The Rationality of Adaptive Agents

John H. Holland

SFI WORKING PAPER: 1993-10-063

SANTA FE INSTITUTE

THE RATIONALITY OF ADAPTIVE AGENTS

John H. Holland

Neo-classical models in economics look toward the equilibria that result when agents of perfect rationality interact. This approach opens the study of economics to powerful mathematical tools, such as fixed-point theorems and allied techniques, and much of our current understanding of economic processes is a direct consequence of this mathematics. Despite these advances, there remain real economic processes that exhibit complexities not anticipated by perfect rationality models. An understanding of the dynamics of contemporary economies turns on an understanding of non-equilibrium processes such as innovation, emergent organization, changes invoked by speculations and anticipations based on partial knowledge, and the like. In other words, we now have to increase our understanding of non-equilibrium conditions mediated by agents of limited rationality.

The usual view of rationality focuses on deductive inference — the ability to derive consequences from facts — and perfect rationality extends this view by setting aside computational limitations. Under the assumption of perfect rationality, knowledge of a set of facts or premises (axioms) is equivalent to knowledge of all the consequences (theorems) the set implies. Human rationality is both more and less limited. It is more limited, because computational limitations can place solutions of problems, even simply defined problems, beyond human reach. For example, the game of chess can be defined with a dozen rules, yet hundreds of years of study of chess have not uncovered an optimal strategy for playing chess. Computer augmentation of human rationality has not substantially altered the outlook. Human rationality is less limited than perfect rationality because humans can often step right around the limitations of deductive inference, going from particulars to useful general-purpose models, a process called *induction*. In chess, despite the lack of an optimal strategy, we do have principles of good play that suggest both strategy and tactics in novel situations.

The objective of this paper is to explore the extent to which we can capture human rationality, both its limitations and its inductive capacities, in computationally-defined adaptive agents. Simulations based on computationally-defined agents open the possibility of exploring analogues of a wide range of economic processes under completely controlled conditions. This, in turn, opens the possibility of discovering patterns that are invariant under changes in parameters and initial conditions. A mathematics based upon such invariants, if we can discover them, would enable us to deal rigorously with the complexities attendant upon disequilibria and limited rationality.

The paper proceeds in three steps: First, it outlines a computationally-based definition of an *adaptive agent* (with suitable references for those who would like to fill in the details). Second, the paper explores characteristics of systems composed of many interacting adaptive agents, concentrating on effects of learning and experience-based anticipation. Finally, the paper sketches a few mathematical results concerning innovation in such interactive systems, to give some suggestion of the kinds of invariants that might lead to a productive theory.

1. Adaptive Agents.

I'll define an *adaptive agent* via the computation-based implementation of two processes: (1) a *performance system* that specifies the agent's capabilities at a given point in time, and (2) a *learning mill* that uses inductive apparatus to modify the performance system as experience accumulates. The next two sub-sections treat each of these processes in turn: the resulting system, called a *classifier system* (Holland et al. [1989]), provides a rigorous definition of an adaptive agent of bounded rationality.

The performance system.

A formal description of a performance system requires, first of all, specification of the system's ways of interacting with its environment (most of which consists of other agents). I will adopt the common view that the state of the environment is conveyed to the performance system via a set of

detectors (e.g., rods and cones in a retina, neurons in the cochlea, etc.) and that the system acts upon its environment via a set of effectors (e.g., muscles). I will also adopt the view that the detectors present the performance system with standardized packets of information called *messages*. With these conventions, we can say that the performance system processes messages from the detectors to determine effector settings that act upon the environment.

There is one further channel of interaction with the environment that must be considered: Under some circumstances the environment provides the agent with payoff (reward, reinforcement, resource acquisition). That is, from a game-theoretic point of view, some situations amount to overt "wins" or "losses", with associated payments. The rate at which the agent acquires payoff is a measure of its performance, and is a vital element in any careful discussion of learning or adaptation (see the discussion of the learning mill).

The detailed definition of the performance system is conveniently carried out in terms of a set of *condition/action rules*. The condition part of each rule "looks for" certain kinds of messages and, when those messages are present, the action part of the rule posts a message in turn. The rules take a particularly simple form if the messages are all of a standard length (say a binary encoding of a certain number of bits, as for a computer register); it is easily established that this restriction does not effect the agent's computational power. Many rules can be active simultaneously, so many messages may be present at a given time. These messages determine both the internal (rule) and external (effector) activity of the performance system. 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. All rules simultaneously check the message list, and active rules add new messages to the list.

It may be helpful to think of the performance system as a kind of office wherein the message list is a bulletin board holding the memoranda that must be handled that day. Each rule corresponds to a "desk" that has responsibility for certain kinds of memos. At the beginning of a day, each desk collects the memos for which it is responsible, processes them, and then, at the end of the day, posts the memos that result from its processing. At the beginning of each day, memos come either from the previous day's work (messages produced by the rules) or from outside the office (messages produced by the environment). Some memos cause actions outside the office (messages that control effectors). The possibilities for concurrency and coordination — *parallelism* — are obvious.

Parallelism, in particular, makes it possible for the performance system to combine rules into clusters that model the environment. This offers clear advantages over using monolithic structures to handle each possible situation. When a separate rule is required for each possible combination of inputs, the torrent of relevant and irrelevant data impinging upon an agent generates a combinatoric explosion. If, instead, we use different rules to describe different aspects of the current situation, we gain two important advantages:
(1) Combinatorics work for the system instead of against it. The advantage is similar to that obtained when one describes a face in terms of components, instead of treating it as an indecomposable whole. If we select, say, 8 components for the face -- hair, forehead, eyes, nose, mouth, chin, and the like -- and allow 10 alternatives for each, then one hundred million faces can be described by combining components, at the cost of storing only 80 individual components.
(2) Experience can be transferred to novel situations. A given rule can be used as a building block in many combinations, just as a single alternative for a facial component, say a particular nose shape, can be used with alternatives for each of the other components (ten million possibilities). If the rule proves useful in a fair sample of these contexts, it is at least plausible to believe it will prove useful in similar combinations not yet encountered.

To exploit these possibilities, the rules must be organized in a way that permits facile reorganization of the system. The first step in this direction is to adopt the tenet that rules serve as hypotheses rather than as incontrovertible facts. As experience accumulates, rules are progressively

confirmed or disconfirmed (see the discussion of the learning mill). The agent's reliance upon a rule is based upon its average usefulness in the contexts in which it has been tried previously. This rating is summarized in a quantity called the rule's strength (see the discussion of credit assignment).

To foster coordination and easy reorganization, we add the requirement that all rules with satisfied conditions at itme t enter a competition for the right to post their messages. The competition is based on a bidding process: Each satisfied rule makes a bid based upon its strength and the specificity of its conditions. A rule that has been useful to the agent in the past (high strength) and uses more information about the current situation (high specificity) will make a higher bid. Higher bidders have a higher probability of winning the competition. Usually there will be many competitors and several winners; only the winners actually post messages. Thus, the competition between rules determines which cluster of rules is active at a given time.

Competition provides a simple, situation-dependent means of resolving conflicts when sets of concurrently active rules attempt to generate responses. Instead of attempting to maintain the consistency of the full rule repertoire, the system treats contradictory satisfied rules as altemative hypotheses about the current situation. When such alternatives are presented, the system uses competition (and local consistency) to decide which rules to favor, even though the particular combination of rules satisfied may not have been encountered before. The result is a flexible procedure for bringing previous experience to bear on a current, possibly novel, situation.

[Chapters 3 and 4 of Holland et al. [1989] provide details concerning condition specificity, emergent default hierarchies, support provided by partial information, and the like.]

The learning mill.

We are now ready to discuss the ways in which experience modifies the performance system. Experience acts upon classifier systems in two ways. First, via a process called *credit assignment* , it modifies the strengths of rules already present in the performance system, and second, via a process called *rule discovery* , it generates new rules (hypotheses) to be tried. I will discuss these processes in order.

Credit assignment.

Credit assignment is not particularly difficult when the system receives a payoff from the environment — the system simply strengthens all the rules active at that time (a kind of conditioning). Credit assignment becomes difficult when credit must be assigned to early-acting rules that set the stage for a sequence of actions leading to payoff. Then the system must decide which rules active along the way actually contributed to the outcome. Parallelism adds to the difficulty. Only a few of the rules active along the way may contribute to the favorable outcome, while others are ineffective or, even, obstructive. Somehow the credit assignment algorithm must sort this out, modifying rule strengths appropriately.

Classifier systems use the performance system's bidding process as the vehicle for credit assignment. To do this, each rule is treated as a "middleman", buying and selling messages via its bids and postings. A rule's "suppliers" at any instant are those rules that have posted messages satisfying its conditions; its "consumers" are those rules that have conditions satisfied by the message it posts. Under this regime, we treat the strength of a rule as an asset that can be cashed. When the rule wins a bidding competition, it must draw down on its strength to pay the amount of the bid to its current suppliers. (A rule only pays when it wins the bidding process). As a result, the winning rule's strength is reduced by the amount of the bid, and the strengths of the suppliers are increased in proportion.

A winning rule can recoup its payment in two ways: (1) Through payments made to it, in turn, by its consumers, or (2) through a share of a payoff from the environment. In case (2) we reach the point, alluded to in the initial discussion of interactions with the environment, where payoff affects the

performance system. Because payoff is shared only amongst rules active at the time of payoff, the system must rely on the "middleman" exchanges to reward stage-setting rules. The overall credit assignment process that implements these exchanges is called a *bucket brigade algorithm*.

In broad outline, the bucket brigade algorithm works because rules become strong only if they are coupled into sequences leading to payoff. To see this, note first that rules consistently active at times of payoff tend to become strong because of the payoff they receive. As these rules grow stronger, they make larger bids because, as noted earlier, bids are proportional to strength. Consider now a rule that consistently sends a message activating one of the "payoff" rules — a supplier of the "payoff" rule. The increased bids of the payoff rule means that this supplier receives larger payments, and becomes stronger in turn. Subsequently, the suppliers of the suppliers begin to benefit, and so on, back to the early stage-setting rules. The whole process, of course, takes repeated "plays of the game".

The bucket brigade algorithm is an important component of the limited rationality of agents based on classifier systems. The algorithm requires no overt memory of the long and complicated sequences leading up to payoff; it only requires knowledge of the immediate suppliers and consumers of each rule. The alternative, a complete memory of all transactions and interactions between payoffs, would involve teasing out relevant strands from a tangled mass of strands that includes unnecessary detours and incidentals. Even for modest parallel systems acting in environments with sporadic payoff, this teasing out process quickly exceeds compuational capacity. The bucket brigade algorithm avoids these complications. Chapter 3 of Holland et al. [1989] provides details of the workings of the bucket brigade algorithm.

Rule discovery.

The second process employed by the learning mill is rule discovery. Because rule discovery is the sine qua non for an adaptive agent, it constitutes the centerpiece of this paper. In a rule-based system, the whole process of induction succeeds or fails in proportion to its efficacy in generating plausible new rules. However, plausibility is not an easy concept to pin down. It implies that experience biases the generation of new rules, but how?

The starting point is the proposal that plausibility is closely linked to the "building block" approach set forth earlier in the discussion of parallelism. Applied to an individual rule, the building block approach requires that the rule be viewed, not as something monolithic, but as an entity constructed from well-chosen parts. If a part appears in several rules, then those rules constitute samples drawn from the set of all rules that can constructed from that part. By calculating the average strength of the rules observed to contain the part, we can estimate the usefulness of the part. Though 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". By treating parts of rules as building blocks we can approach the generation of plausible rules directly.

Note that a rule can be divided up into component parts (potential building blocks) in a great many ways. That is, even a single rule constitutes a sample point for a great many possible building blocks. A fortiori, an agent defined by a few hundred rules provides samples for a vast array of building blocks. Many building blocks will appear in a dozen or more different rules, and different building blocks will appear in different subsets of rules. Each such subset is large enough to provide an estimate of the corresponding part's average usefulness in rule formation. So, an adaptive agent quickly accumulates enough information to make good estimates of the relative usefulness of multitudes of building blocks.

If the learning mill can extract this information, it can bias the rule generation process so that above-average building blocks are favored in the construction of new rules. However, there is a difficulty. The large numbers of building blocks involved make an explicit calculation of all the relevant averages a computationally infeasible task; and, even if the calculations could be made, the biasing

procedure would be difficult to implement. Fortunately, there is a way of achieving the effect of these calculations without carrying them out explicitly.

Recapitulation.

An adaptive agent, to be effective, must continually balance exploration (acquisition of new information and capabilities) against exploitation (the efficient use of information and capabilities already available). As formulated here, the adaptive agent is described in terms of a rule-based performance system. Rules are assigned strengths that are modified by a credit assignment algorithm to reflect past usefulness to the agent. Three mechanisms help the agent to balance exploration and exploitation:
1) *Parallel execution* of rules allows transfer of experience to novel situations by the combined activation of relevant rules -- building block rules -- that describe aspects of the situation.
2) *Tags and rule coupling* provide for directed sequential action, making a "bridge" for credit assignment to stage-setting actions. Tags also provide for adaptive clustering of rules.
3) *Competition* , based on rule specificity and strength, allows rules to be treated as hypotheses to be marshaled and progressively confirmed (by strength revision under credit assignment) as required by the changing environmental situation. ,
To provide new hypotheses, a genetic algorithm treats strong rules as parents, recombining parts of the parents to provide offspring rules that replace weak rules (hypotheses). The resulting adaptive agent is well-defined in computational terms. Via these mechanisms it constructs increasingly sophisticated internal models (default hierarchies, for example) that enable it to anticipate its environment. Its predictions are continually tested against outcomes, with falsifications being used to improve the models, even in the absence of payoff. Though the agent readily improves it performance, it uses only computationally simple procedures to do so. As such, it conforms to reasonable notions of bounded rationality.

---

2. Genetic Algorithms.

The "shortcut" can explained by resorting to a metaphor from genetics: The learning mill selects high strength rules from the agent's repertoire to serve as "parents", then it produceds new "offspring" rules by exchanging parts between the parents. The offspring replace low-strength rules in the repertoire. Though it is not obvious, we will see that this procedure, implemented as a *genetic algorithm* , biases the generation of new rules toward the use of above-average building blocks (see Belew and Booker [1991] for a wide range of papers on genetic algorithms).

To make this notion precise, we need a formal counterpart of the building-block concept. The starting point is the observation that a building-block can be formally identified with the set of *all* objects that have the building-block as a component. Here, the set of *all possible* objects is just the set C of all rules (strings) in the classifier language, so that a particular building-block corresponds to a particular *subset* of C.

Abstractly, C can be taken as the set of all binary strings of some length k: that is $C = \{1,0\}^k$. (It is not difficult to deal with arbitrarily long strings of different lengths, but the same points can be made with strings of fixed length). The subsets of C corresponding to building-blocks are defined with the help of a 'don't care' symbol '*'. Let $C^* = \{1,0,*\}^k$ be the set of all strings of length k over the three-letter alphabet {1,0,*}. Each string s in C^* can be used to designate a particular *subset* of C. To see this, let c_i designate the bit at the ith position of $c \in C$, and let s_i designate the letter from {1,0,*} at the ith position of $s \in C^*$. Then $c \in s$ if and only if, for each position i, (1) if $s_i \neq *$, then $c_i = s_i$, otherwise (2) c_i can equal either 1 or 0. For example, the string 1**** from $\{1,0,*\}^5$ designates the *set* of all binary strings of

length 5 that start with a 1, namely the set {10000, 10001, ... 11111}. The building-blocks (subsets of C) designated by the strings in C* are called *schemata* .

The genetic algorithm does not directly manipulate schemata (building blocks), it only manipulates strings (rules). To express this point formally, let R(t) designate the set of rules used by the performance system at time t. R(t) can be looked upon as a sample set drawn from C. The genetic algorithm uses the strongest rules in R(t) as 'parents' to generate new elements from C. These new elements replace some elements in R(t) to form R(t+1). In forming these new rules, the genetic algorithm implicitly recombines schemata common to the better elements of R(t). As we will see, the best way to understand what a genetic algorithm *does* is to understand its effects upon schemata.

To proceed, in assigning a strength u(c) to each rule $c \in R(t)$, the bucket brigade algorithm implicitly defines a utility function u: C -> reals. The genetic algorithm uses the information u(c), $c \in R(t)$, to act on R(t) as follows (the reader is referred to chapter 6 of Holland [1992] for details):
(1) A probability distribution P(t) is assigned to R(t), with the constraint that $P(c_1,t) > P(c_2,t)$ if $u(c_1) > u(c_2)$. [For example, $P(c) = u(c)/\sum_j u(c_j)$, where the sum is over all $c_j \in R(t)$.]
(2) A set of n pairs is drawn, with replacement, from R(t) using the probability distribution P(t), where $2n < M$, the number of rules in R(t).
[These pairs serve as "parents", generating new trials to be drawn from C.]
(3) The strings in each pair are *crossed* by randomly selecting a different position i for each pair, $1<i<k$, and exchanging the portions of the string to the left of i between the strings in the pair.
[For example, if the pair is (01010, 11100) and i=2 for that pair, the result would be the pair (11010, 01100).]
(4) Each bit in the resulting pair of strings has a small probability p_m of being *mutated* to the opposite value; typically $p_m < .01$.
[For example, if the second bit in the string 11010 were mutated the result would be 10010.]
(5) R(t+1) is formed from R(t) by using the strings just formed by crossover and mutation to replace 2n strings drawn with uniform probability, without replacement, from R(t).
[If 2n is substantially smaller than M, the 'offspring' strings are unlikely to replace the parent strings; instead they replace strings of lower utility. Thus, the new population retains (most of) its information about the better strings it has previously uncovered in the space C of possibilities.]
(6) Increment t to t+1 and return to step (1) for the next iteration.

The Schema Theorem.

Though the *crossover* operation in step (3) of the genetic algorithm seems rather strange as a mathematical operator, it actually produces a sophisticated search that exploits useful building-blocks. The schema theorem makes this point rigorously. It shows that a genetic algorithm increases the rate of use (the sampling rate) of *all* (sufficiently compact) schemata of above-average utility, biassing the sampling of the space of possibilities C accordingly. This is unexpected because the number of distinct schemata in a single binary string of length k is easily shown to be 2^k, so that the number of distinct building-blocks with instances in a set of size M is somewhere between 2^k and $M2^k$. That is, the number of building-blocks being tested and exploited is of order 2^k. If k is at all large, this greatly exceeds the number M of strings being processed. For example, if k=20 and M=1000, the ratio exceeds 1000 to 1. This property of trying out and exploiting many more schemata than the number of strings being processed is called *implicit parallelism* (née *intrinsic parallelism* in the original 1975 edition of Holland [1992]).

The following notation provides for a precise statement of the schema theorem:
M(s,t) = the number of instances of schema s in R(t).
[For example, if s = 1****, then M(s,t) gives the number of strings in R(t) that start

with a 1.]

P(s,t) = M(s,t)/M.

u(s,t) = the average utility of the instances of s in R(t).

u(t) = the average utility of all strings in R(t).

d(s) = the number of defining bits (non-* letters) in the string from C* defining the schema s.

L(s) = the number of possible crossover points between the outermost defining bits in the string defining the schema s (the "compactness" of s is 1/L(s)); if s has only one defining bit, L(s)=0.

Theorem (Schema Theorem). $P(s,t+1) \geq [1+\Delta(s,t)]P(s,t)$

where $\Delta(s,t) = ([u(s,t)/u(t)][1 - L(s)/(k-1)][1-p_m]^{d(s)} - 1)(2n/M)P(s,t)$.

From left to right, the factors on the right-hand side of the equation for $\Delta(s,t)$ have the following interpretation:

[u(s,t)/u(t)] is the expected rate of increase (or decrease) of instances of s in the absence of crossover and mutation:

[1 - L(s)/(k-1)] sets an upper bound on the probability that schema s will be 'broken up' by crossover — it is the probability that a pair of strings will be crossed outside of the outermost defining bits of s, using L(s)/(k-1) as an upper bound on the probability that an instance of s will be broken up by the crossover operator.

$[1-p_m]^{d(s)}$ is the probability that the schema s will not be destroyed by mutation:

(2n/M)P(s,t) is the probability that a given string in R(t) will be in the set of 2n samples drawn from R(t), multiplied by the probability that that string is an instance of the schema s.

[Because the genetic algorithm is stochastic, this equation only provides a bound on expectations. In the terminology of mathematical genetics, the equation corresponds to a deterministic model of the algorithm. The interested reader can find a proof, with 2n = M, as theorem 6.2.3 in chapter 6 of Holland [1992].]

Schemata that consistently appear in above-average rules are used in exponentially increasing numbers as the performance system develops under the genetic algorithm. (Eventually, when a schema comes to occupy a large proportion of R(t), there is an asymptotic 'slowdown' in the rate of increase, as shown by the schema theorem). In this process, crossover's most important effect is to generate new combinations of schemata already in R(t). That is, crossover forms strings that contain schemata not previously on the same string. It does this with increasing efficiency and probability for schemata that occupy large portions of R(t). Crossover also acts to generate instances of schemata not before observed. Acting in concert with the implicit parallelism established by the schema theorem, crossover provides a sophisticated, experience-based rule discovery process.

3. Innovation.

Much that goes on in generating innovations parallels speciation in genetics, particularly speciation in the context of an ecosystem. Each device (genotype) tried amounts to a simultaneous test of all of its component building-blocks. Marketing of variants of a profitable prototype assures that above-average building-blocks are repeatedly tried in new, related contexts. This, in turn, assures that the observation of above-average return, from devices using the building block, is not a statistical fluke, while simultaneously exploiting the building-block in new contexts. Innovations produced by recombination are exemplified by things as different as the chimera of medieval beastiaries, the combustion engine (which combines the geared wheels of a water mill, the pistons of Watt's water pump, Volta's sparking device, and Venturi's vaporiser), and the digital computer (originating with Babbage's combination of geared wheels, Jacquard's loom control cards, and Gauss's arithmetic). Upon occasion, new building blocks are discovered, sometimes by rearrangement of still more elementary building blocks; new building blocks

usually give rise to technological revolutions.

The abstract version of this process is a search through an enormous space of possibilities, biased toward intersections of tested regions (that is, rearrangements of tested parts). The genetic algorithm serves as s a rigorous, simplifed prototype of the process. Innovation, so described, seems so straightforward that one wonders why innovation is not more common (though it is common enough in contemporary economies). A closer look at the action of the genetic algorithm provides a hint of the impediment. The difficulty centers on what, at first sight, seems an unblemished advantage: the exponential increase in the number of instances M(s,t) of any above-average schema s. This rapid 'rise time' of above-average schemata entails a hidden reduction in overall search rate called *hitchhiking* .

To see this effect, consider what happens when the first instance of a schema s, call it c_S, is uncovered. As the genetic algorithm produces new copies of c_S, the particular bit values at loci *other than the defining bits of s* will also appear in the replicas of c_S. Indeed, were it not for the actions of crossover and mutation, all of the 'offspring' of c_S would be copies of c_S. These bit values at other loci "hitchhike" as the number of instances of s increases. The resulting "lock-in" of these bits has an effect much like superstition, producing a bias toward detrimental incidentals that accompany a useful outcome. The lock-in greatly reduces the probability of discovering other schemata disjoint from s.

A rigorous examination can be carried out by concentrating on a defining locus l_j of a schema s' disjoint from s. If exact replicas of c_S should come to occupy the whole of R(t), then all strings in the population would have the bit value at l_j that had hitchhiked along during the 'growth' of s. If this value were different from the one required by s' at that locus, then there would be no samples of s' in R(t). R(t) would be "locked-in" to a configuration that did not allow any samples of s'. The lock-in would hold sway until mutation switched the value at l_j in some future generation R(t+t'), t'>0. Hitchhiking therefore tends to neutralize implicit parallelism.

Setting the mutation probability p_m to a high value, so the bit values at all loci are frequently changed from generation to generation, is one way to deter hitchhiking, but it is *not* a good way. The factor $[1-p_m]^{d(s)}$ in the schema theorem makes it clear that a high mutation rate interferes with the exploitation of schemata already discovered. More generally, mutation is a history-independent operation that does not make use of the system's knowledge or past experience.

Rather, the objective should be to balance the effects of replication and mutation, using a low mutation rate, to retain implicit parallelism while deterring hitchhiking. To examine this possibility, I'll use the following additional notation: Let $P^*(t)$ be the proportion of strings in R(t) that are *free* of hitchhiking bits at one or more of the d defining loci of schema s' disjoint from s; let u+b, b>0, be the average utility of strings that are instances of s, where u is the average utility of strings that are *not* instances of s.

Make the 'worst case' assumption that the offspring of instances of s all carry a complete set of hitchhiking bits at the d defining loci of s', except for strings that undergo a mutation at one of those loci. This sets a lower bound on $P^*(t)$. Using the schema theorem (without the crossover factor), the expected number of such mutated offspring of instances of s is given by

(1) $\quad 2nP[(u+b)/u_{av}][1-(1-p_m)^d]$,

where $u_{av} = P(u+b) + (1-P)u = u+Pb$.

Assume that the offspring of *free* strings are free: that is, assume that the probability is negligible that the d bits at the defining loci of s' mutate 'back' to the particular combination of hitchhiking bits.

Then the expected number of free offspring of the free strings is given by

(2) $2nP^*[u/u_{av}]$.

The random draw, from R(t), of strings that are to be replaced by offspring, reduces the expected number of free strings by the amount

(3) $-2nP^*$.

Summing (1), (2), and (3) gives the net change Δ in the number of free strings following one iteration of the genetic algorithm:

$$\Delta \cong 2n[P(p_m d)(u+b)/(u+Pb) + P^*u/(u+Pb) - P^*],$$

using the approximation $[1-(1-p_m)^d] \cong p_m d$ when $p_m << 1$. Setting $\Delta = 0$ yields an approximation to the non-zero fixed point at which the proportion of free strings is no longer changing.

$$0 \cong P(p_m d)(u+b)/(u+Pb) + P^*u/(u+Pb) - P^*.$$

Solving gives an approximate value for P^* at the fixed point,

$$P^* \cong p_m d(1+u/b).$$

In this equation for P^*, the ratio u/b gives the relative advantage of schema s. This relative advantage can be controlled by scaling b in the genetic algorithm. That is, when an instance of s is first discovered, it can be assigned the value u+b by fiat (assigning u+b rather than the value u(s) directly observed) using the average value, u, calculated for the other strings in R(t).

In a biological setting, a typical value for b would be of the order 0.2u, and a typical value for p_m might be 0.005. With $b = 0.2u$ and $p_m = 0.005$, we see that, for a schema s' defined over d=10 loci, $P^* \cong 0.3$. That is, because of the hitchhiking effect, about 1 sample in 3 will provide new information about the 2^{10} bit combinations possible at the 10 defining loci for schema s'. Stated another way, about $3.3 \cdot 2^{10}$ strings will have to be processed by the genetic algorithm to locate an instance of s'. At these same values, instances of s occupy a proportion of R(t) given by

$$P(s,t) \cong (1-P^*) = 0.7 .$$

Thus, with an appropriate setting for p_m, it is possible to limit the hitchhiking effect (lock-in) while exploiting useful schemata already discovered. These proportions are closely approximated in actual simulations (Mitchell [1993]).

There is one last problem: Schemata with fewer defining bits than the d_{min} used to set the lower bound P^* on the sampling rate will have a sampling rate $P<P^*$. This difficulty can be managed by setting the size M of the initial, randomly generated, population large enough that schemata s' of length $d(s')<d_{min}$ have an expectation of several instances in that initial population. Setting $M > c2^{d_{min}}$, where c is a small integer, will accomplish this; for example, with $d_{min} = 9$ and $c = 5$, the initial population should have size $M > 2560$. With this provision, *all* variants of *all* schemata with $d<d_{min}$ defining loci are present in the initial population, with an average of c copies, making discovery of such schemata a fait accompli.

Summing up: If a newly discovered schema has a value substantially above the population average, $u+b >> u$, then that schema will quickly come to occupy a large part of the population. But this comes at the cost of a substantial hitchhiking effect, which lowers the chances of finding additional above-average schemata. Scaling b to very small values deters hitchhiking, but then new schemata only slowly come to occupy a substantial portion of the population, and they may be lost along the way because of

mutation or crossover. The results just given show that b can be scaled, in conjunction with the mutation rate, so that new schemata can be discovered *and* above-average schemata already discovered rapidly come to occupy a large proportion of the population. Morevover, the results suggest it takes only a small amount of random local variation ($p_m = .005$) to provide a sufficient "insurance policy", sustaining implicit parallelism in the face of hitchhiking effects.

Though the mathematical arguments here are based on particular definitions and algorithms, I think the results offer useful guidelines for more general studies of innovation. The arguments turn on the use of schemata as the formal counterpart of 'building-blocks', but even this restricted definition has direct interpretations in any domain in which the objects of interest can be usefully represented as strings. Schemata can readily be interpreted as coadapted sets of alleles in genetics, active sites of antibodies in immunology, or rule components for agents in economics, for instance. Even when there is no easy interpretation of schemata, many of these results can be re-interpreted for building-blocks more broadly interpreted. Consider, for example, the formal operators used in the arguments: sampling biased by observed utility, recombination (crossover), and local variation (mutation). Each of these operators has counterparts in broader domains. It is commonplace to recombine building-blocks to arrive at *plausible* innovations — as mentioned earlier, we generate everything from unicorns and gryphons to new jet engines in this way. Moreover, we tend to favor building-blocks that have worked in the past in similar contexts (sampling biased by observed utility). It is a way of transferring past experience to new situations.

4. Interactons of Adaptive Agents.

With these preparations we can begin to look at the interactions of agents that are capable of innovation, though of limited rationality. I would submit that such interactions are a common aspect of modern economies and worth direct, rigorous investigation. Indeed such a theory would be of help in understanding a broad range of complex adaptive systems (treated on different time-scales), including ecosystems, the immune system, and cognitive aspects of the central nervous system.

In the present context of performance systems described in terms of rules, meaningful interaction means that the rules of one agent must be able to apprehend characteristics of other agents. One way to accomplish this it to make one or more of the messages on an agent's message list available to other agents. That is, if agents 'display' some of their messages, informed interactions between agents become possible. Such displayed messages, which I will call *tags*, proclaim characteristics of the agent's internal state, serving much like antigen markers in the immune system, phenotypic markers of an organism, or even political slogans and banners.

A population-based version of the Prisoner's Dilemma (PD) provides a simple example of the effects of such "surface" tags in a population of interacting agents. (For a good introduction to studies of the PD using genetic algorithms see Axelrod [1987]). In the simplest version, each agent displays a single message, which is held constant throughout the agent's existence. Agents in the population come into contact via random pairings and, when two agents do come into contact, each is given the option of deciding whether or not it wants to execute one play of the PD. If both agents decide to play, then one play of the game is executed, otherwise no interaction takes place. In this simple version, each agent has only a single rule sensitive to the displayed message of the other agent, and the execution decision is made by that rule. If the condition of that rule is satisfied by the displayed message of the other agent, then the agent offers to execute one play of the PD. If an interaction does take place, each agent determines its particular move ("cooperate" and "defect" are the two options) according to a strategy determined by its other rules.

The four possible outcomes of an interaction (cooperate-cooperate, cooperate-defect, defect-cooperate, defect-defect) set a payoff for each agent determined by one of the standard payoff matrices for the PD (say [3,3], [5,0], [0,5], [1,1] in the given order of outcomes). The rate at which a given agent

collects payoff determines the probability that it will be selected as a parent under the genetic algorithm. This, in turn, determines the frequency with which its building blocks appear in the overall population. In particular, it determines the frequency with which various displayed tags appear in the population. As the frequency of a given kind of tag increases, the frequency of interactions involving that tag also increases.

Earlier experiments with selective mating based on tags are relevant here (Perry [1984]). In those experiments, an early, accidental association of a tag with a trait conferring a reproductive advantage is rapidly amplified because of the higher reproduction rate of the tag's carriers. For example, such an advantage is conferred by a tag associated with "compatible" mates that produce fewer lethal offspring under crossover. The tag, originally meaningless, takes on a meaning. It comes to stand for a particular kind of compatibility. By developing selective mating conditions based on the tags, the agents can react to this compatibility, thereby increasing their fitness.

In the population-based Prisoner's Dilemma, payoff-biased reproduction provides a similar tag amplification. This, in turn, makes it possible for agents to make useful distinctions. For example, an agent developing a condition that identifies tags associated with 'cooperators' will prosper from the increased payoff that results. As in the selective mating experiments, there is strong selection for combinations of tags and conditions that favor profitable interactions. In effect, the agents develop tacit models, anticipating the effects of interacting with agents having certain kinds of tags.

Agents without tags, under random pairing, execute plays of the PD with random unidentified opponents because there is no basis for implementing conditional intereactions In this evolving population, the productive "tit-for-tat" strategy never evolves; interactions settle on the minimax defect-defect mode, which is clearly disadvantageous relative to cooperate-cooperate interactions. On the other hand, agents with tags evolve along an entirely different path. At some point,as the strategies evolve, an agent appears that (1) employs tit-for-tat and (2) has a conditional interaction rule based on a tag carried by a sub-population that is susceptible to tit-for-tat. That is, the agent restricts its interactions to agents having strategies that (often) produce a cooperate-cooperate result under tit-for-tat. The resulting higher reproduction rate, causes both this agent and its cooperating partners to spread through the population. Subsequent recombinations provide agents that play tit-for-tat *and* restrict their interactions to other agents playing tit-for-tat. Once established such a sub-population is very resistant to invasion by other strategies. In biological terms, these agents with their conditional tag-mediated interaction have found an Evolutionarily Stable Strategy (ESS, see Maynard Smith [1978]).

Rick Riolo [1992] has carried out experiments comparing the evolution of adaptive agents with tags (displayed messages) against the evolution of adaptive agents without tags. They confirm the expectation that tags provide an advantage.

Even in the limited confines of the population-based PD, the evolutionary opportunities for adaptive agents with tags go considerably beyond the ESS just discussed. For example, mimicry becomes possible. An agent can present the tag associated with tit-for-tat, while pursuing a different strategy. Thus, the presence of an agent with a tag that has a well-defined functional meaning -- the tag "means" tit-for-tat in this case -- opens new niches for other agents. It is interesting that these niches are usually constrained in size, depending as they do on the continued presence of the "founding" agent. In the case of mimicry, as biological studies suggest, the mimic can only occupy a small proportion of the overall population relative to the agents being mimicked. This is so because other agents begin to adjust to the deception as the proportion of mimics increases. This negative feedback sets a limit on the mimic's expansion. It is typical that tags provide niches of limited "carrying capacity", leading to highly diverse systems with no "super-individual" that outcompetes all comers.

5. A Broader Perspective.

Even a cursory look at natural agents uncovers many examples wherein tags encourage diversity

and complexity. The studies of chemotaxis in the slime mold initiated by Bonner [1947], (for a review of recent work see Kessin and Compagne [1992]) and the studies of cell adhesion molecules initiated by Edelman [1988] (see also Sharon and Lis [1993]) provide two sets of examples of chemical messages and receptors interacting to provide sophisticated organizations. Selective mating based on phenotypic characteristics (Hamilton 1964]), and complex interactions in social insects mediated by pheromones (Hölldobler and Wilson [1990]) provide examples of tag-mediated interactions at a more macroscopic level. If these examples are typical, and I think they are, tags play a critical role in the phylogeny of complex adaptive systems.

To go a step further, I think that tags play a central role in the genesis of the fundamental characteristic of complex adaptive systems: a diverse array of agents that interact in an integrated way. Even a small ecosystem can involve hundreds of thousands of distinct species, a mammalian immune system involves like numbers of antibodies, and a municipality involves thousands of distinct kinds of aggregate social interactions. The diversity is not just a randomly derived conglomeration; rather it is a complex web of interdependent interactions, where the persistence of any agent depends directly on the context provided by the other agents. In the population-based PD, tags made diversity possible by breaking the unexploitable symmetry of random pairing. This observation can be elevated, I think, to a general principle governing complex adaptive systems: Tags break symmetries, providing opportunities for diversity.

The most general point I would make is that, wherever innovation plays an important role, it is important to redirect attention from optimality and equilibrium to improvement and preparation for change. This redirection offers insights even in the simply defined, completely deterministic strategic situations typified by games like chess and Go; it offers still more in the complex, stochastic situations prevailing in modern economies. In chess or Go, it is unlikely that we will ever be able to spell out an optimum strategy (say minimax); instead we search for building-blocks that can be put together to describe board configurations (pawn formation, control of center, tempo, etc.). Then we try to find improvements in (incomplete) strategic rules that exploit these descriptions. (The history of chess strategies since the middle ages nicely illustrates the point). The same point holds a fortiori in the much more complex strategic situations that prevail in modern economies. Provision for sustained exploration in innovative systems is much like making provisions for depreciation: It slows down current exploitation in order to prevent future catastrophes. As in evolutionary biology, rewards go to those that manage to locate improvements.

It will be interesting to see if these points can be confirmed in broader contexts. My own attempt in this direction has been to construct a class of models, the Echo models, that constitute a rigorous cartoon drawn from a broad range of systems that show innovation and learning (see chapter 10 of Holland [1992]). The focus is on complex adaptive systems such as economies, ecosystems, immune systems, and the like. An Echo model employs simplified counterparts of mechanisms common to these systems, such a trade, competition, and interactions conditioned on identification. Under particular settings of the parameters, subsystems in Echo specialize to interactions exhibited by well-studied canonical models, including Wicksell's Triangle, Overlapping Generations models, Prisoner's Dilemma games, Two-armed Bandits, and biological arms races. The object is to translate Bohr's correspondence principle to these nascent studies of complex adaptive systems. (The studies of Marimon et al. [1990] provide an excellent example of this kind of correspondence). Then the evolutionary behaviors observed in Echo can be bridged to well-studied rigorous models. Though the work is still in its early stages, Echo does provide a general framework for thought experiments on innovations generated by the interactions of adaptive agents of limited rationality. With good fortune, the correspondences to canonical models will enable us to start on a useful analytic theory.

References.

Axelrod, R. [1987]. "The evolution of strategies in the iterated Prisoner's Dilemma," *Genetic*

Algorithms and Simulated Annealing. ed. L. D. Davis. 32-41. Morgan Kaufmann.
Belew. R.K. and Booker. L.B.. eds. [1991]. *Proceedings of the Fourth International Conference on Genetic Algorithms* . Morgan Kaufmann.
Bonner. J.T. [1947]. "Evidence for the formation of aggregates by chemotaxis in the development of the slime mold *Dictyostelium discoideum* ." *J. Experimental Zoology 106* . 1-26.
Edelman. G.M. [1988]. *Topobiology* . Basic Books.
Hamilton. W.D. [1964]. "The genetical evolution of social behaviour." *J. of Theoretical Biology 7* , 1-52.
Holland. J.H.. Holyoak. K.J.. Nisbett. R.E.. Thagard. P.R. [1989]. *Induction: Processes of Inference. Learning and Discovery* , Paperback Edition. MIT Press.
Holland. J.H. [1992]. *Adaptation in Natural and Artificial Systems* , 2nd Edition. MIT Press.
Hölldobler. B.. and Wilson. E. O. [1990]. *The Ants* . Harvard University Press.
Kessin. R.H.. and Van Lookeren Campagne. M.M. [1992]. "The development of social amoeba." *American Scientist 80* , 6. 556-565.
Marimon. R.. McGratten. E.. and Sargent. T.J. (1990). "Money as a medium of exchange in an economy with artificially intelligent agents," *Journal of Economic Dynamics and Control 14* , 329-373.
Maynard Smith. J. [1978]. *The Evolution of Sex* . Cambridge University Press.
Mitchell. M. (1993). Personal communication: Mitchell is Resident Director of the Adaptive Computation program at the Santa Fe Institute.
Perry. Z.A. [1984]. *Experimental Study of Speciation in Ecological Niche Theory Using Genetic Algorithms* . University of Michigan Ph. D. Dissertation.
Sharon. N.. and Lis. H. [1993]. "Carbohydrates in cell recognition." *Scientific American 268* . 1. 82-89.
Riolo. R. [1992]. Personal communication: Dr. Riolo is a post-doctoral researcher at the University of Michigan

---

The Speedup Provided by Implicit Parallelism.

The following argument provides some insight into the manner in which implicit parallelism speeds up the search for a particular combination of building-blocks. To simplify, let us assume that the distribution that controls the search of C is uniform random so that all strings are equally likely. (The argument could be developed for a non-uniform distribution but the point remains the same.) Further. for the sake of this example. let us assume that all schemata of interest. $\{s_1, s_2, \ldots, s_m\}$, have the same number of defining loci. d. and that no two schemata have defining loci in common (they are 'non-overlapping'). Under this provision. the probability that a string c randomly chosen from C is an instance of a particular schema s_i is just 2^{-d}; that is. we can associate with s_i a random variable X_i which yields a 'success' with probability $p = 2^{-d}$. In these terms, a single string c drawn from C amounts to a concurrent sample of each of the random variables $\{X_1, X_2, \ldots X_m\}$ corresponding to the non-overlapping schemata $\{s_1, s_2, \ldots, s_m\}$. Formally, we can capture this concurrency by defining a random variable X that is a cross-product of the sample spaces of the random variables $\{X_1, X_2, \ldots X_m\}$.

With these preliminaries we can idealize the action of the genetic algorithm as follows. First note that once an instance of an above-average schema s_i is located, it will rapidly come to occupy a large proportion of the set R(t) because of the exponential increase in the number of its instances. As a consequence. it is very likely that crossover will quickly combine instances of s_i with instances of other above-average schemata already discovered. The result is a string that is both an instance of s_i and an

instance of other already discovered above-average schemata -- the string 'carries' several building-blocks. If that string is also above-average (e.g., if each of the building-blocks makes independent contributions to the utility of the string, or if the building-blocks act synergistically), then that string will in turn come to occupy a large proportion of the set R(t), and the process will be iterated.

Let the set $\{s_1, s_2, \ldots, s_m\}$ be a set of schemata that work together to increase a string's utility, and assume that the exponential 'rise time' of any schema (the time it takes for its instances to occupy a large proportion of R(t)) is small relative to the time it takes to discover that schema. Then the idealization of the genetic algorithm is closely approximated by the following formal model: Define the event E_j as the occurrence of a success in a sequence of trials of X_j, and let W_j be the expected number of trials until the first occurrence of E_j. Then, for a sequence of trials of X, define the event E as occurring when the event E_j has occurred at least once for each of the component X_j, and let W be the expected number of trials until the first occurrence of E. It is the way in which W increases as the number of schemata, m, increases that is of interest: implicit parallelism implies that W is a less-than-linear function of m

For a concrete example of E, think of a trial of X as the simultaneous flipping of m coins. In repeated trials of X, we set aside (stop flipping) each coin as soon as a 'head" shows: the event E occurs for the first time when there are no coins remaining to be flipped. Each coin corresponds to the testing of the defining loci of one of the 'target' schemata $\{s_1, s_2, \ldots, s_m\}$, and a 'head' corresponds to the first occurrence of an instance of the schema. Our earlier assumption that discovery time is long relative to recombination time is used here to let us concentrate on the waiting times E_j without adding in the time required to get discovered schemata 'on the same string'. The waiting time W then gives us an approximation to the search time required to find a string that incorporates all of the building-blocks $\{s_1, s_2, \ldots, s_m\}$.

To derive W in terms of m and p, note first that the probability that the event E_j has occurred in the first t trials of variable X_j is just $1-q^t$, where $q=1-p$ (the form of this proof was suggested by Shevoroskin, see[Shevoroskin (1992)]). It follows that the probability that the event E has occurred sometime in the first t trials is $(1-q^t)^m$. This in turn implies that the event E first occurs exactly on trial t with probability

$$(1-q^t)^m - (1-q^{t-1})^m.$$

Thus,

$$W = \sum_t [(1-q^t)^m - (1-q^{t-1})^m]t .$$

Expanding the two expressions $(1-q^t)^m$ and $(1-q^{t-1})^m$ and subtracting the second sequence from the first, we get

$$W = \sum_t \left[\sum_{j=1}^{m} (-1)^j C(m,j) (q^{jt} - q^{j(t-1)})\right]t$$

$$= \sum_t [\sum_j (-1)^j C(m,j) q^{jt}(1 - q^{-j})]t .$$

where $C(m,j) = [m! / (m-j)!\, j!]$ is the combinatoric function "m choose j".

For $p \ll 1$ we have $(1 - q^{-j}) = (1 - (1-p)^{-j}) \approx (1 - (1+jp)) = -jp$, yielding

$$W \approx \sum_t [\sum_j (-1)^j C(m,j) q^{jt}(-jp)]t .$$

Rearranging gives

$$W \approx \sum_j (-1)^{j+1} (jp) C(m,j) [\sum_t t\, q^{jt}] .$$

But it is easily established that $\sum_t t\, q^{jt} = (1- q^{jt})^{-2} - 1 \approx (jp)^{-2} - 1 \approx (jp)^{-2}$, where the last two steps use the earlier approximation for $p \ll 1$, noting that $(jp)^{-2} \geq (mp)^{-2} \gg 1$ so that the -1 in $[(jp)^{-2} - 1]$ can

be ignored. Thus

$$W \approx \sum_{j=1}^{m} (-1)^{j+1} (jp) C(m,j) (jp)^{-2} = \sum_j (-1)^{j+1} C(m,j) (jp)^{-1}$$
$$= p^{-1} \sum_j (-1)^{j+1} C(m,j) (j)^{-1}$$
$$= p^{-1} \sum_j 1/j ,$$

where the last step follows from a familiar identity.

Finally, $p^{-1} = W_O$ by definition, and $\sum_j 1/j \rightarrow \ln(m)$ as $m \rightarrow$ infinity, so that for large m

$$W \approx W_O \ln(m) .$$

Thus, this model suggests that, when m is large, the waiting time to discover m schemata with a genetic algorithm is approximately ln(m) of the time to discover one schema, instead of increasing directly with m as one might expect. Implicit parallelism can greatly reduce search times if this 'coin-flipping' model reasonably approximates the action of a genetic algorithm, with a corresponding increment in the rate of innovation.