Manifesto for an Evolutionary Economics of Intelligence

Eric B. Baum

NEC Research Institute

Abstract. We address the problem of reinforcement learning in ultra-complex environments. Such environments will require a modular approach. The modules must solve subproblems, and must collaborate on solution of the overall problem. However a collection of rational agents will only collaborate if appropriate structure is imposed. We give a result, analagous to the First Theorem of Welfare Economics, that shows how to impose such structure. That is, we describe how to use economic principles to assign credit and ensure that a collection of rational (but possibly computationally limited) agents will collaborate on reinforcement learning. Conversely, we survey several catastrophic failure modes that can be expected in distributed learning systems, and empirically have occurred in biological evolution, real economies, and artificial intelligence programs, when such a structure is not enforced.

We conjecture that simulated economies can evolve to reinforcement learn in complex environments in feasible time scales, starting from a collection of agents which have little knowledge and hence are *not* rational. We support this with two implementations of learning models based on these principles. The first of these systems has empirically learned to solve Blocks World problems involving arbitrary numbers of blocks. The second has demonstrated meta-learning– it learns better ways of creating new agents, modifying its own learning algorithm to escape from local optima trapping competing approaches. We describe how economic models can naturally address problems at the meta-level, meta-learning and meta-computation, that are necessary for high intelligence; discuss the evolutionary origins and nature of biological intelligence; and compare, analyze, contrast, and report experiments on competing techniques including hillclimbing, genetic algorithms, genetic programming, and temporal difference learning.

1 Introduction

Consider a common event: going into a stationary store to buy a pencil. How did the pencil come to be in the store awaiting your arrival? There is no human on earth who knows how to make that pencil. There are lumberjacks skilled at cutting trees, chemists who know about the yellow paint, miners in Ceylon who know where graphite is to be found, smelters who know about the brass ring, and farmers in Indonesia who grow a special rape seed for oil that is processed into the eraser (Read 1958). These people have no common language or purpose. Adam Smith's "Invisible Hand" organizes widely distributed knowledge to perform a computation, the mass production and distribution of pencils, that would be called cognitive if a human was individually capable of it.

There has been for a while something of a consensus among computer scientists who think about what it would take to implement human-like intelligence, that such an intelligence should be viewed as a multi-agent system with vast numbers of modules specializing in different subtasks and interacting in complex ways (cf. Minsky's Society of Mind (Minsky 1986), Valiant's Rationality (Valiant 1995), Lenat's Eurisko (Lenat 1983), and many many others, (Valiant 1994, Newell 1990, Selfridge 1959, Maes 1990)... see (Franklin 1995) for a survey). This consensus is natural from a number of points of view. On the one hand, divide and conquer is the natural approach to complex problems in computer science. On the other hand, modern learning theory has taught us the importance of compressing data into an explanation whose size is small relative to the size of the data. Identifying product structure gives a lot of compactness. If as I walk, a module in my mind positioning my ankle is independent of the part contemplating my future plans, one has a far more compact representation than one would expect if they were confounded.

Incidentally, there is also a consensus among workers from a vast number of disciplines that the biological brain/mind is a multi-module system. Lesion studies (D'Amasio 1994), neuroanatomy (Luria 1973, Churchland and Sejnowkski 1988), psychophysical experiments like the Wason Selection Test (Cosimidies and Tooby 1992), brain imaging techniques (Toga and Mazziotta 1996), linguistic analysis (Lakoff and Johnson 1980), and theoretical arguments from evolutionary biology (Barkow, Cosimidies and Tooby 1992), all offer a picture of multiple modules with different specialities interacting to solve a range of cognitive tasks. We believe that this diverse evidence can give insight into the computational nature of intelligence. We will briefly survey in §8 selected evidence on the mind as arising from evolution in the form of a multi-module system, and give a speculative, intuitive picture of the nature of biological intelligence.

While the picture of a multi-module intelligence is widely accepted, the notion of an economic picture is not. As exemplified in the example of the opening paragraph, an economy organizes a collection of rational agents to work together. Real economies can serve as a metaphor, lending intuition to design of algorithms, but this is not our main point. We will argue that an economic picture at a more fundamental, non-metaphoric level is valuable for understanding complex multi-agent intelligent systems, and especially if they learn, for four reasons. First, if our complex system is to wind up intelligent, intelligence had better be an equilibrium of the method by which it is constructed, for example learning. In building a "mind", we must construct it of agents that are initially less than "rational". But as the agents evolve/learn/are better programmed, they become more rational. Economics largely discusses equilibria of interacting rational agents. We will give a prescription for organizing diverse rational agents so they collaborate on a common goal. We also hope to make clear that without some related prescription, some global understanding of assignment of credit, there is no reason to expect the system to work as a whole: as the agents become more rational, the system can work worse. In §2, we catalog a number of ways in which such a system, absent a correct economic picture, can be expected to fail.

Let us examine at a high level what it might mean to divide and conquer intelligence, to see if there are any general statements we might make about approaches to constructing a modular intelligence. As a concrete model of intelligence we will discuss reinforcement learning. We will suggest in §7 that reinforcement learning is a much more encompassing task, much closer to what we mean by intelligence in nature and much more useful as well (if we could acomplish it well), than say supervised learning, and potentially is no harder a problem to solve.

The framework of reinforcement learning is that we have a world that we can sense and where we can take actions, and the world makes "money" payoffs in response to a series of "correct" actions. We wish to provide some architecture and computer program that learns to extract money efficiently from the world. If our learning machine is to divide and conquer, it will be made of many learning machines, many agents. If we can set up the system so that each of these agents, individually reinforcement learning, i.e. individually sensing its environment, taking actions on its environment, and trying to learn to increase its money, all act in collaboration to increase the money earned by the system, then we may succeed in dividing and conquering intelligence.

We will describe in §3 a model of an economy of agents that does this: motivates rational agents to cooperate. Each agent does not need to know everything about the world, only to attend to its interactions with other agents and to some part of the world. If the agents are rational in the sense that (1) they seek to maximize their money and (2) they know what the monetary consequence to them of taking an action is, then it follows that (a) any change in the behavior of the system increases the money the system earns and (b) if any group of agents are able to figure out a way of collaborating to increase the money the system earns, they will do it. This is a model of what is meant by the informal concept "the invisible hand" in a reinforcement learning context.

These results are natural if we have a conserved money that agents can transfer, and if everything in the system is owned by some agent. Trivially, if money is conserved, an agent can not increase its money without either increasing the total money pay-in to the system (i.e. our overall goal) or getting the money from somewhere else within the system. If everything is owned by agents, then the money can not come from somewhere else unless the agent owning it agrees to the transaction. But if this agent is rational, it will not agree unless it also profits. Thus conservation of money, plus ownership of everything by some agent in the system, yields a system that does not make transitions to worse performance (assuming the agents are rational). If the agents are free to transfer money, and the system is appropriately set up, then agents that can discover ways of improving performance will be able to recruit all the partners they need, by paying them some of the increased total.

The potential catastophes that we discuss in §2 are equally natural. When we do not enforce such conservation laws and property rights, the agents, instead of collaborating, can eat each other, engage in mutual admiration societies where they do nothing useful but simply create money and pass it around, or otherwise rationally act so as to increase their own lot, but decrease the productivity of the system.

Note that since the agents are computationally limited, they may not know the best computations to perform or actions to take. Indeed the actions they take will change over time if they figure out better ways to earn money. Thus the rationality assumed here is less strong than often assumed in economics, but nonetheless is very strong from the point of constructing an intelligent system, where we must either craft the agents or have them learn. Accordingly we need as well to understand the behavior of such systems for less rational agents.

This brings us to the second reason why an economic picture is useful. We conjecture that evolutionary forces in such systems can lead toward having rational agents. Agents that behave irrationally: i.e. that do not act to maximize their money, can be exploited by agents that do, and thus go broke and are removed from the system. (It has been conjectured that similar forces are the main cause of rationality in real economies (Coase 1997).) We don't know how to prove convergence to a system of rational agents, but we will present in §4 and §5, two systems based on artificial economies that start from random agents which can reasonably be described as idiots and empirically have been seen to learn to solve difficult problems in the domain of Blocks World. Agents that accurately estimate monetary payments and act so as to gain money, survive, so the agents evolve to be more "rational". When the agents are rational, the system evolves to earn more money because new agents can then enter if and only if they improve total productivity.

A third reason for an evolutionary economic picture is that it leads to a picture of intelligence that we believe makes contact with many aspects of human intelligence, some of which do not emerge readily through other models. Humans are often able to learn new concepts from a single example. This requires the ability to apply learned knowledge in learning. The key to this meta-learning, in my view, is assignment of credit to the meta-agents. It is not enough to adjust one's hypothesis, one's learning apparatus must reflectively adjust. Meta-computation is a related problem. Whereas artificially intelligent programs often engage in huge searches, humans seem to expend more computation on deciding where to search. Intelligent creatures must decide what to compute next and when to stop computing and act. Key to doing this well is taking into account the cost of the meta-computation. This is natural within a distributed economic system where agents pay the cost of their computational resources. Both meta-computation and meta-learning require that one take seriously the reflective nature of intelligence: one bootstraps by using one's intelligence to become more intelligent. To do this in both cases assignment of credit at the meta-level is critical. Both of these problems seem very naturally handled within an economic picture that addresses structural credit assignment. Without such a picture, they both seem so insuperable that I suspect few of the readers of this book have spent much time considering them.

More generally, we review in §8 a number of aspects of natural intelligence. Natural intelligence was produced by biological evolution in a process that seems to have built module upon module, with new modules produced by tinkering with old ones. This picture is subjectively similar to the evolution of our systems Hayek1 and Hayek2.

Our goal here is reinforcement learning in the natural world, in spaces with huge numbers of states, with extremely complex behaviors. Such problems can not properly be addressed without some incredibly deep hierarchy of complex functions. The computational resources that might naively be needed to solve such problems greatly exceed the size of the universe. Methods of reinforcement learning that require time proportional to the number of states, e.g. (Kearns and Singh 1998, Dayan and Sejnowski 1994), are seemingly on the wrong tack. Indeed, we wish to solve problems with infinite numbers of states, as we do in producing an algorithm for Blocks World that solves problems with arbitrarily many blocks.

The fourth reason for the kind of evolutionary economic picture we present is the hope that, just plausibly, it may lead to efficient enough procedures to address such huge spaces. If we can really succeed in modularizing and ferretting out product structure, in converging to assign credit accurately, perhaps we can break up these problems, make consistent progress, and solve them given merely vast, but plausible computational resources. Whether this happens will largely be determined by the local maxima structure at the meta-level, which can not be known without experimentation. Our examples Hayek1 and Hayek2 illustrate how complex problems can autonomously be broken into components. More recent work on a variant of Hayek2, described briefly in §7, illustrates how one can autonomously extract modular domain knowledge useful for search and planning.

At the very least, the proposal of representing a potential solution as agents creating agents creating solvers, focusing on credit assignment and finding a way to correctly motivate agents at all levels, is an insufficiently tested approach. We compare our approach to other approaches to learning programs from the point of view of computational efficiency: hillclimbing, and genetic programming, in §6; Temporal difference(TD) learning in §4.5 and Classifier Systems in §4.4. Hillclimbing addresses assignment of credit, but not in a modular way, is not reflective, i.e. does not operate at the meta-level, and has a number of other drawbacks. We discuss how our economic models address each of these problems. Genetic Algorithms (GA's) and Genetic programming(GP) seek in crossover an alternative to explicitly addressing assignment of credit. We discuss why GA's are unsuccessful, but in contrast to a number of recent results in the literature, we found empirically (Baum and Durdanovic 1998) that GP's crossover can be powerful. This result may be of some independent interest to our discussion of economics. The critical problem in TD learning is achieving generalization. Economics offers a modular approach to generalization, and handles credit assignment at the meta-level. Classifier systems are the first economic model we are aware of, and inspired us, but their economy is flawed in ways that allow some of the catastrophes we discuss in §2. We hope these comparisons illuminate how economic approaches can, just possibly, lead toward addressing very complex reinforcement learning problems.

There is a growing and interesting literature utilizing economic concepts in artificial intelligence, cf. a recent special issue of the A.I. Journal (Boutilier, Shoam and Wellman 1997). Clearwater (1996), e.g., collects a number of papers discussing use of equilibrium economic models to allocate resources in a distributed environment. Most of this literature, however, deals with programming environments rather than learning machines or convergence to intelligence, and so is not directly relevant to this paper. The most relevant previous papers are those of Miller and Drexler(Miller and Drexler 1988a), (Miller and Drexler 1988b),(Drexler and Miller 1988) who proposed "Agoric systems", an economic framework for agent interaction. They pointed out the importance of a conserved money and property rights, and described some of the examples of catastrophes that occur when these are violated. Their work is surveyed (and cited) at various points in the present paper. In these reports they did not explicitly discuss the problem of convergence to intelligence, nor attempt to implement a learning machine (being more interested in a programming environment) and so did not come to grips with the issues that arise in bootstrapping from irrational agents.

1.1 Plan of Paper

The plan of this paper is as follows. §2 gives examples of the catastrophes that can occur if conservation laws and property rights are not enforced. §3 elaborates on how our strategy applies "the invisible hand" to reinforcement learning. §4 describes Hayek1, a concrete model of an economy of simple agents, and reviews briefly some empirical results. §5 describes Hayek2, an economy of more complex agents capable of action at the meta-level. §6 compares our approach to other code evolution methods: hillclimbing, genetic algorithms, and genetic programming. §7 discusses reinforcement learning, and motivates our interest in it as a model of intelligence. §8 reviews some selected literature on the evolution of biological intelligence, and psychophysical, linguistic, anecdotal,anatomical, and other evidence as to its nature and speculates on its relationship to our framework. §9 concludes, discussing a number of open questions and directions for research.

2 Conservation Laws and Effects of Their Violation

Economics deals with theorems about stable points. It had better be true in a multi-agent system that the credit assignment is such that "intelligence" is a stable attractor in some sense, or your system has no hope of becoming intelligent. Setting up a framework where a collection of rational agents are motivated to collaborate is by no means trivial, as shown by examples that we describe below where it is violated. If you don't have such a framework, there is not much reason to expect your system will perform well. As your modules individually seek to maximize, they compete and may harm performance.

Our strategy to get the right sort of stable points is by maintaining global economic properties such as a locally conserved money, fungible money, and an apportionment of property rights such that everything is owned. We give examples of systems that do this in Sects. 3, 4, and 5. If money is conserved, an agent can increase his money only by earning money in the world, or by getting money elsewhere within the system. If agents own everything, then there is no elsewhere, and money can only come from another agent. But if we enforce property rights that prevent theft, a rational agent will only pay out money when it also profits. Thus agents are compelled to try to earn money in the world. Relax any of these conditions, though, and agents can more easily take money from each other, or manufacture money without actually solving problems in the world. In this section we give some examples of problems that arise when you do not enforce conservation of money, property rights, and ownership of all.

One problem that arises from nonconservation of money is positive feedback loops, where the system, instead of learning to improve total performance, chases its tail. We will mention three examples here: Biological evolution, Hayek1, and Lenat's Eurisko.

Lenat built an ambitious reasoning, learning, and meta-learning program called Eurisko (Lenat 1983). This work deserves study. Among other accomplishments, it won the world championship in a complex military game two years running, using computers available in 1983. (These and other feats were not from *tabula rasa* but rather the result of a man-machine collaboration in which Lenat interacted with his learning machine. Although our systems Hayek1 and Hayek2 mostly explore learning from near *tabula rasa*, we believe that economic ideas can be valuable as well in such partially programmed systems, cf. §9.) Eurisko used meta-heuristics to assign credit, called "interestingness", to its heuristics, guiding the reasoning and learning. Conservation of interestingness was not enforced. As a result (Miller and Drexler 1988a), the system developed positive feedback loops in which meta-heuristics found ways to assign themselves infinite interestingness, distorting the reasoning and learning, and requiring intervention.

Hayek1 is a learning system based on an artificial economy we will describe in §4. In Hayek1, if agents enter the system with money, the agents have an alternate way of earning money rather than extracting it from the world, namely fleecing novice agents. The system then displays speculative bubbles, and doesn't learn to extract money from the world effectively. Learning about the real world is hard. My experience in playing with these systems is, whenever you leave them any loophole to gain money without actually solving problems in the world, they will exploit it to the detriment of learning about the real world. (This has happened in numerous ways due to conceptual or programming bugs as we experimented with various systems.) I believe the entry of agents with money was a problem as well for Holland's classifier system (Holland 1986), a learning machine based on an economic analog that directly stimulated my thinking on Hayek1. Classifier systems are discussed in §4.4.

Biological evolution, as we will discuss more in §8, is a reinforcement learning process with, however, a funny kind of objective function: number of descendants. This is not conserved, and leads to serious positive feedback effects through sexual selection. If some gene causes females carrying it to prefer males with long tail feathers, then males with long tail feathers have an advantage. But moreover, females with this gene now have an advantage because their children are more likely to have long tail feathers, and thus be better able to attract mates. Thus the gene for prefering long tail feathers will spread. This circular logic has nothing to do with interacting with the physical world, per se, but rather with interacting with other members of one's community. It can lead to peacocks with tails so large they can barely fly and deer with antlers so large they can barely hold their heads up. These kind of phenomena absorb a vast amount of evolutionary effort, and intuitively don't seem useful to a reinforcement learning machine attempting to optimize some complex, externally-supplied objective function, like when we are trying to train a machine to play backgammon or Go or schedule elevators or stack blocks. By contrast, we are arguing that with conservation of money, interactions can promote collaboration of a community so that the community as a whole tends to optimize a complex, externally-supplied objective function.

The Tragedy of the Commons (Hardin 1968, Lloyd 1833) is a second problem that occurs ubiquitously in economies (and other adaptive systems) when not prevented by "property rights". It gets its name from common grazing grounds in England in the 17th century, where (so the story goes[1]) everybody was permitted to graze their cattle. Since everybody collectively bore the cost of damage to the grazing lands, but each person individually reaped the profits of his cattle, everybody's incentive was to overgraze (cf.

[1] According to (Ridley 1996), this story is apocryphal. On the commons in England people had a specific right to graze a certain number of cows.

(Varian 1996, pp571-574) for an elementary calculation of the amount of overgrazing to be expected.) This phenomenon is ubiquitous in economies wherever it is not prevented by property rights or some equivalent enforced custom or regulation. Air pollution can be viewed as stemming from our holding emmision rights in common. Traffic jams can be viewed as arising from our holding roads in common. Special interest politics can be viewed as arising from our holding government income in common. etc. etc.

Comparing an ecology to an economy, the lack of rules – property rights – stands out most glaringly (Miller and Drexler 1988a). Creatures do not even have secure property rights in their own protoplasm: other creatures regard it as lunch. This leads evolution to invest the bulk of its efforts in arms races. Creatures invest heavily in armor and teeth, fast legs, immune systems. Plants build poisons and animals respond with complex livers to digest them. A fantastic amount of evolutionary effort goes into designing these things, and much of the energy and biomass of the creatures is devoted to them. But it is not clear these efforts lead at all to progress, or especially that we would wish them as features in a learning machine. They are deemed "Red Queen" races, after the statement of the Red Queen in Alice in Wonderland who famously opined you have to run as fast as you can just to stay in the same place. Evolution is running in circles. The plants develop toxins, so the animals develop complex livers. In the end, they both have overhead and neither is obviously better off. Note that these interactions are in no way conservative: when A eats B, typically B loses much more than A gains.

Even in the presence of conservation of money, and agents having rights to themselves, Tragedy of the Commons can occur if not everything is owned. An example in the real economy is air pollution. An example in biological evolution is tree trunks. The vast majority of the biomass in a forest is in trunks, the sole purpose of which is to put a tree's leaves higher than the other trees'. The culprit here is that the sunlight has no owner. If the sun had a rational owner, with the right to dispose of the sunlight as he saw fit, building a trunk would be irrelevant. The owner would want to be paid. He would auction off the sunlight to the highest bidder. Money and sunlight would trade hands, but there'd be no wasted investment in trunks. From the point of view of efficiently extracting wealth from the world, grass is a better solution.

Holland's classifier systems are, I believe, plagued by Tragedy of the Commons. In classifier systems several agents are active at the same time. When reward comes in from the world, they split it. This motivates agents to be active when reward is due, independent of whether their action harms total reward in the long term.

Object oriented programming adopts property rights in the form of protected memory and variable names for different objects. Without strong notions of property rights there is little hope of constructing massive complex programs, much less intelligent systems. Our approach will be to have everything owned, and enforce conservation of money. Then we can avoid these pitfalls.

Holland (1975) and other workers (Anderson, Arrow and Pines 1988) have proposed studying "complex adapative systems", looking for laws that might help understand commonality among mind, economies, ecosystems, immune systems, and other large complex systems that adapt. The work reported here builds on the seminal work of Holland, but we focus to a greater degree on differences between the various types of "complex adapative systems". A potential difference between economics and mind is that the agents in economies are highly rational (indeed typically assumed to be perfectly rational). A critical difference between economies and ecologies, pointed out by Miller and Drexler (1988b) is that economies, and particularly a "perfect computational economy" that we may try to construct within our computer, are governed by rules enforcing conserved money and property rights. For this reason economies may be a far better model if we wish to construct algorithms than biological evolution, often cited as inspiration for genetic algorithms. We should be a little circumspect, however, about concluding that positive feedback loops, etc., are bad for producing intelligence. Some speculate, for example, that a sexual-selection based, positive-feedback loop caused or helped the evolution of human speech skills (Baum 1996a, Ridley 1993). Women preferring men who could communicate better, may have driven rapid evolution of language skills just as birds preferring longer tails produced peacocks. We compare our artificial economies, genetic algorithms, and genetic programming at some length in §5.

3 Rational Agents and the Invisible Hand

In this section we will present a simple model of the invisible hand in multi-agent systems performing reinforcement learning. We show here that if one has rational (but computationally limited) agents, a framework can be given so that the system performs reinforcement learning well. The model is similar to the model of Hayek1 discussed in the next section, except that here we assume agents that are rational, and thus derive analytic results. The Hayek1 model is more general in that the model begins with agents that are not rational in any sense, but the results regarding it are empirical.

The First Theorem of Welfare Economics[2] is regarded as a formalization of "the Invisible Hand". The First Theorem of Welfare Economics says that all market equilibria are Pareto efficient. Pareto efficient means that no agent can be made better off without making another agent worse off. The basic intuition behind the First theorem is that if there is any trade that makes everybody better off, the agents will trade to do so. Equilibrium means there is no further such trade, thus it follows that all equilibria are Pareto optimal.

[2] For a discussion of the First Theorem, Pareto efficiency, and all that, cf (Varian 1996).

We offer a version of this that applies to reinforcement learning, and has as criteria not Pareto optimality (which just talks about how well the individual agents are doing) but total reward to the system (in our case, the sum of how the agents are doing). Our model explicitly includes transitions that make some agents worse off: namely agents no longer involved in the solution, analagous to firms in an economy that are out-competed and go broke.

We assume for simplicity that the world presents a series of separate instances to the economy, each of finite temporal duration, as is fairly common in reinforcement learning experiments, as exemplified in the Blocks World problems we describe in §4.1 and experiment on in §4 and §5. In each instance the agents in the economy can take actions and potentially earn money, but we assume actions in one instance don't affect another instance. We make this simplifying assumption to avoid talking about present values. If we have a series of separate, finite instances, we can discuss agents that maximize their money at the end of the instance. We could presumably discuss a more complex situation by invoking a discount rate and discussing present values or in some other means. Note that the individual instances can be arbitrarily complex. Within each instance, there may be many different achievable configurations on which the world will pay money, and even more paths through configuration space achieving some or many of these rewards. Breaking it up into instances, however, lets us talk about agents that adopt one behavior in a given instance, then discover a better approach and adopt another behavior in the next instance.

We now posit the following "constitutional rules" under which the agents interact, similar to those in Hayek1 (discussed in §4). In each instance there is a token denoting the right to take action on the world, and to collect payout from the world. In each instance, one agent initially owns this token[3]. The agent owning this token will be allowed to take any action on the world it likes, and receive any reward paid by the world, but no agent not possessing the ownership can take action on the world or collect reward from the world (although the other agents can compute, communicate, pass money, etc. as they like). Possession of the token also implies the right to sell it to any other agent.

We assume the agents are "rational" in the sense that they will accept an offer that increases their money and won't accept one that doesn't. Note we are not asserting that the agents know the payout for all sequences of actions, or know what the best sequences of actions are, only their pay for sequences in which they engage.

Now we may be in some situation where some collection of agents are collaborating to solve instances, by passing around information, or the token, and money. Say, however, that some agent or collection of agents now proposes a way to earn more money on the next instance. They will always be able to buy the right, for more than the agent currently holding the token

[3] It may be a special agent whose only role is to sell the token to the highest bidder.

was getting. Some agents may be involved in both the new partnership and the old one. These can always be compensated to as to make them better off, and thus amenable to the change.

The simple claim here is that if :

1. there is a token controlling money flow from the world, and access to the world, and
2. agents can not obtain money from other agents unless the other agent agrees to the transaction, and
3. the agents agree to transactions if and only if they make them better off,

then the solution of (maximal total reward + some money transfer) is always preferable to the token holder and a set of agents capable of achieving the maximal total reward in collaboration.

Say the old solution is generating total reward A, and there is some division of this reward among the agents. Say the new proposed solution generates total reward $B > A$. For any division of profits in the old solution, there is a division of the profits such that all the participants in the new solution are better off, and such that the token holder is a participant in the new solution. Note that some agents may have been involved in the old solution but not the new one, and these may be less happy, but are unable to affect the result.

The total reward earned does not depend at all on who initially owns the token. It is always possible to pay the token holder more in solution B. The situation is closely analagous to Coase's Theorem (Coase 1960). Coase's theorem (cited in Coase's nobel award) roughly speaking is the (intuitively suprising) result that it doesn't matter from the point of achieving the outcome maximizing total utility, whether a polluter owns the right to pollute, or the people downwind own the right not to suffer pollution, so long as the rights are saleable. In either case the optimal result will be achieved, possibly accompanied by a payment. From my point of view, Coase's Theorem can be viewed as a statement about games and money. Consider a game with any payoff matrix with positive entries[4], e.g. a prisoners' dilemma matrix. If the players are allowed to pass money, the solution of (maximal total reward + some money transfer) is always preferable to everybody than any other solution; and if you have a law saying: one player can demand the state best for him, e.g. player 1 is allowed to defect and player 2 is not, then there is a well defined outcome (namely cooperation with a money transfer from 2 to 1, although the precise value of that transfer is negotiable within a range). Likewise in the reinforcement learning model we are discussing, whichever agent initially owns the token can be adequately compensated by whichever collection of agents can discover the way of extracting the most reward from the world.

[4] Note(Grove n.d.) that if you allow a payoff matrix with negative entries, this reasoning assumes that at the point of maximal total reward players have enough income to perform necessary transfers (which might be questionable e.g. in the case of residents paying the factory owner upwind to shut down his plant).

At first glance, this may seem so trivial that one is tempted to dismiss it, but the point is that it depends critically on three conditions, which are present in our model, approximated in the real economy, and yet seriously violated in many systems as discussed in §2. These three conditions are rationality of the agents (the agents accept better offers, not worse ones), ownership of everything/ ability to contract (represented here by the token) (else the result is not true– each of a million agents might for example be able to earn $1 to the detriment of total productivity if they act alone on the world), and conservation of money (else for example we could have positive feedback loops, unconnected to earning real reward from the world). As we have discussed, when any of these conditions are violated, there is no longer reason to believe the optimal strategy known to the agents will be followed.

The claim of this section was that, within the system we specified, if any agent (or collection of agents) can discover a better way of having the whole system earn money from the world, they can get everybody relevant to go along. This is a kind of universality result, saying that you can't do better than this system: if we knew a better way of solving, we could simply act as an agent, and get the system to implement our idea with appropriate payoffs. This is potentially a valuable check. Say we think we have a means of solving our problem. We could discard our system, and impose our idea, but if we are wrong, we would achieve a worse result. Instead, as a check, we can implement our idea within the system, making all participating agents (including of course the original token holder) better off. An economy of the type we are discussing can be used as a programming framework for coordinating agents that may be hand coded, as we discuss more in §9, and as discussed in (Miller and Drexler 1988b, Drexler and Miller 1988). From the point of view of creating an intelligence out of a collection of modules, it can not help to put some subset of the system in charge, so long as the modules are individually rational in looking out for their own interests, even though the modules may not be able to discover or understand the whole system.

4 Hayek1

In this section we will sketch a concrete model of a learning machine based on an artificial economy and describe empirical results. A more detailed account of this model and experiments can be found in (Baum 1996b). We first, in §4.1 describe the Blocks World domain where we have performed experiments, and next, in §4.2 describe our learning machine.

4.1 Blocks World

We report experiments on Blocks World. In our implementation of Blocks World, we see a series of instances. Each instance consists of 4 stacks of blocks,

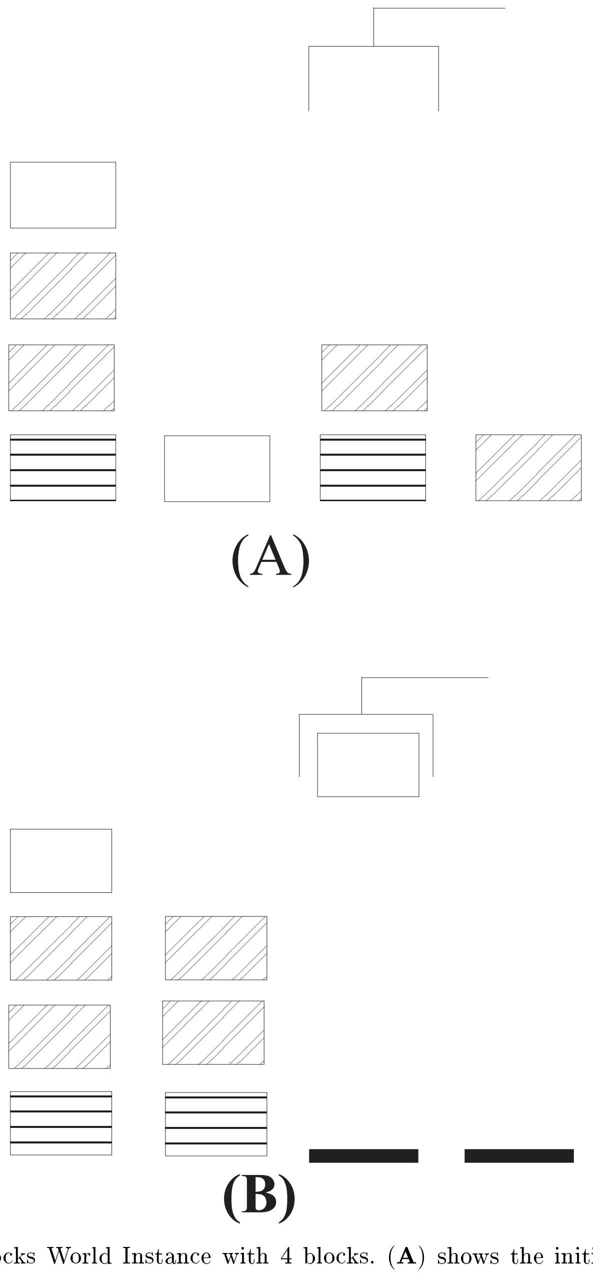

Fig. 1. A Blocks World Instance with 4 blocks. (**A**) shows the initial state. (**B**) shows the position just before solution. When the hand drops the white block on stack 1, the instance will be solved and Hayek will see another random instance.

each colored one of three colors. See figure 1. Our learner controls a hand that if empty may pick up the top block on any stack but the leftmost (aka the zeroth or goal stack), and if holding a block may place it on any stack but the leftmost. If it succeeds in copying the zeroth stack in the first position it gets a reward and a new random instance. If it uses too many actions without copying the goal stack (typically in experiments the threshold was 100 actions) it gets no reward but sees a new instance. Typically in training we present larger instances as our system learns to solve the instances it sees, since our learners receive no useful feedback until they solve problems, and are typically initiated randomly.

Blocks World has been a benchmark domain since the '60's (Winograd 1972) and remains widely studied today, cf. (Bacchus and Kabanza 1995, Estlin and Mooney 1996). It can be treated as a planning problem, or as here as a reinforcement learning problem where one must not only find a plan for solution, but discover the nature of the goal. Note that the number of states is exponential in the number of blocks, and the number of blocks is potentially unbounded. Moreover the goal is abstract and highly non-linear: having the "correct" second through ninth blocks on top of an incorrect first one is not progress. The limited table size creates "Tower of Hanoi"-like configurations that may require long sequences of actions to solve. There is a lot of structure to the problem: a compact algorithm describes an infinite class of instances, and another compact algorithm can solve them. We are interested in how we can construct algorithms that learn about such structure, and learn to exploit it.

Several general purpose AI planning algorithms were recently applied to related Blocks World problems, involving dual goal stacks[5] and a different representation. These programs do extensive, heuristic-guided search. They exceeded resource bounds of 400 CPU seconds for goal stacks totalling over 5 blocks, unless given hand-crafted knowledge specific to Blocks World (Bacchus and Kabanza 1995). With such domain knowledge, problems involving a few tens of blocks can be solved. Within the learning literature, no author I'm aware of has addressed the general goal discovery problem. Whitehead and Ballard (1991) applied a Temporal Difference algorithm to learn to pick up a green block, initially under at most three other blocks. The Schema Mechanism of Birk and Paul (1994) moved a single block around an otherwise empty table. Koza (1992) applied Genetic Programming to build the same fixed stack every instance on an unbounded table, given built in sensors for such pertinent information as the Next Needed Block, and both the top block and the top correct block on the target stack. Each of these learning papers addressed versions of the problem involving neither discovery of an abstract goal, nor the necessity of stacking blocks on top of later needed blocks.

[5] I believe that for a given number of goal blocks, dual stacks makes the problem easier since the goals are then shallower.

4.2 An economy of idiots

We have access to a world that we can sense and that will make payoffs in response to appropriate series of actions, in this case Blocks World. Our goal is to learn how to compute what actions will be appropriate. To do this we will maintain a collection of agents, and let them interact under a set of rules that assign credit in such a way as to motivate the agents to collaborate on solving our problems. Our goal is to give each agent a reward such that if the agent acts to maximize his reward, it is acting to maximize the reward of the system. We will not assume any 'rationality' for the agents. Rather, 'rationality' like that of the last section will emerge (approximately) from evolution of the system.

Computation will proceed in a series of auctions. In each auction, the agents will be permitted to bid. The high bidding agent buys the access to the world, and pays the previous owner of the world (i.e. the agent winning the previous auction) his bid. The agent now owning the world can take any actions it desires on the world, and receives any reward from the world while it owns it. Because the agent owns the world, it is "motivated" to increase its value (immediate reward plus resale value) exactly as we desired.

The winning bid converges to estimate the expected value of the world given that we use the winning agent. This happens because if an agent bids lower than this expected value, a new, similar agent can enter and profitably outbid it. If the agent bids higher, it will go broke and be removed. (Any agent with negative money is removed.) Assuming then that the agents' bids have converged to represent the expected value of using the agent in that situation, some nice things are true. First, using the agent with highest bid is then an eminently sensible choice (from our perspective as owner of the whole system). Furthermore, a new agent can outbid its competitors and enter the economy profitably if and only if it takes the world (on average) to states of higher value than its competitors' bids. Conversely an agent goes broke if it is taking the world (on average) to states of lower value than its bid. Entry of new agents and bankruptcy of old ones stochastically hill climbs in total rate of earning by the system.

A sequence of agents may own the world and take actions before there is any reward from the world. Agents early in this chain are compensated if they sell the world for more than they bought it.

We have experimented with such a system for Blocks World. We maintained a collection of agents each consisting of a Boolean condition, a numerical bid, and an action. At each auction, the Boolean condition was evaluated. If true, the agent posted its bid. The agent posting the highest bid won the auction, paid the bid to the previous owner of the world, and took its action, collecting any reward paid by the world in response, and subsequently being paid the high bid in the next auction The Boolean conditions were of form a conjunction of clauses, each clause asserting the equality or inequality of two locations in the world, specified in Cartesian coordinates, and including existentially quantified variables (written $*$). An example of such an agent is

$$[(1,0) \neq B] \cup [(0,0) = (1,0)] \cup [(0,1) = (*,3)] \rightarrow \text{Bid} = 8; (g, *)$$

This agent will bid 8 in the event that position (1,0) is not Blank (written B), i.e. there is at least one block in stack 1; and the lowest position in stack 0 has the same color block as the lowest (zeroth) block in stack 1; and there is some stack $*$ with a block of the same color at height 3 as the block in stack 0 at height 1 (or else where both positions are blank). If it wins the auction, it will move to this stack $*$ and grab (written g).

New agents were injected into the system by a random process. Half the new agents had conditions drawn randomly from a roughly uniform distribution over possible conditions. The random conditions had i clauses in the conjunction with probability $1/2^i$ for $i \geq 1$. Each clause consisted of equality or inequality with equal probability, and compared block colors at randomly chosen locations. The only free parameter P_* determined the probability of existentially quantified variables, and we set this arbitrarily as well so a smallish constant fraction of the entries were existentially quantified variables. The actions were chosen randomly as well. The other half of new agents were generated by picking a random living agent from the population and making a small random mutation. We initiated with 20 random agents. Two new random agents were added to the system every 10 instances. Agents paid a small rent every 10 instances, which removed agents that never won auctions or earned at too small a rate.

New agents were assigned their bids as follows. The first auction in which a new agent's condition was true, its bid was set to be ϵ greater than the highest competing bid of a veteran agent (i.e. agent with bid already set). Thereafter it was fixed. In Hayek1 we arbitrarily set $\epsilon = .2$. Frequently new agents entered that were functionally equal to an existing one – say with a tautological clause added or deleted– and these had the effect of pushing the bid up to where it was just profitable.

New agents entered with zero money, and hence needed credit to bid initially. We ran a "perfect credit check"– if the agent did not earn enough money to pay its debt, we backed up the instance and reran without the defaulting agent present. This preserved conservation of money. If we allowed agents to default, we found speculative bubbles, with bids rising as agents did not care how much they bid, so long as they could hope for a new agent to enter and pay them money (which it didn't have). Such systems didn't learn to solve instances with more than a block or two on the goal.

This system implements an economy similar to that described in §3, but the agents are initially not in the least "rational". Instead, they are randomly generated. But agents survive only so long as their money is non-negative, i.e. only so long as they are behaving reasonably rationally.

We ran this system on Blocks World, presenting a series of random instances. We initially presented small problems and added a block to the problem size whenever Hayek1 was solving 98% of problems of a given size.

Given no features and no intermediate reward, Hayek1 succeeded in finding sets of up to about 200 agents that solved 98% of "stage 7" problems. "Stage 7" problems had about 4 blocks on the goal stack, and 9 total blocks. These sets of agents are succeeding in collaborating on sequences of moves up to 100 actions long, passing credit back to form a stable system.

Unfortunately the representational capacity of this language is not sufficient to write a general solver: there is no collection of agents you could write that would solve arbitrary Blocks World problems. So next we expanded the representational capability of the agents to allow them to represent a solution. We added one set of features: $Top[i]$. $Top[i]$ was set equal the height at which the next block would be placed on stack i. We also added intermediate reward, giving a Hayek agent that correctly placed a block on stack 1 (i.e. placed a block of correct next color, with all of the blocks below it correct) a reward of 1, rewarded removing an incorrect block from stack 1, and penalized adding an incorrect block or removing a correct block -1.

Note that intermediate reward expands the *representational power* of the Hayek system. A human trying to code a program by writing rule-bid agents of the form used in Hayek1 would naturally use the bid to prioritize the rules. Hayek can not readily do that. Using the bid to prioritize the rules means using a rule with higher bid before a rule with lower bid. But if agents with higher bids are typically followed by agents with lower bids, they will go broke. Such solutions are unstable. However, if we add intermediate reward, such solutions become stable provided that the higher bidding agent gains immediate reward sufficient to keep him profitable.

Hayek now generates (occasionally) a set of agents that collaborate to solve arbitrary Blocks World instances. One example is the following.

```
(1) Bid= 14.4166 (0,{*0})!=(1,{*0}) &
(1,{*0})!=HAND & EMPTY==EMPTY => 1,G

(2) Bid= 14.0712 (0,Top[1])==EMPTY => 3,D

(3) Bid= 13.3013 (1,{*0})!=(2,{*0}) => {*1},D

(4) Bid= 13.2446 (0,Top[1])==(3,{*0}) => 3,G

(5) Bid= 13.2242 (1,15)==({*0},{*1}) => 2,G

(6) Bid= 12.1369 (3,{*0})!=(2,4) => {*1},D
```

This program works as follows. Agent 1 is a universal clearer. Its condition is true if and only if there is a block on stack 1 not corresponding to one on stack 0. It then takes control of the next grab (asserting priority with its high bid) to remove the top block from stack 1. Thus Hayek1 digs on stack 1 until there are no bad blocks on it. Assuming this to be true, the next block we are searching for is a block of color $(0, Top[1])$. Agents 4 and 5 dig for a block of this color. Because of Agent 4's higher bid, it digs from stack 3 if this color block is found there, else 5 digs from stack 2. Thus Hayek always digs on a stack where a useful block may be found. As to dropping these blocks, Agent 2 only bids when stack 1 is higher than stack 0, and drops blocks on stack 3. Otherwise Agent 3 does the dropping, and it drops randomly. Any useful blocks picked up are thus dropped on the target with probability one third by Agent 2, and no correctly placed blocks are ever removed from the target. Hayek always digs on a correct stack– stack 1 if there are any bad blocks on it, otherwise on a stack with a correct next block. Thus Hayek makes steady progress and solves the instance.

There is a class of problems that occurs exponentially rarely, on which this solution can be shown to be quadratically slower than the optimal sequence of actions. Also, this system will actually fail on some small instances[6], that Hayek once knew how to solve, but which it has not been trained on recently and has now forgotten. Other than these exceptions, this set of rules will with high probability solve any instance within a small constant factor of the number of actions taken by the optimal algorithm.

It is unclear to what extent the intermediate reward aided the learning directly, and to what extent it aided by improving the representational power. Note that the set of agents is reasonably stable in spite of being bombarded by millions of new, randomly generated agents. Considerable wealth is being transfered back along the chain by the bids, and this is keeping the set of agents stable. The results without intermediate reward show that the economic organization is able to find collections of agents collaborating on long sequences of actions. My point of view is that we need to experiment with more powerful representations. Experiments along these lines will be discussed in §5. These experiments can not be conclusive whether intermediate reward was more important for guiding learning or increasing representational power in Hayek1 but give conflicting insights. On the one hand, starting from *tabula rasa* with potentially powerful representations, our experiments have not yet generated a set of agents capable of solving arbitrary Blocks World instances, although as will be seen we have had some success and our models continue to improve. On the other hand, once we have such flexible representations, intermediate reward does not seem to help learning much, and indeed for reasons we do not understand actually seems to hurt it sometimes.

[6] There are a few configurations for some instances of less than 12 blocks for which (2,4) is Blank, there are no blocks in stack 3, stack 1 is identical to stack 2, and stack 0 is as high or higher than stack 1, in which case the condition of all dropping rules is False. This configuration of agents was trained on collections of up to 50 blocks, and has not been trained on any instance of under 12 blocks in its last several million presentations.

4.3 Discussion

I view this experiment as something of a proof of concept. We set up a framework that used property rights and conserved money to motivate agents to collaborate. And, although the agents were far from "rational"– in fact they were randomly generated productions, the system evolved to behave rationally. The bids of agents converged toward the expected value of using the agent. Once this happened, new agents could enter (roughly speaking) if and only if they took the system from a state of lower value to one of higher value. (Potential counterexamples to this will be discussed in §4.6.) Agents that survived thus became increasingly "rational" and collections of agents learned to collaborate on solving Blocks World problems.

It is vastly easier to find solutions to Blocks World if you can find them in collaborating pieces. Hayek can, for example generate a rule

$$(0,0) \neq (1,0) \Rightarrow (1,g) \tag{1}$$

that sensibly grabs from stack 1 if the bottom block is incorrect. Hayek1 generates this rule *de novo* with probability roughly 3×10^{-5}. Note that rules like rule 1 are useful even before Hayek has learned much. If they occur in the context of random actions, they lead to solving a higher fraction of small instances. Hayek can then mutate rule 1 to

$$(0,*0) \neq (1,*0) \cap (1,*0) \neq e \Rightarrow (1,g) \tag{2}$$

which is a universal clearer that removes the top block in the target if any location in the target differs from its neighbor in the goal. Once rule 1 is in the population, one can imagine paths modifying it through potentially stable intermediaries that might achieve rule 2 after 10^{-4} or 10^{-5} creations. Creating this rule *de novo* is about a 10^{-8} event. Now, we can also create rules that place good blocks on the target stack. Plausible paths for these also involve only tens or hundreds of thousand of rule generations. Being able to create a program for solving Blocks World in pieces lowers the search space from combinatorially impossible to only hundreds of thousands or millions of trials, if the pieces will be stable as they enter, as is plausible, and empirically observed.

In fact, Hayek's structure makes the search relatively efficient. New rules, as they enter, are immediately judged using the knowledge the system has already as to the value of the state they reach. A new rule survives only if it takes the system from a state of low expectation (bid) to one where it gets paid more. Contrast this to the situtation of Brand X algorithm, where new rules are judged on end reward to the system when they are used. Consider the case where the Brand X system is solving, say, 95% of easy problems, and we are just getting ready to go to the next level. Say a new rule enters that simply wastes an action. Usually the system will recover from this and solve anyway. Say performance with this new rule degrades to 90%. Our new rule, rewarded on the brand X basis of end result, will still be rewarded ten times for every time it is penalized. Such rules will enter the system in droves, and prevent progress. This is another manifestation of the Tragedy of the Commons, when rules simply share in the final reward. From another point of view, this illustrates the observation of Sutton (1988) that it is often far better to perform credit assignment in learning by bootstrapping one's learned knowledge to judge the values of intermediate states than it is to use actual result.

4.4 Holland's Classifier Systems

This Hayek1 model was motivated in part by Holland's proposal of classifier systems (Holland 1986) also based on an economic metaphor involving condition, bid, action agents and a "bucket brigade" assignment of credit. I believe, however, that Hayek1's economic framework is better in several critical respects. First, in Holland's system agents bid proportionally[7] to their wealth. New agents, to bid, must thus enter with money. This violates conservation of money– and thus allows a simpler way to earn money than learning to earn it from the world. In my experiments this pumps money into the system, causing speculative bubbles and inflation. This problem was not as evident in Holland's Classifier systems because they proceeded in generations. At the beginning of each generation all rules are replaced simultaneously (with new rules bred using Genetic Algorithm crossover from rules in the previous population). If one then runs enough instances between generations that the initial money given to the rules is small compared to the money earned, the violation of conservation may perhaps be unimportant (although this is not evident, as any subset of the agents that are actually capable of solving instances together may instead be initially exploited by other agents, and never later get enough money to bid). But running in generations bears a big cost: the system can not evolve smoothly, so it is hard to imagine developing collaboration.

Second, in Holland's Classifier systems many rules act simultaneously. When pay-in comes in from the world, one is faced with a credit assignment problem. Holland's scheme was that the active rules share it. But this is a recipe for the "tragedy of the commons". All rules are motivated to be active

[7] Actually, agents' bids were proportional to their wealth times their "selectivity". Holland's original motivation for this was that "better" (i.e. wealthier and more specialized for the present situation) agents should be used preferentially. However, I view this instead as a recipe for tuning the bid to equal the expected value of using the agent. This occurs because as long as bid is lower than expected pay-in, wealth rises and with it bid, and if bid is higher than expected pay-in wealth falls. Selectivity is irrelevant, in the limit, except in determining the limit value of wealth. I have thus experimented in Hayek1 with setting bids proportional to wealth (but ignoring specificity) as an alternate means of determining the agents' bids, with somewhat worse results to those reported here.

when pay-in is coming in, even if their action lowers payoff, or future payoff to the system.

Holland's classifiers differ from Hayek1 in some non-economic aspects of potential note. They utilize a particular representation that omitted any *a priori* topological bias. This representation was thought to be powerful, i.e. Turing universal (Forrest 1985), but this claim seems to ignore stability issues. It appears to have not been previously realized till I pointed it out (and simultaneously Lettau and Uhlig did also (Lettau and Uhlig 1995)) but classifier systems are less powerful than naively appears because solutions that use the bids for prioritization can be unstable without an intermediate reward as discussed above. This spells trouble for some intuitions Holland had formulated regarding the development of "default hierarchies." Also Holland's classifiers used genetic algorithms to produce new classifiers. I am somewhat skeptical about the efficiency of GA's and will revisit this question in §6.2.

4.5 TD Learning, Q Learning, and All That

In Q-learning (Watkins 1989), one maintains a table assigning a value to every state-action pair. At each step one chooses an action probabilistically with preference for the one of highest value. One learns by updating the value estimates, improving the value estimate at time t using the value estimate at time $t + 1$. Q-learning is proven to converge for Markov systems (Dayan and Sejnowski 1994, Gurvits, Lin and Hanson 1994).

Hayek1 is isomorphic to Q-learning if one only allows agents with condition valid in exactly one state. It then stores exactly the same information (estimated value of a state-action pair), and learns in exactly the same way: by backup from end states. Hayek1's search is randomized by the introduction of new agents.

The interesting question, from my point of view, is how one reinforcement-learns when there are vastly too many states to maintain a state-based representation, and one must generalize. Hayek1 utilizes an agent-based generalization, where agents recognize a set of states according to some Boolean function. This approach has some potential advantages. First, we have an intuition that each update: death or addition of a new profitable agent, improves performance. This will be discussed more in the next section. Second, by keeping around those agents that are profitable, Hayek1 dynamically addresses the bias-variance tradeoff. Profitable agents, by definition, are well trained. Hayek1 can maintain more of them as it has the data to justify a more complex representation. Third, Hayek1 uses both evaluations and an estimate of their reliability in updating. This occurs because agents' bids are honored only to the extent they have (or earn) enough money to justify them. Fourth, and possibly most importantly, Hayek's scheme is a modular way to represent evaluation functions. In very complex environments, such modularity may be critical to learning. We will discuss below more complex generalizations, with more complex representations and more powerful agents, that maintain all of these advantages. When we get into agents able to create other agents, for example, we hope to extend the notions of Real Time Dynamic Programming and Temporal Difference Learning in new and hopefully fruitful directions.

What this manifesto is arguing for is serious study of what I regard as the critical question: how does one do reinforcement learning in vast spaces where one must generalize. One approach is to simply use a neural net, say, to map states into values. But in many contexts, a representation incorporating actions will be critical. A representation able to discover and utilize product structure, or approximate product structure, in the world will be critical. Dealing with quantification will be critical. Dealing with the meta-level, which I will argue is natural within an economic model, and which I know no way of doing within standard TD methods, seems critical.

4.6 Cherrypicking

In Hayek1, agents' bids converge to estimate the expected value of their pay-in, because otherwise new agents enter and outbid them. This expected value is averaged over the instances in which an agent wins the auction, and thus the agent's bid approximates, but only approximates, the value of any given instance. That this is an approximation is a consequence of our attempting to generalize. If the agents recognized only one state, estimates could converge to be exact, but we would reduce to the Q-learning situation discussed in the previous section, where we need at least as many agents as states.

If the agents' bids accurately estimated the expected value for each state, two desirable things would occur. First, in choosing the agent with highest bid, the system would be acting optimally. Second, a new agent would be able to enter if and only if it improved performance of the system. To enter, an agent would have to outbid its competitors, meaning it would have to take the system to a better state. But if it took the system to a better state, it could expect full payment from the next agent, accurately bidding the value of that state. This is an example of the situation discussed briefly in the introduction where the agents are "rational"– they know the value of the state they will reach. We will discuss it more below.

In the real Hayek1, however, because the agent's bid estimates the average over all instances where it applies, it may be inaccurate on any given state, and in systematic ways. This leads to the possibility of "cherrypicking" (analagous to a phenomenon of that name well known in the insurance industry). See figure 2. Say we have a profitable agent A, and now an agent B is introduced into the system whose domain includes the easily solved instances of A, but not the harder ones. So long as B's expected pay-in is higher than A's bid, B may profitably outbid A, even if B's action on these instances is worse than A's, leading to a lower final reward. Thus B's entry may worsen the performance of the system.

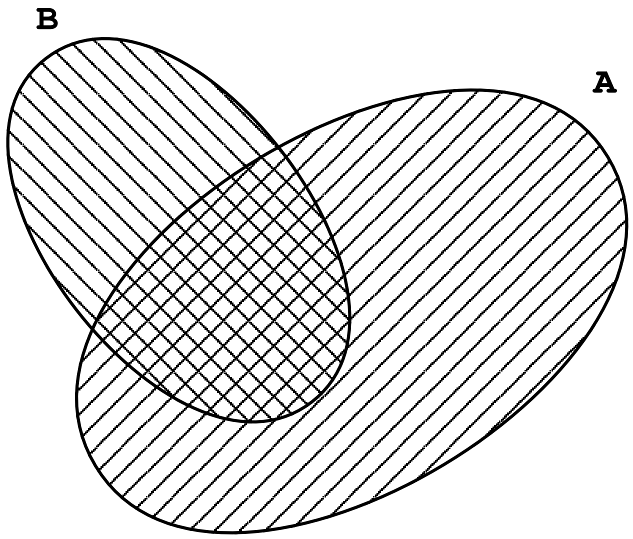

Fig. 2. A Ven diagram in instance space. Region A shows the instances where A won the auction. Agent B enters, now capturing the auction on instances in the intersection.

Moreover, if A can not adjust its bid, it will now go broke, since its bid was estimating average pay-in including the favorable instances. If A goes broke, some other agent C will have to deal with the instances it was handling. Since B is profitable, we know it at least achieves reward better on average than its bid, which was higher than A's bid. But we have no control whatever on how much worse C's performance may be than A's. On the other hand, C's performance may be better than A's. By definition, A was overvaluing these instances. Therefor it wasn't valuing them accurately, and wasn't earning as much money from them as it was elsewhere, perhaps because it was mishandling them.

These considerations have prevented me from proving any kind of convergence result. Nonetheless, cherrypicking does not seem to be great problem in practice for this system. In fact, in Baum (1996b) we describe two experiments adding hillclimbing attachments to Hayek that attempt to prevent cherrypicking. One of these harmed performance, and the other had little effect.

It seems plausible that cherrypicking is not a problem for two reasons. First, new entries tend on *average* to improve performance: the new agent is (by definition) profitable and at a higher bid than the old agent(s). If we assume there is no particular correlation between where the new agent wins the bid and where the old agents are underbidding, (and we assume old agents rather than going broke can lower their bid to the point of marginal profitability) then it would follow that new agents improve performance on average. In our Hayek1 experiments, the displaced agents lower their bid to the point of marginal profitability by going broke[8]: there are typically previous versions of an old agent, with lower bid, in the system. A version of the agent having a bid that is marginally profitable will survive, taking typically the same actions as the displaced agent. Thus the entry of B typically has the benefit of making the overall estimation of value of states more accurate.

Second, and perhaps more importantly, if the new agent actually does discover where a region is of high profitability, this is an important discovery. Such a region might typically tell Hayek a lot about the world. Hayek may then create a mutated version of the new agent that exploits the characteristics of the new region better than it was before. Hayek1 may create a mutated version of the old agent, acting on its remaining region, that takes a better action there. This kind of division of the states of the world in semantically sensible ways is exactly what we are trying to discover and exploit.

5 Hayek2

Hayek1 implemented an economy of very simple agents, able only to bid a single value, and perform a single fixed action. In this section we will consider an economy of more powerful agents, able to perform considerable computation, and able to create other agents as well. For a more detailed discussion of this work see Baum and Durdanovic (1998).

There are several motivations for this generalization. First, our experience with Hayek1 points out the importance of having a sufficiently powerful representation to be able to solve one's problem, and in particular without relying on prioritization using the bid to increase the power of the representation.

Second, Hayek1 generated new agents at random. Random generate and test is the standard method for generating new improvements in local search algorithms, but seems destined to become inefficient in complex spaces and as the solution found becomes more optimal. It seems more powerful to learn a better algorithm for suggesting new improvements, and if we believe in the power of economic approaches, it is natural to try to recurse our economic model on the problem of suggesting new agents. Potentially, of course, we could have an infinite number of such meta-levels. By incorporating agents that can create agents into our system, we hope to have a system that is its own meta-system. Since Gödel, it's been known that there are dangers inherent in having systems be their own meta-system, but we hope to avoid instability and inconsistency by insisting on conservation of money, as will be seen.

Third, we believe that this approach naturally deals with meta-level questions such as meta-learning and meta-computation. Humans clearly meta-learn: they learn to learn. Learning is typically seen to require huge numbers

[8] Another way to arrange this is by making bid proportional to wealth, as discussed in footnote 7 and in more detail in Baum (1996b).

of examples in both the empirical and theoretical learning literature; yet humans frequently learn new concepts from a single example. There is a literature on meta-learning, cf. (Thrun and Pratt 1997) for some recent papers, cf. (Russell and Norvig 1995) for a discussion of some approaches as well as a discussion of examples where humans learn concepts from a single example, but I don't think it is controversial to assert that the techniques understood to date do not seem sufficient to explain human abilities– some new ideas are needed. Presumably achieving human-level abilities to learn concepts from a single example requires recursively bringing to bear one's learned knowledge on the problem of learning new knowledge. This, I submit, requires addressing the assignment of credit problem at the meta-level. It is not enough to judge a hypothesis based on the data, one must also make changes in one's hypothesis generator, so one learns to generate correct hypotheses efficiently. Lenat, for example, attempted this in his Eurisko system by using meta-heuristics that assigned "interestingness" to heuristics. But this generated positive feedback loops in which meta-heuristics succeeded in assigning infinite interestingness to themselves (Miller and Drexler 1988b). Miller and Drexler (1988b) proposed that within economic models it is natural to try to compensate meta-agents as investors in the agents they create. By invoking conservation of money, one can prevent positive feedback loops. Ours is the first attempt we are aware of to implement this idea.

Horvitz (1988) and Russell and Wefald (1991) have cogently argued that understanding real time intelligence requires addressing meta-computation: deciding what to compute next and when to stop computing and act. Russel and Wefald proposed an explicit decision theoretic approach to meta-computation. But the cost of the meta-computation itself is a critical component of such decisions, and I am not aware of any algorithmic ideas (other than the economic ones I will discuss) that seem likely to efficiently estimate utility of computations, taking the cost of the meta-computation into account. Game playing is a good example of what I am talking about. Computer chess programs search tens of billions of positions in the game tree. Warren Smith and I (Baum and Smith 1997) developed a Bayesian approach that estimates the utility of expanding positions at the leaves of a search tree and iteratively expands approximately the most interesting leaf, stopping searching and moving when the utility remaining in the search tree is small compared to the cost of searching it. But our algorithm explicitly ignores the cost of the meta-computation, because we have no idea how to take it into account. Instead, we demanded as a constraint on our algorithm that meta-computation use only a small constant fraction of our thinking time. By doing this, we were able to get an algorithm that can expand within a constant fraction as many leaves of the search tree as alpha-beta, but along more utilitarian lines, and thus beat alpha-beta (at least in Othello, where we have experimented). But what humans do is radically different: instead of expanding billions of leaves, they (according to introspection) expand perhaps 100.

They use the vast majority of their thinking time on meta-computation, deciding which leaves to expand. To achieve such algorithms requires radically different techniques that account for the time spent on meta-computation. But this is naturally implemented in a Hayek-like economy. Make each agent pay rent for computational time it uses. Then the agents will learn to perform only computations that justify their cost. In other words: meta-computation is an inherently distributed task, because gathering the meta-information together is expensive. Divide and conquer the meta-computation at the same time and in the same way as you divide and conquer the computation.

We will compare our approach to local search and Genetic Programming approaches in §6.

5.1 Hayek2's Economy

Now there are two main parts to our program: choosing a constitution for our economy and choosing a language for our agents. Both have evolved as we have come to understand them better. I will survey only the (current) main variant of the constitution here (cf. (Baum and Durdanovic 1998) for more extensive discussion).

As in Hayek1, computation proceeds in a series of auctions. Before each auction all agents have opportunity to compute, to decide if they want to create an agent, or bid, or neither. The high bidding agent wins the auction, buys the world from the previous owner, can do more computation and take actions on the world, exactly in Hayek1 except that the agent executes, potentially, 8000 lines of code and can take multiple actions. Agents that decided to create can also execute up to 8000 lines of code, which code may write instructions for another agent to be added to the population.

The motivation for an agent to create another agent is that it is potentially a way to earn money. Currently a creator extracts compensation in two ways. First, it collects a fraction of profit that its child earns. The default fraction is .5, but this can be modified by the creator by writing instructions to the contrary into the child's code. Of course, when the creator collects this payment, the payment is treated as profit by the creator, which then passes a fraction recursively to its parent in turn. Second, all agents pay a small constant amount to their creator each turn. This allows the creator to ultimately recover all wealth earned by a child. In practice, many agents become inactive since newer agents outbid them. This constant payment to their parent (even when they are not acting or even calculating and paying rent) eventually removes them, returning their money to their parent.

New agents require capital so they can bid or create. Unlike in Hayek1, there is no credit in Hayek2– agents can bid or create only if they have sufficient cash in hand. Creators give their child an amount of money set equal $\min(1.2h, h + 200)$ where h is the high bid of the last 20 auctions. Agents are removed (as unprofitable, and therefor presumably a bad bet) if their wealth ever becomes less than their startup capital, and their remaining assets are transferred back to their parent.

In terms of improving the system by adding new agents, a rational observer would face a complex n-armed bandit problem. There are dozens of creation agents in the system. Which one should we try next? How should we apportion creations among them? Hayek2 is doing this based on capital. Agents' creations are limited by their capital. Agents that earn more create more, typically investing all the money that they earn.

Agents periodically pay rent for the computer time they utilized. Each agent's rent was set at

$$R = cyX \tag{3}$$

where R is the rent, c is a constant, y is the number of instructions executed by the agent, and X is the total number of instructions executed by the whole system per instance. Market analogy suggested having the price per instruction executed by the agent increase with total demand of the system. I do not understand analytically whether price should be linearly proportional to demand, and indeed (Drexler and Miller 1988) has proposed instead a sophisticated auction scheme. Empirically, a linear relationship worked far better than constant price or an exponential relationship (even when constants were tuned to put us in the regime where the exponential is well approximated linearly). If the rent is proportional to X, then the amount of money a marginal agent must earn per instance is proportional to X. But the number of instances per wall clock second is inversely proportional to X. Thus rent proportional to X allows agents earning a fixed amount per second to survive.

We are at present requiring a brand new agent on its first bid to out-bid an existing agent by a margin equal the profitability of the existing agent, estimated as its earned wealth divided by the number of instances it has won auctions. Thus, to win the auction the new agent must bid as much as the old agent is actually earning, not just what it is bidding. This helps to prevent the possibility of new agents entering whose action is actually worse then the action currently being taken, exploiting the under-bid of the current agent, a possibility we discussed in §4.6. It also helps ensure that creators get compensated. When creator $C1$ creates agent A, another creator $C2$ might simply copy agent A getting a new identical agent B that outbids A. Then $C1$ would earn no money. By requiring B to outbid the amount A is actually earning, we require $C2$ to inject some new intellectual property to produce a successful agent. We also credit A (and thus indirectly its creator) with its full gain in intellectual property– A pays as bid the amount the previous agent it outbid was earning. The amount A earns net of this is intuitively a fair measure of the improvement represented by A. (We previously experimented with a more complex intellectual property scheme involving royalty payments, but this did not seem effective. It is, in fact, a much debated question whether intellectual property protection is necessary or beneficial in real economies, cf. (Breyer 1970)).

These design decisions do not represent a fully *laissez-faire* economy. In principle, it might be best to have powerful agents that are free to invest an arbitrary amount of capital in their children, and to remove an arbitrary amount of money from their children as they see fit (in their capacity as owner). Also, we should probably auction off computer time, as opposed to fixing a rent. Our agents could have other capabilities, such as being able to pass information from one to another. All this would be great if our agents were rational beings, but we face the bootstrap problem of creating an intelligence from initially random agents. Our response has been to try to make some decisions for the agents. We have been guided, by what it seemed would be a rational decision for the agents. So, for example, we demanded agents invest $\min(1.2h, h + 200)$ in their children because this seemed an amount likely to be sufficient to allow their child to bid, but not vastly greater. These design decisions arose to some extent from experimentation. For example, we initially demanded agents invest $1.2h$ in their children. This turned out to have the unintended consequence of causing occasional crashes in the system, because the new agent could ratchet up the bid by a factor of 1.2, so a sequence of new random agents would occasionally push the bid up exponentially and drain all money from their creators, and hence the economy.

The economic framework has given us a way to decide how to make these decisions for the agents. We are making these decisions by trying to decide what would be rational from the point of view of the agent. Thus we don't have to understand the whole system, to attempt to make these decisions. But because we are highly limited in being able to decide the best interest even of the agent (we are just coding very simple rules, and don't have access to all the relevant factors) our interference can, and has, led often to unintended consequences. Our economy has improved greatly over the course of the last year's experimentation with rules, and I expect it could improve further.

The system is initiated with a root agent. The root agent was written to produce 2 types of children (with equal probability). Type 1 consists of 256 lines of random code. Type 2 are agents that pick a random agent in the population and copy it with mutations. When Type 2 agents pick another Type 2 agent to copy, they mutate its copying code so gradually complex creation agents develop. Initially nobody has any money so everybody who wants to bid bids 0. Eventually (after a few hundred or thousand instances) an instance gets solved and money starts to flow into the system. Eventually there is a complex economy of several dozen agents solving Blocks World problems.

5.2 Representation Language

We need a language for the agents to use. We would like the language to be expressive enough to be able to solve fairly general problems, such as

Blocks World, without intermediate reward. The language must be able to talk naturally about agents, as well as the world, so we can create other agents. Our language must be a sufficiently good fit to the world so as to learn acceptably fast. Probably this means we should bias in whatever knowledge is generally applicable, such as topology.

For our first cut we were inspired by Ray's Tierra system (Ray 1991). Ray invented a language called "Tierran" somewhat similar to a simplified machine language, with registers and stack operations. Ray wrote an operating system, and a program that copies itself. His operating system apportioned slices of time to all programs in it, and only executed operations stochastically– that is it was engineered to make occasional errors. His replicator copied itself filling the memory, but along the way mutated versions cropped up, and after a while Ray had a complex ecosystem, with programs that acted as parasites on others, hyperparasites, and a wealth of complex and interesting phenomena. Ray's seminal achievement here was getting a language that was evolvable, so it seemed a good place to start for us.

In addition, using a Tierran-like language in an economy seemed to me to be well motivated as a step in the program of developing Ray's work. There has been little progress since Ray's paper on creating more evolvable languages. I believe one reason for this is that there was no standard for evaluating progress. Ray's system doesn't do any useful work– it simply generates biological-like complexity, which is in the eye of the beholder. I hoped to couple a Tierran-like system to an economy. Not only is this, in my view, the logical approach to try to harness a Tierran-like system to do something useful, it also provides a concrete, quantitative yardstick (money earned from the world) for judging whether one language is more "evolvable" than another, which I hope will lead to progress.

Ray believed that Tierra's evolvability came from two features of his language: it has a small instruction set, without numeric operands, and it has address by template (Ray 1991). We implemented languages with both these features, that like Tierra are register and stack based. We refer the reader to Baum and Durdanovic (1998) for a fuller description of the language and will give only a brief summary here.

Agents in Hayek2 consist of a program of up to 1000 instructions that reside in a "Soup" stack. Each agent has 10 additional stacks and 7 registers. The stacks and registers are typed. There are four types: colors (e.g. of blocks), instructions, instruction addresses, and integers. The point of the typing was to inject bias, cutting the search space. So colors are never compared to instructions, and instruction addresses (which can be big numbers loaded onto a stack, say, by a command pushing the address of a pattern) are never compared to small integers built up in a register by a sequence of increments and adds, and naturally representing, e.g. stack number or stack height. The world is accessed through a "look" command that pushes on a color-valued stack the color of the world at position specified by integer registers RX and RY. Because RX and RY have to be manipulated by incrementing and decrementing (and adding and subtracting), we hoped that this would bias in appropriately 2 dimensional topology.

As in Tierra there are two pattern delimiters noop0 and noop1. There were several jump instructions that would jump pointers to the next occurence of a pattern in the soup stack. The pattern was specified by the series of instructions following the jump instruction up until the end of the next sequence of noops. Jump instructions could be used, for example, to move a read pointer to a pattern in some other agent in the soup. Copy instructions could then be used to copy the code of this agent into a child agent. We also supplied stacks specifically for patterns of instructions, and commands to push patterns onto these stacks, and commands, e.g., to copy all instructions starting from the next occurence of a pattern and ending with the next occurence of another pattern. These instructions provided means for creation agents to move chunks of code into their children.

We provided a number of specialized instructions for bidding, for grabbing and dropping blocks, and for specifying the fraction of profits one's child should pay one.

We do not claim this language is in any way optimal. Our results improved when we added typing, and when we expanded the instruction set to include more powerful pattern based meta-operations. I expect much better languages could be designed. The language probably has too many instruction types (92 at the moment), which means we are searching a space consisting of hundreds of agents (in the population at any given time), each hundreds of instructions long, each instruction being one of 92 possibilities. It is a wonder we can learn anything. We might also be much better off with fewer stacks and registers. Also, in spite of this, I expect our meta-instructions are insufficiently powerful. There is currently no effective way, for example, to branch based on comparing instructions.

5.3 Results

We trained this Hayek system up, subjecting it to a series of random instances drawn from a distribution (described below), and giving it reward $1000i$ when it solved an instance of size i. Periodically we tested it on 100 instances each of size $1, ..., 10$, rating its performance by total reward on this set.

In each run Hayek solves an instance within the first few hundred it sees, then starts getting feedback, starts rapidly improving its score, and eventually has a system of agents scoring well. In Fig. 3 we graph the last bid of each auction for a typical run, and pay-in, and in Fig. 4 we graph the sampled money (scored every 50,000 instances). Note the bid converges nicely to approximate average pay-in. Note that except for an occasional overbid incident (quite rare in fact) Hayek makes reasonably steady gains, hitting occasional plateaus and then later discovering a new approach and making more progress. The run ran for 4 days on a 330 MHz Pentium 2. Note that

Hayek2 makes a jump late in the run, and it is not at all clear it has stopped learning.

Hayek2 is definitely succeeding in meta-learning. We compared Hayek2 to a version we call "NoML" with meta-learning turned off. NoML is identical to Hayek2 except that the creation is not allowed to evolve. In NoML, whenever an agent wishes to create another agent, we disregard the code in the creating agent but instead create a child exactly as the children of the root create children: by copying with mutations. NoML learns, but not as well as Hayek2. Hayek2's best score to date was 170, and its average score was 145. (We computed this average by doing 8 runs, each 4 days, discarding the bottom 4, and averaging the scores of the top 4). NoML's best score was 132, and its average 119. This difference has qualitatively been very stable as our economy has evolved. We have now studied numerous versions of Hayek, with somewhat different instruction sets and somewhat different economic rules, and we have compared them to NoML versions. In each case, Hayek has outperformed NoML, and typically by a similar margin.

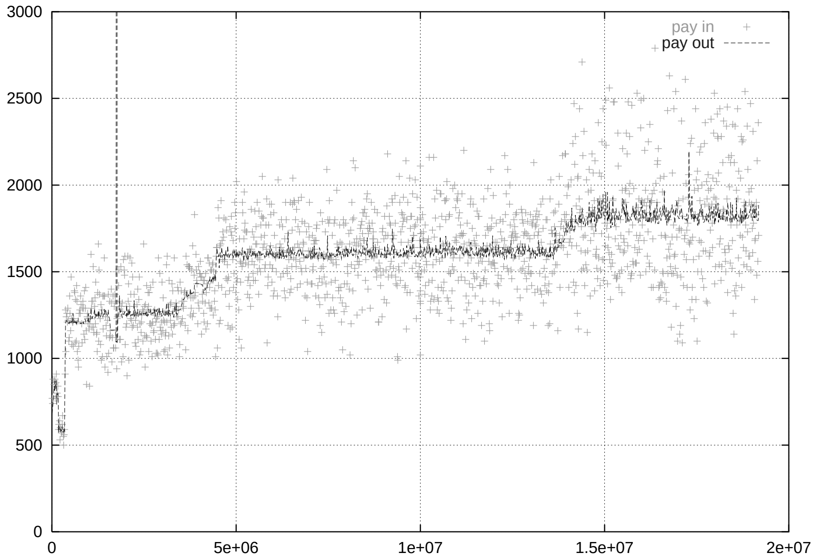

Fig. 3. The bid winning the last auction of each instance is graphed against instance number for a run of Hayek2. Also shown (+ *signs*) is reward paid by the world, averaged over the last 100 instances.

The reason why Hayek2 outperforms NoML is evident from examining graphs of their progress. The final bids in a typical NoML run is shown in Fig. 5. The NoML runs start out very similar to Hayek runs (as is to be expected since initially they are creating identically and are thus identical),

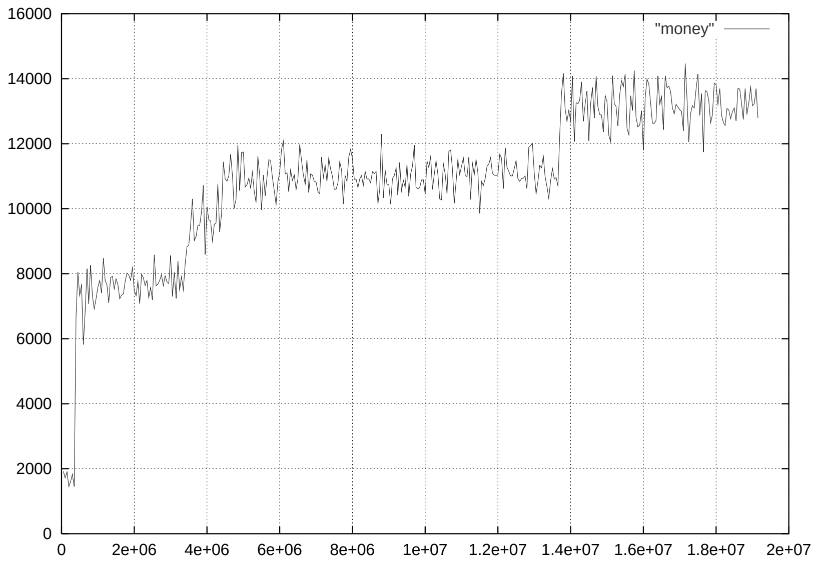

Fig. 4. The "score" calculated every 50,000 instances on a sample of 100 instances each of size 1,...,10 for the run with bid shown in Fig. 3.

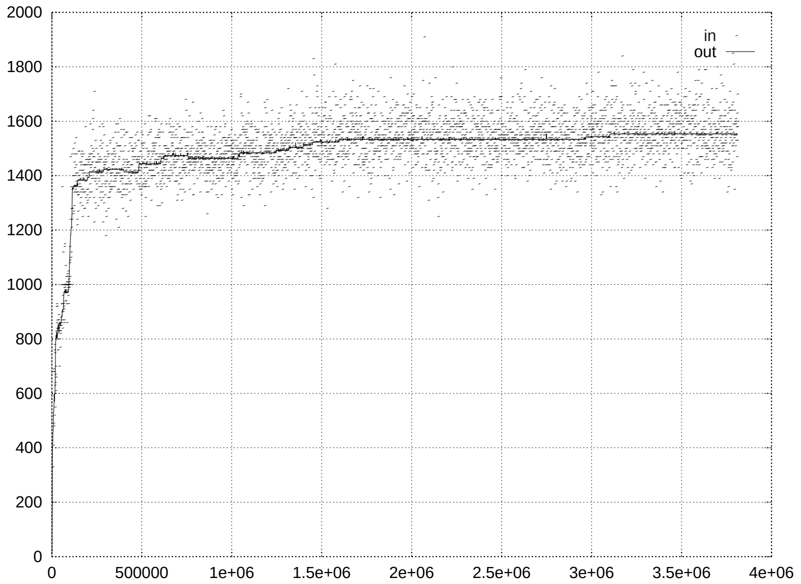

Fig. 5. The bid winning the last auction of each instance is graphed against instance number for a NoML run. Also shown as dots is the final pay-in.

improving rapidly and then plateauing. But they never make the later jumps that Hayek2 makes. Examining the agents in Hayek2 runs, there is typically an identifiable creator associated with the beginning of each jump. Hayek2 discovers a new, improved way of creating agents, initiating a period of rapid progress in which it discovers numerous new, better agents. Then it plateaus again for a while, till it discovers a new, better way of creating agents, and jumps again. Jumps occurred late in many of the Hayek runs, including some we let run 2 weeks. For all we know Hayek2 might continue learning indefinitely, although jumps seem to be somewhat rarer as time goes on. We have examined some of these creators Hayek2 has discovered, but unfortunately been unable to make much sense of their code.

Hayek2's performance has improved over the past year as we improved the economy, and the instruction set. Important improvements included typing the language, adding more pattern based meta-instructions to the language as discussed in §5.2, and modifying the initial capital allocation by parents to their offspring to avoid "speculative bubbles", making the rent proportional to total demand, changing the way agents compensated their parents (by including constant payments, and also by changing way parents specify percentage of profits), changing ways we attempted to compensate for intellectual property by demanding new agents bid more than a present agent is earning, all as discussed in §5.1, and changing the distribution of instances presented. For a more detailed discussion see Baum and Durdanovic (1998).

We also compared Hayek2's performance to two other evolutionary algorithms: a hillclimbing algorithm and a genetic program (GP). We made considerable efforts to optimize these algorithms, and our work on these is, we believe, of some independent interest. Our GP, in particular, demonstrated considerable gains from crossover as against simple mutation. This is a subject of considerable controversy in the literature, and our experiment is the first we are aware of demonstrating such clear gains from crossover. We discuss these evolutionary algorithms in some detail in §6. Hayek2 has outperformed both of these systems. Hayek2's best score was 170 and its average 145; as against a best of 136 for the GP and average 130, and a best score of 125 for our hillclimber and average 102. For reference, an algorithm that simply does 100 random legal actions scores 42. Hayek2's best performance as a function of i is shown in Table 1.

Distributions For all the programs (Hayek2, Hillclimbing, and GP), learning depended importantly on the distribution of instances we presented during the training phase. Typically it was important to present smaller instances first and grow the size of the instances as learning proceeded, so that very roughly half the instances presented are being solved at any given time. This makes sense, since this presumably provides the most bits of feedback per instance. Also, however, given that we were evaluating the programs on a sample chosen uniformly from sizes 1-10, and giving credit proportional to

Table 1. Fraction of instances solved by size n of goal stack for best run of Hayek2

n	Hayek2
1	1
2	1
3	1
4	0.93
5	0.65
6	0.31
7	0.22
8	0.04
9	0.03
10	0.01

the size of the instance, it was important to present occasionally larger instances, so that the training distribution bore some relationship to the testing distributions. We have not experimented much with different distributions, but tried out two or three different distributions for each of Hayek, the GP, and Hillclimber, and in each case report results with training on instances drawn from the empirically best distribution found for that algorithm. (All programs are tested on instances from the same distribution: 100 instances each of size 1-10.)

For Hayek2 the distribution consisted of a sum of two distributions. (This distribution may seem odd, but it was in fact the first distribution we tried! We later experimented with a second distribution for months, before returning to our initial choice.) With probability 1/10, we chose i with probability proportional to $1/2^i$ for $i = 1, ..., 10$ and presented an example of size i. With probability 9/10, the probability of presenting an instance of size i was proportional to $p_i = 1/2^i(1 - \text{solved}(i))$, with solved($i$) defined as the fraction of instances of size i that Hayek solved of the last 10 instances of size i presented. Thus this distribution was initially exponentially weighted toward small instances, but presents larger instances as Hayek learns to solve smaller ones.

6 Other Evolutionary Approaches

The subject of this paper is structural credit assignment in reinforcement learning. There are two standard evolutionary approaches to credit assignment. The first is local search. At any given time we have a current favorite program (CFP) that decides what to do in the world. We make small changes to this program getting a new candidate program and evaluate the candidate. If the candidate is better we replace our current favorite with the candidate. We iterate this procedure. Tabu search and simulated annealing are simple variants on this that attempt to escape from local maxima. Local search does credit assignment because it evaluates the small change in the context of the rest of the program.

The second evolutionary approach is genetic algorithms and genetic programming. Genetic approaches do not do explicit credit assignment, but rather attempt to achieve "implicit parallelism" by swapping around building blocks. This is an interesting alternative proposal and we will discuss it in §6.2 and §6.3.

6.1 Hill Climbing

Local search is a powerful algorithm, cf. (Papadimitriou and Steiglitz 1982). In large, complex, real world problems with no particular structure one might hope to exploit cleverly, it is perhaps the best you could generically hope for. For example, large random travelling salesman problems are well addressed by local search (Johnson, Aragon, McGeoch and Schevon 1992). Local search is a vast improvement in such systems on random search because it makes systematic progress. Random search draws values from some distribution. If you assume, that the distribution will typically be normal, multiple draws will gain only logarithmically. But systematic progress can gain substantially, depending on the structure of the fitness landscape.

Local search works well because it addresses assignment of credit. The new small change is considered in the context of the whole complex system, and a decision is made about its merit taking into account its interactions with the rest of the system.

Local search has several problems, however, particularly when applied to programs for reinforcement learning. First, from what space do we draw local changes? If we draw changes that are too large, the value of the changed state is decoupled from the pre-change state, i.e. it is no longer local search, but reduces to random search. As search proceeds and we get to a better state, it will typically become hard to find good changes. If we draw changes from a set that is too small, we will hit a local maximum sooner; e.g.3-opt (changing links between 3 cities) works better than 2-opt for travelling salesman. A related problem is that random generate and test may become an increasingly inefficient way to find useful changes as the system improves. Hayek2 is attempting to deal with these problems by meta-learning a change set. Effectively it is doing local search at the meta-level to find a good change set, and escape from local optima, and suggest good changes.

Second, how do we evaluate states? In reinforcement learning contexts in particular (but also elsewhere) we are faced with an evaluator that may be stochastic. We can evaluate a program for solving Blocks World problems by trying it on some instances. A change may perform better on the current instance, but worse overall. How many instances should we try it on to decide if it is an improvement? A local search algorithm in the presence of a stochastic evaluator is apt to accept changes that actually make its solution worse. This limits the performance of the local search algorithm. Say, hypothetically, that for any CFP, we can suggest changes that with probability p improve it. Say p decreases slowly as the CFP becomes better. Then, if we could accurately evaluate the CFP, we could get an arbitrarily fit program after a while. But, if we accept changes that actually worsen the fitness of our solution, we will get stuck at the fitness when the expected decrease in fitness of the change we accept equals the expected increase. An alternative way to look at this problem is that there is a risk that we will overestimate the fitness of our CFP. In fact, our evaluation of our CFP may be higher than the true fitness of any solution in its neighborhood, including itself. Then our local search algorithm is stuck. Baum, Boneh and Garrett (1995) analyzed how local search in the presence of noise can get stuck in this way even for a simple toy problem with no true local optima. To avoid these problems, we will need to test using a lot of instances, in order to get an accurate evaluation. But this is expensive.

Hayek2 is attempting to deal with this problem by what amounts to a backtracking algorithm. New agents are tested as they enter. If they lose money on their first instance, they are immediately rejected. But if they make money, they stick around and undergo further testing. A newer candidate agent may be proposed with a higher bid, in which case the first agent will cease being tested. But when an agent goes broke, we backtrack to the previous agents with lower bids and resume testing them till their wealth goes negative. One might view the problem as really described by a complex n-armed bandit, with the arms being all the candidate programs we have yet proposed and evaluated on some instances, and our option to test one of these candidate programs more, or to suggest yet another new program. Hayek's proposal may not be the optimal solution to this n-armed bandit (which I have not yet succeeded in analyzing) but it is an intuitively reasonable one. The proposal used by hillclimbing is much less flexible– never backing up at all.

Genetic programs, which we will discuss shortly, deal with this problem by using a beam search. This is also a sensible proposal. One's evaluation of one's programs will fluctuate at each step because of the stochastic evaluator, but if you are keeping, say, the best 50 programs in your population around to the next generation, you will not lose the program with true best fitness unless your evaluation succeeds in fluctuating it down to position 51. I hope to analyze these issues of evaluation in a noisy environment for evolutionary algorithms in more detail elsewhere (Baum n.d.).

A third problem of the hillclimbing approach is that it uses end reward to evaluate the solution. This, as argued e.g. by Sutton (1988) can be very inefficient. It may be much better to criticize our choice of move 22 in a chess game in terms of our evaluation of the position after move 23 than by the end result at move 62, because several errors may occur between moves 23 and 62 decoupling the final reward. Hayek's bid scheme is intended to address this problem. The bid in any state represents the system's knowledge regarding the value of that state, and it is this that we use for update. An example of how this can be important was discussed in §4.3.

A fourth problem of hillclimbing is that it has no modular structure.

Hillclimber for Blocks World We constructed and tested a hillclimbing algorithm for Blocks World, using the same instruction set as Hayek2 (but omitting the meta-instructions, since it does no meta-learning), and with some optimizations to address problems 1 and 2 above.

The first CFP was generated in an initiation phase where we proposed programs consisting of 256 lines of random code, and tested these on 1000 random instances. The first one solving 20% was declared the CFP.

We then iteratively created new candidates by randomly deleting i lines of code in the CFP with probability $1/2^i$ and randomly inserting j lines of code with probability $1/2^j$. The new candidate was tested on a similar test of 1000 instances, with cutoffs at 1, 10, and 100 instances. If it either solved as high a fraction of instances or earns as much money per instance, it survived each cutoff, else it was rejected. If it earned more money on the full test of 1000 instances, it became the new CFP.

The test on 1000 instances seemed sufficient to avoid serious problems with fluctuations in the evaluator, but smaller tests were too small. The use of cutoffs was an attempt to avoid the cost of a 1000 instance test. The new candidate generation process was chosen after some empirical testing of alternatives.

This hillclimber worked pretty well, generating a Blocks World program that scored 128, but not as well as the genetic program or Hayek2.

6.2 Genetic Algorithms and Credit Assignment

Genetic algorithms are a conceptually different approach. They do not attempt to do credit assignment per se. Rather they attempt to do better by an approach inspired by biological evolution. Biological evolution swaps genes among individuals. The fit genes are those that produce many offspring. This is context dependent– how a gene works will depend on what other genes are in the population. To paraphrase Dawkins, cf. (Dawkins 1989, pp91-92), this is something like the way a coach might select a rowing team for an 8 man shell. Rather than decide who are the best 8 individual rowers (which might be impossible, and anyway might not produce a good team) he can swap all his potential rowers among boats. The rowers that are often on winning boats will then be judged strong rowers.

Many variations of genetic algorithms have been discussed, but for illustration we will first dicuss a popular one. One has some function one is trying to optimize. Fed an n bit string, it returns a fitness. One initializes with a population of M random strings, finds their fitness, and then draws a new population of M strings. A popular way to do this is by drawing strings from the old population with probability proportional to their fitness, and using "crossover" to create new strings. Given two strings A and B, crossover might involve picking a random component i between 1 and n, and forming a new string with components 1,...,i from string A and components $i+1, ..., n$ from string B. One may then further modify the strings with some point mutations.

Now a major motivation for the field was Holland's Schema theorem (Holland 1975) which models the selection of "building blocks". Holland defines a "schema" as an axis parallel hyperplane, i.e. an assignment of some component values. Any string satisfying this assignment is said to be a member of this schema. The notion is that certain schema are building blocks that contribute to fit strings. Let H be a schema. Holland defines the fitness $f(H)$ of a schema to be the mean fitness of its members in the population. Let μ denote the average fitness of all strings in the population. Now when we draw strings proportional to their fitness, the expected frequency of all schema is proportional to their fitness. Let $\delta(H)$ be the distance between the first and last defined components of H. When we cross over, the probability of disrupting H is $\delta(H)/n-1$. Let o_H be the order of H, the number of components defined in H. When we mutate components with probability p_m, the probability we disrupt H is only $p_m o_H$. Let $m'(H)$ denote the number of examples of schema H in the population after the reproduction and $m(H)$ denote the number before. Thus we have Holland's schema theorem:

$$m'(H) \geq m(H)\frac{f(H)}{\mu}\left[1-\frac{\delta(H)}{n-1}-o_H p_m\right] . \tag{4}$$

This says that for any schema fitter than average, of low enough order and length, its frequency in the population increases exponentially with the number of generations. This holds for a vast number of schema simultaneously, which intuitively might mean that this kind of search shuffles a vast number of "building blocks" in an efficient search for fit strings.

Now the question is, when does this really lead to a better optimization algorithm? One problem is that the implicit credit assignment given by selection proportional to fitness is too slow. It is easy to see (Baum et al. 1995) that, if we have a population of strings of mean fitness μ and deviation σ, if we draw a new population by drawing proportional to fitness (not even doing any crossover) the mean fitness of strings in the new population will be $\mu+\sigma^2/\mu$.

Consider now how this works on a toy problem, the Additive Search Problem (ASP). Here each component of our strings has L possible values. For each component there is one distinguished value, and the fitness of a string is the number of components it has right. Then we have $\mu \sim O(N)$ and $\sigma \sim O(\sqrt{N})$. So for the ASP problem we have a gain in mean fitness per generation of order 1. This means that in order to solve the ASP problem,i.e. to discover the fittest string, we naively would take of order N generations. But we need to use a population size large enough so that fluctuations do not remove critical schema from the population altogether. For the ASP problem, it is not hard to see this means we need to use a population size of order N, which means we need to do of order N^2 evaluations to solve this problem by this genetic approach. But the naive hill climbing approach: keep a current favorite string, iteratively change one component, and accept the change if it doesn't worsen the score, is easily seen to solve this ASP in time linear in N. Thus this 'standard' genetic algorithm is quadratically slow compared to naive hill climbing.

This problem can, however, be dealt with using a more aggressive selection algorithm. If, instead of drawing strings proportional to fitness, one discards all strings less fit than the mean, one gains about one σ in fitness of the population per generation. (Selection of strings fitter than the mean can be shown optimal under very general conditions among a class of culling algorithms (Baum et al. 1995).) For the ASP problem, this means we gain of order $\sqrt{N}$ in fitness per generation, and thus naively solve the problem in only $\sqrt{N}$ generations. Because we now expect to solve more quickly, there is less time for fluctuations to remove critical schema from the population, and hence we wind up needing a smaller population as well. If the evaluator is noisy (i.e. instead of returning the true number of correct components, returns this number plus some noise), the resulting genetic algorithm can be shown to be faster than hill climbing, or any other algorithm known for this problem. If the evaluator is exact, the genetic algorithm is conjectured to be faster than hillclimbing. For a full discussion see Baum et al. (1995).

These results show there is something to the idea of genetic search. But a second obstacle is more serious. For the ASP problem defined above, there are no important schema that depend on more than one component, so there is no structure for crossover to destroy. The question is, can genetic approaches succeed in the presence of such structure? Unless the components of the string are very well ordered indeed, good solutions may depend on schema of very long length, which are disrupted frequently. Also, the fitness of a particular schema may depend critically on many other schema. If you don't do credit assignment, you will be doomed.

In Baum et al. (1995), to get a handle on this obstacle, we analyzed a slightly more complex test problem, that we called k-ASP. Here the components were divided into N/k groups of k components each. You got a group right when you got all of its components right. The fitness of a string was the number of groups it got right. We were able to analyze the performance of some genetic algorithms on this problem, and to empirically test others, and they did not perform well compared to algorithms doing explicit credit assignment, even when the groups were formed of sequential components. We were able to give an algorithm that used binary search to perform explicit credit assignment that performed very well on this problem.

I believe this second obstacle to be almost insuperable in string based genetic algorithms. There are few if any problems for which GA's have been shown better than a simple hillclimbing approach. Even constructing toy problems on which GA's can beat hillclimbing has been problematic (Forrest 1985, Baum et al. 1995). However, genetic programs are potentially a different story.

6.3 Genetic Programming

In genetic programming one maintains a population of *programs*. One assesses their fitness, selects the fitter programs, and then creates a new population of programs from these by some combination of crossover, mutation, macro-mutations, and simple copying.

A key feature is that now there is no fixed order of the components. The components are instructions in a computer program. They can be moved around. Thus, for any collection of instructions that need to act as a unit, they can potentially be brought close together, and thus made less likely to be chopped by crossover.

A typical (though by no means unique, cf. (Banzhaf, Nordin, Keller and Francone 1998) for a survey of alternatives) representation for programs in Genetic Programming is as S-expressions. These are isomorphic to trees. See Fig. 6. Crossover between two trees is done by randomly picking a subtree of each and swapping them. (If the language is strongly typed, than the type of the expression at the root of each subtree must match, as in the figure.)

Now the question is: does crossover in fact achieve anything in Genetic Programming? In 1995 Lang published a paper showing that crossover with random trees was substantially more effective than crossover within the population on a particular problem (Lang 1995). Crossover with random trees was dubbed "headless chicken mutation" and considered as a macro-mutation operator. Two groups (O'Reilly and Oppacher 1994, Angeline 1997) within the genetic progamming literature compared macromutations to crossover on a suite of other problems and found no strong gains from crossover. To quote a recent text on Genetic Programming (Banzhaf et al. 1998, p155) "*The empirical evidence lends little credence to the notion that traditional GP crossover is, somehow, a more efficient or better search operator than mutation-based techniques... On the state of the evidence as it exists today, one must conclude that traditional GP crossover acts primarily as a macromutation operator.*"

So it was that after having performed the experiments discussed above where we applied hillclimbing and an artificial economic model to Blocks World, we implemented the best genetic program we could to attack the same problem in full confidence that it would be a dud and prove yet again the supremacy of our economic approach. In fact, our genetic program did very well, not as well as our economic model, but vastly better than a genetic

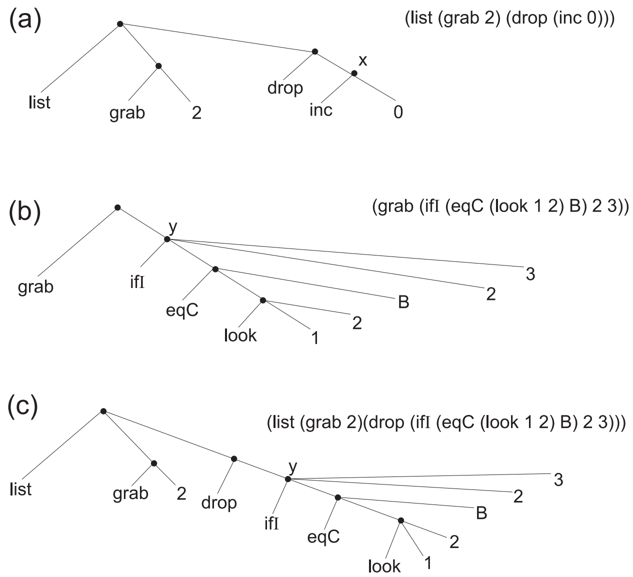

Fig. 6. Crossover of S-expressions. (**a**), (**b**), and (**c**) show S-expressions written both as trees and as expressions. Each open parenthesis in the expression corresponds to a node in the tree with the first expression in the parenthesis corresponding to the first branch. The first expression is some function (that may be a tree itself) and the other branches correspond to its arguments. The tree in (**c**) is given by crossover from the trees of (**a**) and (**b**) by replacing the subtree rooted at node (x) in (**a**) by the subtree rooted at (y) in (**b**). Note the language is typed. For example, "ifI" is an statement that returns an integer. "drop" expects an integer argument, so the subtrees at y and x can legally be swapped. "ifI" takes three arguments, the first Boolean and the other two integers. It returns either the second or third of its arguments, depending on whether the Boolean is true or false. In this case, its Boolean is "eqC", a Boolean function that expects two color typed arguments and returns true if their colors are equal. Its arguments here are (look 1 2), which returns the color of the block at location (1 2), and B which is the color blank.

program without crossover. Crossover GP achieved a best run of 136 and a top-half mean of 130, compared to a Headless chicken GP max of 121 and top-half mean of 97, and a Hayek best of 170 and top half mean of 145.

Our initial population was randomly grown from the top down respecting type constraints. Each generation we tested each S-expression on a sample of instances, then selected the top 10 trees to copy intact into the next generation, then filled out our population of 100 trees by crossover among the top 50 trees followed by 10% mutation. One advantage of Genetic Programming over hillclimbing (commented on in §6.1) is that by executing beam search we are somewhat protected from fluctuations in our noisy evaluation. The best tree's evaluation would have to fluctuate down to 10th to not be copied, making it somewhat more likely we won't drop the best tree in our population for random reasons.

It was evident from the form of the solutions as well as the scores that crossover was achieving some implicit assignment of credit, at least at the expression level. Our GP contained an expression "Look" which took two integer arguments i, j and returned the color of the block at i, j. Several of our GP runs with crossover succeeded in purging all "Looks" from their entire population. This means, they were totally unaware of the state of the world. What they were doing is finding optimized stacking routines that blindly stacked blocks on stack 1, then taking them off and rearranging them, thus performing a search. By doing this in an optimized fashion, using their 100 actions, they were able to solve decent size problems. The crossover succeeded in discovering that grabs and drops were more important than looks and ifs, and found good sequences of grabs and drops. Essentially it was solving the Blocks World problems by touch.

Our GP's used strongly typed S-expression trees, and thus a very different (and much simpler) language than our Hayek runs. A widespread intuition within the genetic programming community (however by no means universally shared nor proven as fact) is that tree based representations are more susceptible to mutation and crossover because the branches (which represent closed parenthesis in a linear representation) form natural units for swapping or modifying. We are currently experimenting with Hayek runs in a similar tree-based language. I note in proof: these have now generated far superiour performance to the machine language Hayek2 reported on here. We will describe these in more detail elsewhere.

A general problem with Genetic Programs (Banzhaf et al. 1998) is that as evolution progresses, the programs "bloat", i.e. become huge, eventually halting progress by slowing the program or exceeding memory. (Our runs grew to hundreds of megabytes.) From an economic point of view this is natural: space is a commons, that we are not charging for. In runs where we charged for size, there was far less bloat, but also they did not perform well.

Our conclusion is that there may be something to crossover as an implicit assignment of credit approach, although I continue to advocate research on explicit assignment of credit.

7 Reinforcement Learning and Planning

We have focussed in this paper on reinforcement learning. In this section we motivate our focus on reinforcement learning, arguing reinforcement learning may be as easy to solve as e.g. supervised learning, offers a better window on natural intelligence, and is necessary for understanding high order intelligence. We also discuss planning in hard environments, and how Hayek has learned domain dependent search control knowledge, which will be necessary for planning in hard environments.

The papers in this volume mostly describe approaches to intelligence that begin by carving off some relatively small, seemingly sensible sub-problem. This allows you to make good analytic progress, and sometimes to produce a program doing something useful, like character recognition or medical diagnosis. On the other hand, from the perspective of standing back and looking at the big picture, its not obvious you are making progress in understanding human abilities any more than (to paraphrase Dreyfus discussing the field of AI as a whole (Dreyfus 1972)) you progress in reaching the moon by climbing a tree. Human abilities seemingly involve, at a minimum, logical operations that go well beyond the standard picture dealt with by the Bayes net or neural net communities, including quantified variables, temporal reasoning, planning, and language. I emphasize this because many of the researchers in the field seem oblivious to these problems (and sometimes even disdainful of mainstream AI researchers who are more conscious of them).

Indeed, carving up intelligence into seemingly well motivated subfields may introduce intractability into the problem that in principle could be avoided in a more wholistic approach, or perhaps simply by a different dichotimization of the problem. For example[9], Khardon and Roth (1994) have been able to give formal examples where learning a representation of the world is intractable, and/or reasoning from a representation of the world is intractable, but learning to reason directly is tractable. Their point is that interaction with the world is important, and the knowledge representation you want depends critically on the functionality.

Reinforcement learning is a relatively understudied subarea of learning, itself a small part of the field of "artificial intelligence"; but we have three motivations for our focus on reinforcement learning. First, from a practical point of view, if we hope to create a robot in our lab to do tasks for us, we are going to have to be able to tell the robot what we want it to do. If it can reinforcement learn, we will be able to just write a program rewarding it when it achieves the state of the world we desire, and train it rather than program it. Writing such a reward program may often be much easier than writing a program saying how to achieve the desired state, which involves, at a minimum, solving planning problems. Such a reward program might, incidentally, contain guidance in the form of rewards for intermediate states. If we can state what we want our learning system to do, we can reward it when it does it. Thus if we can solve reinforcement learning, we can provide a declarative programming language.

Second, from a theoretical point of view, studying reinforcement learning is motivated by the results of Khardon and Roth discussed above. Reinforce-

[9] Another conjectured example of analytic methods introducing intractability is Carver Mead's proposal (Mead 1987) that hard correspondence problems arise in vision when computer scientists discretize vision, but are avoided entirely in biology by use of analog computation.

ment learning involves interaction with the world, and addressing simultaneously the "learning" and "reasoning" or "planning" problems.

Third, I suggest it is edifying to regard human or animal intelligence as created by a three facet reinforcement learning process. Facet 1 was biological evolution, that can be regarded as a reinforcement learning process with a somewhat strange reward function: roughly number of grandchildren. Facet 2 is learning within our lifetime, which I propose should be viewed largely as a reinforcement learning process making use of a complex intermediate reward function, programmed into us at birth by evolution. This complex intermediate reward function is quite detailed, including not just pleasure at eating when hungry, but pain at overeating; pain at hearing one's child crying; and perhaps pleasure at pleasing one's parents. As will be discussed more in §8, one should regard evolution as compiling knowledge into us in other ways than simply through a reward function, but surely such a built in reward function is an important influence on our development. Facet 3 is the evolution of technology, or more generally culture[10], which is largely a process whereby innovations survive that are successful in dealing with the world. To the extent that you accept this picture of animal and human intelligence as arising from a reinforcement learning process, studying reinforcement learning lends insight into animal intelligence, an end in its own right, and seeks as well to understand the only example of intelligence we have.

Recently Kearns and Singh (1998) have reported dramatic theoretical progress on reinforcement learning. Given a world represented as a Markov process they described a learning algorithm that outputs an optimal strategy in time polynomial in T^*, the "mixing time" of the best policy, and N, the number of states of the best policy. They showed that any learning algorithm not utilizing knowledge about the world must use time growing at least linearly in N and in T^*. Unfortunately, this work is at the opposite pole of the situation I wish to consider. For typical worlds of interest, the number of states N is (at least) exponential in some reasonable parametrization that exploits structure. I am interested in how we can learn about underlying structure, and learn to exploit it. The difference is somewhat analagous to the difference, in a supervised learning context, between memorizing a set of examples, and learning a very compact representation allowing prediction of future unseen examples. Kearns and Singh's algorithm uses enough examples to essentially memorize the structure of the transition matrix of the world (or at least the important component of it). We are interested in situations where one might be able to predict the structure of the space from vastly fewer examples.

Ultimately to solve hard planning problems, we are going to have to learn to exploit the compact structure of the world. Blocks World has such compact structure, as was clear from Hayek1's discovery of a finite set of agents capable of solving Blocks World problems of arbitrary size. To exploit such

[10] Culture is of course a property of animal intelligence too, cf. (Bonner 1980).

compact structure requires learning about the particular problem domain, and integrating this knowledge with planning. The planning literature has mostly avoided this issue, focussing on domain independent heuristics for organizing search, or human-crafted domain specific heuristics. Domain independent heuristics can make a big difference, as shown, e.g., by the nice algorithm of Blum and Furst (1997), but ultimately this can take you only so far. I propose that it is time to face up to the hard problem of how it can be possible to learn to exploit compact structure in planning, and that the key to addressing this is addressing structural assignment of credit.

We can get additional perspective on how reinforcement learning integrates domain knowledge into search from very recent experiments (done in collaboration with Igor Durdanovic) on a version of Hayek2 using typed S-expressions like those in our Genetic Programming experiments (c.f. §6.3) instead of the Tierran like assembler language described in §5.2. This version of Hayek2 has in some experiments been far more effective at learning to solve Blocks World problems than our assembler language version was, and has the additional advantage of producing agents that are far more human understandable. What we are finding is that this Hayek is engaging in a search, much like classical planners do, stacking blocks in an attempt to find a solution. Of course, classical planners (cf discussion in §4.1) are given a domain model and allowed to engage in extensive virtual search. These classical planners engage in an extensive search. Hayek2 is not given a domain model, but can only execute actions on the world, and is allowed only 100 actions, so it must use a very efficient search. In fact, it learns to stack blocks on stack 1, and then rearrange them on stack 1, searching for a correct stack. Elaborate collaboration evolves between stacking agents (who stack on column 1 up to the height of stack 0) and unstacking agents (who unstack column 1). This represents a use of considerable domain knowledge in its search. From among the possibilities for searching, it has learned to search only sequences of stacks on stack 1, to the appropriate height, and then sequences of unstacks, and then sequences of stacks on stack 1 again. This exhibits the kind of learning of domain knowledge that will be necessary if planning is to be possible in ultra-complex environments.

8 Biological Insight

This section will describe some selected aspects of biological intelligence. It can be read independently of the rest of the paper. I include it because I think there are a number of insights that are underemphasized in the learning theory and artificial intelligence literature, and some ways in which our economic models have similarities to biological intelligence.

We know the origin of intelligence in biology. It arose from evolution. Evolution can be thus be viewed as a learning process: something that interacts with the world and produces intelligence. In this case it is probably best viewed as addressing reinforcement learning but with a rather unusual reward function: number of descendants. This unusual reward function leads, as discussed in §2, to some undesirable features from the point of view of one crafting an algorithm to create intelligence.

Evolution started from near *tabula rasa*. The only way I know of where the initial representation had significant, useful bias, is that it presumably worked with molecules that are objects in three dimensions. This perhaps builds in a topological bias. When we come to write algorithms in our computers, if we treat the world as mere strings of bits, we can easily create a starting point lacking even a bias to the topology of the world. We have no existence proof that it is possible to recover from a *tabula* so *rasa* as not even to bias in topology. These molecules also interact causally, in time, potentially biasing in some aspect of causality.

Evolution works by making small changes and keeping what works. Thus it is akin to a local search algorithm (cf. (Papadimitriou and Steiglitz 1982) and §6 for discussion of local search algorithms). By doing such tweaking, evolution somehow succeeds in building up complex behaviors that are instinctive: inherited, and showing no signs of "conscious understanding". One famous example (Franklin 1995) is the egg laying behavior of the wasp *Sphex*. This wasp goes through an elaborate series of "subgoals", building a burrow, finding and paralyzing a cricket, dragging it to the burrow, checking the burrow, putting it in the burrow, laying eggs in the burrow, sealing the burrow. But the wasp does not seem to "understand" what it is doing in a "conscious sense". If the cricket is moved a few inches away while the wasp is checking the burrow, the wasp will drag the cricket back to the threshold and check again. This process has on one occasion been repeated 40 times. The wasp has no notion that it has just checked the burrow, but apparently goes back into its programmed behavior. Another example of an instictive behavior in a higher animal is the egg rolling behavior of birds. Many ground nesting birds, if you displace their egg a few feet from their nest will roll it back in. They will do the same if you place a softball near their nest, or anything else vaguely the right shape and size.

Indeed evolution has built in rather complex behaviors. Most labrador retrievers love water. Take one for a walk in the woods, and it'll jump into any pond, or large puddle, you pass. They also are notoriously friendly. Get a sheltie, instead, and you can expect it to herd your children if it can't find sheep. Pointers point. These complex personality differences are the result of only a few hundred years of selective breeding. Beaver dams are another example of a complex behavior. And remarkable character traits are presumably controlled in humans as well by inheritance. Experimenters collected pairs of identical twins raised apart from birth (often in radically different environments) and, in addition to all kinds of correlation on tests, etc. found even more remarkable anecdotal similarities (Bouchard, Lykken, McGue, Segal and Tellegen 1990): one pair of women wore seven rings; a pair of men independently and correctly pointed out that a wheel bearing needed replacement in the experimenter's car after he picked both of them up. Another pair of twins reunited for the first time sent each other identical birthday gifts (that crossed in the mail); another pair flew in from different countries wearing blue double-breasted epauletted shirts, mustaches, and wire rimmed glasses, both had hasty tempers and enoyed sneezing in elevators, flushed toilets before and after using them, kept rubber bands around their wrists, and dipped buttered toast in coffee. Not that there is necessarily an individual gene for keeping rubber bands around your wrist: but identical twins share all their genes, and the genes directly control (apparently to a considerable degree) personality in people as well as dogs, and also heavily influence the learning processes that build personality.

How we learn during our lives is heavily influenced by evolution. All sorts of inductive biases are built in. Both dogs (Garcia and Ervin 1968) and rats (Garcia and Koelling n.d.) are known to associate very readily an electical shock to a location or action and poison induced illness to a food, but not to associate readily an electrical shock to a food or illness to a location. In another experiment (Mineka 1980) monkeys raised in a laboratory were found to have no fear of snakes, and would reach over a snake to get a banana. Shown, a single time, a video of a monkey reacting in terror to a snake, the monkeys developed fear of snakes. Shown a video of a monkey reacting in terror to a flower, the monkeys do not develop a fear of flowers. They are biased to learn a fear of snakes, and yet not born with it. Other monkey species are born with fear of snakes– evolution appears to be able to predispose the learning, or build in the fear directly. Both might be imagined to have selective advantages in different environments: having the knowledge built in saves learning it, which might be fatal; having to learn it allows more flexibility, that can be beneficial, e.g. if you happen to live in an area where the snakes are not poisonous.

Language, in humans, requires learning, but starts at birth with enormous bias. Humans must learn language (they're not born speaking), and who they learn it from determines whether they speak English or Japanese. But languages are similar, and most likely the vast majority of the learning has been biased in by evolution. Note, incidentally, that it may well be much easier for evolution to create language than it would be to learn a language from *tabula rasa*: the two are different problems. The one is analagous to creating an encryption system and the other to decrypting one. From this point of view, designing machines that learn to understand language may be very difficult indeed: evolution does not provide an existence proof that it can be done.

Evolution has built in modules into our bodies, and brains. Our bodies have hearts, lungs, livers. Our brains have hippocampuses and cerebral cortexes. These are organized in a hierarchic way, out of cells and cell assemblies (Churchland and Sejnowkski 1988). Within our brains, there is all sorts of evidence for modules with complex behaviors. Localized strokes can cause interesting deficits from being able to write but not read, to making one unable to perceive that the stroke has paralyzed one side of one's body[11]. Brain imaging techniques (Toga and Mazziotta 1996) are now giving pictures of local modules interacting in the context of controlled experiments. These pictures may soon give considerable insight into the modular nature of thought. More generally, by comparing the various different pictures of, and different evidence for, modular thought across disciplines one can perhaps achieve considerable insight.

Psychophysics in particular gives an interesting window on modularity. One celebrated test within the newish field called evolutionary psychology is the Wason Selection test. Subjects are asked to perform minimal manipulation of a set of cards given a simple semantic to verify or falsify a statement like "every time I go to Cleveland, I always drive my car." It turns out people are very bad at such logical reasoning. But asked logically isomorphic questions, like performing minimal manipulation to verify or falsify a statement like "every one who drinks beer must be at least 21" they perform very well. Extensive experimentation with different sets of questions, and different cultures, has established that people reason very quickly and accurately about such questions when verifying social obligations is the issue. In one set of experiments, people's performance improved radically just by inserting the word "must" into the sentence. The hypothesis, now largely accepted within the evolutionary psychology community, is that people have a module[12] specifically for reasoning about and detecting cheating. This is just what evolutionary psychologists would expect. Our intelligence evolved, and a major selective force was likely verifying social obligations and keeping from getting cheated. Our logical reasoning generally is either a spin off from this, or otherwise evolved but under considerably less selection pressure.

In a fascinating book called "Metaphors We Live By" (Lakoff and Johnson 1980), Lakoff and Johnson point out that metaphor is central to our language, and thus possibly central to our thought. For example, "time is money" in a very real way: you can borrow time, save time, waste time, invest time profitably, spend time, lose time, budget time, run out of time, have time to give somebody, etc.. From my perspective this suggests a module for valuable resource management that is being recruited to help understand time, and to allow us to make rational decisions about how to manage our time. Of

[11] Such patients deny they are paralyzed. If handed a heavy tray of glasses, they will reach out with their one good hand and try to take it. When it crashes to the floor, they make excuses for their clumsiness, but never admit they are paralyzed (Ramachandran 1995).

[12] This suggests to me that one should look for such a module with brain imaging techniques. The Wason selection test is a simple test where one compares two tasks that seem logically isomorphic. Having such a control task is necessary for brain imaging. It would be interesting to compare what areas of the brain are active on the two logically isomorphic tasks.

course in other contexts we use other metaphors to understand time. For example, we treat time as a moving object, with the future coming toward us: " The time will come when...let's meet the future head-on." Likewise an argument can be a building: we construct a strong foundation for our theories, lay out a concrete framework, buttress them with solid arguments, etc.; or an argument can be a journey: we set out to prove something, we arrive at conclusions, we proceed in a step by step fashion, etc. A full survey of (Lakoff and Johnson 1980) would be beyond the scope of the present paper. I will just mention here that the hundreds of metaphors they highlight suggest to me that the mind is a complex program with modules (themselves some complex collection of code, e.g. "a module for valuable resource management" is a rather complex program) interacting with modules like objects interact with objects in a complex C++ program.

More abstract concepts are typically grounded in more concrete, physical ones, especially metaphors relating to body position. So, for example, happy, healthy, alive, control, good, status, rational etc. are all up, as in "You are in high spirits... Things are looking up... he's at the peak of health... the discussion fell to the emotional level, but I raised it back up to a high level, intellectual discussion." The modules interact in complex ways, but coherency is maintained. This is all just what one might expect from some evolutionary origin. One learns modules for concrete, physical operations first, then tweaks and reuses these in more and more complex ways to understand and deal with more complex concepts. Probably to some extent this learning was done by evolution, and to some extent during our lives, although strongly biased in by evolution. The question is: can we hope to get a learning algorithm to follow this general path within our computers?

An extensive survey of the evolutionary origins and modular structure of the brain, describing evidence from numerous areas, and particularly connecting with computation, would I think be fascinating and illuminating to the AI/learning community, but is beyond the scope of this paper. We will summarize the points we want to take from this here as follows.

- Intelligence was produced by evolution.
- Evolution started with inductive bias potentially incorporating Euclidean topology and causality, but possibly little else.
- Evolution builds by local search: tweaking solutions and keeping what works. It makes extensive reuse of previous discoveries, arguably often in new contexts and applications.
- Evolution presumably built reactive systems first. Real time performance is critical to survival. Intermediate state is hard to evolve and probably came later.
- Evolution gradually produced very complex behaviors, perhaps first without "conscious thought", but later produced humans. Reflection emerges from reaction?

- Learning during life is one mechanism evolution uses. Evolution builds in complex inductive biases. Sometimes it builds these in in the form of intermediate reward functions, discussed more below.
- Ultimately the result of this is a complex modular structure, with a web of modules that perform relatively complex computations interacting in a complicated way.

We are going to set out to create intelligence by an "evolutionary" approach. We are going to hope eventually to achieve complex systems with interacting complex modules, specializing in computational tasks, passing information and signals, and together collaborating. If one is ever to achieve true intelligence on a remotely human level this seems necessary. It is daunting because of the depth and complexity of the structure required. How can one ever produce such a thing? For any calculation using any intermediate product whatever there is a critical chicken and egg problem: why should one agent calculate the intermediate product before the other agent uses it, and how can the other agent use it before the first learns how to produce it? This is the same question, translated to our simulations, that is posed to evolutionary biologists by skeptics of biological evolution: what good is half an eye? The human eye is a complex machine, and requires an even more complex cortex to process the information. How could such a complex system ever evolve, since take away part of it and it is disfunctional? Our answer must be the same as Dawkins (1988). A slighly light sensitive region connected to flagella is potentially a big advantage compared to no sensitivity to light at all. To evolve an eye, every step along the way must be favored. So too we will hope to evolve complex webs of interacting agents, with a complex hierarchy of data representations. But every step along the way must provide utility at that point. You can't add a module just to compute some concept that may be useful later. Every module, when it is introduced, must improve reward, and thus must contain knowledge about actions[13].

In this context it is interesting to observe that one of the principle ways evolution has programmed us to learn during our lives is by equipping us at birth with a complex, derived reward function. Evolution started with a simple reward function: number of grandchildren. But we are born with a complex reward function, including such subgoals as eating (when hungry),

[13] In the game of Go there is a concept called an eye, unrelated to an eye that sees. A key theorem in Go is that group of stones with two eyes can not be killed. This is a difficult, highly abstract theorem. An agent that did not know about this theorem, but however understood the *functional* knowledge: do not play in your eyes, would still be at great advantage to one that did not, and once it survived might evolve to knowing about the importance of a second eye. Conversely, an agent that recognized when a group can make two eyes (in general an incredibly hard concept, even for humans) would nonetheless be useless and die, unless something already existed that knew how to make functional use of this knowledge.

not eating (when full), orgasm, pain, desiring to have one's kids not cry, etc. Each of these seem to involve considerable inborn distinctions, i.e. not all possible foods, orgasms, pains, etc. are regarded as identical, and many of the preferences seem innate. From this context it is interesting to study learning in the presence of both simple end reward, and some elementary derived reward functions. It is interesting to contrive learning systems where intermediate reward can be learned (as evolution learned the intermediate reward system it supplies us with.) Potentially various kinds of arbitrageurs within economies could provide intermediate reward.

It is evident that self organization of human-like reasoning and planning capabilities may require significant computation. Say we had a computer with the brain's capabilities, perhaps 10^{15} cycles/second, and we ran the brain's learning algorithm on it for a full year. The result of this massive investment would be a system with the capabilities of a 1 year old baby. We wouldn't regard this as practically useful. Perhaps this is one reason why the field of AI disappointed some of the early optimists. Duplicating evolutionary time is an even more daunting task. Perhaps 10^{30}-10^{33} viruses[14] have been created on earth since the dawn of time. Nonetheless, the cycles available to the brain or to evolution have not been infinite, nor even exponential in the size of the problems solved. The computational structures created, by contrast are huge and deep, and naively would require immense time to find. So, if we are right that what evolution is doing is tweaking modules and reusing the tweaked modules in different places, and demanding that every new tweak be effective in generating improved reward, this must be a reasonably effective algorithm on natural problems.

In this context, it is perhaps worth noting that the continuing advance of computer firepower makes it plausible that computers will pass humans in raw hardware abilities within our lifetimes. Myhrvold, for one, predicts that the cost of computing will continue to drop by 10^3 per decade for the next four decades (Myhrvold 1994). Assuming this comes to pass, what will we do with all these cycles? To date the primary sink of cycles has been graphics, but Myhrvold notes that this use will soon be saturated. We predict many cycles will be used to self-organize computation. Programs will be too complicated to be written by hand, and will require automatic robustness. The advent of faster computers is, sociologically, one driving force behind increased interest in understanding evolutionary computation. They allow the kind of experiments we are reporting, and provide motivation for pursuing

[14] A human infected with HIV produces about 10^9 new viruses per day. Figuring loosely that this is typical of viral infections, and that at any time of order 10^9 people are infected with viruses, we have 10^{18} viruses created per day by humans. Maybe this is a tenth to a ten thousandth of the total number of viruses created per day by the biosphere? Multiplying this by about 10^{11} days of life on earth gives the estimate.

this line of research, which if fully successful could define the use of most of the world's cycles.

Notwithstanding the growing power of computers, it is clear our resources pale compared to evolution. Nonetheless there is a ray of hope: we conjecture that biological evolution is an extremely inefficient algorithm precisely because (as discussed in §2) evolution does *not* utilize a conserved, fungible money and property rights. Perhaps we can hope to do substantially better.

9 How to Construct an Intelligence

To summarize our economic approach, we are proposing the following plan for constructing "intelligence".

1. Choose a framework giving correct incentives.
2. Choose a representation language.
3. Write an initial agent or agents.

We have argued at some length that it is critical to get the right incentives by maintaining a conserved money, enforcing property rights, and ensuring everything is owned by agents. When such a framework for agent interactions is enforced, then if the agents make local decisions rationally, i.e. maximize their money, the whole system acts rationally to maximize its money. The system may not act optimally in some global sense, but it acts as well as any of its agents can figure out. If the agents are not rational, then there is certainly no guarantee it will perform well, but agents will exploit other agents who act less "rationally", so that evolution tends to produce rational agents and thus a system that behaves well. When such a framework is not enforced, agents instead act to earn money by taking it from other agents, or compete to exploit unowned resources in ways deleterious to the system as a whole, or create money out of thin air in ways unrelated to actually solving problems in the world, and learn to exploit the bugs in the framework rather than solve problems.

We have made a proposal for point 1: auction of the world to the high bidding agent. This solves (in a trivial way) the problem of getting the whole world owned, and thus avoids the Tragedy of the Commons. We have elaborated this in allowing agents that can create agents. It is far from clear that our proposal is the only or the optimal one. For instance, we can raise many open questions.

(a) Should intellectual property be explicitly protected? (If so, how?) Otherwise, it may be that creators are not properly motivated to collaborate, since a sequence of creators act, each adding improvements, but ultimately the last creator may reap most of the benefits. This question is debated in human economics as well (Breyer 1970). A related question is:

(b) Should we enable more information and money passing between agents? We have so far only allowed passing of information in the form of code which can be read by meta-agents. For example, meta-agents can collaborate by creating new agents, each modifying the agent created by the last. Passing information between agents in other ways seems natural, but is hard to envision evolving because of the chicken and egg problem: why should one agent create information before the other agent can use it. How can the other agent learn to use it before it is created? How can they come to agree on a common language? My belief is every bit of knowledge has to be functional when it is first discovered how to compute it. This is, I believe, what happened in biological evolution as well. A meta-agent for creating some functional module can then collaborate with other meta-agents for other modules to use multiple modules within an agent.

We have only enabled passing of money between agents through selling the world. To what extent does this suffice? Should we allow more general exchange? For instance:

(c) Should it be possible to sell rights without selling the whole world? In principle, different agents could own different pieces of the world. For example, we could establish some framework for agents that buy rights to specified later payoffs. Such arbitrageurs could act as an effective intermediate reward function. If one contemplates addressing hard problems, e.g. the game of Go is a good thing to think about, one probably will want effective ways of being able to combine information from many experts. Each expert needs, however, to receive credit assignment so it learns to predict effectively. Arbitrageurs receive feedback, and sensitivity to the number of arbitrageurs currently bidding would combine a number of experts opinions. A related question is:

(d) The framework we have adopted of charging rent for time (and in some cases space) is almost certainly inaccurate, because we are essentially fixing the price of time. We have found it important to charge rent proportional to demand, but if this is correct, why? Presumably it would be more accurate to auction time, but this involves not only some computational overhead (for an efficient algorithm see Drexler and Miller (1988)) but more importantly, yet another thing the agents have to learn how to handle.

(e) More generally, there are questions regarding the evolution to rationality. Rather than demand our agents learn everything, we have made many critical decisions for them, for example, capital allocation to children, and also decisions about how parents extract money from children. The market framework provides us with a basis to make these decisions: we make them to serve the interest of the individual agent whose property is being affected, thus, e.g., capital allocation to children is made from the point of view of the parent. However, our decisions are clearly not optimal, and (have frequently been critically buggy at first because of consequences we did not foresee). In principle, these decisions should be placed under control of the property owner, and the system should then learn to make them sensibly.

(f) Should agents learn? For example, they might use a neural net to set their bid, explicitly trying to predict the value they will receive? We have so far made an explicit decision not to do this, adding new modified agents to a given collection of fixed agents. The point of this is, the new agent can adopt himself to a fixed environment. If the agents were modifying their behaviors, then they would be playing a complex game with each other, which would introduce complexity. Instead, we have fixed all the other agents' strategies, and allow the most recent agent to optimize in this context.

We have begun exploration of point 2, although we have not discussed this much here. Finding languages that are powerful enough, yet efficiently evolvable is perhaps the most critical question. We have argued that biasing in topology, and perhaps causality, are likely to be worthwhile, and have attempted to do this by accessing the world through "Look" expressions that take two integer arguments that are manipulated using increments and decrements. Typing machinery seems to help as well in biasing in structure. We have experimented with machine-like (i.e. Tierran-like) languages, and are currently experimenting with S-expressions. The interesting success of our genetic programming efforts suggest typed S-expressions may be powerful. They have substantial structure, that may cut down the search space in important ways. See Fig. 6 for a picture of S-expressions.

Lisp-like S-expressions are reasonably natural in our context, because Lisp is adept at processing Lisp code and thus seems to be a natural for reflective processing. However, Lisp is not typed, and imposing typing constraints and other kinds of useful biases in a Lisp environment while allowing powerful meta-processes is an open problem.

In our Tierran-like language, we found that increasing the power of the meta-operations with pattern based meta-instructions that would look for patterns in agents, and then insert a chunk of code, was valuable. Genetic programming just tried swapping randomly chosen subtrees from useful agents, and found this better than random mutations. Our goal is to build up useful chunks of code, useful subtrees for example, and recognize where to insert them. We hope to have meta-agents that are given credit for their success, and learn useful chunks to insert, useful places to insert them. Again, the philosophy must be that each chunk is useful when created. The meta-agent creating it is credited for the creation. It proceeds to continue trying to use this chunk, inserting it in likely places, including building up more complex meta-agents incorporating this chunk and other chunks.

If one can find usefully constrained language, and a constitution that suitably assigns credit, there is no theoretical reason why one can not solve very complex problems autonomously. It is an interesting theoretical question whether one can produce models where one could show rapid convergence. One must be able to find product structure, and at each step to find improvements with a tolerable amount of search. In the real world, we know evolution has succeeded in massive problems, using massive, but by no means exponential, resources.

In the meantime, there are ways to inject knowledge and improve performance, e.g. point 3, writing agents. Our discussion of Hayek2 was mostly based on a single simple "Root" that produced random children, and children copying other children with mutations. We have also experimented with a Root that in addition injects a hand coded agent to perform a function: cleaning. The cleaner agent bids whenever stack 1 contains an incorrect block, and if it wins the auction clears all blocks off stack 1 until there are no more incorrect blocks. In experiments that included the cleaner agent, Hayek did much better. This illustrates the possibilities of injecting knowledge into Hayek in the form of hand crafted agents. Indeed, Miller and Drexler's early proposal for economic organization (Miller and Drexler 1988a) envisaged using an economic framework simply to organize interacting hand coded programs.

In general, there are at least 4 ways we could inject knowledge into a Hayek-like system:

1. Write an intermediate reward function program.
2. Write some initial agents.
3. Add features to the language.
4. Set up a well chosen training schedule teaching useful subconcepts.

Hayek1 illustrated points 1 (with a hand coded intermediate reward function) and 3 (with the topi features). We tried training Hayek2 with a hand coded intermediate reward function, and it didn't seem to make any difference. We hope to address more complex problems, and for such problems we may wish to inject knowledge in any way we can.

Acknowledgements All the programming on Hayek1 was done by Charles Garrett and Michael Buro. Hayek2, and the work on GP was done in collaboration with Igor Durdanovic. I thank Michael Buro, Igor Durdanovic, Adam Grove, and Dale Schuurmans for conversations, and Harold Stone for suggestions on improving the presentation.

References

Anderson, P. W., Arrow, K. J. and Pines, D. (eds): 1988, *The Economy as an evolving complex system*, Addison Wesley, Redwood City, CA.

Angeline, P.: 1997, Subtree crossover: Building block engine or macromutation, *in* K. et al. (ed.), *Genetic Programming 1997: Proc. 2nd ann. Conf.*, Morgan Kaufmann, San Francisco, CA, pp. 9–17.

Bacchus, F. and Kabanza, F.: 1995, Using temporal logic to control search in planning. Unpublished document available from `http://logos.uwaterloo.ca/tlplan/tlplan.html`. A short version was presented at the European Workshop on Planning, 1995.

Banzhaf, W., Nordin, P., Keller, R. E. and Francone, F. D.: 1998, *Genetic Programming, An Introduction*, Morgan Kaufmann, San Francisco.

Barkow, J. H., Cosimidies, L. and Tooby, J.: 1992, *The Adapted Mind*, Oxford University Press, New York.

Baum, E. B.: 1996a, Did courtship drive the evolution of mind, *Behavior and Brain Sciences* **19**, 155–164.

Baum, E. B.: 1996b, Toward a model of mind as a laissez-faire economy of idiots, extended abstract, *in* L. Saitta (ed.), *Proc. 13th ICML '96*, Morgan Kaufman, San Francisco, CA, pp. 28–36. Longer version, submitted for publication, and available at `http://www.neci.nj.nec.com:80/homepages/eric/eric.html`.

Baum, E. B.: n.d. in preparation.

Baum, E. B., Boneh, D. and Garrett, C.: 1995, On genetic algorithms, *COLT '95: Proceedings of the Eighth Annual Conference on Computational Learning Theory*, Association for Computing Machinery, New York, pp. 230–239.

Baum, E. B. and Durdanovic, I.: 1998, Toward code evolution by artificial economies. Submitted for publication, and available at `http://www.neci.nj.nec.com:80/homepages/eric/eric.html`.

Baum, E. B. and Smith, W. D.: 1997, A bayesian approach to game playing, *Artificial Intelligence* **97**, 195–242.

Birk, A. and Paul, W. J.: 1994, Schemas and genetic programming, *Conference on Integration of Elementary Functions into Complex Behavior*, Bielefeld.

Blum, A. and Furst, M.: 1997, Fast planning through planning graph analysis, *Artificial Intelligence* **90**, 281–300.

Bonner, J. T.: 1980, *The Evolution of Culture in Animals*, Princeton Univ. Press.

Bouchard, T. J., Lykken, D. T., McGue, M., Segal, N. L. and Tellegen, A.: 1990, Sources of human psychological differences: the minnesota study of twins reared apart, *Science* **250**, 223–228.

Boutilier, C., Shoam, Y. and Wellman, M. P.: 1997, Economic principles of multi-agent systems, *Special Issue of Artificial Intelligence* **94**.

Breyer: 1970, The uneasy case for copyright: A study of copyright in books, photocopies, and computer programs, *84 Harvard Law Review* **281**, 292–301.

Churchland, P. and Sejnowkski, T. J.: 1988, Perspectives on cognitive neuroscience, *Science* pp. 741–745.

Clearwater, S. H. (ed.): 1996, *Market-Based Control, A Paradigm for Distributed Resource Allocation*, World Scientific, Singapore.

Coase, R. H.: 1960, The theory of social cost, *Journal of Law and Economics* **3**(1), 1–44.

Coase, R. H.: 1997, Interview in January 1997 issue of *Reason*.

Cosimidies, L. and Tooby, J.: 1992, Cognitive adaptations for social exchange, *in* J. H. Barkow, L. Cosimidies and J. Tooby (eds), *The adapted mind*, Oxford University Press, New York.

D'Amasio, A. R.: 1994, *Descartes' Error, Motion, Reason and the Human Brain*, Grosset/Putnam, New York.

Dawkins, R.: 1988, *The Blind Watchmaker*, W. W. Norton., New York.

Dawkins, R.: 1989, *The Selfish Gene*, Oxford University Press, Oxford.

Dayan, P. and Sejnowski, T. J.: 1994, Td converges with probability 1, *Machine Learning* **14**(3).

Drexler, K. E. and Miller, M. S.: 1988, Incentive engineering for computational resource management, *in* B. A. Huberman (ed.), *The Ecology of Computation*, Vol. 2 of *Studies in Computer Science and Artificial Intelligence*, North Holland, pp. 231–266.

Dreyfus, H. L.: 1972, *What computers can't do: a critique of artificial reason*, 1st edn, Harper and Row, New York.

Estlin, T. A. and Mooney, R. J.: 1996, Multi-strategy learning of search control for partial-order planning, *Proceedings of the Thirteenth National Conference on Aritificial Intelligence*, pp. 843–848.

Forrest, S.: 1985, Implementing semantic network structures using the classifier-system, *Proc. First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 188–196.

Franklin, S.: 1995, *Artificial Minds*, MIT Press, Cambridge.

Garcia, J. and Ervin, F. R.: 1968, Gustatory-visceral and telereceptor-cutaneous conditions – adaptation in internal and external milieus, *Communications in Behavioral Biology* **A**(1), 389–415.

Garcia, J. and Koelling, R.: n.d. Discussed in (Hutchinson 1994, pp22-23).

Grove, A.: n.d. Helpful discussion.

Gurvits, L., Lin, L.-J. and Hanson, S. J.: 1994, Incremental learning of evaluation functions for absorbing markov chains: New methods and theorems. Unpublished report.

Hardin, G.: 1968, The tragedy of the commons, *Science* **162**, 1243–1248.

Holland, J.: 1975, *Adaptation in natural and artificial systems*, MIT Press, Cambridge.

Holland, J. H.: 1986, Escaping brittleness: the possibilities of general purpose learning algorithms applied to parallel rule-based systems, *in* R. S. Michalski, J. G. Carbonell and T. M. Mitchell (eds), *Machine Learning*, Vol. 2, Morgan Kauffman, Los Altos, CA, pp. 593–623.

Horvitz, E. J.: 1988, Reasoning under varying and uncertain resource constraints, *Proc. 7th National Conf. on AI*, Morgan Kaufmann, San Mateo, CA, pp. 11–116.

Hutchinson, A.: 1994, *Algorithmic Learning*, Clarendon Press, Oxford.

Johnson, D., Aragon, C., McGeoch, L. and Schevon, C.: 1992, Optimization by simulated annealing: an experimental evaluation, part iii, the travelling salesman problem, *Operations Research* **40**.

Kearns, M. and Singh, S.: 1998, Near-optimal reinforcement learning in polynomial time, *Proceedings of International Conference on Machine Learning*.

Khardon, R. and Roth, D.: 1994, Learning to reason, *Proc. 12th National Conference on Artificial Intelligence*, AAAI Press/ MIT Press, pp. 682–687.

Koza, J.: 1992, *Genetic Programming*, MIT Press, Cambridge. pp459-470.

Lakoff, G. and Johnson, M.: 1980, *Metaphors we live by*, Univ. of Chicago Press.

Lang, K.: 1995, Hill climbing beats genetic search on a boolean circuit synthesis task of koza's, *The Twelfth International Conference on Machine Learning*, pp. 340–343.

Lenat, D. B.: 1983, EURISKO: a program that learns new heuristics and domain concepts, the nature of heuristics III: program design and results, *Artificial Intelligence* **21**(1,2), 61–98.

Lettau, M. and Uhlig, H.: 1995, Rule of thumb and dynamic programming. Unpublished manuscript, presented at the European Economic Association Meeting, Maastricht (1994), available at: `http://greywww.kub.nl:2080/greyfiles/center/1995/27.html`.

Lloyd, W.: 1833, *Two Lectures on the Checks to Population*, Oxford University Press, Oxford.

Luria, A. R.: 1973, *The Working Brain, an Introduction to Neuropsychology*, Basic Books, New York.

Maes, P.: 1990, How to do the right thing, *Connection Science* **1**(3).

Mead, C. A.: 1987, Neural hardware for vision, *Engineering and Science* **L**(5), 2–7.

Miller, M. S. and Drexler, K. E.: 1988a, Markets and computation: Agoric open systems, *in* B. A. Huberman (ed.), *The Ecology of Computation*, number 2 in *Studies in Computer Science and Artificial Intelligence*, North Holland, New York, pp. 133–176.

Miller, M. S. and Drexler, K. E.: 1988b, Markets and computation: Agoric open systems, *in* B. A. Huberman (ed.), *The Ecology of Computation*, number 2 in *Studies in Computer Science and Artificial Intelligence*, North Holland, New York, pp. 51–76.

Mineka, S. e. a.: 1980, ear of snakes in wild- and laboratory-reared rhesus monkeys (macaca mulatta), *Animal Learning and Behavior* **8**, 653–663.

Minsky, M.: 1986, *The Society of Mind*, Simon and Schuster, New York.

Myhrvold, N.: 1994, Roadkill on the information highway, Stanford, Calif.: University Video Communications, Distinguished Lecture Series IX; Leaders in Computer Science and electrical Engineering.

Newell, A.: 1990, *Unified Theories of Cognition*, Harvard University Press, Cambridge.

O'Reilly, U.-M. and Oppacher, F.: 1994, Program search with a hierarchical variable length representation: Genetic programming, simulated annealing, and hill climbing, *in* Y. Davidor, H. P. Schwefel and M. R. (eds), *Parallel Problem Solving from Nature-PPSNIII*.

Papadimitriou, C. H. and Steiglitz, K.: 1982, *Combinatorial Optimization, Algorithms and Complexity*, Prentice Hall, Inc., Englewood Cliffs, NJ.

Ramachandran, V. S.: 1995, Anosognosia in parietal lobe syndrome, *Consciousness and Cognition* **4**, 22–51.

Ray, T. S.: 1991, An approach to the synthesis of life, *Artificial Life II*, Vol. XI, Addison-Wesley, Redwood City, CA, pp. 371–408.

Read, L.: 1958, I, pencil: My family tree as told to leonard e. read, *Freeman* .

Ridley, M.: 1993, *The Red Queen: Sex and the evolution of human nature*, Macmillan, New York.

Ridley, M.: 1996, *The origins of virtue*, Penguin, New York.

Russell, S. and Norvig, P.: 1995, *Artificial Intelligence, A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ. cf. pp625-648.

Russell, S. and Wefald, E.: 1991, *Do the Right Thing*, MIT Press.

Selfridge, O.: 1959, Pandemonium: a paradigm for learning, *Proceedings of the Symposium on Mechanisation of Thought Process*, National Physics Laboratory.

Sutton, R.: 1988, Learning to predict by the methods of temporal differences, *Machine Learning* **3**, 9–44.

Thrun, S. and Pratt, L. (eds): 1997, *Learning to Learn*, Kluwer Academic Publishers, Boston.

Toga, A. and Mazziotta, J.: 1996, *Brain Mapping, the Methods*, Academic Press, San Diego.

Valiant, L.: 1994, *Circuits of the Mind*, Oxford University Press.

Valiant, L.: 1995, Rationality, *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, pp. 3–14.

Varian, H. P.: 1996, *Intermediate Microeconomics, A Modern Approach*, W.W. Norton, Inc.

Watkins, C.: 1989, *Learning from delayed rewards*, PhD thesis, Cambridge University.

Whitehead, S. and Ballard, D.: 1991, Learning to perceive and act, *Machine Learning* **7**(1), 45–83.

Winograd, T.: 1972, *Understanding Natural Language*, Academic Press, New York.