Machine Learning 35, 155–185 (1999)

Toward a Model of Intelligence as an Economy of Agents

ERIC B. BAUM eric@research.nj.nec.com
NEC Research Institute, 4 Independence Way, Princeton, NJ 08540

Editor: Sridhar Mahadevan

Abstract. A market-based algorithm is presented which autonomously apportions complex tasks to multiple cooperating agents giving each agent the motivation of improving performance of the whole system. A specific model, called "The Hayek Machine" is proposed and tested on a simulated Blocks World (BW) planning problem. Hayek learns to solve more complex BW problems than any previous learning algorithm. Given intermediate reward and simple features, it has learned to efficiently solve arbitrary BW problems. The Hayek Machine can also be seen as a model of evolutionary economics.

Keywords: reinforcement learning, multi-agent systems, planning, evolutionary economics, tragedy of the commons, classifier systems, agoric systems, autonomous programming, cognition, artificial intelligence, Hayek, complex adaptive systems, temporal difference learning, evolutionary computation, economic models of mind, economic models of computation, Blocks World, reasoning, learning, computational learning theory, learning to reason, meta-reasoning

1. Introduction

There is a growing consensus that one should attempt to understand the mind as arising from the interaction of many modules or agents which are much simpler than the whole, yet complex compared to individual neurons. This view is separately expressed by workers in a wide variety of disciplines, each discipline offering independent empirical evidence, cf. the literatures of evolutionary psychology (Cosimides & Tooby, 1992), cognitive psychology (Newell, 1990), brain imaging (Toga & Mazziotta, 1996; Fox, 1997), philosophy (Dennett, 1991), neuroanatomy (Luria, 1973) and elsewhere. Much work within the computer science literature focusses on a constructivist approach to intelligence. Divide and conquer is a natural approach for complex computational tasks. Here too one finds a growing consensus that high level intelligence must somehow originate in the interaction of many simpler agents cf. (Selfridge, 1959; Lenat, 1983; Holland, 1986; Minsky, 1986; Maes, 1990; Newell, 1990; Drescher, 1991; Valiant, 1994, 1995).

Arguably the critical problem in producing intelligence as a multiagent system is attributing credit when several agents together collaborate on a project. As Minsky wrote already in 1961 (Minsky, 1961) "suppose that one million tasks are involved in a complex task (such as winning a chess game). Could we assign to each decision one-millionth of the credit for the completed task?... For more complex problems, with decisions in hierarchies... the

running times would become fantastic." Backpropagation (Rumelhart, Hinton, & Williams, 1986) addresses this problem at the level of the individual neuron, but after 15 years of study it does not seem likely to provide the whole answer. For example, it is not evident how backpropagation can apply to interactions of modules that are complex compared to individual neurons, or for tasks involving reasoning or planning.

In this paper I take the point of view that a multiagent system must learn how to reward its agents as it learns to perform tasks. This picture is similar to that studied in the Reinforcement Learning community (Sutton & Barto, 1998) in that one learns policy and value simultaneously, except that I am proposing that the rewards themselves be computed by agents, possibly in collaboration. I define an *economy of agents*[1] to be any set of agents that pass around rewards and modify their behavior in response to these rewards. It is proposed to seek an appropriate constitution for an economy of agents that promotes the evolution of intelligence. Many previous models can be viewed as economies of agents, including prominently Eurisko (Lenat, 1983) and Classifier Systems (Holland, 1986). A new definition is useful if it leads to insight. We discuss below economic insights into assignment of credit in Eurisko, Classifier Systems, and a new model presented here, and pose several open questions.

To my knowledge, Holland was the first author to use economic terminology and insight in a multi-agent learning machine. His Classifier Systems work attracted considerable interest, but has not succeeded in rewarding long chains of agents to collaborate (Wilson & Goldberg, 1998). This suggests that the assignment of credit for agents collaborating in sequence is flawed. I discuss several possible explanations in Section 5.1. First, a "Tragedy of the Commons" (Hardin, 1968) can arise when rewards are shared as in Classifier systems—if each active agent shares in the payoff then the incentive for all agents, including those agents who hurt performance, is to be active when payoff is coming. Second, reward (money) can enter the system that is not earned from the world. It is much easier for the system to learn to cheat than to solve the hard problems we are posing it. Classifier-like systems also are seen to have substantially less representational power than naively apparent, because many useful sets of classifiers are dynamically unstable.

Miller and Drexler (1988a, 1988b) have proposed applying economic ideas to create a complex programming environment for distributed computing, and also commented briefly on the possibility of producing intelligence as an economy of agents. They identified the cheating problem mentioned above, and proposed that enforcing local conservation of "money" would solve it. In particular they noted that without such a conservation law, positive feedback loops would corrupt assignment of credit, and remarked that such corruption plagued Lenat's multi-agent intelligence Eurisko (Miller & Drexler, 1988b).

This paper describes an economy of agents I call[2] "The Hayek Machine". Hayek interacts with an external world, modelled here as a complex system where Hayek may take actions, and which will make payoffs to Hayek in response to appropriate series of actions. Our experiments to date have tested Hayek on a particular benchmark "world", "Blocks World", described below. The economy learns from reinforcement to solve Blocks World problems.

Hayek consists of a collection of agents, initialized at random. Each agent is a function which maps states of the world (including the internal state of Hayek) into a bid, action pair. Hayek's computation proceeds in steps as follows. At each step only the agent with

high bid is active.[3] It takes its action, pays its bid to the previously active agent, and receives any reward paid by the world on that step. As agents pay and are paid their wealth fluctuates. Bankrupt agents, i.e., those with negative wealth, are removed, and new agents are created by a randomized process.

What motivates Hayek's design? Hayek's access to the world is a valuable resource because correct actions will create wealth. But he doesn't know how to utilize this resource. So he sells it to the high bidder among a bunch of agents. The winning bidder buys the world as property—that is, the winning bidder has the right to take action changing his property, the right to sell it, and the right to collect any rewards from the world while he owns it. Because the agent owns the resource, it is motivated to maximize its value. The agent whose action adds the most value can afford to outbid all others. Conversely, competitive bidding forces agents to bid close to the expected value of the world after their action. When bids estimate value, choosing the agent with high bid is rational.

Thus Hayek divides complex problems down to a set of subproblems, and does so in a way that gives to each agent the incentive of improving wealth production by the system as a whole. Note that this was our goal: to create a framework apportioning credit among our agents so that the agents are motivated to increase the value of our access to the world. With the exception of important caveats described below and in Section 3.3, a new agent can enter Hayek's economy if and only if its entry increases Hayek's overall wealth production. That is: an agent that on average takes the world to states of higher value will be profitable and can enter. An agent that on average takes the world to states of lower value will go bankrupt and be removed. Thus Hayek executes a randomized hill climbing search in "collection of agent space". Each update of Hayek, that is each entry or death of an agent, improves overall performance, on average (modulo caveats).

The main caveat stems from the fact that Hayek is bootstrapping its evaluation of states. New agents may be able to enter to exploit misvaluations even if their entry degrades the performance of the system. The phenomenon is analagous to "cherrypicking" in insurance markets. However, the entry of such agents advances Hayek's learning by improving the valuation of states.

In this paper Hayek's agents are restricted to be productions with a single fixed bid and action. We will call the version experimented on in this paper Hayek1. (More complex agents are experimented with in (Baum & Durdanovic, 1998a, 1998b). Thus for this paper each agent has a triplet: condition, action, bid. The condition is a Boolean conjunction. If the condition is true, the agent makes its bid, otherwise it bids zero. Thus each agent is a specialist.

Hayek1 as experimented with here derives from Holland Classifier systems the general notion of condition action bid agents with an agent paying its bid to a preceding agent in what Holland called a "bucket brigade". It differs in selling access to the world to one agent, thus establishing clear title to assignment of credit and avoiding any Tragedy of the Commons, in imposing conservation of money, and in its agent representation language. Hayek1 is compared to the classifier system literature in Section 5.1 and to the reinforcement learning literature in Section 5.2.

In the "Blocks World" (BW) problem, Hayek1 sees a series of instances. Each instance consists of 4 stacks of blocks, each block one of three colors. The first stack is called the goal and the second the target. Hayek1 controls a hand which if empty may grab the top

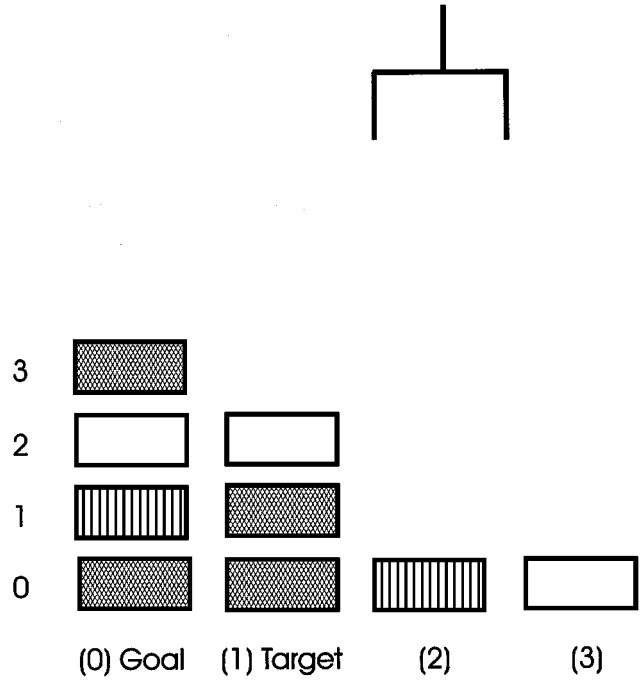

Figure 1. An instance on which the agent of (Eq.) 1 would bid. $*y = 1$ makes both conditions true, since the second block in the goal and target stacks are of different colors. (See Section 3).

block off any stack but the goal, and if holding a block may drop it on top of any stack but the goal. If Hayek1 succeeds in copying the goal stack in the target position within 100 actions he is rewarded. Otherwise he sees a new random instance. A picture of a BW problem is shown in figure 1.

Blocks World has been a benchmark domain since the '60's (Winograd, 1972) and remains widely studied today, cf. (Bacchus & Kabanza, 1995; Estlin & Mooney, 1996; Dzeroski, Blockeel, & DeRaedt, 1998). It can be treated as a planning problem, where the solver is told the objective, but must construct a plan for solution. Alternatively it can be a learning and planning problem, as here, where the learner only receives reinforcement and must discover the goal. Our BW problem is challenging for three reasons. First, the state space is combinatorially huge, which raises the "curse of dimensionality" and hence frustrates standard learning algorithms. Second the goal is abstract and highly non-linear—correctly stacking the third through tenth blocks on top of an incorrect second one gets you nowhere. Third, the limited table size creates a "Tower of Hanoi"-like aspect in that long sequences of actions may be necessary for solution, and one must often stack on top of blocks that will soon be needed. Note that Blocks World has a compact algorithmic description, yet includes instances of arbitrary size.

Recent experiments have applied several sophisticated general purpose AI planning algorithms to Blocks World problems, finding that they could solve problems involving only

small, roughly single digit, numbers of blocks, unless given hand crafted knowledge specific to Blocks World (Bacchus & Kabanza, 1995). With such domain knowledge, problems involving a few tens of blocks can be solved. The Blocks World problems in these experiments were similar but not identical to those experimented with in this paper.

No author that I'm aware of has addressed the abstract goal discovery problem in Blocks World, although a number of authors have been interested in learning in Blocks World contexts. Whitehead (1991) studied how Temporal Difference algorithms could be modified to deal with large state-spaces. Unfortunately, in their experiments their algorithm learned only to pick up a single green block from 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 papers addressed BW versions involving neither discovery of an abstract goal, nor the necessity of stacking blocks on top of later needed blocks.

After four days of training on a 150 MHz MIPS R4400, Hayek1 learned to solve BW problems involving ten total blocks (thus with up to five on the goal stack) if started from *tabula rasa*. Given simple features and intermediate reinforcement Hayek1 has learned to solve arbitrary instances,[4] and moreover to use within a constant factor as few actions as possible on all but exponentially rare instances. (Note that to solve large instances requires more than 100 actions.) While Hayek1 is slow to learn, once trained it solves novel instances instantly, by applying successively a series of agents. By comparison, hand coded planners do an extensive, time-consuming search in addressing a given Blocks World instance.

Above I motivated this approach as an attempt to solve the assignment of credit problem in multi-agent systems so as to be able to divide and conquer hard problems. From this perspective, the most promising observation may be that Hayek1 has here succeeded in producing sets of agents that collaborate successfully, engaging in long, indeed potentially infinite, series of actions. The universal solver set, for example, consists of a set of 6 agents. Our random agent generator would take of order 10^7 tries to generate the most complex agent alone, and of order 10^4 tries for each of the 5 others. Thus, to exhaustively search for such a set of agents as a set would be combinatorially prohibitive.

This optimism should be qualified for several reasons. The scheme used here was on the one hand aided, but on the other hand limited, by its representation. The representation implicitly encodes the topology of the problem, as discussed further in Section 3. This rendered the search more tractable, and is a feature that can perhaps extend to other domains. The representation was, however, sufficiently limited that it could not describe a general solution until aided with both an added feature (top of stack) and an intermediate reward function. The intermediate reward function expanded the *representational* capabilities by stabilizing some sets of agents. A natural way to encode algorithms in a representation like here or Holland's consisting of productions and bids, is to prioritize the productions by using the bids. However, without intermediate rewards, stable agent sets must on average have agents with higher bid acting *after* agents with lower bid, else the higher bid agents will go broke. Because the intermediate reward guided the learning as well as increasing the representation power, an important open question is whether a similar economic system

with more powerful representation could solve general BW problems without intermediate reward.

The constitution used here is limited in other ways. It is impossible for an agent to represent negative knowledge such as: "under condition X, don't take action Y". Agents can only take actions, there is no possibility of agents simply communicating the results of computations to other agents. In general, no claim is asserted that the constitution of Hayek1's economy is optimal. Devising other, and perhaps better, constitutions is posed as an open problem. Ongoing experiments with generalizations to constitutions allowing more complex agents are reported on elsewhere (Baum and Durdanovic, 1998a, 1998b).

Section 2 describes the Blocks World environment in more detail. Section 3 gives more details on Hayek1 including four subsections: Section 3.1 describes how new agents are generated; Section 3.2 describes how bid values are generated; Section 3.3 analyzes convergence properties; and Section 3.4 gives some statistical analysis of Hayek1 and the Blocks World. Section 4 describes the empirical results on BW, including four subsections: Section 4.1 describes Hayek1's results when given a simple feature; Section 4.2 describes Hayek1's results from *tabula rasa*; Section 4.3 compares a simple strawman; and Section 4.4 assesses these experimental results. Section 5 discusses related work including three subsections: Section 5.1 discusses Hayek1 in the context of previous Holland Classifier systems, and makes some conjectures about problems in some versions of these; Section 5.2 discusses Hayek1 in the context of Temporal Difference/Real Time Dynamic Programming algorithms, and argues that Hayek1 has some advantages; and Section 5.3 discusses previous work on economic models of computing. Section 6 suggests that The Hayek Machine can be viewed from a very different perspective as a model of economics. Section 7 concludes.

2. Blocks World

The physical Blocks World domain described in the introduction is encoded as a set of vectors of form $(c, a_0, a_1, a_2, \ldots, a_{n-1})$. $c \in \{0, 1, 2, 3\}$ denotes the stack number. 0 is the goal stack and 1 the target. $a_i \in \{0, \ldots, 3\}$ denotes the color of the block at height i. $a_i = 0$ means there is no block at that height. If $a_i = 0$, then $a_j = 0$ for $j > i$. Hayek1's actions are written (i, a) where $i \in \{1, 2, 3\}$ and $a \in \{g, d\}$. Action (i, g) (respectively (i, d)) moves the hand to column i and grabs (respectively drops) the top block. If holding a block already, this is invalid.

Hayek1 is given a series of 'instances'- each a random set of 4 stacks of blocks. If it reproduces the 0 stack in the 1 position, it is rewarded and goes onto the next instance. If it makes 100 actions without such a solution, it proceeds to the next instance without reward.

We[5] have experimented with three reward schemes. In scheme one Hayek1 is rewarded for partial success. Agents removing the top block from an incorrect partial stack (in position 1) or placing a correct next block on a correct partial stack are given reward 1. Agents removing a correct block or placing an incorrect block on stack 1 or placing any block on an incorrect partial stack 1, are penalized 1. An agent who solves an instance is given a reward of 10.

In scheme two agents are only rewarded for successfully solving the instance. Hayek1 was trained on successively larger instances. At stage i, instances had $\lfloor \frac{i+2}{2} \rfloor$ blocks distributed

over the target and goal stacks and $\lceil \frac{i+2}{2} \rceil$ blocks distributed over stacks 2 and 3. Hayek1 sees instances from a given stage until it has learned to solve 95% of them and then moves onto the next stage.

In scheme three Hayek1 is "taught" both by intermediate reward and by successively larger instances.[6]

At least two considerations motivated study of intermediate rewards. In coming decades computers may be much faster and programs more complex, and hence less robust. My hope is that one might train, rather than program, one's computer by guiding it with an intermediate reward function. Also, in my view, intermediate rewards are a feature of human learning, which I seek to understand.

3. Hayek1

This section describes our economy of agents. We first discuss how and when agents act and pay money, then the encoding of agents, then the creation of agents, then the assignment of bids to agents, then make remarks on the evolution of the economy.

Hayek1 consists of a collection of agents. Each agent has a condition, a numerical bid, and an action. Computation proceeds in steps. At each step the validity of all agents' conditions is evaluated. If the condition, a Boolean formula, is true, and the action is legal,[7] the agent bids. At each step, the single agent bidding highest is "active". Its action is taken, it pays its bid to the previous active agent,[8] and it receives both any payoff from the world on its turn, and pay from the next active agent. The initial wealth of every agent when it is first created is zero. Each agent, active or not, was charged a small "rent" after every fifth BW instance in order to remove agents that are never active and keep the population small enough for Hayek1 to run fast. Except for this rent, an agent's wealth changes only when it is active, i.e., wins the auction. Any agent ending any instance with negative wealth is removed.

The agents have form

$$A_1 \wedge A_2 \wedge \cdots \wedge A_n \Rightarrow (a, b) \quad \text{Bid} = N$$

Here the condition of the agent is $A_1 \wedge A_2 \wedge \cdots \wedge A_n$ and its action is (a, b). The condition is thus the conjunction of the clauses A_i. The A_i in turn are expressions of form $u(op)v$ where (op) is either $=$ or $\neq$ and u and v, which I denote "atoms" may be either E (meaning empty), or H (the current content of the hand) or a grid location of form (x, y) where $x \in \{*x, 0, 1, 2, 3\}$ is a stack and $y \in \{*y, 1, 2, \ldots, h\}$ is a height. $*x$ and $*y$ here are existentially quantified variables, analagous to Holland's wildcards (Holland, 1986). The condition is said to be valid, if there is any assignment of $\{0, 1, 2, 3\}$ to $*x$ and $\{0, 1, 2, \ldots, \}$ to $*y$ rendering the Boolean condition true. The system searches for assignments in a randomized order, so if there are many such assignments, one is selected randomly. N is some fixed number associated with the agent. The action (a, b) moves the hand to column a and performs action b. Here b takes values in $\{g, d\}$ (i.e., grab or drop), and a may be any number 0–3 or $*x$. A grab action is legal if and only if the hand is empty and there is a block in the stack b from which it grabs; a drop action is legal if and only the

hand is holding a block. The agent will bid N if and only if its condition is valid and its action is legal.

For example, we might have an agent

$$(0, *y) \neq (1, *y) \wedge (1, *y) \neq E \Rightarrow (1, g) \quad \text{Bid} = 8.3 \tag{1}$$

The condition of this agent has two clauses, the first being $(0, *y) \neq (1, *y)$. This clause is true if there is some height $*y$ such that the block at height $*y$ in the 0 (or goal) stack is not the same color as the block at height $*y$ in the 1 stack, or where neither stack is as tall as $*y$, so both positions are equal by virtue of both being equal to Empty. The second clause is $(1, *y) \neq E$ which is true if this height $*y$ can be further restricted so that the 1 stack in fact has a block at this height. Thus both clauses are true exactly when there is some block in stack 1 of different color than the position next to it in stack 0. In the example of figure 1 this is fulfilled for $*y = 1$. If there is such a $*y$ satisfying both clauses, this agent will bid 8.3 if its hand is empty. If 8.3 is higher than the bid of any other agent, so that it wins the auction, it will move the hand to position 1 and grab, thus clearing a block from stack 1. Note in passing that this is potentially a very useful rule: a "universal cleaner" that grabs from stack 1 whenever we need to clear blocks off stack 1 to reach our goal.

Note that our agent encoding departs from *tabula rasa* through its use of Cartesian coordinates. The use of existential quantifiers over either stack values, or block height, but not some arbitrary mixture, implicitly encodes topology. I am interested in minimizing encoded domain specific knowledge in hopes of the Hayek Machine robustly solving a wide variety of problems. However topology seems broadly useful.

3.1. Agent creation

Hayek1 creates its agents by a random process. This random process was specified as follows with the goal of sampling roughly uniformly in a natural way from the set of possible clauses. Half the rules are created from scratch and half are mutations of previous rules.

When a rule is created from scratch, it is given $i > 0$ clauses in its condition with probability $p_i = 1/2^i$. As discussed in the previous section, each clause is of form $u(op)v$ with (op) either $=$ or $\neq$. We chose the operator to be $=$ with probability $P_=$, and set $P_= = 1/2$. The lefthand side u of a clause is set to E(mpty) with probability $P_E = .2$, to H(and) with probability $P_H = .2$, else it is of form (a, b) where a and b are specified as follows. a is $*x$ with probability P_*, else it is the index of a stack chosen uniformly from $\{0, \ldots, 3\}$. b is $*y$ with probability P_*, else it is a height value chosen uniformly from $\{0, \ldots, \text{numblocks} - 1\}$, where numblocks is the total number of blocks in the system. The only parameter we experimented with was P_*, for which we tested two values of P_*: $P_* = 0.1$ and also $P_* = 0.5$, which were about equally effective.

The action is a pair (a, b) where a is $*x$ with probability $P_{*x} = .2$, is a random number generator with prob $P_{\text{Random}} = .2$, else it is the index of a stack chosen uniformly in $\{1, \ldots, 3\}$. b is the grab action with probability $P_{\text{grab}} = 1/2$ and the drop action otherwise.

A mutation consists of the insertion or deletion of a random clause, or the random replacement of an atom. A random agent in the population is chosen. A random clause

in its condition is chosen. With probability 1/4 the clause is removed, with probability 1/2 one of its atoms is instead replaced with a atom drawn from the same distribution as when producing new random agents, and with the remaining probability 1/4 no clause is removed but a new random clause is added.

Note that the above parameters were chosen to sample roughly uniformly from the set of possible clauses, with no parameter tuning whatever, except on P_* as mentioned. We also experimented with a naive meta-learning scheme, in which all parameters were themselves learned from *tabula rasa*. Neither the choice of P_* nor the replacement of parameters by metalearning noticeably impacted Hayek1's performance. Ongoing work on more sophisticated meta-learning is reported elsewhere (cf. Baum & Durdanovic, 1998a, 1998b). We mention these naive meta-learning experiments here only as further evidence that our results were not generated by sophisticated choice of parameters in agent creation.

3.2. Bid assignment

Human contractors and purchasers think hard about when and how much to bid. Hayek's agents must decide on bids autonomously. The simplest approach is for each agent to be assigned a fixed, random bid. This strategy learns the bid exactly the same way as the condition and action: agents with appropriate bids survive. A slight variation on this approach was used to avoid having to determine an appropriate range for the random bid generation. Agents when created were called novices. The first time a novice agent's condition was true, its bid was fixed at ϵ more than the high bid from applicable veteran rules[9] and it became a veteran.

Mutations create new versions of successful agents. Often these are similar, or even tautologically equivalent to their parents. But now the mutated rule has a higher bid and displaces its parent. Thus bids rise till further mutated agents are no longer profitable. That is, competition drives an agent's bid to the expected value of the states its action reaches. This is analagous to entrepreneurs copying a successful business, and undercutting his prices till the profit margin is near zero.

When new agents bid, they promise to pay the preceding agent, and yet they enter with zero money. This requires that they be temporarily extended credit. In real economies, entities extending credit take considerable care to assess credit worthiness. In Hayek1, I ran a perfect credit verification, in which agents extending credit were allowed to look into the future and see first that they would be paid. This was done by backing up the computation whenever any agent ended an instance with negative wealth. That agent was removed, the remaining agents' wealths were reset to where they were, and we ran the instance again. This is computationally simple—requiring at most one backup per agent, since each agent can die but once. In experiments without this credit verification, new agents injected fictitious money into the system causing inflation, and speculative bubbles arose where agents bid ridiculous sums in the confidence that a new novice would bid more. No such run ever progressed beyond stage 3. Fleecing novice agents became the main goal of the system, rather than generating wealth in the world.

We experimented with other means of bid selection. As will be discussed more in the next subsection, we want the bid to converge to the expected value of using the agent, given

that it wins the auction. A direct approach is to learn this value by Real Time Dynamic Programming (RTDP) or Temporal Difference type (TD) learning (Sutton, 1988). Let b be the bid of an active agent, b' the bid of the following agent, r any reward on that particular step, and $\delta \in (0, 1)$ a small parameter, and update by $b \rightarrow b(1-\delta) + (b'+r)\delta$. b' here is an estimator of the value of being in the subsequent state. $b'+r$ thus estimates the value of being in the current state and using the given agent. b averages such estimates over a timescale determined by δ. Our experiments with this method were done before we instituted the backup, described above, and so this method did not work well, but rather generated inflationary speculative bubbles on small instances, never progressing beyond stage 2.

Holland (1986) set bids in his classifier systems by $b = \alpha s W$ for α a small constant, where s is the "specificity" of a classifier, and W its wealth. His motivation was to set the bid (a dependent variable) so as to prefer wealthier and more specific rules. However I view this rule as setting the wealth in terms of the bid. If the expected pay-in exceeds the bid, the rule is profitable and hence its wealth rises, and with it the bid, until the bid equals the expected pay-in. Thus I ignored specificity as irrelevant in determining the limit bid.

Setting bids by $b = \alpha W$ requires initiating the wealths of new agents. If these wealths were non-zero, we found speculative bubbles, inflation, and all the problems discussed above. Instead, I merged the Holland rule with the fixed bid scheme in a three stage process, as follows. When first created, agents are novices. Novice agents' bids are fixed when first active, exactly as in the fixed bid scheme, except that the agent is then dubbed an "apprentice". When an apprentice accumulates sufficient capital to justify its bid, i.e., when first $W \geq b/\alpha$, it is designated a "veteran" and subsequently its bid b varies as $b = \alpha W$. This allows flexibility in adjusting to market conditions. In practice, becoming a veteran is a lengthy process and at any typical time most agents were apprentices, and thus fixed bid. I call this scheme "Modified Holland".

3.3. Convergence

We described in Section 3.2 evolutionary pressures pushing agents' bids to estimate the expected value of the states they lead to. Two desirable things occur when the agents' bids estimate this accurately. First, the agent with high bid is then the one most likely to advance solution. Thus the system plans "rationally", at any time choosing to apply next the agent most likely to be useful. Second, new agents can then enter the economy precisely if and only if they improve overall performance of the system. To enter, a new agent must profitably bid more than its competitors, which means (under the assumption that the other agents' bids accurately estimate the value of states) that it must lead to a more valuable state. Conversely, any new agent which leads to higher value states can afford to outbid its competitors, because it will be paid an accurate estimate of the higher value of the state it leads to. Thus we expect that each profitable entry or bankruptcy will improve Hayek's performance, but there are important caveats. These stem from the fact that in practice an agent's bid estimates the average over all instances[10] where the agent applies, and hence may be inaccurate on any given instance.

First, the system is stochastic. In any given instance, the bid is a noisy estimate. Some agents may go bankrupt, or enter for a time, just due to runs of luck. Note that, since

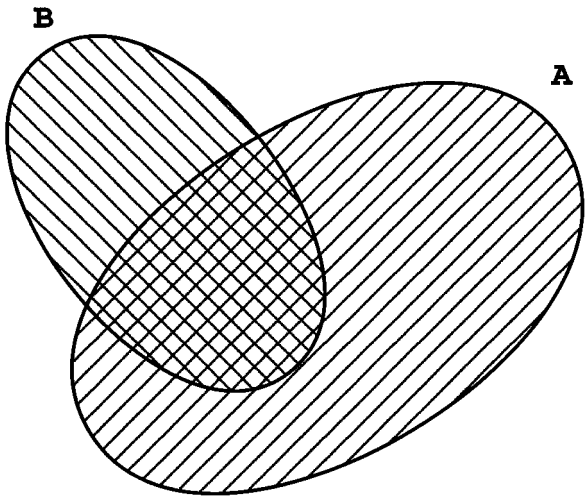

Figure 2. A Venn diagram in instance space. Region A shows the instances where *A* won the auction. Agent *B* enters, now capturing the auction on instances in the intersection.

agents' bids are pushed close to their expected pay-in by competition, they are typically only marginally profitable and are thus particularly subject to such bad runs.

Second, there is the possibility of "cherrypicking" (see figure 2). Say we have a profitable agent *A*, and an agent *B* is introduced into the system whose domain includes the easily solved instances of *A*, but not the harder ones. Then *B* may cherrypick *A*'s best instances, so that *A* can no longer profit at its old bid. *A* might even be better than *B* at handling these best instances, but be unable to capture them since its bid reflects its handling as well of harder instances. The possibility of cherrypicking is inherent if you have poor price discrimination, i.e., if Hayek is not accurately evaluating states. Our agents, being simple rules, are inflexible in assigning bids to configurations. However, cherrypicking events tend to make the price assignments more accurate, which is a valuable learning step in itself. *B* prices its subset of *A*'s client states more accurately than *A* did. If *A*'s bid can adjust (e.g., if *A* is a "veteran" rule in the modified Holland pricing), then *A* may readjust its bid to more accurately reflect the value of its remaining clients than it did before. If a certain region of configuration space represent easily solved instances there may be a reason for it. Discovery of this region may be a step to learning a better solution.

It is, at least in principle, important that prices be able to adjust downward. If *A*'s bid can not readily adjust, it may go broke. If other agents have learned to expect to be followed by *A*, they may also go broke and we have the possibility of a crash. Typically in our systems there are several similar agents with slightly different bids. If one goes broke, the next starts winning auctions, providing graceful price decreases.

In our experiments Hayek1 applies knowledge gained in small instances to learn to solve large ones. This would not occur if intermediate market crashes caused loss of memory. Hayek1's success indicates that its economy achieves considerable stability.

We also experimented with specific hill climbing attachments to prevent crashes. We maintained in storage the best set of agents yet found, as measured by a moving average of

solution efficiency. Whenever a new best set[11] was found, the stored best set was updated. If the system ran for N steps without updating its best set, then the system was reset to the stored best set. Since the instances and the agent creation process are randomized, this resulted in a second search from this set in an effort to improve it. Each time we returned to a stored set, N was increased to allow a longer search, and when a new best set was found, N was reset to a fixed search value. Interestingly, Hayek1 with such forced hill climbing was far inferior empirically to Hayek1 without it. These results seem to indicate that Hayek1 is making very long excursions of marginal agent changes in between making discoveries allowing significant gains. These results also provide further evidence that Hayek1's economy is not much troubled by market crashes.

3.4. Statistical speculations

In this section we make some rough estimates of how likely Hayek1 is to find interesting agents. Consider the sensible rule: if the bottom block in the target stack is different from the bottom block in the goal stack, remove the top block in the target stack. Hayek1 writes this rule as:

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

The probability of generating this rule from scratch in our random rule generation can be seen to be about 3×10^{-5} if $P_* = 0.1$, or about 2×10^{-6} if $P_* = 0.5$. (Recall from Section 3.1 that P_* determines the probability of wildcards in random rule creation.) Many other rules usefully grabbing from stack 1 could also arise. Any rule that grabs from 1 but doesn't use $*y$ could be modified by adding a clause $(1, *y) \neq e$ without hurting its performance. This clause is generated with probability about $10^{-2} P_*$, so maybe 1 out of every few hundred grabbing rules might contain this clause. If a grabbing rule contains this clause, it could acquire a clause $(0, *y) \neq (1, *y)$ Such a clause is generated with probability very roughly about $10^{-2} P_*^2$. This rule now looks the universal cleaner rule of Eq. (1) discussed in Section 3:

$$(0, *y) \neq (1, *y) \wedge (1, *y) \neq e \Rightarrow (1, g) \tag{3}$$

Hayek1 could also create this rule from scratch, rather than by mutation, with probability roughly 2×10^{-8} if $P_* = .1$ and 4×10^{-7} if $P_* = .5$. One gets similar interesting estimates for producing useful rules for placing correct blocks on target.

Such exercises yield several insights. First, we can expect to sort through large, but potentially tractable, numbers of rules for each useful one found.

Hayek1's price mechanism speeds this search. Each time a novice rule is applied, Hayek1 immediately estimates whether it helps or hurts, using all the knowledge available in the system-expressed as the market price. Contrast, e.g., the alternate Brand X approach defined as follows: we reward a rule if the actual final result of an instance it is used in is successful, rather than using an estimate of the next state's value. Say we have learned pretty well, and are now solving 95% of simple instances, hoping soon to progress to more complex

instances. Say a new rule is introduced which simply wastes an action. This will worsen performance, but only slightly. Say performance degrades to 90%. Brand X will reward this rule 10 times for every time it is penalized. Brand X must apply this rule 10–100 times before it can decide to remove it. Such rules will enter Brand X in droves, and strictly limit its performance. This is a manifestation of Tragedy of the Commons, when rules simply share in the final reward.

Note that rules like those of Eqs. (2) or (3) might be useful even before Hayek1 has learned much. If Hayek1 simply took random actions, it would solve some small instances. If it used one of these rules in addition to random actions, it would solve a higher fraction of small instances.

Hayek1's local search on collections of agents is a big win over, say, random search. Producing even the one universal clearing rule above (e.g., Eq. (3)) is much harder if it must be assembled intact, rather than by mutation of smaller useful components. Discovering a set of rules that work together would be combinatorically improbable if such sets were not built of useful agents discovered separately.

Better tuned parameter settings in principle could impact greatly the probabilities of finding useful rules. If $P_* = 0.1$ Hayek1 has a much easier time producing Eq. (2), but a much harder time generalizing it to Eq. (3). A mixture of different creation types, some with $P_* = 0.1$, some with $P_* = 0.5$ might be significantly faster. The experiments reported in this paper, however, do not use any combination of rule creation procedures, nor did we optimize parameters.

4. Experiments

4.1. Explicit variables

The language for representing agents, as specified in Section 3.1, is not sufficiently flexible to allow description of a general solution to arbitrary BW problems. Consider for illustration the rule: "if holding the next needed block, place on target". This could be written

$$(0, *y) = H \wedge (1, *y) = E \wedge (1, *y - 1) \neq E \Rightarrow (1, d)$$

But there is no notion of "$*y - 1$", so Hayek1 can not express this (cf. Baum & Durdanovic, 1989a, 1998b, for more powerful languages).

In the meantime, I introduced three terms: top[1], top[2], top[3]. top[i] is the height at which the next block will go, if placed on stack i. We modified the rule creation process so that whenever a new random condition is created, wherever a term of "type" height might appear, top[i] occurs with probability $(1/3)P_{\text{top}}$.

We set[12] $P_* = .25$ and $P_{\text{top}} = .25$, and ran with incremental rewards as well as staged learning. The experiments reported all used fixed bid agents.

In our first run, we ran Hayek1 for over a week on a MIPS R4400 CPU at 150 MHZ with a rent of 0.5. Hayek1 then discovered a set of agents that solves arbitrary BW instances (so long as they have more than 7 blocks in the goal) given enough actions. This solution is also efficient, i.e., the rule set will solve almost all instances using within a constant

factor as few actions as possible. (Pathological instances which occur exponentially rarely can require quadratically more actions than optimal.) Here is the first rule set it found (omitting a few dozen rules that are strictly superceded by more general rules with higher bids and thus never win auctions, and in steady state would eventually be removed by the rent):

1. Bid = 14.4166 $(0, *y) \neq (1, *y) \wedge (1, *y) \neq H \wedge E = E \Rightarrow (1, g)$
2. Bid = 14.0712 $(1, *y) \neq E \Rightarrow (3, d)$
3. Bid = 13.3013 $(1, *y) \neq (2, *y) \Rightarrow (*x, d)$
4. Bid = 13.2446 $(0, \text{top}[1]) = (3, *y) \Rightarrow (3, g)$
5. Bid = 13.2242 $(1, 15) = (*x, *y) \Rightarrow (2, g)$
6. Bid = 12.1369 $(3, *y) \neq (2, 4) \Rightarrow (*x, d)$

Agents 1, 4, and 5 are grabbing rules. One of them is active when the hand is empty. Agent 1 is a universal clearing rule: whenever a bad block is anywhere on the target, it lifts the top block off target. (It is identical to the rule of Eq. (2), because HAND = EMPTY for a grabbing rule to apply.) Agent 4 grabs from stack 3 if stack 3 contains the next useful block, else agent 5 grabs from stack 2.

Agents 2, 3, and 6 place blocks. 2, with higher bid, is active whenever the target is taller than the goal. It places blocks on stack 3. Otherwise, one of agents 3 and 6 drop on a random stack. Note that if stack 2 has as many as 4 blocks, the condition of rule 6 is valid because for some height, stack 3 is empty. If stack 2 has fewer than 4 blocks, and stack 3 has any blocks, rule 6 still bids. Likewise rule 3's condition is valid unless stacks 1 and 2 are identical. Therefore one of rules 3 and 6 will bid for any instance of more than six goal blocks whenever the hand is empty. Note, however, that for instances of less than seven blocks all three dropping rules can fail to be valid, in which case this rule set fails. Hayek1 was trained on increasing stages, and forgot how to solve some small instances.

This program works as follows. Agent 1 keeps bad blocks off the target. This is the highest priority. So assume there are no bad blocks on the target. Agents 4 and 5 dig for useful blocks. Because of agent 4's higher bid, it digs from stack 3 if a useful block is found there, else 5 digs from stack 2. Thus it always digs on a stack where a useful block may be found. Any useful blocks picked up are dropped on the target with probability one third, and no correctly placed blocks are ever removed from the target. Thus it makes steady progress and solves the instance.

There are a few reasons why this set of rules is less than optimally efficient. Random dropping causes inefficiency by a small constant factor. Say it has picked up a block from stack 3 in hopes of getting at a block beneath. With a third of a chance it will replace it on 3, and with probability a third it will it place on 1. In either of these cases it will again pick up the same block. Each time it picks up a block, it drops it on the correct stack with probability one third. This random retrying to get the block where it wants it costs it slightly more than a factor of 3 in efficiency, against simply dropping it correctly. Always picking from stack 3 if there is a good block there is also inefficient—a block of the color you want may be closer to the top in stack 2. One can even construct examples where this ruleset uses $\Omega(n^2)$ actions when $O(n)$ actions would suffice for a more sophisticated algorithm able to

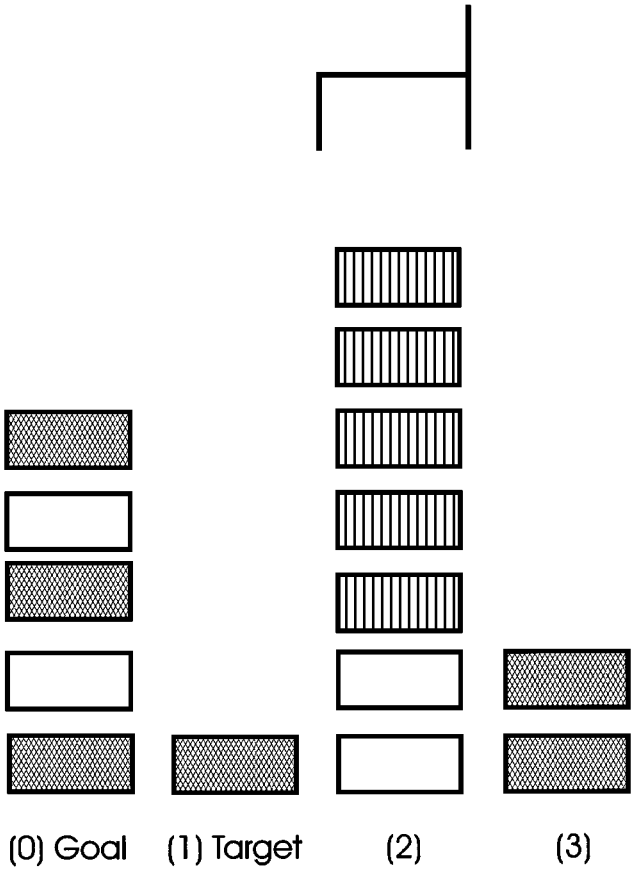

Figure 3. An instance on which ruleset 1 takes time $O(n^2)$. Between placing each correct block on the target, it must shift all the vertically hashed blocks between stacks 2 and 3. Such instances are exponentially rare—for a random instance, the probability that the next block wanted is buried under k blocks is about 3^{-k-1}.

use stack 1 as a staging ground (see figure 3). But it is possible to show that instances where ruleset 1 is inefficient by more than a constant factor occur exponentially rarely. Say you have a random instance. After placing the correct jth block on the target, there is a third of a chance the block you want next is on top of stack three, and only a chance of about 3^{-k} the next correct block is buried under k others. For the ruleset to take more than kn time to solve an n block instance, requires that the *average* block be buried at depth $\Omega(k)$ which occurs only for exponentially rare instances.

Note that intermediate payoffs are essential to the stability of this program. This algorithm utilizes the differing bids to prioritize agent actions. But in Hayek1's economic model, without intermediate payoffs, agents' bids must get larger the *later* they are used in each instance (on average). Hayek1 does not have flexibility to assign an agent a high bid precisely so it will be used early—such an agent would get paid less than it pays, and go broke. Thus the intermediate payoffs allow greater flexibility in *representation*.

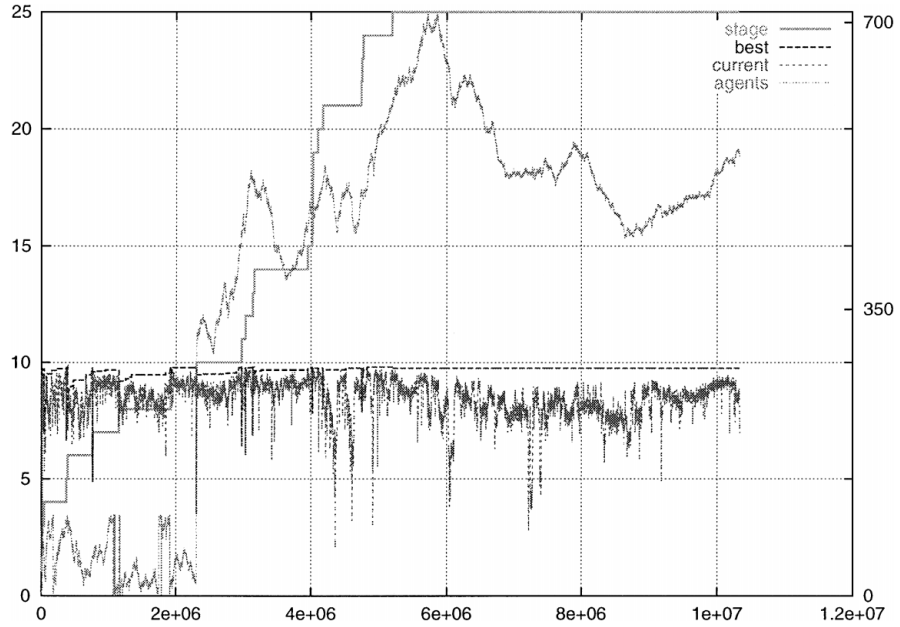

Figure 4. Evolution of a sample BW run with topi and intermediate reward. The horizontal axis is instance number. We graph the stage reached vs. instance number, with stage reached indicated on the left vertical axis. We graph the number of agents alive vs. instance number, with number of agents indicated on the right vertical axis. We graph the best performance on 100 consecutive instances within the current stage vs. instance, and the fraction of the last 100 instances solved vs. instance. In both of these latter two graphs, % correct is shown on the left vertical axis, with 10 on on the left hand scale meaning 100%.

4.1.1. Further Runs with Topi. We restarted a number of additional runs from scratch, and generated strong BW solvers, but did not produce another run that could be demonstrated to be a universal solver. We discuss this further in Section 4.4. Results are reported in Table 1. The evolution of another sample run is shown in figure 4. Our additional runs 1–12 each lasted 4 days on a MIPS R4400 cpu at 150 MHz. Runs 13–18 ran a few additional days on the same machines. (Our initial run discussed above that did produce a universal solver ran for a long but unmeasured time, possibly a few weeks). Runs 1–6 were at rent $= .05$. In each of runs 1–6 (except the least successful) the population size grew gradually, although noisily. It seemed likely that the rent was substantially lower than optimal, both because our large agent sets were greatly slowing computation, and because our previous run with rent $= 0.5$ had generated a universal solver (albeit after a longer run). So we next ran 6 runs with rent $= .5$, each run again lasting 4 days.

Column 2 in these tables reports the final stage reached. Recall Stage i has $i + 2$ total blocks, and the system goes on to stage $i + 1$ when solving 95% of instances at stage i. We took the best set of agents produced in the final stage of each run (measured by a running count of number of the last 1000 instances solved) and ran it on a stage 50 problem, as a rule set with learning turned off. In these runs we allowed 1000 actions to solve each

Table 1. Runs with Topi and Intermediate Reward. Stage is final stage reached during learning. % on stage 50 gives fraction right when best rule set was applied to stage 50 problems, and allowed to use up to 1000 actions for solution. # steps gives number of steps taken to solve the stage 50 problems, averaged over those actually solved. # Agents final is number of agents in population actually participating in stage 50 problems. # Instances total is total number of instances seen during training. Runs 13–18 continued runs 7–12, with a modified algorithm on the additional instances. # Instances Add'l is additional number of instances seen in this continuation.

Run	Stage	% on stage 50	# Steps	# Agents final	# Instances total
			Rent = 0.05		
1	15	88%	88	502	7E6
2	16	93%	184	684	1.1E7
3	10	29%	187	314	9E6
4	10	25%	163	447	1.2E7
5	10	73%	155	246	7E6
6	22	89%	117	581	7E6
			Rent = 0.5		
7	10	57%	84	25	2.4E7
8	18	85%	141	53	1.1E7
9	22	93%	141	40	1.8E7
10	21	93%	155	44	1.1E7
11	20	96%	151	59	1.6E7
12	25	96%	100	59	3.0E6
			Rent = 0.5, Cherrypicking prevention		
Run	Stage	% on stage 50	# Steps	# Agents final	# Instances add'l
13	10	35%	261	31	7.8E6
14	20	87%	139	53	3.1E6
15	48	99%	144	108	1.2E6
16	34	94%	127	71	1.6E6
17	49	98%	129	56	1.4E6
18	31	93%	261	49	2.1E6

problem. (Note at least 100 actions are required to move 50 blocks). The fraction of stage 50 instances solved is reported in column 3 and the average number of actions used in solution of those instances actually solved is reported in column 4. The number of agents active (that is, actually winning auctions) in these agent sets is reported in column 5. The total number of training instances is reported in the last column.

The rent .5 runs are stronger than the rent .05 runs, but within each class there is considerable consistency, especially if, say, the top 50% of the runs are considered. The rent .5 runs saw more instances than the rent .05 runs, because they maintained a smaller agent

population. It is plausible that this accounts for the full difference, and that the rent .05 runs might eventually have caught up in performance if run longer.

In all of these runs two new agents were inserted every tenth instance. If the current agent set was in fact a perfect solver, these new agents nonetheless had the opportunity to cherrypick and enter, destroying the perfect solver. I thus modified the algorithm[13] so that a new agent was inserted every fifth instance, but only if the current agent set had failed on an instance and thus was known not to be perfect. If an agent set were found that was a perfect solver, no additional agents would thus be added to destroy it. We took the agent sets from the rent = .5 run, and restarted them from stage 1 with the cherrypicking modification for about a million additional learning instances. Because we anticipated going to higher stages, we allowed 300 actions per instance during training, instead of 100 as had been the limit. For this reason the final stage reached is not directly comparable. The performance on stage 50 is directly comparable. Disregarding the worst run in both cases, the mean on stage 50 of the unmodified runs was 92.6% with deviation 1.8% and the mean of the anti-cherrypicking augmentation runs was 94.2% with deviation 1.9%. The improvement in the anti-cherrypicking runs was thus not statistically significant, especially as these runs had additional learning time.

We also ran 6 runs, four days each, with rent = .05, with Topi variables but without intermediate reward. One of these reached stage 6, and the other five reached stage 8. These did not exhibit any trend toward larger agent population, finishing up with about 200 agents. Without the ability to use bids for priority, the topi variables are not sufficient (so far as I know) to allow representation of solutions to arbitrary problems, and we observe (by comparison to the next section) that they do not improve Hayek1's performance. Note that the optimal rent should presumably be lower in the absence of intermediate reward, because less money is paid into the system.

4.2. Empirical results without topi features

This section describes results without using the top[i] features.

TD systems and "straight" Holland-style systems, without perfect credit verification, experimentally exhibited interesting phenomena such as speculative bubbles, but could not solve BW instances with more than a few blocks.

We ran the fixed bid system, with perfect credit verification, 6 times on the multistage problem with end payoff only, and 6 times with intermediate payoff, with tax set at .05. Again each run ran for 4 days on a MIPS R4400 cpu at 150 MHz. First we discuss the runs with end payoff only. One of these reached stage 6, one stage 7, and the other three reached stage 8. These runs each generated 1–3 million total agents. The number of agents alive at any given time fluctuated, apparently randomly, between 50 and about 300. At the end, five of the runs had about 200 agents alive, and the sixth had about 100.

One of the runs with intermediate payoff reached stage 8, two reached stage 9, and the other three reached stage 10. These runs all generated between 1 and 1.4 million total agents. They had less total agents than the runs with end payoff only, because they ran for the same CPU time but with more agents at any given time-between 200 and 600.

We also did some runs with the Modified Holland system, which seemed comparable, but we did not make an explicit statistical comparison.

Fixed Bid and Modified Holland systems were also able to solve with intermediate payoffs and 2 blocks per stack but no staged instances. Applied to 3 blocks per stack with incremental payoffs, they solved only about 25% of the instances.

We did another test to make sure that the rule sets Hayek1 produced would in fact generalize.[14] We trained one instance up to stage 9, holding out a random subset of the instances. The resulting rule set solved 946 out of 1000 instances from its training set and 939 out of 1000 from the separate test set. Thus Hayek1 seems to generalize well to examples drawn from the same distribution as its training set.

4.3. "Strawmen"

I have argued that Hayek utilizes its economic model to perform an efficient hillclimbing search in collection of agent space. Such a search is vastly more efficient than, say, an exhaustive search of collections of agents. We estimated in Section 3.4 that the likelihood of generating even single useful rules is very small, so that the likelihood of randomly generating useful rule collections should be infinitesimal. Moreover we discussed there that it is not clear how many instances one should use to evaluate a set of rules, if doing some sort of exhaustive search, before proceeding to try the next set. Nonetheless, I have searched for a simple non-economic hill climbing approach, and tried a few without success. None seem worth reporting. I will report here, however, on exhaustive search for a perfect rule set.

We generated random rule sets, with the topi variables, using Hayek1's random rule generator, and assigned the rules random fixed bids. We applied these rule sets. Whenever the system solved 20 instances in a row, we increased to the next stage. Whenever it failed on any instance, we deleted a random rule and created a random rule. This is an efficient way to search through random rule sets with random prioritization. It looks for rule sets which are either universal solvers, or at least able to solve small block instances with high probability. We searched in this way through over 7 million rule sets. We failed to find a rule set able to progress to stage 4, indicating that no rule set was able to successively solve 20 stage-3 instances.

4.4. Summary and discussion of experiments

We were able to discover one universal solver, using topi variables and intermediate reward. Runs with topi variables and intermediate reward frequently achieved agent sets able to solve almost all 20-block instances, and well over 90% of 52-block instances. Without topi variables or internal reward, Hayek1 was able to solve about 10-block instances, and without topi variables but with internal reward, Hayek1 learned to solve instances of about 12 blocks. Occasionally a run would get stuck in an apparent "local maximum" not nearly as strong as similar runs, but overall the performance is fairly consistent from run to run. In particular, the performance in the best half of the runs of each type is quite consistent. A modification designed to reduce the possibility that cherrypicking was preventing convergence to a perfect solver did not improve performance by a statistically significant amount.

In every experiment, a substantial fraction of the runs improved their stage late in the run. It thus seems likely that the learning process had not converged in any or at least most runs. This may explain why our additional runs were not able to generate another universal solver.

It is also possible that some of the other strong solvers would solve arbitrary instances, given sufficient actions. In the universal solver run, I was able to understand the algorithm used by the agents after some analysis. I have not been able to understand the algorithms used by the other runs. Typically, substantially more agents are involved in computations. Later experiments performed by Erik Rauch have generated very strong solvers which may well be universal solvers, given sufficient actions, but which were not readily analyzable (E Rauch, personal communication).

I view these experiments as validating the concept. We have observed considerable stability in the economy and an ability to solve hard BW problems. We have not, as yet, engaged in many experiments aimed at optimizing the technique. For example, we have not determined the optimal rent (although it was evident from our experiments that the size of the rent impacts performance), nor established clearly whether the fixed bid is better or worse than the Modified Holland bid scheme, nor optimized over the agent creation parameters at all. We introduced 2 agents every 10 instances. It is plausible we could have introduced agents many times faster without upsetting stability, gaining a big speedup.

The Blocks World planning problems Hayek1 learns to solve are complex. They require discovery of an abstract goal. Solution of these problems can often require placement of blocks on top of blocks which will later be needed. As discussed in the introduction, I am unaware of previous authors reporting an ability to learn Blocks World problems with these features. Moreover our simple efforts to handprogram in Hayek1's restricted representation (simply as a test of our software), without the top[i] variables, were inferior to the solutions Hayek1 produced.

I do not have a clear analytic understanding of how to optimize the choice of rent. We performed our experiments with a rent of .05 or .5. The following briefly discusses some ideas regarding the effect and choice of rent. Without rent, the agent's bids tend to rise to the expected payin of the agent. Charging a per instance rent depresses the expected payin of the agent (by the amount it pays in rent) and so should, in principle, depress all bids by a small amount, but more generally applicable agents—who have more profit opportunities over which to amortize each rent payment—should see a smaller decline than specialists. Thus higher rent favors using more general agents over more specific ones. This favoritism is perhaps useful for learning, but is precisely the opposite of what intuition (Holland, 1986) says we should do at any given time to solve the current instance—i.e., favor the more specific, and hence presumably more pertinent, agent. A higher rent does, however, help to remove marginally unprofitable agents who might otherwise survive for some time by luck. If we hypothesize that the agent's wealth executes a random walk (as they earn money in some instances and lose it in others), agents with a stochastic payoff but slight negative expectation have an expected lifetime of forever before they randomly walk to negative wealth and are removed, and the entry of such could in principle clog the system and depress performance.

5. Related work

The origin of mental capabilities in the interaction of smaller units has been widely studied, e.g. (Drescher, 1991; Holland, 1986; Lenat, 1983; Maes, 1990; Minsky, 1986; Newell, 1990; Selfridge, 1959; Valiant, 1994, 1995). These models are not explicitly economic. We commented in the introduction that non-economic multi-agent models risk unintended consequences arising from distorted implicit incentives. For one example, Miller and Drexler (1988b) discuss how imprecise motivation of meta-agents led to parasitic behavior in Lenat's "Eurisko". We discuss some distorted incentives in Holland style classifiers below. Valiant's neuroid model (Valiant, 1994), proposes many specialized agents, but assumes a peripheral system that will arbitrate in large measure their interaction. This may be a hard task. Minsky's "Society of Mind" (Minsky, 1986) proposes many subsystems at a high level, but does not address their dynamic interaction. Specifying this interaction without vast unintended consequences may be a challenging task, akin to centrally managing a complex economy.

5.1. Classifier systems

Holland (1986) first proposed an explicitly economic model of intelligence, also based on condition, action, bid agents. His seminal work has sparked a whole literature, and greatly influenced my own ideas. However, some have viewed the empirical results with Holland-style "classifier systems" as disappointing, cf. (Wilson & Goldberg, 1998). My research indicates that problems may stem from misguided incentives. One critical problem in my view is that agents enter Holland's economy with initial wealth.[15] This creates an unfortunate incentive for agents to fleece new agents, rather than to earn money in the real world. In my experiments without perfect credit verification, the injection of initial wealth causes inflation and speculative bubbles. Perhaps to mitigate this problem, Classifier systems often proceed in generations, with all or many existing rules (or at least their bids) replaced simultaneously. If one then runs long enough so that the total amount earned from the world is small compared to the initial wealth, the distorting effect can be minimized. But the cost of running in generations is that the system can not evolve smoothly. By contrast Hayek1 learns continuously, slowly adding new agents.

Many rules act simultaneously in Holland's Classifier Systems. When payoff arives from the world, all active rules share it (Holland, 1986). This invites a phenomenon called "the tragedy of the commons" (Hardin, 1968; Lloyd, 1833). The "tragedy of the commons" is a ubiquitous problem in economies wherever property rights are shared or unclear. Just like herders are motivated to overgraze land held in common, each rule's motivation is to be active even if its action lowers the payoff. Hayek1 by contrast allows only one rule at any given time to act. Property rights are clear and not shared, and thus there is no tragedy of the commons. In ongoing work (Baum & Durdanovic, 1998a, 1998b) Hayek is generalized to allow multiple computational actions in parallel, but with only one auction winner able to take actions affecting the world. That rule then collects any payoff.

Some of Holland's non-economic design choices should also be reconsidered (and see also (Schuurman & Schaeffer, 1989)). Holland Classifiers use a particular representation

rather different than that used to date by Hayek1. This representation has been shown Turing equivalent (Forrest, 1985). However, universal computers require infinite memory. Practical considerations force size limitations. It is unclear whether Holland's representation is a particularly efficient or learnable realization of finite state machines. In particular, it omits any *a priori* topological bias.

The bucket brigade algorithm used both in Hayek1 and Classifier Systems has the property that the bids, which naively can be used to prioritize the agents, in fact are restricted by the requirement that the agents must be profitable to survive. Thus if agent A typically precedes agent B, agent B must have a higher bid, else agent A will go broke. But this is precisely the opposite of what one would like, if using the bids to specify default hierarchies. Thus the representation is less flexible than naively appears (cf. Section 4.1). This problem has been independently pointed out by Lettau and Uhlig (1999).

Hayek1 (and classifier systems) are therefor intrinsically not free to use bids to specify default hierarchies. It is unclear whether this is a flaw. One certainly needs some way that the representation can be flexible enough so that Hayek can learn to solve its problems. Ongoing work is experimenting with ways in which Hayek can expand its representations to achieve the necessary flexibility. In this paper, intermediate reward functions yielded the desired expressive power.

Holland (and much if not all of the following literature) used Genetic Algorithms (GA's) to train classifier systems. Recently (Baum, Boneh, & Garrett, 1995) has shown that some GA's can in some contexts be more efficient than Hill Climbing. However, the same paper, as well as much experience, indicates that in many contexts GA's are dramatically worse. At best, their performance is very sensitive to the choice of crossover structure, and in learning environments which are even marginally *tabula rasa*, the optimal crossover structure will be unknown. Genetic Programming is a related technology to Classifier Systems. A recent comparison of Genetic Programming to a strawman hillclimbing algorithm found the hillclimbing algorithm faster by a factor of 50 on a small Boolean function learning task, and found a gradient descent neural algorithm yet faster than the strawman hillclimbing algorithm by a factor of several hundred (Lang, 1995a, 1995b). Hayek1 is currently trained by hill climbing. Ongoing work is experimenting with metalearning approaches that might be viewed as vaguely genetic, since they have the ability to combine other agents, but use an economic model to assign credit for useful combinations (Baum & Durdanovic, 1998a, 1998b).

5.2. Temporal difference learning/RTDP

There is a large literature (cf. Bertsekas & Tsitsiklis (1996); Sutton & Barto (1998)) on learning methods that maintain an evaluation function $U(i)$ mapping states to values, update the evaluation of a state in terms of the evaluation of the next state reached (plus any reward in transition), and make move choices by choosing the action maximizing the expected evaluation of next state, with some randomization for exploration. I will refer to this class of methods as Real Time Dynamic Programming (RTDP) or Temporal Difference Learning (TDL) methods. If $U(i)$ is a lookup table this approach can be shown to converge for Markovian environments (Dayan & Sejnowski, 1994; Gurvits, Lin, & Hanson, 1994).

The use of a lookup table $U(i)$ mapping state-action pairs to values is called Q-Learning (Watkins, 1989). Unfortunately, in environments with a huge number of states, one must regularize by using an evaluation mapping which is many to one, e.g., a neural net, and also give up on exploring the entire state space. Under such circumstances, the interaction of action choice, evaluation update, and learning bias is complex enough that there is not only no mathematical reason, but also no strong intuitive reason, to expect updates to yield improvement, to say nothing of convergence of the algorithm overall. Nonetheless, there have been some notable empirical successes, cf. (Tesauro, 1992; Crites & Barto, 1996; Zhang & Dietterich, 1996).

By this definition, the Hayek Machine can be viewed as a kind of RTDP method. Indeed, if one runs Hayek1 with agents whose condition is true for exactly one state, it basically reduces to Q-learning.[16] Note that in this case the system has perfect price discrimination, and cherrypicking can not occur. In appropriate environments, I believe that the proof of convergent of Q-learning could be carried through for such a simplified Hayek machine, but I have not written out a formal proof.

Thus from this point of view, the Hayek Machine is a particular approach to generalization among RTDP methods. There are, however, several arguments why it may be a particularly interesting approach. First there is the intuition that each update in Hayek1—i.e., each addition of a new agent or each death of an old agent, can be expected to improve it, either by improving performance, or by improving valuation.

In proposing the Hayek Machine I have abandoned the state-space based view of standard RTDP approaches from the outset, and sought an agent based view. In this paper, we have seen how it is possible to exploit the compact structure of Blocks World in such a view. Standard RTDP approaches rely on generating an evaluation function able to guide progress. It is not clear whether such an evaluation function can be learned for Blocks World. I have been unable to do so, and I know of no other author who has. Whitehead (1991), for example, remarked that non-Markov properties are likely to arise when one attempts a representation able to exploit compact structure of the space.

Hayek1 maps state, action pairs to values. This is a particular form of bias which may intutively be particularly effective in environments with combinatorial numbers of states, but relatively few actions (Wilson, 1995). By keeping around those agents which are profitable, Hayek1 dynamically addresses the bias variance tradeoff. Profitable agents, by definition, are well trained and so Hayek1 can maintain more of them as it has data to justify a more complex representation.

Within the framework of The Hayek Machine, we are free to explore more complex representations (i.e., more complex types of agents) while still maintaining the intuition that each update improves performance. Indeed, I am currently exploring complex representations that would be difficult to conceive within standard RTDP frameworks (Baum & Durdanovic, 1998a, 1998b).

Collections of agents are a modular way to represent evaluation functions. In very complex environments, and especially in systems like the mind that must function in many different environments, this modularity may be critical to learning.

When new agents enter, their bid may be an unreliable guide to the actual value of using that action in that state (Venturini, 1994). But by backing up to ignore agents which go broke,

Hayek1 is conservative in its use of these estimates. Agents' bids are used only so far as (a) subsequent bids/results indicate the bid was justified, or (b) the agent has obtained wealth in previous successes to justify any discrepancy. Thus Hayek1 uses both evaluations and an estimate of their reliability in its updating. There is no theorem saying the approach used in Hayek1 is the optimal way to use reliability estimates, but it is a sensible way. Experiments using a TD-like update of bids within Hayek1, but no backup of dying agents, displayed speculative bubbles and no success at solving BW instances of more than a couple of blocks.

Possibly the most impressive example of RTDP success is Tesauro's backgammon player, which starting from zero knowledge trained a neural network to serve as a strong evaluation function for the game of backgammon (Tesauro, 1992). If started with hand coded features, the evaluation function produced sufficed to play at near championship level. TD was so successful at backgammon, however, largely because there is a linear evaluator that is powerful for backgammon (Tesauro, 1995). By contrast *any* useful evaluator for our Blocks World problem must inherently be highly nonlinear, since it must compare different locations.

On the other hand, it seems unlikely that Hayek1, at least in its present form, would deal as effectively with problems such as early vision or perhaps backgammon, where linear structure is a critical guide. Analagously to how TD was effective at backgammon largely because of the existence of a powerful linear evaluator, a skeptic might conjecture that Hayek1 was effective at BW because of the existence of relatively small agents that make significant progress, and of small sets of agents solving BW, cf. Section 4.1, and because this is a problem in which 2 dimensional topology, built in the form of wildcards and cartesian coordinates, was very helpful.

There have been several recent attempts to apply reinforcement learning in multi-agent contexts. Crites and Barto (1996) divided the problem of operating four elevators up among four agents (in the obvious way). They gave a reward to each agent equal to the total reward to the system. This approach attempts no assignment of credit among the agents. I conjecture this reward scheme would not work well for large numbers of agents, but it was effective in their application. Humphrys (1996) discusses several alternative approaches.

5.3. Agoric computation

There is a sizeable and interesting literature discussing use of equilibrium economic models to allocate resources in a distributed environment (cf. Clearwater (1996)). Much if not all of this literature, however, deals with programming environments rather than learning machines. Wellman's WALRAS (Wellman, 1993) is a programming environment where users may describe agents and provide them with appropriate cost and demand curves. Prices are then determined by equilibrating cost and demand for various commodities. The WALRAS environment implements general equilibrium theory. By contrast, the present paper deals with a learning machine, not a programming environment, and utilizes non-equilibrium economics, where the entry of new agents and bankruptcy of old ones is central, and where rational pricing and decisions are not programmed in, but rather learned, enforced by bankruptcy.

Perhaps more pertinent to the current paper is the work of Miller and Drexler (1988a, 1988b) which proposed 'Agoric Computing'. Miller and Drexler were also chiefly

concerned with creating a programming environment, but they did comment on the possibility of using Agoric computing for learning. Because they did not instantiate a learning approach, however, they did not come to grips with the critical problem of how agents can learn to make appropriate pricing decisions. Miller and Drexler conjectured that computational markets could be set up to avoid all imperfections of real ones. So for example, computational markets can in principle absolutely rule out theft, and fraud. No negative externalities need exist—there is no analog of pollution. It is a priori possible that a computational market can be constructed without any analog of "public goods"—I know of none in Hayek. No one need have moral compunctions if there is extreme income inequality among agents. The present paper illuminates, however, some limitations of the perfect computational market that arise because agents are imperfectly able to learn optimal pricing.

5.4. Blocks World as a planning problem

Bacchus and Kabanza (1995) experimented with SNLP McAllester and Rosenblitt (1991); Soderland, Barrett, and Weld (1990), Prodigy 4.0 Carbonell et al. (1992), and their own TLPlan Blocks World Problems closely related to (but different from) ours. These are sophisticated programs representing many years of effort, utilizing hand coded general purpose heuristics, doing extensive searches, and of course given a specified goal. Arguably they represent the state of the art in General Planning approaches. Run on a Blocks World with an unbounded table, SNLP, Prodigy, and TLPlan all exceeded resource bounds on problems of about six blocks. TLPlan was then augmented with hand-coded special-purpose knowledge about Blocks World. This allowed it to solve problems with around 50 blocks. TLPlan was then run on a table with room for only three blocks. With simple handcoded rules, it solved problems with about 12 blocks. With a complete backtrack-free strategy, it solved 35-block problems.

My BW problems are not identical to Bacchus and Kabanza's. While the physical task of copying stacks of blocks is basically the same, the encoding and the distribution of examples are different. Their blocks are uniquely labelled, while mine come in three colors. Thus it is not clear that the 10-block problems Hayek1 learns to solve with end payoff only, or the 12-block problems it learns to solve with incremental payoff, are harder than the 5-block problems these planning algorithms solve without hand coded domain knowledge. It does seem plausible, however, that these problems are of roughly comparable difficulty, viewed strictly as planning problems. But Hayek1 is further solving the goal discovery problem. These planning programs take 400 CPU seconds on search. Hayek1, once trained, solves instantly, and uses few actions. Hayek1's solutions involve systems of tens or in some cases several hundred participating agents. And note that, given topi variables and incremental payoff, Hayek1 generated a general and efficient solution to arbitrary size problems the first time it evidently could express one.

6. Motivation as a model of economics

Orthodox economic models typically make several assumptions. All economic agents are often postulated to have complete information (i.e., no agent knows anything not known

to the others), and perfect rationality, i.e., infinite computational power. Real economies, however, involve interaction between humans who, while computationally powerful, are not infinitely so, and who have differing information.

A literature on "bounded rationality", cf. Simon (1987) attempts to improve economic analysis by perturbing away from perfect rationality. One drawback of this approach is that it is not evident exactly what direction to perturb in (Palmer et al. (1994)).

Moreover, orthodox economics models typically discuss equilibrium phenomena. Yet, to quote one text (Nelson & Winter, 1994), "It is, however, an institutional fact of life that in the Western market economies—the economies that growth theory purports to model—much technical advance results from profit-oriented investment on the part of business firms. The profits from successful innovation are a *dis*equilibrium phenomena, at least by the standard of equilibrium proposed in the models in question."

Increasingly the nascent field of evolutionary economics has attempted to deal with these questions by modelling the interaction of agents who initially have little rationality but learn (cf. Palmer et al., 1994; Anderson, 1996). The Hayek Machine can be seen as another entrant into this approach, with some novel features.

Hayek1's individual agents are simple, automatically generated rules which only make one decision (to bid or not to bid), and hence might be expected to be far from perfectly rational. Since each has a different condition, which is all it knows about the world, they see incomplete and differing information. If one finds that the results which can be proved to hold when one assumes infinitely rational agents, also hold at the other extreme in Hayek1's economy of agents, one has more confidence they apply to the real world. To the extent that Hayek1 suggests alternate explanations for phenomena: e.g., markets are efficient because of disequilibrium phenomena, it may call into question accepted lore. For example, the agents surviving evolution apparently behave very rationally. Coase (1960) has intuitively remarked that such a selection mechanism might be the cause of rationality in economic systems.[17] It is perhaps interesting to present a computer model which realizes this picture.

The Hayek Machine also models interaction with a concretely defined, complex physical world from which wealth may be extracted by careful, complex, and coordinated actions, rather like if a collection of people get together and build a car, they create wealth in our world. This is a considerable departure from the standard economic paradigm, where a central tenet is that value is not absolutely defined, but rather derives solely from the utilities of the individuals. Because of its assumption of an externally defined value, the Hayek Machine addresses technological progress in a novel way. Over time its collection of agents evolves, learning as a society to extract more wealth from the world.

I stress that I am not suggesting Hayek as a fleshed out model of economics. I am proposing that Hayek raises many open questions that might be worthy of consideration by the economics community. If one is willing to couple an economy to an absolutely defined external reward function, one raises the question: for what class of worlds, and for what class of interactions between the agents, can one demonstrate efficient wealth extraction? For simpler systems than that experimented with here, formal mathematical results may be obtainable. In simulation, interesting phenomena observed in real economies, like speculative bubbles and inflation (cf. Section 3.2), cherrypicking (cf. Section 3.3), intellectual property (Baum & Durdanovic, 1998b), formation of corporations (Baum & Durdanovic, 1998b),

and technological progress arise in Hayek. The way in which imperfections in price discrimination can lead to steps which do not increase global wealth creation (discussed in Section 3.3 suggests an important, and perhaps sometimes underemphasized, role for accurate price discrimination[18] in the efficient working of real economies.

7. Conclusions and discussion

A strategy was proposed that uses property rights to autonomously divide complex computational problems among a collection of agents in such a way that each agent has the incentive of improving the performance of the system as a whole. I argued that such a division is critical in extremely complex problems, and that controlling incentives of all the agents is perhaps the critical problem in multiagent systems, mentioning several examples (e.g., Eurisko, Holland Classifier systems) where I believe misguided incentives have caused problems. An instantiation of my strategy was described and applied to difficult Blocks World problems where it was able to solve interesting size problems from *tabula rasa* and, given a feature and intermediate reward, was able to generate a system of agents that interacts to solve an arbitrary BW problem of our form. It was also proposed that "The Hayek Machine" has interest as a new evolutionary model of economics.

If one hopes to understand how reinforcement learning can attain the kind of computational abilities humans display, one is going to have to address the critical problem of how to reinforcement learn in state spaces too large to enumerate. To solve this problem, one must exploit the existence of a compact description of the state space. Our approach abandons the state-space view from the outset. In the limit where our agents recognize only one state, our strategy reduces to Q-learning, but in the more interesting regime studied here where agents generalize it was argued to have several important advantages over standard reinforcement learning strategies. We have a picture potentially capable of exploiting modular structure in problem domains. We have seen empirically here that our approach can powerfully exploit compact structure. Moreover our approach admits generalization to more complex agents, as will be discussed below.

There are many open questions. It would be interesting to determine how much performance would be improved by simple tuning e.g., in parameters of agent creation, rate of agent creation, method of bid selection, rent, etc. More interesting would be an analytic understanding of how these parameters should be set. Another question is whether one can prove interesting analytic results about toy economies of this type. It would be interesting to apply Hayek to several different types of problems, simultaneously, to see whether transference of knowledge occurs and what kind of niche structure develops in the economy.

In the years between when this work was first presented (Baum, 1996) and when the present paper appeared, my research has focussed on creating an economy with more powerful agents and on understanding convergence properties. Baum (1998) shows that given the kind of economic framework we propose, a collection of rational agents will cooperate to solve problems as best they know how, surveys catastrophes that will occur in multi-agent systems when an appropriate economic framework is not imposed, compares genetic programming and hillclimbing approaches, and surveys insight into biological intelligence as a

multi-agent system arising from many disciplines. Baum and Durdanovic (1998b) discusses an economy of agents able to create other agents as well as to implement complex series of actions in the world, and demonstrates meta-learning effects. Baum and Durdanovic (1998a) improves the model of Baum and Durdanovic (1998b) to yield the following. We initiate an economy with a single agent that creates other agents that are random programs, themselves potentially capable of creating other agents. The system evolves to a collection of hundreds of agents that interact to solve complex Blocks World problems. Learning from end reward only, they are able to solve essentially 100% of problems with 20 high goal stacks, and 90% of problems with 200 high goal stacks. Solution of each instance involves the sequential application of ten or so agents, each of which takes tens of actions. Thus, compared to the present paper, we do indeed find strong learning without intermediate reward when the agents are able to employ powerful representations. My current work is focussing on extensions to other problem domains.

Acknowledgments

Charles Garrett wrote all the code and ran all the experiments in the first year, from high level (English language) specifications. I thank him for his efficient and expert assistance which was integral to making this paper a reality. Michael Buro rewrote the code and ran later experiments (again from English language specifications). I would also like to thank W.B. Arthur, A. Birk, M. Buro, C. Garrett, A. Grove, M. Kearns, M. Mitchell, S. Omohundro, H. Stone, D. Waltz, J.-C. Weill for useful comments on a draft or a talk, and F. Bacchus, E. Rauch, J. Scheinkman, R.S. Sutton and W.D. Smith for conversations. An extended abstract of this paper appeared in ICML '96 (Baum, 1996).

Notes

1. Holland (1995) and others (Anderson, Arrow, & Pines, 1988) have similarly generalized economies as well as many other natural systems as "Complex Adaptive Systems". CAS include what I am calling economies of agents. Unfortunately, I do not know of an extent compact definition of CAS.
2. F.A. Hayek (1900–1992), winner of the 1974 Nobel Memorial Prize in Economics, illuminated the emergence of spontaneous order in economies, societies, and the mind.
3. Other agents may perform computations, but can not take actions affecting the world.
4. All instances except a small (finite) collection of instances on which Hayek1 wasn't trained, and which Hayek1's solution could not solve without further training.
5. All computer code was written by Charles Garrett or Michael Buro. See the acknowledgment at the end of the paper.
6. In some runs with the second and third schemes, larger final rewards were given for solving problems from higher stages. The motivation for this scaling was to avoid bankrupcies at the transition to a higher stage. This scaling of rewards had no impact. Runs with the same initial random seed but with and without graduated reward were nearly identical.
7. An agent whose action is to grab (respectively, drop) a block will not bid when the hand is holding (respectively, not holding) a block.
8. In our experiments, the first active agent in a given BW instance pays out its bid, but no other agent receives it.
9. If two novices applied, one was selected randomly. ϵ was set by fiat to be 0.1. If no veteran applied than the novice's bid was fixed to be ϵ.

10. This happens because our agents generalize, bidding a contant value whenever their condition is valid. If conditions are chosen so as to be valid on only a single state, Hayek1 reduces to Q-Learning, as discussed in Section 5.2.
11. We also tried an alternative approach where the stored best state was only updated at the end of N steps (and then only if it had improved).
12. When a term of form (a, b) was created, b was $*y$ with probability $P_* = .25$, top$[i]$ with probability $P_{\text{top}}/3$ for each of the three i values $1, \ldots, 3$, else it was chosen uniformly from $(0, \ldots,$ numblocks $- 1)$.
13. In the modified algorithm we initialized a flag to be 1. An agent was inserted every fifth instance if and only if the flag was at least 1. The flag was decremented by 1 when a novice agent got positive wealth. The flag was set to 1000 if we failed to solve an instance unless an agent who at one time had wealth above 50 died in the instance, in which case the flag was set to 0. The flag was set to 0 whenever an instance that at one time had wealth above 50 died. This ensured the flag would be zero if we had not failed on an instance since we last added an agent, or if we had not failed on an instance since an established rule last died, creating essentially a new agent set to test. The "50" threshold was heuristically chosen so as not to consider an agent set "new" and worthy of testing whenever a novice agent gained wealth and then immediately died.
14. In addition, I have described tests with topi variables where we tested rule sets on much larger instances than those on which they had been trained.
15. For an interesting paper applying a bucket-brigade like idea in a different context, and conserving money (see Schmidhuber, 1989).
16. Q-learning maintains a table assigning a value to every state, and at each step goes probabilistically to the next state, with preference to the one of highest value. It learns by dynamic backup of values from end states. Using single state agents, Hayek1 stores exactly the same information and learns state values the same way, by backup from end states. Hayek1's search is randomized by the randomized introduction of new agents. Of course, the rate of introduction of new random agents achieving optimal tradeoff between exploitation and exploration would depend on the number of states. You need to have enough novice agents around so that a finite fraction of instances explore novel ground. In Q-learning the evaluation of a state is typically updated to that of the best following state. One can duplicate that in Hayek1 by introducing new agents with bid ϵ less than their successor, rather than ϵ higher than their competitor, as done in the experiments reported here.
17. Coase went further to note that this mechanism has predictive power: such a mechanism might be expected to enforce rationality strongly on firms, but not so strongly on individuals in their personal lives.
18. Although human or corporate agents may have more computational resources available in fixing prices than Hayek's agents, they are frequently constrained in their ability to price discriminate. A well known example is that airlines have to resort to the clumsy mechanism of pricing based on Saturday night stays in order to achieve a degree of price discrimination.

References

Anderson, E.S. (1996). *Evolutionary Economics: Post-schumpeterian contributions*. London: Pinter Publishers.

Anderson, P.W., Arrow, K.J., & Pines, D. (1998). *The economy as an evolving complex system*. Redwood City, CA: Addison Wesley.

Bacchus, F., & 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.

Baum, E.B. (1996). Toward a model of mind as a laissez-faire economy of idiots, extended abstract. In L. Saitta (Ed.), *Proc. 13th ICML '96* (pp. 28–36). San Francisco, CA: Morgan Kaufman.

Baum, E.B. (1998). Manifesto for an evolutionary economics of intelligence. In C.M. Bishop (Ed.), *Neural networks and machine learning*. Springer-Verlag.

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

Baum, E.B., & Durdanovic, I. (1998a). Emergent planning by an artificial economy. Submitted for publication.

Baum, E.B., & Durdanovic, I. (1998b). Toward code evolution by artificial economies. In L.F. Landweber and E. Wintree (Eds.), *Evaluation as Computation*, Springer Verlag, 1999, and available at `http://www.neci.nj.nec.com:80/homepages/eric/eric.html`.

Bertsekas, D.P., & Tsitsiklis, D.P. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

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

Carbonell, J.G., Blythe, J., Etzioni, O., Gill, Y., Joseph, R., Khan, D., Knoblock, C., Minton, S., Perez, A., Reilly, S., Veloso, M., & Wang, X. (1992). *Prodigy 4.0: The manual and tutorial*. Technical Report CMU-CS-92-150, School of Computer Science.

S.H. Clearwater (Ed.). (1996). *Market-based control, a paradigm for distributed resource allocation*. Singapore: World Scientific.

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

Cosimides, L., & Tooby, J. (1992). Cognitive adaptations for social exchange. In J.H. Barkow, L. Cosimidies, & J. Tooby (Eds.), *The adapted mind*. New York: Oxford University Press.

Crites, R.H., & Barto, A.G. (1996). Improving elevator performance using reinforcement learning. In D.S. Touretsky, M.C. Mozer, & M.E. Hasselmo (Eds.), *Advances in neural information processing systems* (Vol. 8, pp. 1017–1023). Cambridge, MA: MIT Press.

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

Dennett, D.C. (1991). *Consciousness explained*. Brown, Boston. Little.

Drescher, G.L. (1991). *Made-up minds*. MIT Press.

Dzeroski, S., Blockeel, H., & DeRaedt, L. (1998). Relational reinforcement learning. In J. Shavlik (Ed.), *Proceedings of the 12th International Conference on Machine Learning*, San Mateo, CA: Morgan Kaufman.

Estlin, T.A., & 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 classifiersystem. *Proc. First International Conference on Genetic Algorithms* (pp. 188–196). Hillsdale, NJ: Lawrence Erlbaum Associates.

Fox, P. (1997). Functional volume models: System level models for funcational neuroimaging. In *International Conference on Neural Networks*.

Gurvits, L., Lin, L.-J., & 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.H. (1986). Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), *Machine learning* (Vol. 2, pp. 593–623). Los Altos, CA: Morgan Kauffman.

Holland, J.H. (1995). *Hidden order*. Reading, MA: Addison-Wesley.

Humphrys, M. (1996). Action selection methods using reinforcement learning. In P. Maes, M. Mataric, J.-A. Meyer, J. Pollack, & S.W. Wilson (Eds.), *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior* (pp. 135–144). Cambridge MA: MIT Press/Bradford Books.

Koza, J.R. (1992). *Genetic programming* (pp. 459–470). Cambridge: MIT Press.

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

Lang, K. (1995b). Comments on 'a response to...'", August 18.

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., & Uhlig, H. (1999). Rule of thumb and dynamic programming. American Economic Review, in press.

Lloyd, W. (1833). *Two lectures on the checks to population.* Oxford: Oxford University Press.

Luria, A.R. (1973). *The working brain, an introduction to neuropsychology*. New York: Basic Books.

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

McAllester, D., & Rosenblitt, D. (1991). Systematic nonlinear planning. *Proceedings of the AAAI National Conference*.

Miller, M.S., & 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 (pp. 133–176). New York: North Holland.

Miller, M.S., & Drexler, K.E. (1988b). Comparative ecology. In B.A. Huberman (Ed.), *The ecology of computation*, number 2 in Studies in Computer Science and Artificial Intelligence (pp. 51–76). New York: North Holland.

Minsky, M. (1986). *The society of mind*. New York: Simon and Schuster.

Minsky, M. (1995). Steps towards artificial intelligence. In E.A. Feigenbaum & J. Feldman (Eds.). *Computers and thought*. Menlo Park: AAAI Press.

Nelson, R.R., & Winter, S.G. (1994). *An evolutionary theory of economic change*, volume 5th Printing. Harvard University Press.

Newell, A. (1990). *Unified theories of cognition*. Cambridge: Harvard University Press.

Palmer, R.G., Arthur, W.B., Holland, J.H., LeBaron, B., & Tayler, P. (1994). Artificial economic life: A simple model of a stockmarket. *Physica D 75* (pp. 264–274).

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by error propagation. In D.E. Rumelhart & J.L. McClelland (Eds.), *Parallel distributed processing*. Cambridge: MIT Press.

Schmidhuber, J. (1989). The Neural Bucket Brigade: A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science*, *1*(4), 403–412.

Schuurmans, D., & Schaeffer, J. (1989). Representational difficulties with classifier systems. *Proceedings of International Conference on Genetic Algorithms* (pp. 328–333), Fairfax, VA.

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

Simon, H.A. (1987). Bounded rationality. In J. Eatwell, M. Millgate, & P. Newman (Eds.), *The new palgrave: A dictionary of economics*. London and Basingstoke: Macmillan.

Soderlan, S., Barrett, T., & Weld, D. (1990). The snlp planner implementation, contact bug-snlp@cs.washington.edu.

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

Sutton, R.S., & Barto, A.G. (1998). *Reinforcement learning, an introduction*. Cambridge: MIT Press.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, *8*, 257–277.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, *38*(3), 58–68.

Toga, A.W., & Mazziotta, J.C. (1996). *Brain mapping, the methods.* San Diego: Academic Press.

Valiant, L. (1994). *Circuits of the mind*. Oxford University Press.

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

Venturini, G. (1994). Adaption in dynamic environments through a minimal probability of exploration. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior* (pp. 371–379). Cambridge, MA: MIT Press.

Watkins, C.J.C.H. (1989). *Learning from delayed rewards*. Ph.D. thesis, Cambridge University.

Wellman, M.P. (1993). A market oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, *1*, 1–23.

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

Wilson, S.W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation*, *3*(2), 149–175.

Wilson, S.W., & Goldberg, D.E. (1998). A critical review of classifier systems. *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kauffman.

Winograd, T. (1972). *Understanding natural language*. New York: Academic Press.

Zang, W., & Dietterich, T.G. (1996). High-performance job-shop scheduling with a time-delay td (lambda) network. In D.S. Touretszky, M.C. Mozer, & M.E Haselmo (*Eds.*), *Advances in neural information processing systems* (Vol. 8, pp. 1024–1030).

Received November 13, 1995
Accepted February 21, 1997
Final manuscript November 4, 1998