

Pets That Learn

by

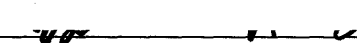
William H Coderre

SB, Electrical Engineering and Computer Science
Massachusetts Institute of Technology
1986

Submitted to the Media Arts and Sciences Section
in partial fulfillment of the degree of
Master of Science in Visual Studies
at the
Massachusetts Institute of Technology
February 1989

Copyright © 1988 Massachusetts Institute of Technology. All rights reserved.

Signature of Author



William H Coderre

Media Arts and Sciences Section

September 16, 1988

Certified by

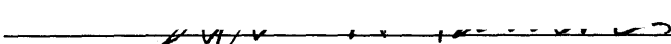


Marvin L Minsky

Donner Professor of Science

Thesis Supervisor

Accepted by



Stephen A Benton

Chairman

Departmental Committee on Graduate Students

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

FEB 24 1989

1

RECEIVED

Rotch

Pets That Learn

by

William H Coderre

Submitted to the Media Arts and Sciences Section
on September 16, 1988 in partial fulfillment
of the requirements for the degree of
Master of Science in Visual Studies.

Abstract

Two years ago I proposed a system called Petworld for modelling behavior. In Petworld a hierarchy of agents determines the behavior of generic animals in a simulated environment.

Now I propose a new system called SHERMAN that extends pets into learning entities. Each agent in a hierarchy will be able to recognize, remember, and predict when its subordinate agents will produce a useful result. Thus SHERMAN is capable of organizing and refining the hierarchy.

The initial demonstration of the system is based on a Petworld environment, and shows how the system, given a collection of agents, will learn to connect them in a propitious manner, using a variety of connections.

Thesis Supervisor: Marvin L Minsky

Title: Donner Professor of Science

Table of Contents

Table of Figures	4
Introduction	5
<i>Overview</i>	
 Section One: About The Petworld	
Chapter 1: The Petworld.....	7
<i>Mechanics of the Petworld · About Pets</i>	
Chapter 2: Pet Intellect	10
<i>Pets as Knowledge-Based Systems · Why a Hierarchy? ·</i>	
<i>What Kind of Hierarchy?</i>	
Chapter 3: An Example Pet Analyzed	16
<i>Rules from the Sample Pet · Petworld Anecdotes</i>	
 Section Two: SHERMAN — Agent Learning in the Petworld	
Chapter 4: Learning in a Hierarchy.....	21
<i>Why Learn? · How AI's Learn Now ·</i>	
<i>A Simplistic Model of Learning · About Goals</i>	
Chapter 5: How Agents Learn	25
<i>Additional Mechanisms for Learning · How Agents Run</i>	
Chapter 6: Memory and Modelling in Agents.....	28
<i>Details of Experiences · Why Build Models? ·</i>	
<i>Models, Clumps, and Cases · Details of Models in SHERMAN ·</i>	
<i>Creating and Refining Models · Retrieving Relevant Cases ·</i>	
<i>Adding Cases to Memory · Conclusions about Modelling</i>	
 Section Three: Conclusions, Applications, and Extensions	
Chapter 7: Additional Learning Agents	35
<i>Goal Chaining · Agent Generation</i>	
Chapter 8: Conclusions and Applications.....	38
<i>Applications</i>	
 Acknowledgements	40
References.....	41

Table of Figures

Figure 1	A Petworld screen dump.....	8
Figure 2	A pet in completed nest	9
Figure 3	A ranking and what it represents.....	10
Figure 4	A hierarchy	10
Figure 5	A decision tree.....	12
Figure 6	Branching down.....	13
Figure 7	Burbling up.....	14
Figure 8	An example pet brain.....	16
Figure 9	Schematic of the learning cycle.....	23
Figure 10	Cases and clumps	31
Figure 11	Goal chaining.....	35
Figure 12	Agent generation	36

Introduction

Two years ago I proposed a system called Petworld for modelling the behavior of simulated animals. Petworld uses a hierarchy of agents to model behavior, where each agent gives to its superior a ranking (a weighted list) of potential actions. The superior can resolve conflicts between agents with methods suggested from animal behavior study: by choosing one agent outright, by compromising between agents, and by ignoring conflicting agents and allowing another agent to be selected.

Despite the simple mechanism and small amount of internal state the pets contain, an interesting set of behavior can be modelled. An example pet is presented, which is capable of foraging, interacting with other pets, building a nest, and exploring the world. Emergent behavior is also demonstrated.

As an extension to the Petworld, I will describe SHERMAN (named after the cartoon character), an approach to machine learning. Practically all animals do some form of learning — from the simple associations that sea slugs make between light and food, to the predictive behavior of predatory cats. As a teaching tool, SHERMAN allows a student to train a pet and then study its behavior.

With SHERMAN, each agent learns by remembering when its subagents produced useful behavior. A set of behavioral goals drives the system. When a goal is satisfied, the agent creates an experience record noting how the subagent filled the goal. By making models of and generalizations about the experiences, the agent is able to fill in its knowledge, and by adding goals, the agent is capable of extending it. Last, by adding an agent generator capable of creating plausible agents to handle new situations, the pet as a whole is able to restructure the hierarchy, and adapt to a changing environment.

One of the key features of SHERMAN is that it is capable of learning in a structured and organized manner. As behavioral skills are learned, they are used to construct a hierarchical network of agents like the ones in Petworld.

SHERMAN can learn by refining an existing network — it can start with a marginally useful network designed by hand (or borrowed from another application) and allow the system to tune itself to the new environment.

Overview

In Section One of this thesis I will talk about the Petworld system for modelling behavior: the simulated environment, the “pet” generic animal and its capabilities, and the structure and capabilities of the pet intellect. I’ll conclude with a sample brain capable of foraging, interacting with other pets, building a nest, and exploring the world.

In Section Two, I will describe SHERMAN, my approach to learning. Each agent in the pet brain will be given an experience memory — a means of remembering which subordinate agent was good under what circumstances. I’ll analyze ways of organizing, using, and extending the experiences collated, as well as the more practical issues of controlling the explosive amount of experience information to be stored and perused. I’ll dub this kind of learning *agent learning*, since each agent in the brain learns.

In Section Three, I will describe the uses of agent learning in other settings. One additional method I will describe is *learning by imitation*, where the network learns by mimicking an expert source. I’ll describe how this method of learning can be used in a teaching environment.

I’ve considered SHERMAN as both a logical learning extension to a simple cognitive model and as a first step to a learning machine for Society Of Mind-like intellects. Insights from SHERMAN should prove of interest to both behavior simulation and machine learning.

SECTION ONE: ABOUT THE PETWORLD

Chapter 1: The Petworld

Petworld is a system for modelling the behavior of simulated animals as a hierarchy of simple interacting agents. Each agent passes a ranking (a weighted list) of recommendations to its superior, which can resolve conflicting rankings by straight choice, compromise, displacement, or substitution. When the rankings have burbled up to the top of the hierarchy, the pet executes the highest-ranked action.

This model is quite simple but can *leverage* knowledge — use a little knowledge at the top of the hierarchy to control the information flow.

Pets contain very little internal state. Pets function in nearly a purely reactive fashion: “What they see determines what they do.” Although this is almost the opposite of many AI approaches, it is well suited to the simulation of many kinds of animals.

Petworld was designed for pedagogical uses as well as an exercise in AI. As part of the Apple/MIT Vivarium Project, grade- and high-school students could use Petworld in a science curriculum studying animal behavior.

Mechanics of the Petworld

Petworld is a world of pets, rocks, and trees which inhabit a bounded region of a two-dimensional cartesian plane. Time passes in discrete quanta, and simultaneous action by all the inhabitants is simulated. Each action executed in the world takes one tick, and brain calculations are instantaneous. This is similar to a single-clock state machine. (I also tested a model with longer-duration actions and computer-style interrupts. Although both offer about the same computational power, I found the state machine model adequate and simpler.)

Pets have a body orientation and a limited field of view. Pets can move for a limited distance per round in the direction of their body orientation. Pets cannot push or throw things, but can carry one rock at a time. Trees are food sources, and pets are browsers. Each time a pet eats, the tree loses some meal points and dies when the count reaches zero. Trees also grow (accrue meal points) as time passes. New trees appear spontaneously around the play field. Famines are prevented by creating a new tree when the last one dies.

Several pets are usually living in the world at once. In general, pets are mutually antagonistic, and can attack each other. Pets cannot reproduce, but can die from starvation or wounds.

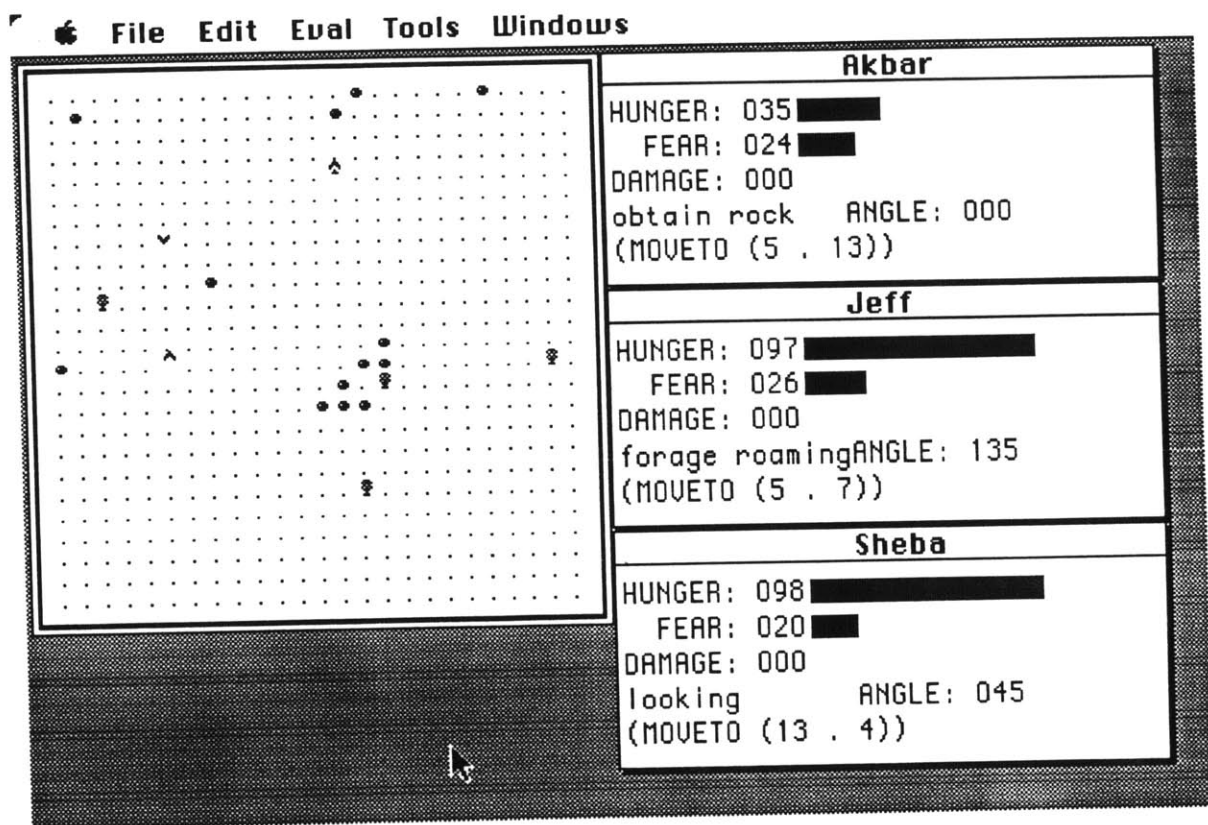


Figure 1: A Petworld screen dump. Each pet has a window showing its current state, which part of its brain hierarchy is being utilized, and its next move. Pets, rocks, and trees are indicated in the world window with a distinguishing character set.

About Pets

Each pet has a very limited set of internal states which indicate the condition of the animal. HUNGER, FEAR, and INJURY have values between 0 and 100. A hunger of 0 indicates being sated, and a hunger of 100 indicates death by starvation. Eating reduces hunger, and the passage of time increases it. Fear is determined by the distance of other pets. A low fear indicates that others are far away, and a high fear tells proximity. Injury is caused when one pet attacks another. Each attack increases the pet's injury count, and with time the count goes back down. If an animal's injury count exceeds 100, the animal dies from its wounds. The flags PAYLOAD, IN-NEST, and NEST-COMPLETE have boolean values. If the payload flag is true, the pet is carrying a rock. NEST-COMPLETE is set when the pet has matched its internal nest pattern. The IN-NEST flag is set if the pet is in the nest.

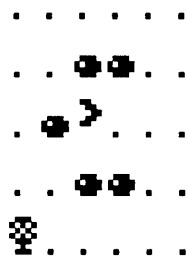


Figure 2: A pet in a completed nest.

In each time-unit cycle, pets perform a "SEE THINK DO" loop similar to a "READ EVAL PRINT" loop in Lisp. First, every pet is given a chance to perceive the world, then a chance to decide what to do, then all actions are performed. Since there is still an order in which pets act (which might be important when, say, there is very little food left), this order is shuffled occasionally.

The pets' actions are MOVE-TOWARDS, TURN, LIFT, DROP, EAT, and ATTACK.

Chapter 2: Pet Intellect

A pet brain is a hierarchy of modules called *experts*. Each expert has inputs from its subordinates and the world, and outputs a *ranking* of possible actions. A ranking is a list of actions assigned *weights* assessed by the expert. The pet executes the top-ranked action of the topmost expert.

<MOVE-TOWARDS <12 . 1> 1.0>	
<MOVE-TOWARDS <12 . 0> 0.5>	
<MOVE-TOWARDS <12 . 2> 0.5>	
<MOVE-TOWARDS <11 . 1> 0.0>	
<MOVE-TOWARDS <11 . 0> -0.5>	. . > . . >
<MOVE-TOWARDS <11 . 2> -0.5>	
<MOVE-TOWARDS <10 . 1> -1.0>	
<MOVE-TOWARDS <10 . 0> -1.5>	
<MOVE-TOWARDS <10 . 2> -1.5>	

Figure 3: A ranking and what it represents. This ranking shows the opinion of the MOVE expert in the situation shown at right.

Decision information in the form of rankings flows up the hierarchy, processed by each expert along the way. Typical expert processing strategies involve assigning weights to rankings, filtering rankings to remove unwanted elements, merging two rankings while possibly emphasizing one more than another, and reducing the elements of a ranking to a single object. Thus, the ethological constructs of competitive, compromise, and displacement behaviors can be directly implemented.

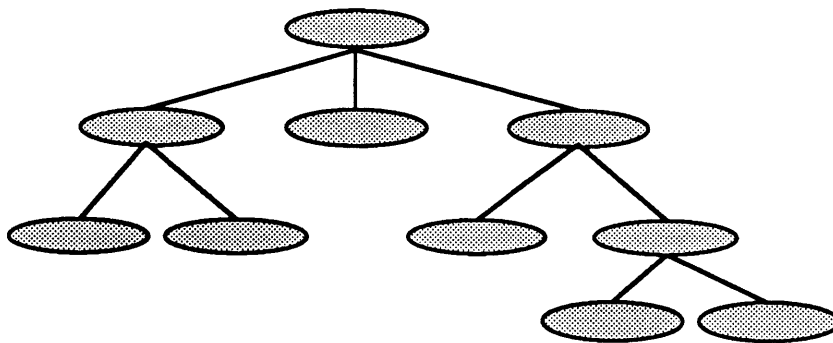


Figure 4: A hierarchy.

Pets as Knowledge-Based Systems

Pet brains contain strategic knowledge. In this sense pets are knowledge-based systems where the knowledge is organized as a rigid hierarchy of interacting agents. Unlike a decision tree, where the tree is used as a polynomial to score different moves, or a branching tree, where program control flows from top to bottom, the Petworld tree is similar to a dataflow network, where information is passed up from the bottom of the tree, being processed or decided upon at each node in the hierarchy.

Typically, agents near the bottom of a tree tend to process information, as the MOVE agent above ranks different directions of motion on how well they approach food. Agents near the middle of the hierarchy tend to make content-based decisions. In the example pet below, the FORAGE agent decides whether to issue an EAT or MOVE command. Agents near the top of the hierarchy tend to perform the strategic decisions — for example the BRAIN5 agent will decide whether to FORAGE or COMBAT based on pet state.

Why a Hierarchy?

Using a hierarchy to encode knowledge offers several useful advantages:

Managing complexity: In a knowledge based system, it is easy to end up with thousands of knowledge elements. The knowledge must be structured — if not for efficiency in running the system, then for clarity maintaining it. Structure indicates redundant elements, for example common clauses in rules, and allows them to be centralized.

Controlling execution: Grouping ideas together suggests that some groups might control others. For example, one group of rules could resolve conflicts between other groups. (This is the root of meta-rules as described by Randall Davis.) It could resolve conflicts not only by having a built-in preference for one of the subordinates, but also by effecting a compromise between groups. If an animal is both scared and hungry, it might choose to run away in a direction that is favorable for finding food. Another possible resolution is called *displacement behavior*.

Dogs for example, when unable to decide between running from an enemy and fighting, sometimes abruptly sit down and scratch an itch. A related concept in AI is sometimes called *Papert's Principle*. If two experts conflict, both are ignored and a third is employed.

Increasing leverage: By taking the grouping strategy one step further and arranging experts into something resembling a hierarchy, a phenomenon similar to mechanical leverage arises: a little knowledge at the top of the hierarchy can have the same effect as attaching a precondition to many rules lower in the hierarchy.

What Kind of Hierarchy?

The organization of the hierarchy of experts controls the flow of decisions. There are several approaches to this form of hierarchical structuring. I examine them below.

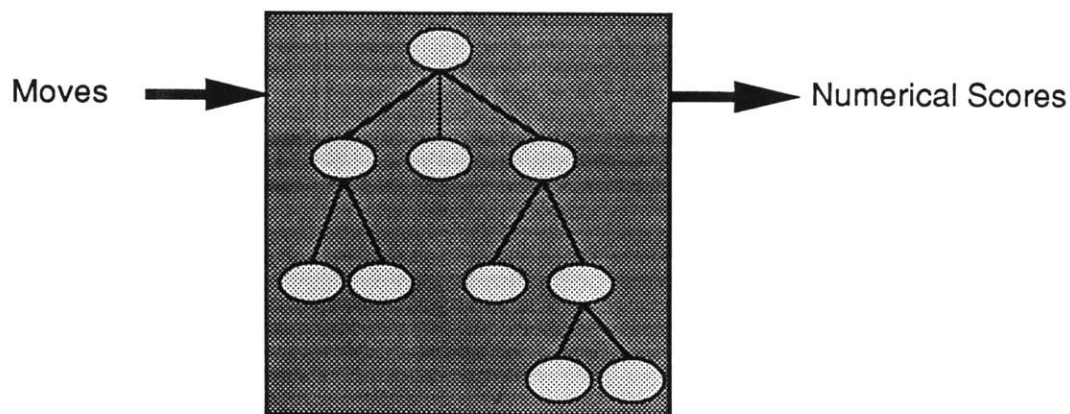


Figure 5: A decision tree.

- A simple approach, called a decision tree, is based on a science known as operations research, and is similar to how many computer games work. A move generator feeds the hierarchy. Each expert passes to its superior a numeric rating for each move based upon world state, pet state, and the opinions of its subordinate experts. The action that gets the best numeric rating executes.

This approach has two drawbacks. First, it evaluates a lot to find out a little — the entire tree might be calculated for dozens of possible moves. It also requires that compromises be finagled as weightings of the scoring functions, rather than being written out as rules. Some of the decision knowledge is hidden in the rating scheme, an undesirable circumstance.

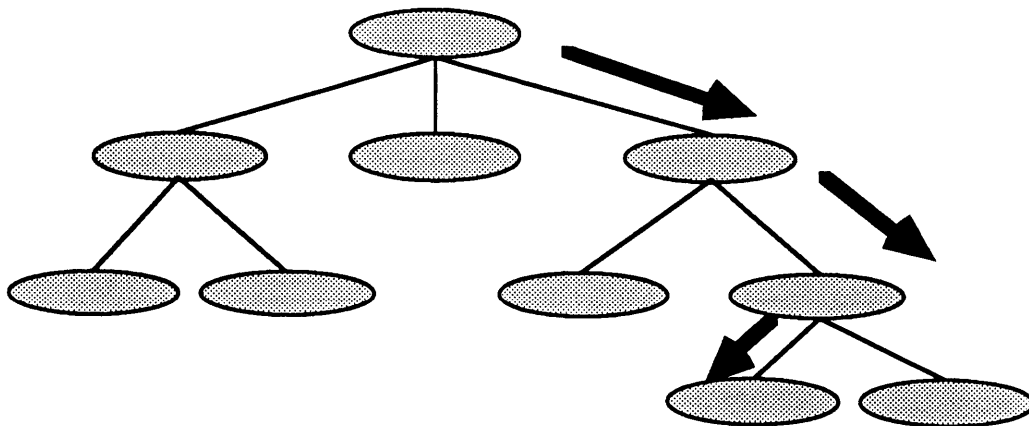


Figure 6: Branching Down.

- Another approach is similar to the meta-rule and grouping approaches extended to their logical extreme: control branches down the hierarchy by running rules in each expert along the path until an expert at the bottom of the tree is chosen. One of its rules then fires and action is finally taken.

This approach, although theoretically as powerful as any other, in practice still has the reconciliation problem. To effect a compromise between two experts, a “compromise between a and b” expert will have to be constructed, and more rules will have to be added to the superior expert to decide when to choose it.

- Marvin Minsky proposes a more complex version of an agent hierarchy in his recent book, *Society of Mind*. Agents take control, and can invoke other agents to assist them. Other agents also attempt to take control, and agent conflicts are an interesting part of behavior.

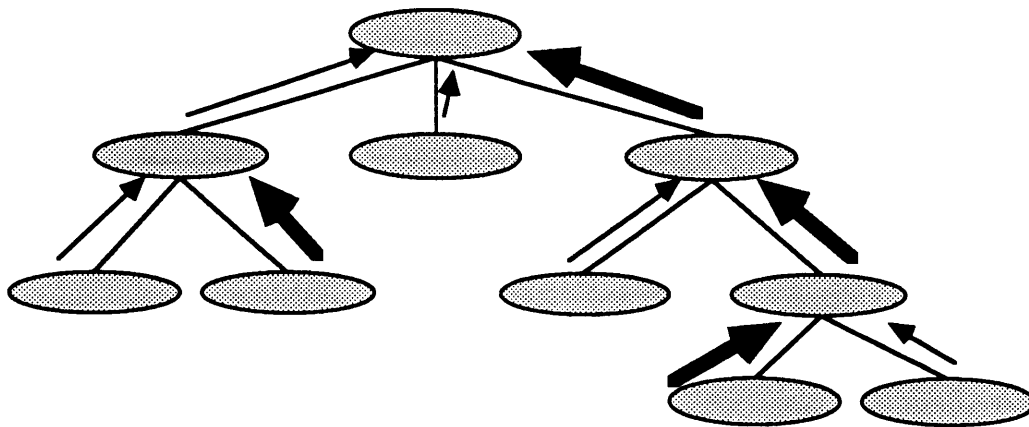


Figure 7: Bubbling up.

- In my chosen method, low-level experts “bubble up” rankings from which higher levels choose actions. Each expert will be concerned with compromising between actions presented to it, either by choosing one action, or by creating a new action. Knowledge about making decisions is stored explicitly in the experts. The experts output only a relative ranking, not an absolute one.

One of my other design constraints was to build pets with as little local state as possible. Although originally undertaken as an exercise in seeing how much behavior could be obtained with how little a brain, I have since decided the decision was a good one since it echos the constraints of most of the animals I saw my system as capable of simulating.

I encountered a problem implementing *concentration*, the ability to stay focused on one problem for some length of time. Concentration requires local state. The solution I chose adds a history of actions to the pet.

An example: pets must concentrate on finding food when they cannot see any. To find food, the pets employ a simple strategy (called EXPLORE in the example pet): turn around to look behind, and when that fails, go to another region on the board and try looking around there. (As soon as food comes into sight, the MOVE-TOWARD-FOOD agent will take over.) Some form of local memory is required to turn around just once, and then move for a long duration before trying again.

To this end, I have incorporated a notion of agent history to allow concentration. Each expert can remember a history of its recommendations. In the case of the exploring expert above, the mechanism allows it to turn around precisely once, and then move several times in a row in a straight line. This history mechanism allows much of the power of Minsky's Society of Mind, with a simpler mechanism.

Chapter 3: An Example Pet Analyzed

As an example of the Petworld system, I offer a sample pet which is capable of several basic behavior patterns: interacting with other pets, foraging for food, building a nest and staying in it once it is built, and exploring areas not visible. This sample brain was made by hand.

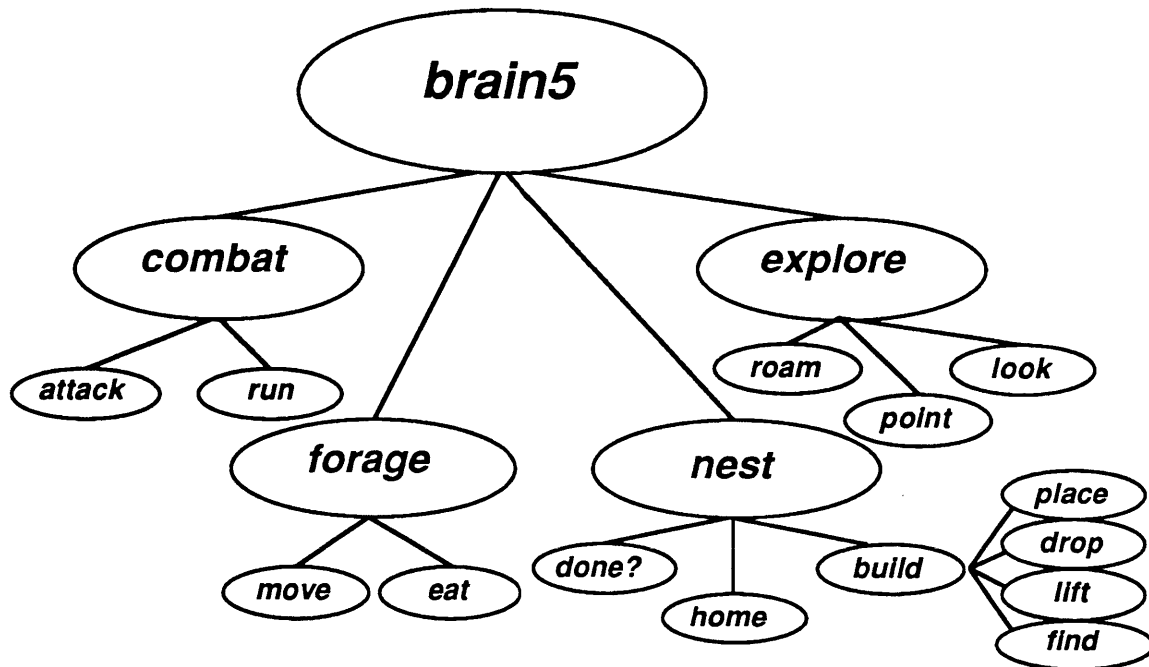


Figure 8: An example pet brain. This brain is discussed below.

The strategies of the pet are decided between by the topmost expert, BRAIN5. Finding food and interacting with other pets are most important, followed by building a nest, exploring the world, and homing into the nest.

In the case of conflicting recommendations, several decision strategies are pursued. If there is a conflict between foraging and combat, a compromise is made, dependent on how serious the danger of starvation or attack is. Thresholds are also used to ignore experts when the situation is stable. Assuming no life-or-death situations are imminent, nest building takes priority over exploring.

The basic combat strategy is to avoid other pets. Another pet is attacked only when the other is close enough and damage is low.

Foraging consists of either eating adjacent food, or moving toward the closest food. If the pet cannot see any food, the foraging expert recommends nothing.

Nest building models a strategy similar to one sometimes suggested for bird nest building: “If you are in your nest and have a rock, put the rock in the nest; otherwise get a rock and bring it back.” If the pet cannot see any rocks to get, then the nest building expert recommends nothing. If the nest is already completed, the nest building expert recommends returning to it.

Last is the exploring expert, which produces what Tinbergen calls “appetitive behavior.” It is called by default, when nothing else works. Since pets have limited sensory range, they will often be in places where no useful objects are visible. In that case, they should first turn around, then go somewhere else. As soon as the exploring expert brings something useful into view, another expert will be able to take control. Since the exploring expert is able to remember what it was doing previously, it can act as if it were performing long-duration actions.

Rules from the Sample Pet

BRAINS

1. If both HUNGER and FEAR are high, effect a tradeoff between COMBAT and FORAGE.
2. If FEAR is high, COMBAT.
3. If HUNGER is high, FORAGE.
4. If FORAGE is recommending that there is a food element immediately available, then FORAGE.
5. If BUILD has some non-trivial action to perform, then BUILD.
6. Otherwise, HOME.

COMBAT

1. If you have an available attack, and your damage is low, then recommend a tradeoff of attacking and running away.
2. Otherwise, recommend running away from any visible pets.

ATTACK

Construct a ranking of things you can attack standing right where you are.

RUN

Construct a ranking which scores movements on how well they bring you away from other pets. (This includes just the current positions of the other pets, and does not take into account their potential movement.)

FORAGE

1. If you are standing next to food, then recommend EAT.
2. Otherwise, recommend MOVE. If none are visible, no recommendation is made.

MOVE

Construct a ranking which scores movements on how well they bring you toward trees.

EAT

Construct a ranking of things you can eat standing right where you are.

NEST

1. If the nest is already built, then recommend HOME.
2. Otherwise, BUILD.

BUILD

In any of the following cases, if no forward progress can be made toward some destination, a displacement behavior of wandering randomly is undertaken.

1. If you have a rock, and you are standing in the right spot, then drop the rock. (DROP)
2. If you have a rock and are not in the right spot, then move toward the right spot. (PLACE)
3. If you don't have a rock, and are standing next to one that is not part of your nest, then pick it up. (LIFT)
4. If you don't have a rock, and are not standing next to one that is not part of your nest, then move toward the closest rock that is not part of your nest. (FIND)

FIND

Return a ranking scoring movements on how effectively they bring you towards the closest rock that is not already part of your nest. If no rock is visible, no recommendation is made.

PLACE

Return a ranking of movements on how effectively they bring you towards the top-ranked spot in the nest to place a rock. (The nest is defined as a ranking of squares in which to place rocks.)

HOME

Return a ranking scoring movements on how effectively they bring you towards “home” (the position on the board in which you started the simulation). The pet always knows where home is.

EXPLORE

Exploration is allowed when no other strategy fires, probably due to a lack of visible objects. I have chosen a very simple exploration strategy: first turn about to look all around, then move in a random direction a few units and try again.

1. Recommend LOOK enough times in a row so that one complete revolution is made.
2. Recommend POINT once, to attain a random direction to travel in.
3. Otherwise, recommend ROAM several times in a row.

LOOK

Turn clockwise by one field of view.

POINT

Turn some random amount.

ROAM

1. If you can't move forward, turn right 90 degrees.
2. Otherwise, move forward.

Petworld Anecdotes

Figure 1 shows a frame from a pet simulation run. Three pets are shown in the example, all with identical brains and all with display windows. (The practical limits on Petworld as implemented on a Macintosh II computer seem to be about 10 pets in a world about 50 by 50 elements, constrained by running speed of one brain cycle every five seconds.)

BRAIN5 contains several thresholds, but is surprisingly insensitive to the values used. My first raw guesses of setting thresholds at about two-thirds of the range of values worked well, and later tests at three-quarters and one-half range values made little difference until a critical value was reached. For example, the threshold of when to panic about finding food is well defined in relation to world size (*i.e.* how long it will take to get to food), and unrelated to the threshold of avoiding conflict.

Additionally, BRAIN5 is capable of *emergent behavior*. When two pets are building nests close to each other, the strategy of finding nearby rocks leads to pets poaching rocks from each others' nests, which in turn causes a great deal of conflict between pets. These conflicts are usually stalemates, and get resolved when one pet backs off to forage.

Also, because a pet cannot see its nest when it is far away from it, it will deposit rocks near the nest while returning from foraging expeditions. This behavior is in retrospect very advantageous, and an unexpected result of the interplay of several other behaviors.

SECTION TWO: SHERMAN — AGENT LEARNING IN THE PETWORLD

Chapter 4: Learning in a Hierarchy

Learning is characterized by an improvement in performance brought on by the useful acquisition of knowledge.

—Ron Rivest

Why Learn?

In Part One of this thesis, Petworld is presented as a pedagogical model of animal behavior. A simple hierarchy is used to drive the behavior, and can produce complex and even emergent behavior as a result of relatively simple code. The hierarchical structuring allows for *leverage* of the knowledge, where more general knowledge at the top of the hierarchy controls the use of knowledge developed lower down.

The most common criticism of Petworld was its lack of learning. Just about every animal learns at some level. Sea slugs can learn to associate temperature and light level with food concentration. Fish can learn a map, remembering food sources and hiding places in their territory. And primates can learn new strategies such as deception.

SHERMAN is a first attempt at incorporating machine-learning techniques in Petworld's hierarchical scheme. SHERMAN will present learning at an agent level — agents will have a learning engine and each one will learn separately. This technique will allow a large amount of knowledge to be acquired — knowledge for each agent to decide which subagent is likely to be helpful under which circumstances. But first an examination of common machine-learning techniques is in order.

How AI's Learn Now

There are several approaches to machine learning that are relevant to my work:

- *Learning From Examples* creates and refines a model of a concept based on positive and negative examples identified and presented by a teacher. Although much research has been done on the subject, the work of Winston remains definitive.
- *Case Based Reasoning* takes modelling one step further — models are made of experience *cases* from all aspects of planning in problem solving, and these model experiences are used to guide the system, and are refined and upgraded as experiences accumulate. The learning model employed in SHERMAN is based on key features of CBR, in particular the work of Kolodner.
- *Knowledge Based Systems* (KBS) offer excellent tools for modelling complex and poorly understood systems. Systems can be built incrementally, allowing refinement of the knowledge base, and knowledge can be compartmentalized, so a system can be specified in small chunks organized topically. Learning in KBSs typically tends to involve *chunking* productions together — learning the tendency of productions to fire in sequence. A thoroughly developed example is presented by Klahr.
- *Classifier Systems* (by Holland, among others) are very similar to the most simple of KBSs. The precondition and action are encoded as simple patterns, together with a score of how well the rule is perceived to function. No overall organization of the rule base is provided. New rules are created by simple pattern manipulations: randomly mutating an element, twisting a subpattern, or “crossing over” from one rule to another.
- *Neural Networks* operate by adjusting weights connecting simulated neurons in a lattice. The network runs by sending pulses from input gates, between neurons, and into the outputs. The network is capable of a form of learning commonly called *training*. Examples are presented as both input conditions and correct outputs, and a process called *simulated annealing* is used to adjust the weights to cause the network to correctly mimic the examples.

A Simplistic Model of Learning

Functionally, a learning system improves its behavior over time. It remembers past experiences, and uses those experiences as guidelines in the future. If a pet performs an action that results in a reward or punishment — for example, eating reduces hunger — it should remember that experience and use it in the future to achieve that reward again. Learning systems can be said to learn from their successes and failures.

So there are four pieces to a learning engine:

- An *experience memory* which contains information linking situations and actions,
- A *decision engine* which finds information in the memory relevant to the current situation and uses that knowledge to determine behavior,
- A *goal memory* which contains information about the desirability of conditions and actions, and
- A *feedback engine* which adds information to the memory based on the satisfaction of goals.

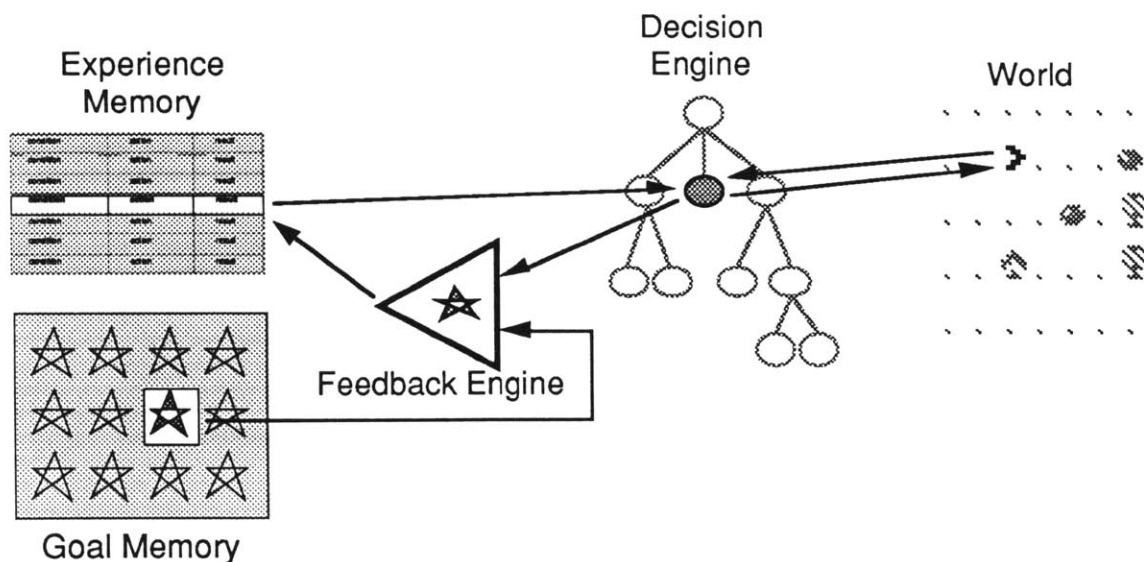


Figure 9: Schematic of the learning cycle.

SHERMAN will contain each of these parts in learning engines imbedded in certain agents in a Petworld brain.

The storage and retrieval mechanisms will be heuristically-driven, hard wired code components of the system, and will work identically for each agent. Each learning agent's memory will store experiences about which subordinate is good when, and will be able to extract common features of these experiences to build model experiences. These model experiences will allow knowledge to be extended by generalization, and then tested and updated.

Each agent will also have *goals* which provide feedback. The satisfaction of goals will cause experiences to be remembered, and the creation of new goals will allow knowledge to be extended.

About Goals

Learning always involves feedback. Feedback in SHERMAN will comprise a set of goals that pay off credit or blame when conditions in the pet or in the world are satisfied. I use the term *goal* in a simple sense: a condition of pet and world state that results in a reward or punishment. These payoffs affect the future behavior of the agent by educating models of when an action will trigger the payoff. Typical conditions that result in payoffs: hunger just went down, damage just went up, or the pet deposited a rock in the nest.

A feature of goals is that a clever strategy can create more of them as information is acquired, fleshing out the pet's behavior. This could occur by allowing a payoff in a higher level agent create a goal in a lower level agent, for example. When FORAGE pays off, it will be propitious to create a goal in EAT.

Goals can also be created. One technique creates goals by chaining new goals from the preconditions of old ones. A state that leads to the goal state is probably nearly as desirable as the goal state, so it would be propitious to create a secondary goal for that state.

These techniques will be discussed further in Chapter 7.

Chapter 5: How Agents Learn

Intelligent behavior is to be repeatedly successful in satisfying one's psychological needs in diverse, observably different, situations on the basis of past experience.
— P.J. van Heerden

As described before, the pets' intellect is made of a hierarchy of agents, each of which gets input from the world and its subordinates and passes on recommendations to its superiors in the form of rankings.

Chapter 4 presented an overview of the learning method of SHERMAN, where goals drove the hierarchy to strengthen pathways by remembering correct and incorrect decisions that were made.

The next few chapters will describe a proposed implementation of these tactics. First I will examine of the additions to the Petworld system to accommodate learning. Then I'll discuss the way that the new system runs, and how it acquires knowledge.

Additional Mechanisms for Learning

Several mechanisms will be needed for agents to learn. An *agent history* will let the agent perform actions that take more than one tick. *Experience memory* will let the agent remember under which circumstances subordinate agents are likely to provide winning recommendations. And a *manager* will direct the more complex execution of the new agent.

Agent history

Agent history is a record of what the agent has done. Since the agent can look back at the record, it can perform actions that take longer than one tick to execute (such as spinning around in place to look behind).

Each entry in the history will contain a record of the ranking that the agent recommended to its superior. It will also store the experience record, if any, on which the agent is basing its action, the subordinate agent which was chosen for

recommendation, and the other subordinate agents that recommended nearly the same things.

There will also be an operator that tells how many times in a row the same action was recommended at the top of the ranking. Thus, to spin once around, the pet should turn 90 degrees four times.

Experience memory

Each agent will also need a memory mechanism, for storing experiences about previous successes and failures. These experiences will be used in similar situations in the future.

Each experience has three parts: a precondition, an action, and a result. The precondition records the circumstance under which the action was taken, the action remembers which subordinate agent was chosen for execution, and the result tells the outcome — which goals, if any, were satisfied by the action. Common features of situations that caused the same payoff will be extracted into generalized models.

Manager

Each agent also will have a manager to control the execution of the decision strategy. The manager will fetch experience memories that are similar to the current situation, resolve any conflicts between the experiences, create an appropriate recommendation for the agent's superior, and create experiences to be stored in memory.

How Agents Run

The first step of an agent's cycle is the acquisition of sensory information from the world and from its subordinates. This information is provided in advance by the underlying Petworld system.

The manager will then retrieve from memory all of the experiences that are similar to the current situation, based on the preconditions of the experiences stored in

the agent: nearby objects in the world, the condition of the pet, and the recent actions recommended by the agent.

If there are no relevant experiences, the manager will pick an action either by default or at random. If there are too many experiences, it will resolve the conflicting experiences either by doing what's common to all, choosing one experience, or recommending nothing and allowing displacement behavior.

Once the manager has decided what to do, it will create an appropriate recommendation ranking to give to its superiors. It will also put into the agent history a record of which experiences it used, what recommendation it made, and which subordinate agents made that recommendation — both the one chosen by the manager and the other agents which happened to produce the same recommendation.

After the pet runs an action, credit — or lack of credit — should be assigned. For each agent on the path of execution, an experience record will be created with the precondition encoded from the situation, the action chosen by the manager, and whatever goals were satisfied. The memory unit will then attempt to incorporate the new experience with the existing model experiences, probably the ones that were recalled at the beginning of the loop.

Chapter 6: Memory and Modelling in Agents

Experience is what causes one to make new mistakes instead of old.

—from a fortune cookie

This chapter describes the organization and operation of the experience memory that will be part of every decision agent — the details of storing cases in an experience memory, the advantages and mechanics of compressing cases into models, and the retrieval of relevant experiences.

Many of the ideas concerning experience memory are based on a strategy called Case Based Reasoning, and in particular work by Janet Kolodner.

Details of Experiences

Each agent will have an experience memory that stores information about previous situations in records called *cases*. Each case has three parts: a *condition*, which describes the state of the world and the pet; an *action*, which tells what the pet did; and a *result*, which records payoffs the action triggered, if any.

Condition

The condition contains three parts: *world state*, *pet state*, and *agent history*.

World state contains the relevant objects in the world — the closest objects in each category, as well as the objects that are close enough to manipulate:

*the nest · the closest rock · the closest tree · the closest pet
rocks that are close · trees that are close · pets that are close*

The exact identity or position of the objects is less important than their existence or relative position, so objects will be recorded in body-centered coordinates relative to the pet. For example, under the category *closest rock* might be stored the coordinate (0 1), indicating the closest rock is one “south” of us.

Pet state is very simple: the variables hunger, fear, damage, and the flags payload, nest-complete, and in-nest are used from the Petworld system.

Agent history will incorporate the agent's most recent recommendations.

Action

The action part of an experience tells the agent which subordinate to recommend under the circumstances.

Result

The result part of an experience tells what resulted. If a goal paid off, the result is the name of the goal. If a certain goal was expected to pay off (*i.e.* if a model was being used to predict a result) and it didn't, the result is *not* <goalname>. If a payoff was neither expected or received, the result will be *no payoff*. If several payoffs occurred, then several experience records are created, one with each payoff.

Why Build Models?

Memory is a repository for the experiences used to guide the agents in the pet. Memory has two functions: storing experiences, and recalling poignant ones.

Yet if memory were merely to record and recall distinct experiences without structure, several problems would arise:

- It would require enormous amounts of storage. Much of the information contained in experiences is redundant.
- Finding experiences similar to a given situation would require an exhaustive search of the experience base, and an expensive metric of similarity applied to each experience.
- But most important, learning is impossible without structure. Consider experiences as discreet data points in the *condition-space* of situations. It should be possible to find regions of winning behavior by "connecting the dots" between known experiences, and then learning from the feedback if the generalization paid off.

The need to make models from experiences is clear. Modelling reduces storage requirements, metricizes experiences, speeds retrieval, and predicts trends.

Models, Clumps, and Cases

An agent's experience memory is filled with *cases* — condition-action-result triples. If these cases are grouped by result and action, they will define areas in condition-space where a given action has a given result, both by positive examples, which are always recorded, and by negative ones, which are recorded when the prediction of a model fails.

Modelling involves building *clumps* by collecting the relevant common features of cases. Models can contain clumps and cases, which together define a region of condition-space that a subagent will produce a predicted result. There might be a fairly general clump and a specific exception case, for example.

As a model improves, it shows more clearly the set of circumstances under which a given subagent is likely to pay off. And as the model becomes more general, it predicts areas of condition-space that are likely to produce the given result, causing the model to be more clearly defined.

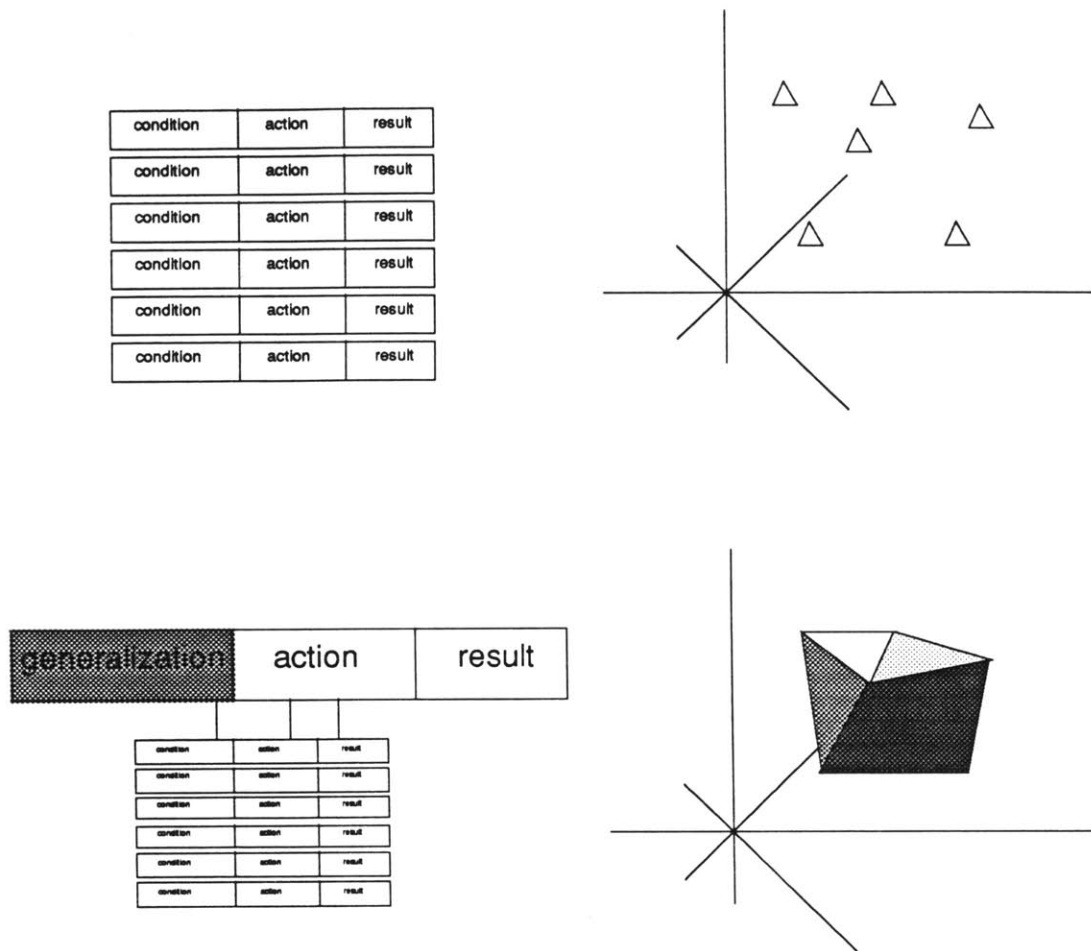


Figure 10: Cases and clumps. As clumps generalize cases, they predict regions of likely outcome.

Details of Models in SHERMAN

Clumps are built by the process of *generalization*. Cases that are similar are examined, and common *features* (condition slot values) are extracted. Features that are not the same among the cases are generalized — values are replaced with *generalization values* that matches all the cases. Since condition slots use such a simple representation, models can be constructed almost trivially via pattern matching techniques.

The three generalization values of condition slots are *some*, *don't care* and *this many*. When a slot is labelled *this many* with a number, it matches any conditions which have that many entries in the slot. A *don't care* matches zero or more entries in the slot. *Some* matches one or more. As before, an empty slot matches exactly

zero entries, and a slot filled in with a specific value matches that value. There is also a marker *inherit* for inheriting the slot value of a more general model.

This method allows for a uniform representation across a hierarchy of models. Specific cases can simply allow inherited parameters for some or all of the slots, and supply any missing or exceptional slot values.

Creating and Refining Models

Creation and refinement of models is driven by heuristics which are currently hard wired into the system. These heuristics are run whenever a new experience is acquired.

- When incorporating a new experience into a model, generalize an existing clump if there is one that fits well. Otherwise hold the experience separately until more experiences accumulate.
- Make the smallest generalization possible.
- Don't create a new clump until there are several (perhaps five or ten) similar experiences to generalize.
- If a set of experiences to be added is greatly variant from the target clump, make another clump.
- If the experiences to be added to a clump seem to have the same exception over and over again (*i.e.* the clump claims that eating requires a rock in a certain position and the experiences repeatedly indicate not), then the model is too specific and should have that feature removed.
- If the experiences seem to have the same additional feature over and over again (*i.e.* they all indicate that nearby food is necessary for eating), then the model is too general and should have that feature added.

Retrieving Relevant Cases

Retrieval of relevant experiences has a fairly straightforward goal: find the most specific models that cover the current situation. Breadth-first search is ideal for this process.

- If there are no models that are both similar to the given situation and more general than it, the agent can choose between several strategies: an action might be chosen at random, a default action could be used, or an overly-specific but similar model might be chosen.
- If retrieval yields more than one applicable model, their conflicting recommendations must be resolved. The simplest heuristic is to keep the most specific model. Yet it might be propitious to recommend the most popular action, or the action with the best potential payoff. It is important to keep the set of experiences small, but improvising between several models might lead to improved behavior.

Adding Cases to Memory

Often, the results of actions will provide useful experience information, and should be made into cases in memory.

- If the action chosen is based on one or more models, update each model by creating a case with either positive or negative result in each applicable model. Then try to generalize the model by creating or refining a clump.
- If the action chosen is not based on a model, then create a case only if the result is positive.

Conclusions about Modelling

Modelling reduces storage requirements, since most cases will end up shrinking to just a few differences from an appropriate clump.

Modelling allows experiences to be compared intelligently, since features that are not common in cases are not in the model. To find cases that are most similar to a given situation, use a breadth-first search. This will match the most relevant features first, since those features which maintain precise instead of general values will be the deciding ones.

Last, modelling allows for the fleshing out of knowledge, filling the gaps between known cases. When a new case arises, a recommendation can be made based on existing models and then tested, refining the operative region of the model.

SECTION THREE: CONCLUSIONS, APPLICATIONS, AND EXTENSIONS

Chapter 7: Additional Learning Agents

So far SHERMAN is able to learn which subagent is good when by deciding when goals are satisfied. This, however, is a very limited kind of learning since hierarchies only will learn about states immediately preceding the goal. SHERMAN is unable to learn to trade off between agents. SHERMAN also cannot reorganize the hierarchy, a mark of advanced learning systems. I'll briefly introduce several new learning mechanisms to address these concerns.

Goal Chaining

By adding goals to an agent, we can expand the circumstances in which a goal is used. Several heuristics can be employed to add new goals to an agent.

For a given *target goal*, there is a set of circumstances under which a certain action will achieve the goal. In the agent in question, this is represented by a condition-action-result model. But the *target condition* is almost as good as the target, since it is only one action away. So create a new, *secondary goal*, which rewards the achievement of the given condition.

An example: for the goal `HungerGoingDown` and the action `EAT`, the precondition will model the pet standing next to food. This condition becomes a secondary goal, which might be called `BeingNextToFood`.

This particular strategy for goal creation is called *goal chaining*, since it creates chains of goals leading to an eventual payoff, and resembles what machine learning calls *backward chaining goal propagation*.

If *the model itself* of the condition leading to the target goal is used as the secondary goal, then the new goal is continually refined as more information about the circumstances leading to the eventual goal is acquired.

As the target condition is refined, some secondary conditions might no longer lead to the payoff. Over time, these conditions will be discarded from the experience memory.

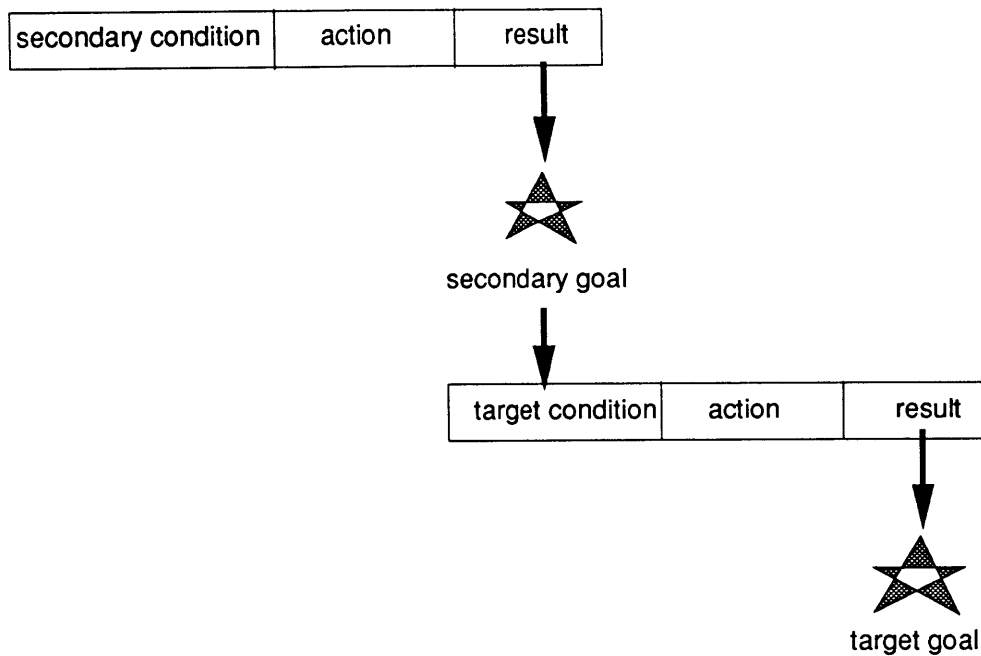


Figure 11: Goal chaining.

Agent Generation

Heuristic knowledge can also be used to generate agents. SHERMAN agents are already capable of learning which subagents are good when, and so can narrow down choices of subagents. Adding an agent generator will allow new agents to be added to the hierarchy, which will allow new structures to evolve. This technique is reminiscent of work by Lenat.

As an example, consider *tradeoff agents*: a new agent that returns a tradeoff between two existing agents. An agent is created when there are two agents that have overlapping circumstances where they might pay off. As an initial condition, the agent generator could suggest that the new agent would be useful in the intersection of regions of utility of the other two agents. The superior agent could then learn when a tradeoff is advantageous.

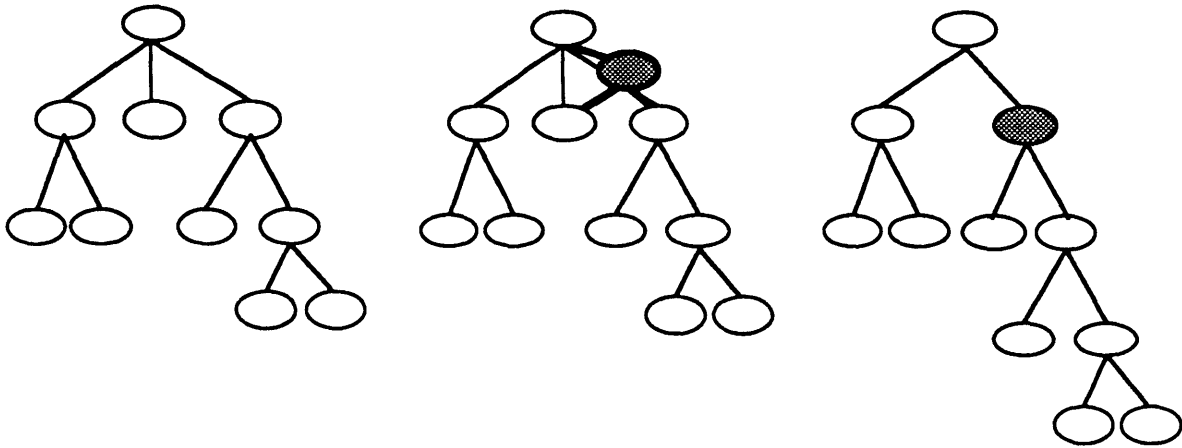


Figure 12: Agent generation. A tradeoff agent is generated, and proves to be more useful than the other two agents it replaces. It therefore causes a restructuring of the hierarchy.

Chapter 8: Conclusions and Applications

I've shown that SHERMAN is capable of performing a specific kind of learning — agents can learn which subordinate agent to choose when. It is also possible for agents to generate new goals from old ones, thus expanding the regions that can be learned. And it is possible for a hierarchy to grow new agents — adding to the repertoire of transformations on the ranking.

SHERMAN is a powerful system capable of learning on several levels and even building a structured network and adapting it to changing situations.

I expect that SHERMAN will offer insights into both pedagogical applications of learning techniques and into Machine Learning.

Applications

Up until now, I have shown how SHERMAN could learn from feedback from the environment as interpreted by a set of goals built into the starting hierarchy of agents. But feedback can come from several sources, including an “expert” that the pet is to be trained to be like. I'll call this form of learning *Learning By Imitation*. An expert might be a body of case knowledge, a set of rules, or a child “driving” the pet with a control panel. As the pet makes its suggestions, it is constantly corrected by the expert, and this knowledge is built up and tunes the network to behave more like the expert.

This is a pedagogical instance of the machine learning field of knowledge acquisition. It's clear that the network must be very well developed if it will be able to correctly classify the new knowledge and decide where in the hierarchy to store it (this is one of the hardest problems in knowledge acquisition), but this isn't really a problem in these pedagogical circumstances.

Now this form of learning — predictive learning based on mirroring the movements of an “expert” — is similar to the kind of learning we expect neural nets to do. But SHERMAN has the additional advantage of structure to its network, which would allow analysis of the structure of the knowledge acquired. Addi-

tionally, case-based memory can reorganize itself if necessary, can defer model-building until a clear consensus seems to be available, and can adapt to changing situations, allowing the use of a hierarchy from another body of knowledge to be used as a starter seed.

Acknowledgements

Dedicated to the memory of Thomas Trobaugh, who kept me going in times of trouble and provided the key insights that made the original Petworld happen.

Without Mario Bourgoïn, Mike Bove, Janet Kolodner, Henry Lieberman, and CJ Silverio, this thesis would not exist.

I'd like to thank Jim Davis, Max Hailperin, Alan Ruttenberg, and Mike Travers for their conspicuous insights.

The chocolate award goes to Steve Benton and Linda Peterson for administrative support above and beyond the call.

References

- Agre, Phil. "Routines." *MIT AI Laboratory Memo 828*. 1985.
- Agre, Phil, and Chapman, David. "A recipe is Not an Algorithm: What Plans Are and Aren't." *Slides from a talk*. 1987.
- Alkon, Daniel L. "Learning in a Marine Snail." *Scientific American*. 1983. pp. 70–84.
- Batali, John. "Computation Introspection." *MIT AI Laboratory Memo 701*. 1983.
- Becker, Joseph D. "A Model for the Encoding of Experiential Information." Bolt, Beranek, and Newman, Inc.
- Beer, Colin. "Philosophical Questions about Animal Intentionality." *Unpublished manuscript*. 1987.
- Braitenberg, Valentino. **Vehicles: Experiments in Synthetic Psychology**. MIT Press. 1984.
- Brooks, Rodney A. "Achieving Artificial Intelligence Through Building Robots." *MIT AI Laboratory Memo 899*. 1986.
- Burstein, Mark and Adelson, Beth. "Analogical Learning: Mapping and Integrating Partial Mental Models." *Unpublished manuscript*. 1986.
- Burstein, Mark and Adelson, Beth. "Learning by Analogy: A Concept Paper." *Unpublished manuscript*. 1986.
- Camhi, Jeffrey M. "The Escape System of the Cockroach." *Scientific American*. December 1982. pp. 158–172.
- Chapman, David, and Agre, Philip E. "Abstract Reasoning as Emergent from Concrete Activity." **Reasoning about actions and plans. Proceedings of the 1986 workshop**. Morgan Kaufman, 1987.
- Coderre, William H. "PETWORLD: An Animal Behavior System Using Rules." *Bachelor's thesis*. MIT. 1986.
- Davis, James R. "Pesce Retrospective," *Vivarium Technical Note*, MIT Media Laboratory, February 1988.
- Davis, Randall. "Meta-Rules: Reasoning about Control." *MIT AI Laboratory Memo 576*. 1980.
- Dennet, D. C. "Intentional Systems." *The Journal of Philosophy*. February 25, 1971. pp. 87–106.
- Drescher, Gary L. "Genetic AI: Translating Piaget into Lisp." *MIT AI Laboratory Memo 890*. 1986.
- Goldstein, Ira P. "Bargaining Between Goals." **Proceedings of the Fourth IJCAI**. 1976. pp. 175–180.
- Groelich, Horst. **Vehicles**. *Software package for Apple Macintosh*. MIT Press. 1986.

- Haase, Kenneth W., Jr. "TYPICAL: A Knowledge Representation System for Automated Discovery and Inference." *MIT AI Laboratory Technical Report 988*. 1987.
- Halbert, Daniel C. "Programming by Example." *Xerox Corporation Report OSD-T8402*. December 1984.
- Hoare, C. A. R. "Hints on Programming Language Design." *Stanford AI Laboratory Memo AIM 224*. 1973.
- Holland, John H. "Adaptive Algorithms for Discovering and Using General Patterns in Growing Knowledge Bases," *Policy Analysis and Information Systems*, v. 4(3), September 1980.
- Holland, John H. "Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems," *Machine Learning II*, 1986.
- Hofstadter, Douglas R. "The Copycat Project: An Experiment in Nondeterminism and Creative Analogies." *MIT AI Laboratory Memo 755*. 1984.
- Jacobs, Walter. "How a Bug's Mind Works."
- Johnston, Timothy D. "Introduction: Conceptual Issues in the Ecological Study of Learning." *Issues in the Ecological Study of Learning*. 1985.
- Kay, Alan C. "Computer Software." *Computer Software. Scientific American*, 1984.
- Kehler, Thomas P., and Clemenson, Gregory D. **KEE, The Knowledge Engineering Environment for Industry**. Intelligenetics, Inc., 1983.
- Kolodner, Janet L. "Maintaining Organization in a Dynamic Long-Term Memory." *Cognitive Science* v. 7, pp. 243-280, 1983.
- Kolodner, Janet L. "Experiential Processes in Natural Problem Solving." GIT-ICS-85/23, Georgia Institute of Technology. 1985.
- Kolodner, Janet L. "Case-Based Inference: A Collection of Papers." GIT-ICS-87/18, Georgia Institute of Technology. 1987.
- Lebowitz, Michael. "Generalization from Natural Language Text." *Cognitive Science* v. 7, pp. 1-40, 1983.
- Lehnert, Wendy G. "Case-Based Reasoning as a Paradigm for Heuristic Search." University of Massachusetts, Amherst.
- Lenat, Douglas B. "Beings: Knowledge as Interacting Experts." *Proceedings of the Fourth IJCAI*, pp. 126-133, 1975.
- Lenat, Douglas B. "The Role of Heuristics in Learning by Discovery: Three Case Studies." *Machine Learning*, 1983.
- Lenat, Douglas B., and Brown, John S. "Why AM and EURISKO Appear to Work." *Artificial Intelligence*. 1984. pp. 269-294.

- Lenat, Douglas B., and Harris, Gregory . "Designing a Rule System that Searches for Scientific Discoveries." *CMU Department of Computer Science*. 1977.
- Lorenz, Konrad. **King Solomon's Ring**. Thomas Y. Crowell Company, 1952.
- Macphail, Euan M., "Ecology and Intelligence," **Memory Systems of the Brain**, Guilford Press, 1985.
- MacLaren, Lee S. "A production system architecture based on biological examples." *Doctoral thesis*. University of Washington, Seattle. 1978.
- Maruichi, Uchiki, and Tokoro. "Behavioral Simulation Based on Knowledge Objects." *Proceedings, ECOOP '87*. **Springer-Verlag Lecture Notes in Computer Science #276**. 1987.
- Minsky, Marvin. **The Society of Mind**. Simon and Schuster, 1986.
- Rissland, Edwina L. and Ashley, Kevin D. "Credit Assignment and the Problem of Competing Factors in Case-Based Reasoning." University of Massachusetts, Amherst. 1988.
- Robot Odyssey I. *Computer program*. The Learning Company.
- Selfridge, Oliver G. **Tracking and Trailing: Adaptation in Movement Strategies**, *unpublished manuscript*, 1978.
- Stefik, Mark, et al. "Knowledge Programming in Loops: Report on an Experimental Course." *AI Magazine*. Fall 1983.
- Scientific American, eds. **Brain: A Scientific American Book**. 1979.
- van Heerden, P.J. "The Foundation of Empirical Knowledge." Wassenar, The Netherlands: Wistik. 1968.
- Wilson, Stewart W. "Knowledge Programming in an Artificial Animal" Rowland Institute for Science. 1986.
- Winston, Patrick H. "Learning New Principles from Precedents and Exercises: The Details." *MIT AI Laboratory Memo 632*, 1981.
- Winston, Patrick H. "Learning by Augmenting Rules and Accumulating Censors." *MIT AI Laboratory Memo 678*, 1984.
- Winston, Patrick H., Binford, Thomas O., Katz, Boris, and Lowry, Michael. "Learning Physical Descriptions from Functional Definitions, Examples, and Precedents." *MIT AI Laboratory Memo 679*, 1983.