10. Interim and Prospectus

Adaptation in Natural and Artificial Systems, after a seven-year gestation, made its appearance in 1975. It is now 1991, and much has happened in the interim. Topics that were speculative in 1975 have been carefully explored; extensions, applications, and new areas of investigation abound. More than 150 papers were submitted to the 1991 International Conference on Genetic Algorithms (Belew and Booker 1991), and several new books have been written about genetic algorithms (e.g., Davis 1987 or Davis 1991). There is even a textbook (Goldberg 1989). Most of this new research has been reported in the published proceedings of the genetic algorithm conferences of 1985, 1987, 1989, and 1991 (Grefenstette 1985, Grefenstette 1987, Schaffer 1989, and Belew and Booker 1991) and is readily accessible there, so I will not attempt to review it here—the review would be, at best, little more than an annotated listing. Instead, I'll follow the pattern of the rest of the book, using this new chapter to report on lines of research I've pursued since 1975. A new edition also provides an opportunity to correct errors in the original edition. Most of these are simple and innocuous, but an error in one proof, discovered and corrected by Dr. Daniel Frantz, is subtle and important. By good fortune, after the correction the theorem involved stands as stated. Finally, a new chapter offers an opportunity to look further into the future; this too I'll attempt.

1. IN THE INTERIM

Classifier Systems

Classifier systems, a specialization of chapter 8's "broadcast language," are a vehicle for using genetic algorithms in studies of machine learning. Classifier systems were introduced in Holland 1976 and were later revised to the current "standard" form in Holland 1980. There is a comprehensive description of the standard form, with examples, in Holland 1986, but there are now many variants (see Belew and Booker 1991). A classifier system is more restricted than the broadcast language in just one

major respect: A broadcast unit can directly create other broadcast units, but a classifier, the broadcast unit's counterpart in a classifier system, cannot directly create other classifiers. This restriction permits a much simpler syntax based on only three atomic symbols, {1,0, # ("don't care")}. A classifier system creates new classifiers through the action of the genetic algorithm on the system's population of classifiers.

Classifier systems aim at a question that seems to me central to a deeper understanding of learning: How does a system improve its performance in a perpetually novel environment where overt ratings of performance are only rarely available? A learning task of this kind is more easily described if we think of the system as playing a strategic game, like checkers or chess. After a long sequence of actions (moves), the system receives some notification of a "win" or a "loss" and, perhaps, some indication of the strength of the win or loss. But there is almost no information about what moves should have been changed to yield better performance. Most learning situations for animals, including humans, have this characteristic—an extended sequence of actions is followed by some general indication of the level of performance, with little information about specific changes that would improve performance.

In defining classifier systems (see Figure 15), I adopted the common view that the state of the environment is conveyed to the system via a set of *detectors* (e.g., rods and cones in a retina). The outputs of the detectors are treated as standardized packets of information—*messages*. Messages are used for internal processing as well, and some messages, by directing the system's *effectors* (e.g., its muscles), determine the system's actions upon its environment. Beside the interactions with the environment provided by detectors and effectors, there is a further interaction that is critical to the learning process. The environment must, upon occasion, provide the system with some measure of its performance. Here, as earlier, I will use the term *payoff* as the general term for this measure.

The computational basis for classifier systems is provided by a set of *condition/action* rules, called *classifiers*. To simplify the computational basis, all interactions between rules are mediated by messages. Under this provision a typical rule, under interpretation, would have the form

> IF there is (a message from the detectors indicating) an object left of center in the field of vision
> THEN (by issuing a message to the effectors) cause the eyes to look left.

That is, the *condition* part of the rule "looks for" certain kinds of messages, and when the rule's conditions are satisfied, the *action* part specifies a message to be sent. Messages both pass information from the environment and provide communication

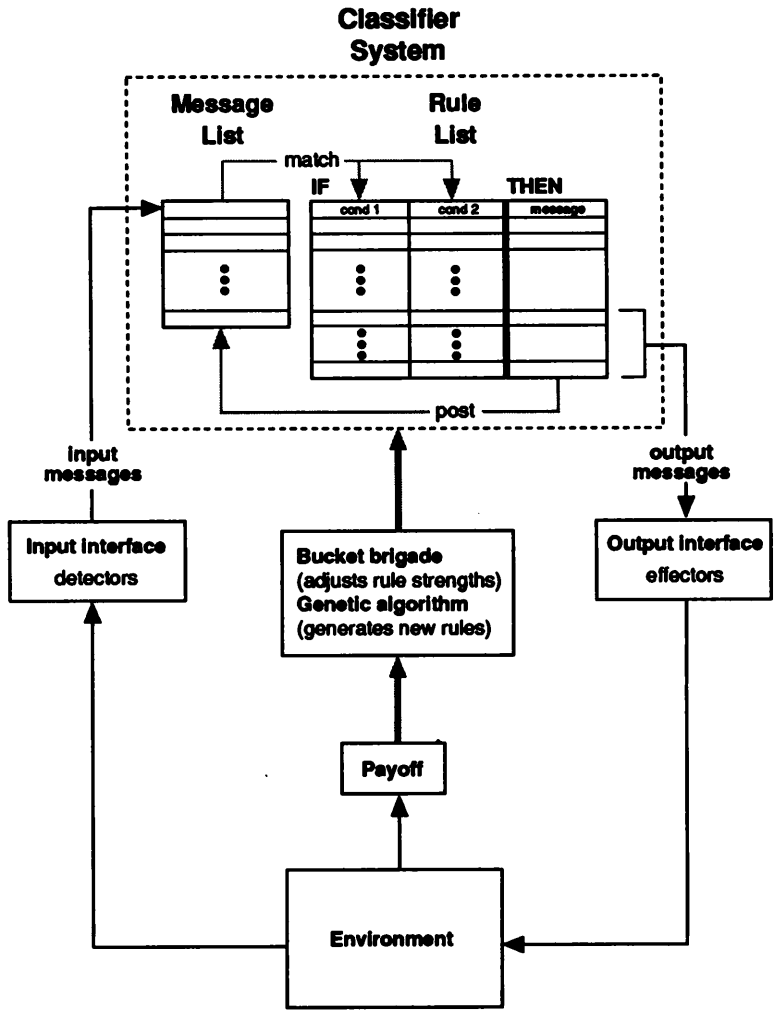


Fig. 15. A classifier system

between rules, as in the broadcast language (where they were called *signals*). Thus each rule is a simple message processor. Many rules can be active simultaneously, so many messages may be present at any given instant. It is convenient to think of the messages as collected in a *list* that changes under the combined impetus of the environment and the rules.

Parallelism, the concurrent activity of many rules, is an important aspect of classifier systems. Parallelism makes it possible for the system to combine rules into clusters that model the environment, providing two important advantages:

1. Combinatorics work for the system instead of against it. The system builds a "picture" of the situation from parts, rather than treating it as a monolithic whole. The advantage is similar to that obtained when one describes a face in terms of component parts. Select, say, 8 components for the face—hair, forehead, eyebrows, eyes, cheekbones, nose, mouth, and chin. Allow 10 variants for each component part—different hair colors and textures, different forehead shapes, and so on. Then $10^8 = 100$ million faces can be described by combining these components in different ways. This at a cost of storing only $8 \times 10 = 80$ "building block" components.
2. Experience can be transferred to novel situations. On encountering a novel situation, such as a "red car by the side of the road with a flat tire," the system activates several relevant rules, such as those for "red," "car," "flat tire," etc. When "building-block" rules, such as those for "car," have proved useful in past combinations, it is at least plausible that they will prove useful in new, similar combinations. To exploit these possibilities, the rules must be organized in a way that permits clusters of rules to be activated in changing combinations, as dictated by changing situations. Building-block rules then give the system a capacity for transferring experience to new situations.

To define a standard classifier system, we first require all messages to be bit-strings of the same length, k, much as one sets the register size for a computer. Formally, then, messages belong to the set $\{1,0\}^k$. The condition part of a rule is specified by the use of a "don't care" symbol, #, reminiscent of the "don't care" used to define schemata. Thus, the set of all conditions is the set $\{1,0,\#\}^k$. For $k=6$, the condition 1##### is satisfied by any message that starts with a 1, while the condition 001001 is satisfied by one and only one message, the message 001001. It is worth noting that a condition's *specificity* (the reciprocal of the number of messages that satisfy it) depends directly upon the number of #s in the condition—the more #s, the lower the specificity.

In the standard system, all rules consist of two conditions and a single outgoing message, which is sent when the two conditions are satisfied. Rules are specified in the format

1#####,001001/000011.

This format is interpreted as follows:

> IF condition 1 is satisfied (in this case, by a message, on the message list, that starts with 1),
> AND condition 2 is satisfied (in this case, by a second, specific, message 001001),
> THEN the message in the action part (in this case, 000011) is posted to the message list on the next time-step.

Conditions may be negated: For example, $-1\#\#\#\#\#$ is satisfied if there is *no* message on the message list that begins with a 1. With these provisions it is easy to show that a classifier system is computationally complete, in the sense that any program that can be written in a standard programming language, such as Fortran, C, or Lisp, can be implemented within a classifier system.

Without any changes to this definition, rules can be given an "address" that can be used by other rules when that is useful. Consider a rule r with a condition of the form $111\# \ldots \#$. Any message that starts with three 1s will satisfy this condition. If this particular prefix, 111, is reserved for the rule r alone, then any message with that prefix will be directed to r and only to r. Such reserved prefixes (they can also be suffixes, or indeed any part of the message) are called *tags*. Of course, several rules might have the same reserved tag; that simply means that all of them receive messages so tagged, acting as a cluster with respect to that tag. Appropriate use of tags also permits rules to be coupled to act sequentially.

The basic execution cycle of the classifier system consists of an iteration of the following steps:

1. Messages from the environment are placed on the message list.
2. Each condition of each classifier is checked against the message list to see if it is satisfied by (at least one) message thereon.
3. All classifiers that have both conditions satisfied participate in a *competition* (to be discussed in a moment), and those that win post their messages to the message list.
4. All messages directed to effectors are executed (causing actions in the environment).
5. All messages on the message list from the *previous* cycle are erased (i.e., messages persist for only a single cycle, unless they are repeatedly posted).

Because the message list can hold an arbitrary number of messages, any number of rules can be active simultaneously; because the messages are simply uninterpreted

bit-strings, there are no *consistency* problems in the internal processing. Consistency problems do arise at the effectors; when different, simultaneous messages urge an effector to take mutually exclusive actions, they are resolved by competition.

Competition plays a central role in determining just which rules are active at any given time. To provide a computational basis for the competition, each rule is assigned a quantity, called its *strength*, that summarizes its average past usefulness to the system. We will see shortly that the strength is automatically adjusted by a *credit assignment* algorithm, as part of the learning process. Competition allows rules to be treated as hypotheses, more or less confirmed, rather than as incontrovertible facts. The strength of a rule corresponds to its level of confirmation; stronger rules are more likely to win the competition when their conditions are satisfied. Stated another way, the classifier system's reliance upon a rule is based upon the rule's average usefulness in the contexts in which it has been tried previously. Competition also provides a means of resolving conflicts when effectors receive contradictory messages.

A rule, then, enters a competition to post its message any time its conditions are satisfied. The actual competition is based on a bidding process. Each satisfied rule makes a bid based upon its strength *and* its specificity. In its simplest form, the bid for a rule r of strength $s(r)$ would be

$$\text{Bid}(r) = c \cdot s(r) \cdot \log_2[\text{specificity}(r)],$$

where c is a constant <1, say 1/10. A rule that both has been useful to the system in the past (high strength) and uses more information about the current situation (high specificity) thus makes a higher bid. Rules making higher bids are favored in the competition. Various criteria for winning can be employed. For example, the probability of winning can be based on the size of the bid, or all rules making bids at least equal to the average bid can be declared winners. Usually there are several winners, so that parallelism is exploited.

This completes the description of the performance part of the system; we are now ready to discuss the system's learning procedures. There are two basic problems, *credit assignment*, already mentioned, and *rule discovery*. Credit assignment rates the rules the system already has. Rule discovery replaces rules of low strength and provides new rules when environmental situations are ill-handled.

Credit assignment

Let us begin with the credit assignment problem. Credit assignment is not particularly difficult where the situation provides immediate reward or precise information about

correct actions. Then the rules directly involved are simply strengthened. Credit assignment becomes difficult when credit must be assigned to early acting rules that *set the stage*, making possible later useful actions. Stage-setting moves are the key to success in complex situations, such as playing chess or investing resources. The problem is to credit an early action, which may look poor (such as the sacrifice of a piece in chess) but which sets the stage for later positive actions (such as the capture of a major piece in chess). When many rules are active simultaneously, the problem is exacerbated. It may be that only a few of the early acting rules contribute to a favorable outcome, while others, active at the same time, are ineffective or even obstructive. Somehow the credit assignment algorithm must sort this out, modifying rule strengths appropriately.

Credit assignment in classifier systems is based on competition. The bidding process mentioned earlier is treated as an exchange of "capital" (strength). That is, when a rule wins the competition, it actually "pays" its bid to the rule(s) that sent the message(s) satisfying its conditions. The rule acts as a kind of go-between or broker in a chain that leads from the stage-setting situation to the favorable outcome.

In a bit more detail, when a rule competes, its *suppliers* are those rules that have sent messages satisfying its conditions and its *consumers* are those rules that have conditions satisfied by its message. Under this regime, we treat the strength of a rule as capital and the bid as payment to its suppliers. When a rule wins, its bid is apportioned to its suppliers, increasing their strengths. At the same time, because the bid is treated as a payment for the right to post a message, the strength of the winning rule is reduced by the amount of its bid. Should a rule bid but not win, its strength is unchanged and its suppliers receive no payment. The resulting credit assignment procedure is called a *bucket brigade* algorithm (see Figure 16).

Winning rules can recoup their payments in two ways: (1) They, in turn, have winning consumers that make payments to them, or (2) they are active at a time when the system receives payoff from the environment. Case (2) is the sole way in which payoff from the environment affects the system. When payoff occurs, it is divided among the rules active at that instant, their strengths being increased accordingly. Rules not active at the time the payoff occurs do not share directly in that payoff. The system must rely on the bucket brigade algorithm to distribute the increased strength to the stage-setting rules, under repeated activations in similar situations.

The bucket brigade works because rules become strong only when they belong to sequences leading to payoff. To see this, first note that rules consistently active at times of payoff tend to become strong because of the payoff they receive from the environment. As these rules grow stronger, they make larger bids. A rule that "supplies" one of the payoff rules then benefits from these larger bids in future transactions.

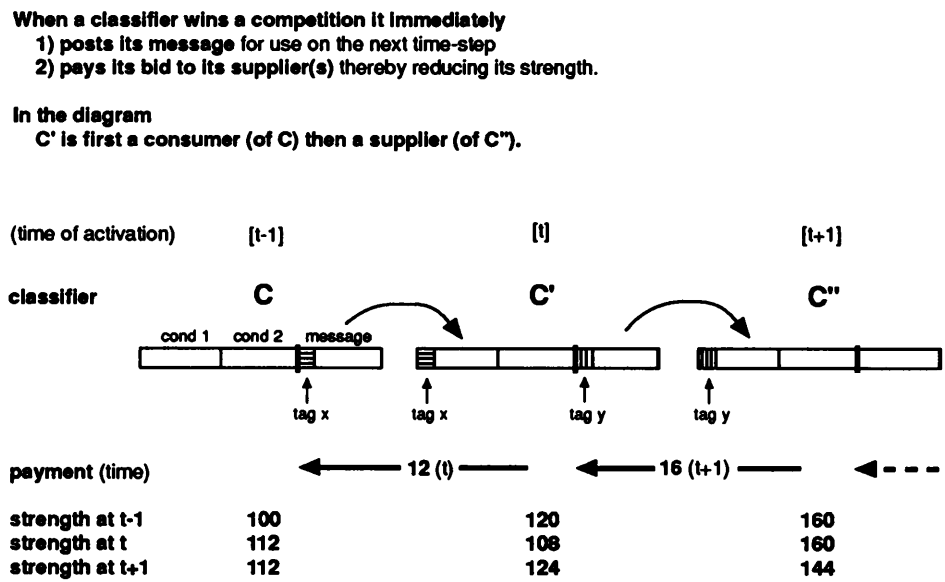


Fig. 16. The bucket brigade algorithm

Its strength increases because its income exceeds its payout—it makes a "profit." Subsequently, the suppliers of the suppliers begin to benefit, and so on, back to the early stage-setting rules.

Things can go wrong. A supplier might, through a message to the effectors, convert an environmental state to one that diverts its consumer rule from a payoff-directed path. That is, it might fail in its stage-setting role. In that case, the consumer suffers because the diversion will prevent it from receiving payments from *its* consumers; however, the diverting supplier rule generally suffers even more, because it is at an earlier stage in its "getting rich" effort. Or it may be that the consumer has a condition that attends to the state of the enviroment and does not even bid when the diverting state occurs. In either case, the diverting supplier soon loses enough strength so that it no longer wins the competition. It then ceases to influence subsequent activity.

The whole process, of course, takes repeated plays of the game. But it only requires that a rule interact with its immediate suppliers and consumers. It requires no overt memory of the long and complicated sequences leading to payoff. Avoiding extensive overt memories is almost a *sine qua non* for large, parallel systems acting

in perpetually novel environments with sparse payoff. Overt memories in such situations necessarily involve many tangled strands, including unnecessary detours and incidentals. To tease out the relevant strands at the time payoff occurs would be an overwhelming, hardly feasible task. Over repeated trials the bucket brigade carries out this task, but in an implicit fashion.

Rule Discovery

Generating *plausible* replacements for rules assigned low strength under the credit assignment algorithm is an even more daunting task than credit assignment itself. In a rule-based system, the whole process of induction succeeds or fails in proportion to its efficacy in generating plausible new rules, rules that are not obviously incorrect on the basis of experience. However, *plausible* is not an easy concept to pin down computationally. It implies that experience biases the generation of new rules, but how?

I propose that the concept of plausibility is closely linked to the "schema" concept set forth in the discussion of genetic algorithms. Because the rules in a classifier system are presented by strings defined over a three-letter alphabet, {1,0,#}, we can think of the strings as chromosomes defined on three alleles. Accordingly, we can interpret the set of rules used by the classifier system as a population of chromosomes. Moreover, the strength of each rule can be interpreted directly as its fitness (though it should be noted that there are interesting variants that base fitness on strength in a less simplistic way). A genetic algorithm, then, is easily applied to such a population of rules, and, indeed, classifier systems were designed with just this objective in mind.

In this application of the genetic algorithm, schemata serve as building blocks for rules. The usefulness of any given schema can be *estimated*, in the usual way, from the average observed strength of the rules that are instances of the schema in the population. Though these estimates are subject to error, they do provide an experience-dependent guideline. Both the possibility of error and role of experience are consonant with the term *plausibility*. As always, the genetic algorithm exploits these estimates implicitly (*implicit parallelism*, née *intrinsic parallelism* in chapter 4) rather than explicitly, but this does not affect the plausibility of the new rules generated thereby.

It helps in understanding the evolution of a classifier system to note that simple schemata (those with few defining positions) generally have more instances than more complex schemata in a population of fixed size. From a sampling point of view, this

means that simple schemata accumulate samples more rapidly. It is not difficult to show that the rate of accumulation falls off exponentially with the complexity.

This automatic differential in sampling rates has a strong influence on what schemata play an important role in rule generation at any point in time. Early on, the system has reliable information only about simple schemata. But simple schemata usually only provide building blocks and estimates for coarse discriminations. Though the classifier system can exploit this information, rules built from these simple schemata are exposed to frequent surprises, departures, and exceptions in more complex contexts. Over time, the system gains more experience, and it gains information about more complicated schemata. This information biases the genetic algorithm toward the construction of more sophisticated, more specific rules. As a consequence, as the classifier system accumulates experience, it is prone to build hierarchies of rules of increasing specificity. These hierarchies grow from early "default" rules, based on simple contexts, to layers of "exception" rules based on later, more detailed contextual information.

When simultaneous messages satisfy both a simpler default rule and a more complex exception rule, the latter tends to outcompete the former (though there can be complications; see Riolo's paper in Belew and Booker 1991). The higher specificity of the exception rule causes it to outbid the default rule if their strengths are comparable. The exception rule only survives under the bucket brigade if it corrects inappropriate actions of some default rule; otherwise, the strength of the exception rule diminishes until it is no longer a factor in the competition. When the exception rule does correct the default rule, a kind of symbiosis results. By saving the default rule from paying a bid in a situations where it would not make a profit, the exception rule actually helps the default rule to retain its strength. Thus both the default rule and the system as a whole are better off for the presence of the exception rule.

Because successive layers of exception rules are only added as the necessary information becomes available, these rule hierarchies provide a sophisticated, incremental way of modeling the environment. The formal structures corresponding to these default hierarchies, called *quasi-homomorphisms*, have been defined and studied in Holland et al.(1986).

Genetic algorithms have another critical effect on the development of classifier systems. Recombination, under the algorithm, discovers useful schemata for tags in just the way it discovers useful schemata for other parts of the rule. For example, a genetic algorithm can recombine parts of established tags to invent new tags. As a result, established tags spawn related tags, providing new clusters of rules, and new

couplings between established clusters. Tags survive (or, more carefully, the rules using them survive) if they contribute to useful interactions. Under these evolutionary pressures, the tags develop into a system of experience-based "symbols" for interior use (cf. Hofstadter's [1979] concept of an "active symbol"). The associations provided by these tags flesh out the default hierarchy models. The resulting structures can be quite sophisticated, enabling the system to model new situations by coupling appropriate clusters of established (strong) rules. Moreover, these models can be used in a "lookahead" fashion, permitting the classifier system to act in anticipatory fashion, selecting actions on the basis of future consequences. The interested reader is referred to Holland 1991 and Riolo 1990.

Each of the mechanisms used by the classifier system has been designed to enable the system to continue to adapt to its environment, while using the capabilities it already has to respond instant-by-instant to that environment. In so doing the system is constantly trying to balance exploration (acquisition of new information and capabilities) with exploitation (the efficient use of information and capabilities already available).

2. THE OPTIMAL ALLOCATION OF TRIALS REVISITED

Pride of place in the correction category belongs to Dan Frantz's work on one of the main motivating theorems in the book, Theorem 5.1. This theorem concerns the "optimal" allocation of trials in determining which of two random variables has a higher expected value (the well known 2-armed bandit problem). In chapter 5, an "optimal" solution is a solution that minimizes the losses incurred by drawing samples from the random variable of lower expectation. The theorem there shows that these losses are minimized if the number of trials allocated to the random variable with the highest observed expectation increases exponentially relative to the number of trials allocated to the observed second best.

Because schemata can be looked upon as random variables, this result illuminates the treatment of schemata under a genetic algorithm (née *genetic plan* in chapter 7). Under a genetic algorithm, a schema with an above-average fitness in the population increases its proportion exponentially (until its instances constitute a significant fraction of the total population). If we think of the genetic algorithm as generating samples of n random variables (an n-armed bandit), in a search for the best, then this exponential increase is just what Theorem 5.1 suggests it should be.

The problem with the proof of the theorem, as given, turns on its particular

use of the central limit theorem. To see the form of the error, let us follow Frantz by using $F_n(x)$ to designate the distribution of the normalized sum of the observations of the random variable X. For the 2-armed bandit, F is the distribution of the *difference* of the two random variables of interest. Using the notation of chapter 5, $q(n) = 1 - F_n(x)$ when $x = bn^{1/2}$. That is, $1 - F_n(x)$ gives the probability of a decision error, $q(n)$, after n trials out of N have been allocated to the random variable observed to be second best. Because x is a function of n, the proof given in chapter 5 implicitly assumes that, as $n \rightarrow \infty$, the ratio

$$[1-F_n(x)]/[1-\Phi(x)] \rightarrow 1,$$

where $1-\Phi(x)$ is the area under the tail of a normal distribution. However, standard sources (see Feller 1966, for example) show that this is only true when x varies with n as $o(n^{1/6})$. This is manifestly untrue for Theorem 5.1, where $x = bn^{1/2}$.

The main result of theorem 5.1 can be recovered by using the theory of large deviations instead of the central limit theorem. The theory of large deviations makes the additional requirement that the moment-generating functions for the random variables exist, but this is satisfied for the random variables of interest here. Let the moment-generating functions for the two random variables, corresponding to the two arms of the bandit, be $m_1(t)$ and $m_2(t)$. Then the moment-generating function for X, the difference, is $m(t) = m_1(-t)*m_2(t)$. There is a uniquely defined constant c such that

$$c = \inf_t(m(t)).$$

Define $S(n)$ to be the sum of n samples of X. Then the appropriate theorem on large deviations yields

$$\Pr\{S(n)\geq 0\} = [c^n/(2\pi n)^{1/2}]d_n(1+o(1)),$$

where $\log d_n = o(1)$. Making appropriate provision for ties, this yields

$$q(n) \sim b'c^n/(2\pi n)^{1/2},$$

where b' is a constant that depends upon whether or not X is a lattice variable. This relation for $q(n)$ is of the same form (except for constants) as that obtained for $q(n)$ under the inappropriate use of the central limit theorem. Substituting, and proceeding as before, Frantz obtains

THEOREM 5.1 (large deviations): *The optimal allocation of trials n^* to the observed second best of the two random variables corresponding to the 2-armed bandit problem is given by*

$$n^* \sim (1/2r)\ \ln[(r^3c^2N^2)/(\pi \ln(r^2c^2N^2/2\pi))],$$

where $r = |\ln c|$.

This theorem actually goes a step further than the original version, directly providing a "realizable plan" for sample allocation. The original version was based on a "ideal" plan that could not be directly realized, requiring section 5.2 to show that the losses of the ideal plan could be approached by the losses of an associated "realizable" plan. Section 5.2 is now superfluous.

The revised constants for the realizable plan do not affect results in later chapters, because the further analysis of genetic algorithm performance does not depend upon the exact values for the constants. The basic point is that genetic algorithms allocate trials exponentially to the random variables (schemata) corresponding to the arms of an n-armed bandit. Coefficients may vary among schemata, but the implicit parallelism of a genetic algorithm is enough to dominate any differences in the coefficients.

There are two other errors that may trouble the close reader, though they are much less important. The first error occurs, at the top of page 71, in the example giving estimated values for schemata. $x(3)$ should be .1000010 . . . 0, with the consequence that

$$\hat{f}_{\square 1 \square\square 01 \square \ldots \square} = (f(x(1))+f(x(4)))/2.$$

The second error occurs, on page 103, in the discussion of the effect of crossover on the increase of schemata. In the derivation just below the middle of the page, the approximation $1/(1-c) \geqq 1+c$, for $c \leqq 1$, is invoked. But this approximation is in the wrong direction for preserving the inequality for $\hat{\mu}_\xi(t)$; therefore the line that follows the mention of this approximation should be deleted.

Finally there is a point of emphasis that may be troublesome. In the discussion of the role of payoff in the formal framework, near the bottom of page 25, the mapping allows the payoff to be any real number, positive or negative. It would have helped the reader to say that payoff is treated as a nonnegative quantity throughout the book, particularly in the discussion of genetic algorithms.

Other than these corrections, I am only aware of a few (less than a dozen) typographical errors scattered throughout. They are all obvious from context, so there's no need to list them here.

3. RECENT WORK

My most recent work stems from my association with the Santa Fe Institute in Santa Fe, New Mexico. About five years ago the Santa Fe Institute, then newly founded, began developing a new interdisciplinary approach to the study of adaptive systems. The studies center on a class of systems, called *complex adaptive systems*, that have a crucial role in a wide range of human activities. Economies, ecologies, immune systems, developing embryos, and the brain are all examples of complex adaptive systems. Despite surface dissimilarities, all complex adaptive systems exhibit a common kernel of similarities and difficulties, and they all exhibit complexities that have, until now, blocked broadly based attempts at comprehension:

1. All complex adaptive systems involve large numbers of parts undergoing a kaleidoscopic array of simultaneous nonlinear interactions.
   Because of the nonlinear interactions, the behavior of the whole system is not, even to an approximation, a simple sum of the behaviors of its parts. The usual mathematical techniques of linear approximation—linear regression, normal coordinates, mean field approaches, and the like—make little progress in the analysis of complex adaptive systems. The simultaneity of the interactions poses both a challenge and an opportunity for the massively parallel computers now coming on the scene.
2. The impact of these systems in human affairs centers on the aggregate behavior, the behavior of the whole.
   Indeed, the aggregate behavior often feeds back to the individual parts, modifying their behavior. Consider the effect of government statistics on the plans of individual businesses in an economy, or the effect of the aggregate retention of nutrients in a rain forest, despite leached, impoverished soils, upon species diversity and niches therein.
3. The interactions evolve over time, as the parts adapt in an attempt to survive in the environment provided by the other parts.
   As a result, the parts face perpetual novelty, and the system as a whole typically operates far from a global optimum or equilibrium. Standard theories in physics, economics, and elsewhere are of little help because they typically concentrate on "end points," whereas complex adaptive sys-

tems "never get there." Improvement is usually much more important than optimization. When parts of the system do settle down to a local optimum, it is usually temporary, and those parts are almost always "dead," or uninteresting, if they remain at that equilibrium for an extended period.

4. Complex adaptive systems anticipate.
   In seeking to adapt to changing circumstance, the parts develop "rules" (models) that anticipate the consequences of responses. At its simplest, this is a process not much different from Pavlovian conditioning. Even then, the effects are quite complex when large numbers of parts are being conditioned in different ways. The effects are still more profound when the anticipation involves lookahead toward more distant horizons. Moreover, aggregate behavior is sharply modified by anticipations, even when the anticipations are *not* realized. For example, the anticipation of an oil shortage can cause a sharp rise in oil prices, whether or not the shortage comes to pass. The effect of local anticipations on aggregate behavior is one of the aspects of complex adaptive systems we least understand.

The objective of the Santa Fe Institute is to develop new approaches to the study of complex adaptive systems, particularly approaches that exploit interactions between computer simulation and mathematics. Computer simulation offers new ways of carrying out both realistic experiments, of flight-simulator precision, and well-defined gedanken experiments, of the kind that have played such an important role in the development of physics. For real complex adaptive systems—economies, ecologies, brains, etc.—these possibilities have been hard to come by because (1) the systems lose their major features when parts are isolated for study, (2) the systems are highly history dependent, so that it is difficult to make comparisons or tease out representative behavior, and (3) operation far from equilibrium or a global optimum is a regime not readily handled by standard methods.

The Institute aims to exploit the new experimental possibilities offered by the simulation of complex adaptive systems, providing a much enriched version of the theory/experiment cycle for such systems. In conjunction with these simulations, the common kernel shared by complex adaptive systems suggests several possibilities for theory (cf. the work on the schema theory of genetic algorithms). In an area this complex, it is critical for theory to guide and inform the simulations, if they are not to degenerate into a process of "watching the pot boil." Theory is as necessary for sustained progress here as it is in modern experimental physics, which could not proceed outside the framework of theoretical physics. We need experiment to inform

theory, but without theory all is lost. The broadest hope is that the simulations will suggest well-informed conjectures that offer new directions for theory, while the theoretician can test deductions and inductions against the simulations. Only then can we fully reincarnate, for complex adaptive systems, the cycle of theory and experiment that is so fruitful for physics.

Echo

While interesting models of complex adaptive systems can be built with classifier systems, and classifier systems have indeed been used for this purpose (Marimon et al. 1989), there is an advantage to having a simpler model that places the interactions in a simpler setting, giving them sharper relief. *Echo* is one such model, properly a class of models, designed primarily for gedanken experiments rather than precise simulations. Echo provides for the study of populations of evolving, reproducing agents distributed over a geography with different inputs of renewable resources at various sites (see Figures 17 and 18). Each agent has simple capabilities—offense, defense, trading, and mate selection—defined by a set of "chromosomes." Though these capabilities are simple, and simply defined, they provide for a rich set of variations illustrating the four kernel properties of complex adaptive systems previously described. Collections of agents can exhibit analogues of a diverse range of phenomena, including ecological phenomena (e.g., mimicry and biological arms races), immune system responses (e.g., interactions conditioned on identification), evolution of metazoans (e.g., emergent hierarchical organization), and economic phenomena (e.g., trading complexes and the evolution of "money").

A precise description of Echo begins with definition of the individual agents (see Figure 19). The capacities of an agent are completely determined by a small set of strings, the "chromosomes," defined over a small finite alphabet. In the simplest Echo model, this alphabet consists of four letters $\{a,b,c,d\}$, called *resources*, and there are just two classes of chromosomes, *tag* chromosomes and *condition* chromosomes. The tag chromosomes determine the agent's external, phenotypic characteristics, and the condition chromosomes determine an agent's responses to the phenotypic characteristics of other agents.

There are just three tag chromosomes in the simplest model: (1) offense tag, (2) defense tag, and (3) mating tag. It is convenient to think of the tags as displayed on the exterior of the agent, counterparts of the signature groups of an antigen or the trademarks of an organization. These tags are a kind of identifying address, quite similar to the tags employed by classifier systems. There are also just three condition

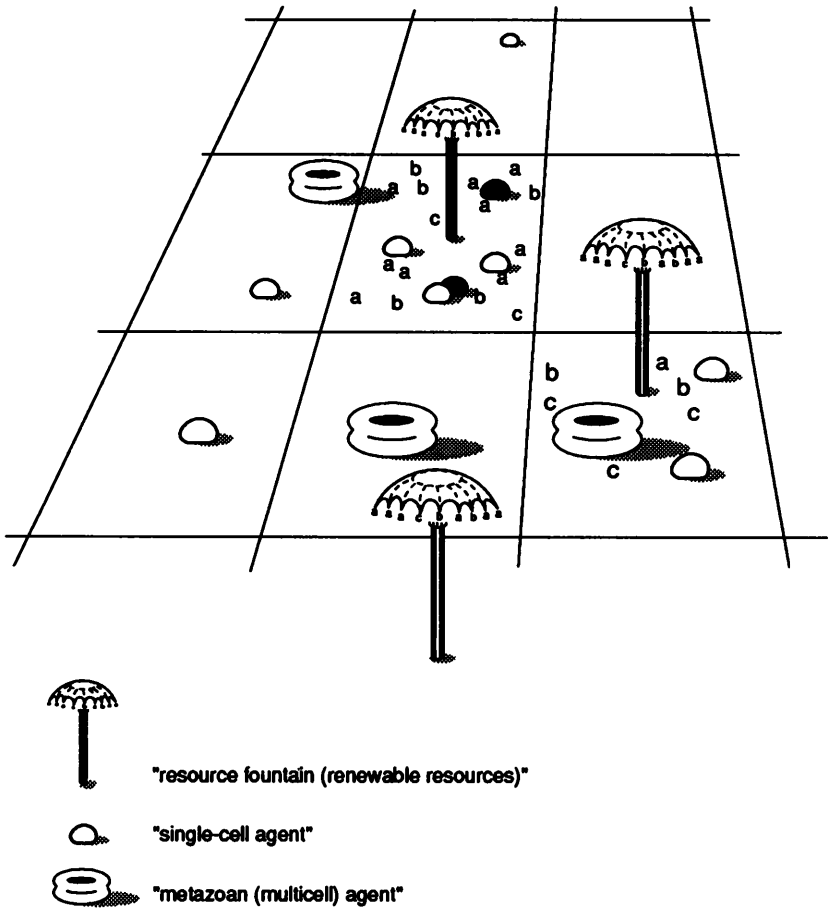


Fig. 17. Echo's "world"

chromosomes: (1) condition for combat, (2) condition for trading, and (3) condition for mating. Conditions serve much as the condition parts of a classifier rule, determining what interactions will take place when agents encounter one another.

The fact that an agent's structure is completely defined by its chromosomes, which are just strings over the resource alphabet $\{a,b,c,d\}$, plays a critical role in its reproduction. An agent reproduces when it "collects" enough letters to make copies of its chromosomes. As we will see, an agent can collect these letters through its

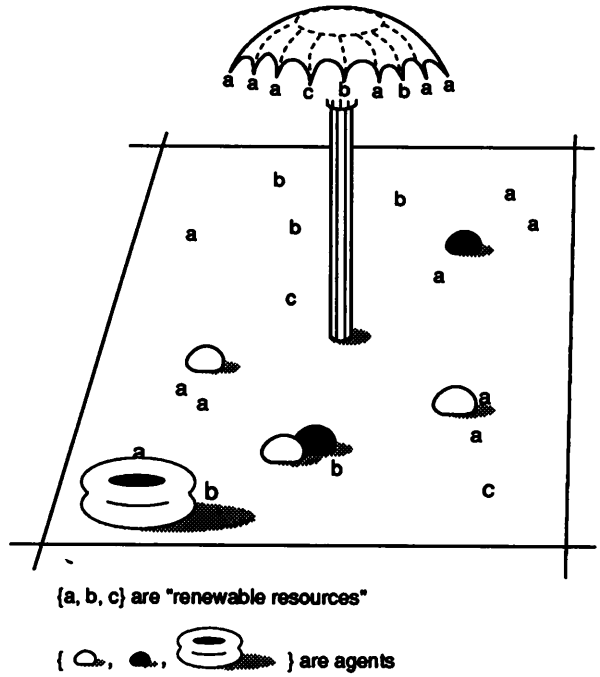


Fig. 18. A site in Echo

interactions: combat, trade, or uptake from the environment. Each agent has a *reservoir* in which it stores collected letters until there are enough of them for reproduction to take place.

Interactions between agents, when they come into contact, are determined by a simple sequence of tests based on their tags and conditions. In the simplest model, they first test for combat, then they test for trading, and finally they test for mating:

1. *Combat* (see Figure 20). Each agent checks its combat condition against the offense tag of the other agent. This is a matching process much like the matching of conditions against messages in classifier systems. For example, if the combat condition is given by the string *aad*, then this condition is matched by any offense tag that begins with the letters *aad*. (The condition, in this example, "does not care" what letters follow the first three in the tag, and it does not match any tag that has less than three letters.)

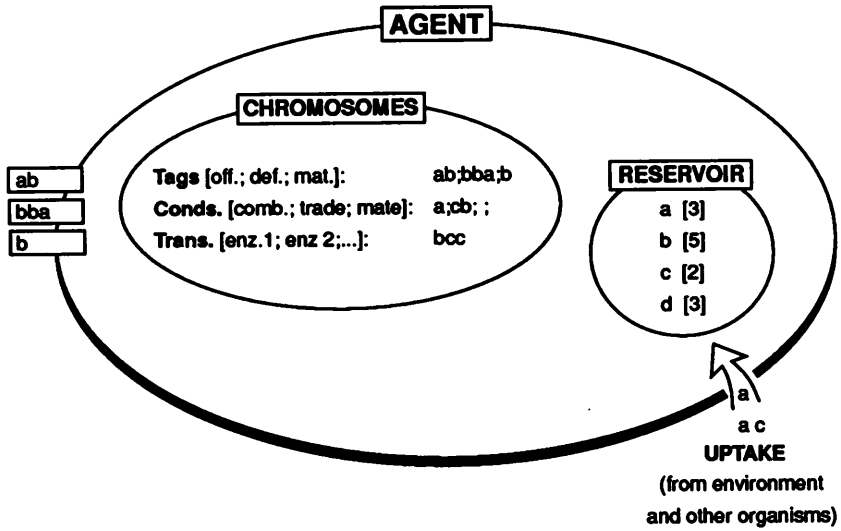
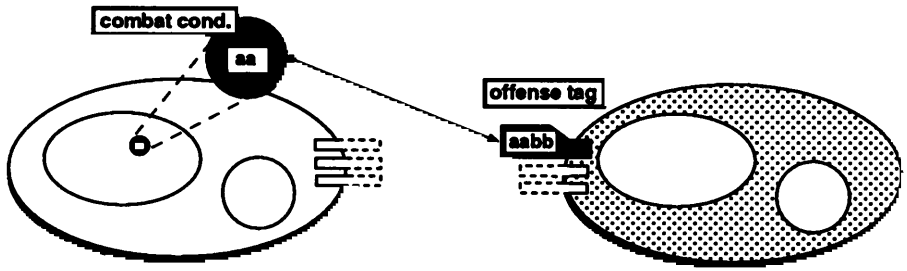


Fig. 19. A single-cell agent

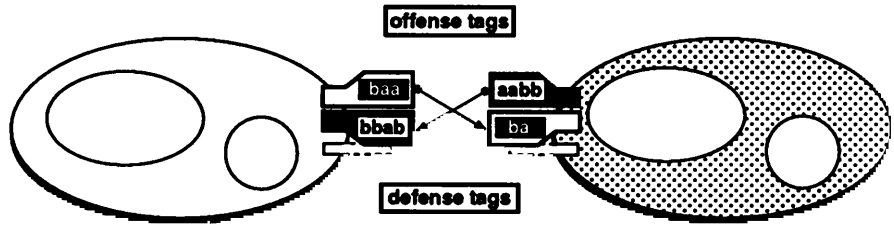
If the combat condition of either agent matches the offense tag of the other, then combat is initiated. That is, combat can be initiated unilaterally by either agent. If combat is initiated, the offense tag of the first agent is matched against the defense tag of the second and a score is calculated. In the simplest case, this score is calculated on a position-by-position basis, adding the results to get a total. For example, the score for a single position could be obtained from a score matrix that is used to score the match between corresponding letters in the two tags:

Offense	*a*	*b*	*c*	*d*
Defense				
a	4	0	2	1
b	0	4	2	1
c	2	2	4	0
d	2	2	0	4

1) When agent 1 [◯] moves into the vicinity of agent 2 [◯], combat occurs if agent 1's combat condition matches agent 2's offense tag.



2) Agent 1 is assigned a score based on the match between its offense tag and the defense tag of agent 2; a similar score is calculated for agent 2. The agent with the higher score is the winner.



3) The winner acquires from the loser the resources in its reservoir and the resources tied up in its chromosomes and tags. The loser is deleted.

Fig. 20. A typical interaction between agents

Under this matrix, the offense tag *aab* matched against the defense tag *aaaad* would yield a score of 4+4+0 = 8. (In this simple example, the additional letters in the defense tag do not enter the scoring; in a more sophisiticated scoring procedure, the defense might be given some extra points for additional letters). A score is also calculated for the second agent by matching its offense tag against the defense tag of the first agent. If the score of one agent exceeds the score of the other, then that agent is declared the winner of the combat. In an interesting variant, the win is a stochastic function of the difference of the two scores.

The winner collects the loser's resources, both the resources in its reservoir and the resources tied up in its chromosomes (broken into individual letters). In some models, the winner collects only some of the resources of the loser, the rest being dissipated. The provision of separate offense and defense capabilities, with possible asymmetries, allows the system to evolve intransitive relations between agents wherein, for example, *X* can "eat" *Y*, and *Y* can "eat" *Z*, but *X* cannot "eat" *Z*. As a consequence, various kinds of "food webs" can evolve.

2. *Trading*. If combat does not take place, then the first agent in the pair checks its trading condition against the offense tag of the second agent, and vice versa. Unlike combat, which can be initiated unilaterally, trading is bilateral—a trade does not take place unless the trading conditions of both agents are satisfied. The trading condition in the simplest model has a single letter, as a suffix, that specifies the resource being offered for trade. If the trade is executed, then each agent transfers any *excess* of the offered resource (amounts over and above the requirements for its own reproduction) from its reservoir to the reservoir of its trading partner. Though this is a very simple rule, with no bidding between agents, it does lead to intricate, rational trading interactions as the system evolves: Trades that provide resources needed for reproduction increase the reproduction rate, assuring that agents with such rational trading conditions become common components of the population.

3. *Mating*. While an agent can reproduce asexually, simply making a copy of each of its chromosomes when it has accumulated enough resources (letters), there is also a provision for recombination of chromosomes. When agents come into contact and do not engage in combat, the mating condition of each agent is checked against the mating tag of the other. As with trade, mating is only executed as a bilateral action: Both agents must have their mating conditions satisfied for recombination to take place. If this happens, then the agents exchange some of their chromosome material, as with crossover under the genetic algorithm. (The procedure is reminiscent of conjugation between different mating types of paramecia). This selective recombination provides a powerful mechanism for discovering and exploiting useful schemata. The effect is very like the effect that the schema theorem(s) of chapter 7 project, though the schema theorems cannot be applied directly to Echo's agents because they have no explict fitness function.

Sites

In addition to these three agent-agent interactions, there is one direct interaction with the environment. The geography of Echo consists of a set of *sites*, laid out in some

regular, or irregular, array (see Figures 17 and 18). Each site has a well-defined set of neighboring sites, and each site can contain a subpopulation of agents. In addition, each site is assigned a *production* function that determines how rapidly the site produces and accumulates the various resources. For example, one site may produce 10 units of resource *a* per time step, and nothing of *b*, *c*, or *d*, whereas another may produce 4 units each of *a*, *b*, *c*, and *d*. If the site is unoccupied by any agents, these resouces accumulate, up to some maximum value. In the example of the site that produces 10 units of resource *a* per time step, the site could continue to accumulate the resource until it had accumulated, say, a total of 100 units. Agents present at a site can "consume" these resources. Thus an agent located at a site that produces the resources it needs can manage reproduction without combat or trade, if it survives combat interactions with other agents. Different agents may have intrinsic limits on the resources they can take up from the site. For example, an agent may only be able to consume resource *b* from the environment, being dependent upon agent-agent interactions to obtain other needed resources. Resouces available at a site are shared among the agents that can consume them.

When neither agent-agent nor agent-environment interactions are providing at least one needed resource at a given site, an agent may migrate from that site to a neighboring site. For example, consider an agent that has already acquired enough of resources *a* and *b* to make copies of its chromosomes but that is not acquiring needed resource *c*. Then that agent will migrate to some neighboring site; in the simplest models the new site is simply selected at random from the neighboring sites.

The Simulation

The actual Echo simulation is designed so that, in effect, the populations at each of the sites in the model undergo their interactions simultaneously. In other words, Echo is well suited to execution on a massively parallel computer. The interactions at each site are carried out by repetition of the following basic cycle.

(1) Pairs of agents from within the site are selected for interaction. (In the simplest model, these pairs are simply selected at random from the local population). Each pair is tested for the kind(s) of action that will ensue following the procedures outlined above.

(1.1) First the pair is tested for combat, which may be invoked unilaterally.

(1.2) If combat is not invoked, then the pair is tested for trade, which can only be invoked bilaterally. The same pair is then tested for mating compatibility. If the agents are compatible, then, with low probability, recombination of their chromosomes will follow.

(2) Each agent in the site executes uptake of resources produced at the site.

(3) Each agent in the site is charged a "maintenance" cost, which must be payed by the "subtraction" of specified resources from its reservoir. If the cost cannot be met, the agent is deleted (some of its resources may be returned to the site, depending on the particular model). Each agent also has a small random chance of being deleted "without cause."

(4) Each agent in the site tests to see if it has accumulated enough resources in its reservoir, via steps (1) and (2), to make a copy of its chromosomes. If so, it replicates itself, with infrequent mutations.

(5) Each agent in the site, other than the replicates produced in step (3), tests to see if it has acquired at least one of the resources it currently needs for reproduction. If not, the agent migrates to one the neighboring sites.

(6) The production function associated with the site adds a specified number of units of each resource to the site, for later uptake in step (2).

The details of this basic cycle can be filled out in a variety of ways, depending upon the particular range of gedanken experiments of interest. Even the simplest models show surprisingly sophisticated evolutions. One of the earliest models produced evolving sequences of agents with ever longer, more complicated chromosomes, accompanied by a corresponding increase in the complexity of their interactions. The result was a "biological arms race" (Dawkins 1986), wherein defense tags became ever longer and offense tags developed ever more sophisticated matches to overcome the increasing defensive capabilities. More recent models, by a modification of the basic cycle, provide for the evolution of "metazoans"—connected communities of agents that have internal boundaries and reproduce as a unit. With this provision, agents belonging to a connected community can specialize to the advantage of the whole community. For example, one kind of agent belonging to the community can specialize for offense, while a second kind specializes in resource acquisition (somewhat reminiscent of the stinging cells and cavity cells of the hydra). It is easy to show that intracommunity trading between these specialists yields a net increase in the reproduction rate of both. As a consequence the metazoans come to occupy a significant place in the overall ecology of agents. Many of the mechanisms investigated by Buss (1987) can be imitated by this model, including the evolution of cooperation between cell lines (cf. Axelrod and Hamilton 1981) and the origin of such developmental mechanisms as induction and competence (see Figure 21).

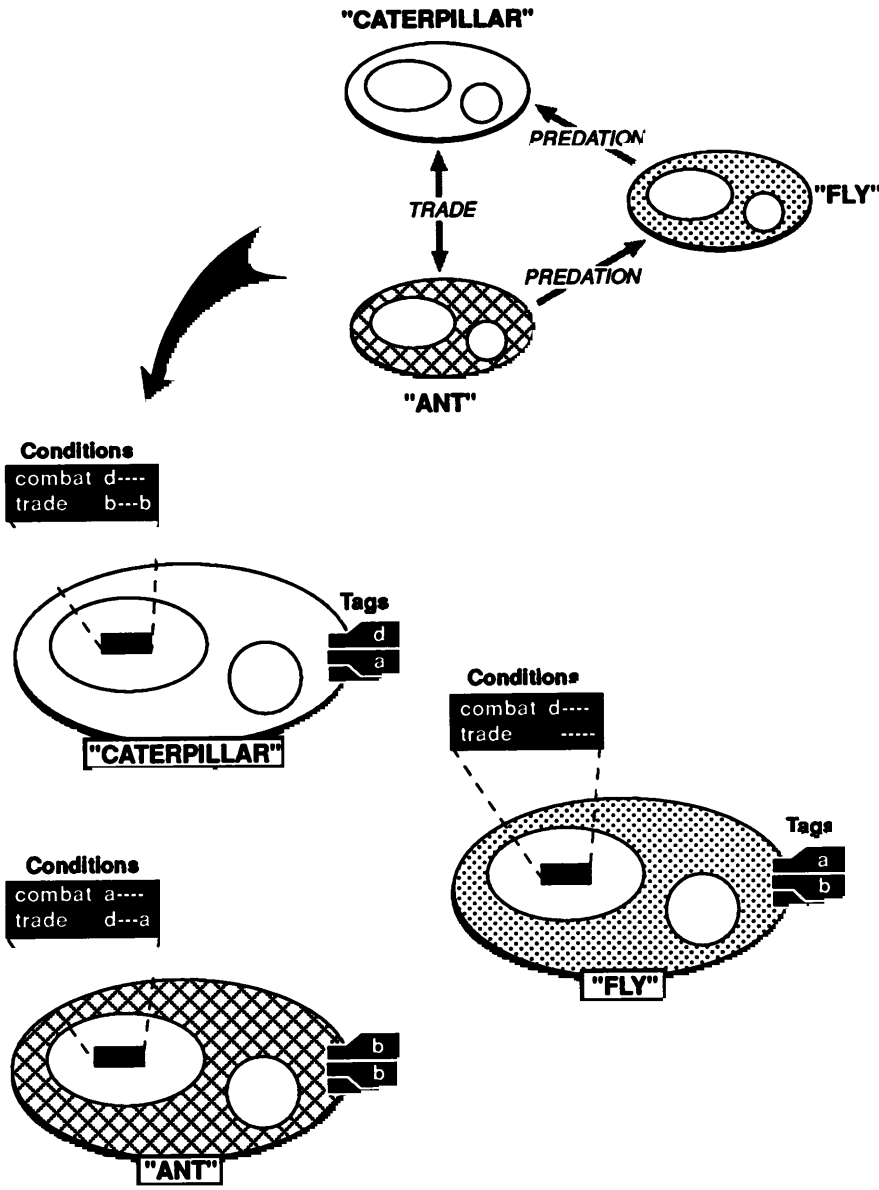


Fig. 21. A small ecology in Echo

Classifier Systems and Echo: A Comparision

Echo and classifier systems are similar in many ways. The conditions employed by an agent in Echo to determine its actions are quite similar to the condition/action rules of a classifier system. However, the actions in Echo (combat, trading, mating) are much more concrete than the rule-activating messages used by a classifier system. They are much easier to interpret when one is trying to understand aspects of distributed control and emergent computation in complex adaptive systems. Tags also play a critical role in both Echo and classifier systems, but again a tag's effects are much more directly interpretable in Echo.

Echo differs from classifier systems in two important ways. First, geometry is critical in Echo. This goes back to the origin of the Echo models (Holland 1976) where geometry played an important role in the spontaneous emergence of autocatalytic structures. In a similar way, the sites in Echo, with their differing resource production characteristics, encourage sophisticated agent ecologies. Second, there are no explicit fitness functions in Echo. The reproduction rate of an agent depends solely on its ability to gather the necessary resources in the context of other agents and sites. There is no number corresponding to the payoff used by a genetic algorithm, nor is there a counterpart of the payoff-derived strength of a classifier system rule. An economist would say that fitness has become endogenous in Echo, whereas it is exogenous in genetic algorithms and classifier systems. As a consequence, the emergent structures (agents) in Echo are much more a function of the overall context and much less a function of external constraints. This can be both an advantage and a disadvantage, but it does allow studies of emergent *functional* structures free from the confounding effects of external constraints.

4. POSSIBILITIES

Both Echo and classifier systems point up a salient characteristic of complex adaptive systems: In these multiagent systems it takes only a few primitive activities to generate an amazing array of structures and behaviors. Moreover, when the primitives are chosen with care, counterparts of these structures and behaviors can be found in all kinds of complex adaptive systems. Echo's primitives (combat, trade, and mating) and the phenomena they generate (arms races, cooperation, etc.) directly illustrate the point. Though the range of structures exhibited by complex adaptive systems is daunting, this "generator" characteristic offers real hope for a future general theory.

In pursuing a general theory, there is a traditional tool of physics that can be

brought to bear—the gedanken experiment. As the name implies, a gedanken experiment is a thought experiment. It extracts a few elements from a process in order to examine, logically, some critical effect produced by the interaction of these elements. Computers offer a way of extending the scope of gedanken experiments to much more complex situations. Echo has been designed as a computational base for gedanken experiments on complex adaptive systems.

Echo, and other models of complex adaptive systems, are readily designed for direct simulation on massively parallel computers. It is also possible to design interactive interfaces for these simulations that permit ready, intuitive interactions with the ongoing simulation, much as is the case with flight simulators. Thus, the "logic" of the simulation can be combined with the human's intuition and superb pattern recognition ability to provide quick detection of interesting patterns or events. This has the double value of providing reality checks on the design, while allowing investigators to bring their scientific taste and intuitions to bear in creating and exploring unusual variants.

By looking for pervasive phenomena in these gedanken experiments, we can study complex adaptive systems with a new version of the classic hypothesize-test-revise cycle. The "test" part of this cycle is particularly important because complex adaptive systems, as mentioned earlier, typically operate far from a steady state. They are continually undergoing revisions, and their evolution is highly history dependent. This, combined with the nonadditive nature of the internal interactions, makes it difficult to do controlled experiments with real complex systems. Computer-based gedanken experiments should help fill the gap.

In examining complex adaptive systems, there is one property that is particularly hard to examine *in situ*. Complex adaptive systems form and use *internal models* to anticipate the future, basing current actions on expected outcomes. A system with an internal model can look ahead to the future consequences of current actions without actually committing itself to those actions. In particular, the system can avoid acts that would set it irretrievably down some road to future disaster ("stepping over a cliff"). More sophisticated uses of an internal model allow the system to select current "stage-setting" actions that set up later advantageous situations (as in Samuel's [1959] use of "lookahead"). As pointed up earlier, the very essence of attaining a competive advantage, whether it be in chess or economics, is the discovery and execution of "stage-setting" moves. Internal models distinguish complex adaptive systems from other kinds of complex systems; they also make the emergent behavior of complex adaptive systems intricate and difficult to understand.

Internal models offer a second advantage in addition to the advantage of

prediction. They enable a system to make improvements in the absence of overt payoff or detailed information about errors. Whenever a model's prediction fails to match subsequent outcome, there is direct information about the need for improvement. An appropriate credit (blame) assignment algorithm can even determine what part(s) of the model should be revised. This is a tremendous advantage in most real-world situations where the rewards for current action are usually much delayed. Internal models enable improvement in the interim.

Though we readily ascribe internal models, cognitive maps, anticipation, and prediction to humans, we rarely think of them as characteristic of other systems. Still, a bacterium moves in the direction of a chemical gradient, implicitly predicting that food lies in that direction. The repertoire of the immune system constitutes its model of its world, including an identity of "self." The butterfly that mimics the foul-tasting monarch butterfly survives because it implicitly forecasts that a certain wing pattern discourages predators. A wolf bases its actions on anticipations generated by a mental map that incorporates landmarks and scents. Because so much of the behavior of a complex adaptive system stems from anticipations based on its internal models, it is important that we understand the way in which such systems build and use internal models.

A general theory of complex adaptive systems that addresses these problems will be built, I think, on a framework that centers on three mechanisms: *parallelism*, *competition*, and *recombination*. *Parallelism* lets the system use individuals (rules, agents) as building blocks, activating sets of individuals to describe and act upon changing situations (as described in the discussion of classifier systems). *Competition* allows the system to marshal its resources in realistic environments where torrents of mostly irrelevant information deluge the system. Procedures relying on the mechanism of competition—credit assignment and rule discovery—extract useful, repeatable events from this torrent, incorporating them as new building blocks. *Recombination* underpins the discovery process, generating plausible new rules from building blocks that form parts of tested rules. It implements the pervasive heuristic that building blocks useful in the past will prove useful in new, similar contexts. Overall, these mechanisms allow a complex adaptive system to respond, instant by instant, to its environment, while improving its performance. In so doing, as with classifier systems, the system balances exploration with exploitation.

When these mechanisms are appropriately incorporated in simulations, the systems that result are well founded in computational terms, and they do indeed get better at attaining goals in perpetually novel environments. It should be possible to take a first step toward a general theory of complex adaptive systems by formalizing

a framework based on these mechanisms. A second step would incorporate a mathematics that emphasizes process over end points. This mathematics would emphasize the discovery and recombination of useful components—building blocks—rather than focusing on fixed points and basins of attraction. At that point, the incipient theory should begin to provide the guidelines that make the computer-based experiments more than uncoordinated forays into an endlessly complex domain. Once we get that far, we come at last to a rational discipline of complex adaptive systems providing genuine predictions.