otherwise} \end{cases}$$

In Fig. 11, we set $N = 10$ and $n = 3$. Note that the equilibrium probabilities are normalized binomial coefficients, as predicted, except at $k = k^* = 1$ (where $f = 0.1$), where the probability is greater because of the longer persistence time. □

EXAMPLE 4. To illustrate the convergence of the algorithm and the accuracy of the approximation over a larger range of reward functions, we consider a system with $N = 5$ and $n = 3$. We use the set of reward functions $r(f)$ such that $r(f) = 0.4$, 0.6, or 0.8, for all $f = 0.0$, 0.2, 0.4, 0.6, 0.8, or 1.0. There are thus $3^6 = 729$ different reward functions. We examine the equilibrium probabilities under all 729 functions after 100 cycles (or time units) and 1,000 cycles.

If we denote the simulation probabilities by $\Phi_{sim}(k)$ and the approximation probabilities by $\Phi_{approx}(k)$, then we define the error between the two as

$$\text{error} = \sum_{k=0}^{N} \left[\Phi_{\text{sim}}(k) - \Phi_{\text{approx}}(k)\right]^2$$

In Fig. 12, we give the distribution of error across all 729 functions. For instance, the curve for 1,000 cycles is approximately equal to 0.23 at zero. This means that about 23% of the functions produce an error of between 0.00 and 0.01 after 1,000 cycles. After 100 cycles, the approximation has a median error of about 0.10, and after 1,000 cycles, the median error is only about 0.03. After more than 1,000 cycles, the error distribution does not change much. This suggests both that the approximation is very close to the actual result, and that the algorithm converges quickly. Examination of the simulation traces reveals that the approximation is most accurate when the reward function is most nearly linear, and least accurate when there are many changes in slope (particularly at $f = 0.4$ or 0.6). The scope of this paper does not allow us to go into greater detail about this correspondence.

Our model can easily be generalized to allow more than two outputs, by adding more "arms" to the automaton design. (Consider that our current model has two—a positively numbered arm, and a negatively numbered arm—and that the sign has no semantics other than to distinguish the two arms.) For complete details, the reader is referred to [7].

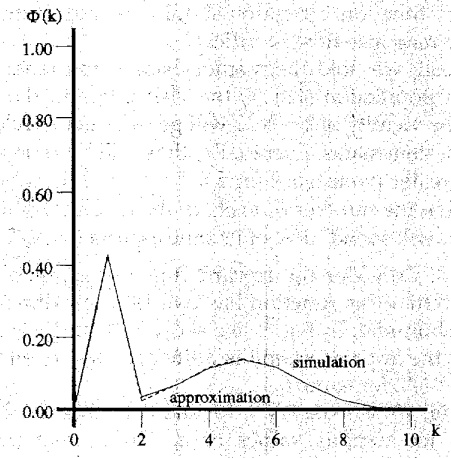


Fig. 11. Steady state probability distribution for Example 3.

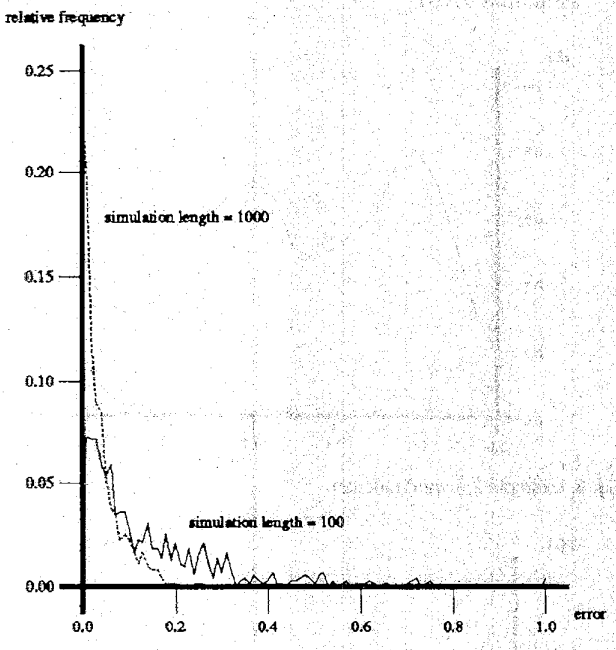


Fig. 12. Error distribution across general reward functions.

4 EXAMPLE: COOPERATIVE ROBOTICS

The Commotion Laboratory[1] at UCLA conducts research into all aspects of mobile robots (or "mobots"). One of these is the prospect of some means of automated cooperation. Is it possible to get the mobots to complete a complex task (that is, one that requires the cooperation of many) without individually directing each one through every subtask?

1. The Commotion Laboratory is supported by the National Science Foundation under Grant No. CDA-9303148.

In this example, we shall see how the Gur Game paradigm can be used to produce exactly such cooperation. The experiment described here is based on measurements taken at the Commotion Laboratory, although it was not conducted in its entirety. For that, we have chosen to conduct simulations (for simple expediency).

The scenario for this experiment is as follows. Consider a landscape containing pieces of ore (represented by tin cans). It is desired that the ore be collected and sorted by type (represented by different colored paper labels on the cans). For this experiment, consider two types, sorted into two separate bins.

This is a task that can be completed correctly by one mobot, but it is clearly faster to utilize more than one. Suppose that we have a population of six mobots. Because of variations in the physical components of the mobots, some mobots are better than others at collecting, and some are better at sorting. Furthermore, physical components wear out, and relative abilities may change over time as a result.

In our scenario, the mobots have access to a single shared access communication channel. Access to this channel can, in fact, be coordinated by use of the Gur Game, but in this case, it was preassigned in a repeating frame of six slots, one for each mobot. This channel is used by the mobots only to inform the other mobots of their current actions. These actions are one of:

1) *Collecting*. This consists of searching out ore, retrieving it with the mobot arm, and placing it in a sorting bin.
2) *Sorting*. This consists of retrieving ore from the sorting bin, sorting it based on its color (with a bank of RGB sensors), and placing it in the correct finished bin.

These actions clearly map onto A_0 and A_1 of our model. For the purposes of this example, we ignore the quality of the sort; it is the efficiency that interests us here.

A mobot which finishes sorting a piece of ore sends out a signal to indicate that. This signal is used as the reward/penalty signal; we were unable to use a remote control to trigger a base station to send a signal.

This signal is translated into a reward or a penalty as follows: Time is divided into a repeating cycle of six periods (not to be confused with the communication channel time division), one for each mobot. If a mobot receives a signal during its period, it is rewarded; otherwise, it is penalized. At the end of the cycle, the mobots change the state of their automata and, if applicable, their actions. The cycle then repeats.

For this experiment, we used automata with a memory size of $n = 3$, and a cycle 30 sec long, consisting of six periods of 5 sec each.

In actual mobots, variations in abilities, while measurable, are small enough that the effects of these variations take a long time (on the order of an hour) to manifest themselves. In our simulations, to speed matters, we made the variations artificially large, as in Table 1.

We assume that collection times are uniformly distributed, since the ore is uniformly distributed.

The scope of this paper does not allow us to go into full detail on the results of our experiments. We conducted 1,000 simulation trials, with each mobot initially collecting. (This produced quite a full collection bin!) On average, it took 38 cycles for behavior to settle on mobots 1 through 4 collecting, and mobots 5 and 6 sorting, as expected. These 38 cycles last about 20 minutes.

To simulate degradation of abilities, we slowed down the collection time of mobots 1 and 2 to 40–60 sec. (In actuality, this can be effected by detuning the mobots' "eyes.") On average, it now took 32 cycles for behavior to settle on the following curious arrangement: either mobot 1 or 2, and either mobot 5 or 6, sorting, and the rest collecting. It turns out that these arrangements are optimal for the given speeds!

Because of the random walk nature of the Gur Game, however, this "settling" was not permanent; in almost all of the simulation runs, the population oscillated back and forth between the original division and the new one, although more time was spent in the new arrangement, by a ratio of approximately 3:2.

5 Summary

We have examined the problem of how to design automata so that they may work together cooperatively to find the maximum of a given reward function. 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 is quite accurate, as demonstrated by computer simulation. The approximation displays a decomposition form, which clearly illustrates the essential behavior of the system. This method is robust in that it overcomes changing conditions, such as shifts in the population or reward function. We have illustrated this method and its performance in a simple application.

In future research, we will investigate the application of this scheme to the solution of difficult real world problem formulations. We will characterize the space of problems that are resolvable by this technique, and difficult to solve by other methods. We also propose to give specific solutions to some standard problems, and to detail necessary modifications to this scheme.

Acknowledgment

This work was supported by the Defense Advanced Research Projects Agency under Grant No. MDA-972-91-J-1011, Advanced Networking and Distributed Systems.

TABLE 1
TABLE OF MOBOT SPEEDS FOR TWO TASKS

Mobot	Collection Time (sec)	Sorting Time (sec)
1	20-30	20
2	20-30	20
3	20-30	20
4	20-30	20
5	30-45	15
6	30-45	15

REFERENCES

[1] V.A. Borovikov and V.I. Bryzgalov, "A Simple Symmetric Game between Many Automata," *Avtomat Telemekh*, vol. 26, no. 4, 1965.

[2] D.R. Hofstadter, "Metamagical Themas," *Scientific American*, May 1983.

[3] L. Kleinrock, *Queueing Systems, Volume 1: Theory*. John Wiley & Sons, 1975.

[4] L. Kleinrock, "On Distributed Systems Performance," *ITC Specialist Seminar: Computer Networks and ISDN Systems*. pp. 209–216, 1990.

[5] P.J. Ramadge and W. Wonham, "Supervisory Control of a Class of Discrete Event Processes," *SIAM J. on Control and Optimization*, vol. 25, no. 1, 1987.

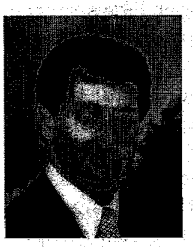
[6] M.L. Tsetlin, "Finite Automata and Modeling the Simplest Forms of Behavior," PhD thesis, V.A. Steklov Math. Inst., 1964

[7] B. Tung, "Distributed Control Using Simple Automata," PhD thesis, Univ. of California, Los Angeles, 1994.

[8] V.A. Volkonskiy, "Asymptotic Properties of the Behavior of Simple Automata in a Game," *Probl. Peredachi Inform.*, vol. 1, no. 2, 1965.

[9] W. Wonham and P.J. Ramadge, "Modular Supervisory Control of Discrete-Event Systems," *Math. of Control, Signals, and Systems*, vol. 1, no. 1, pp. 13–30, 1988.

Brian Tung received his BS degree in electrical engineering and computer science (EECS) from the University of California at Berkeley, in 1989. He received his PhD degree in computer science from the University of California, Los Angeles, in 1994, working on distributed optimization and control. Currently he is a computer scientist at the University of Southern California Information Sciences Institute where he does work on network security and access.



Leonard Kleinrock received the BS degree in electrical engineering from City College of New York in 1957 and the MS and PhD degrees (both in electrical engineering) from the Massachusetts Institute of Technology, Boston, in 1958 and 1963, respectively. He is a professor in the Computer Science Department at the University of California, Los Angeles. While at MIT, he worked at the Research Laboratory for Electronics, as well as with the Computer Research Group of MIT Lincoln Laboratory in advanced technology. He joined the faculty at the University of California at Los Angeles in 1963. His research interests focus on performance evaluation and design of many kinds of networks (i.e., packet switching networks, packet radio networks, local area networks, metropolitan area networks broadband, gigabit networks, and parallel and distributed systems).