# Two Kinds of Training Information for Evaluation Function Learning

Paul E. Utgoff
Jeffrey A. Clouse
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003 U.S.A.

COINS Technical Report 91-11
February 1, 1991

## Abstract

The paper identifies two fundamentally different kinds of training information for learning search control in terms of an evaluation function. Each kind of training information suggests its own set of methods for learning an evaluation function. The main result is the conclusion that one can and should integrate the methods and learn from both kinds of information.

# Contents

# 1 Introduction

This paper identifies two fundamentally different kinds of training information for learning search control knowledge in terms of an evaluation function. The main point is that one can and should seek to learn from both kinds of information, rather than be concerned with which one is better than the other. Both kinds of information are often available, and there is no point in ignoring one of them.

The discussion focuses on the problem of learning an evaluation function for the purpose of making control decisions during search. An evaluation function maps each state to a number, thereby defining a surface that can be used to guide search. If the number represents a reward, then one can search for a sequence of control decisions (operator applications) that will lead to the highest foreseeable payoff. Analogously, if the number represents a cost or penalty, then one searches for a minimizing sequence.

# 2 Sources of Training Information

There are two fundamental sources of training information for learning an evaluation function. The first is the payoff achieved by executing a sequence of control decisions from a particular starting point. Sutton (1988) has illustrated via his temporal difference (TD) methods that one can learn to predict future value from a state by repeatedly correcting an evaluation function to reduce the error between the local evaluation of a state and the backed-up value that is determined by lookahead search. This is similar to an idea of Samuel (1963), but Sutton has broadened it considerably and related it to several other lines of thought.

The second source of training information is identification of the control decision made by an expert, given a particular state. In the literature, such an instance of an expert choice istypically called a *book move* (Samuel, 1967), but it need not have been recorded in a book. Instead, one can simply watch an expert in action, or ask an expert what to do in a particular situation, and thereby obtain the control decision that the expert would make. Whenever an expert's choice is available, one would like to be able to learn from it. Such a choice is the result of the expert's prior learning, and therefore should be quite informative. Indeed, learning to make the same choices as an expert would be a sensible approach to building an expert system.

# 3 State Preference Methods

When making a control decision based on the value of each successor state, the exact value of a state is irrelevant with respect to making the choice. Only the relationship of two values is needed for the purpose of identifying the one with the higher value. The objective is to identify the most preferred state and then move to it. We refer to a method that selects a successor state by identifying a most preferred state as a *state preference (SP)* method. Many of the standard search algorithms that employ an evaluation function for the purpose of selecting a most preferred succesor state are examples of state preference methods. Such algorithms include hill-climbing and best-first search.

Given that a control decision does not depend on the particular values returned by an evaluation function, one does not need to learn an exact value for each state. One needs

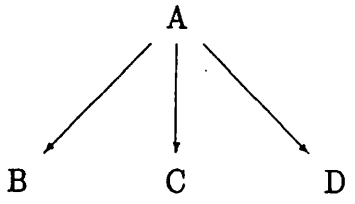


Figure 1. One Ply of Search.

only to learn a function in which the relative values for the states are correct. Whenever one infers, or is informed correctly, that state $a$ is preferrable to state $b$, one has obtained information regarding the slope for part of a correct evaluation function. Any surface that has the correct sign for the slope between every pair of points is a perfect evaluation function. An infinite number of such evaluation functions exist, under the ordinary assumption that state preference is transitive. One would expect the task of finding any one of these evaluation functions to be easier than the task of finding some particular evaluation function.

Because one wants to learn to select a most preferred state from a set of possible successors, one should be able to learn from examples of such choices (Utgoff & Saxena, 1987; Utgoff & Heitman, 1988). By formally stating the problem of selecting a preferred state, and expanding the definitions, a procedure emerges for converting examples of state preference to constraints on an evaluation function. One can then search for an evaluation function that satisfies the constraints, using standard methods.

Assume that a state $x$ is described by a conjunction of $d$ numerical features, represented as a $d$-dimensional vector $\mathbf{F}(x)$. Also assume that the evaluation function $H(x)$ is represented as $\mathbf{W}^T\mathbf{F}(x)$, where $\mathbf{W}$ is a column vector of weights, and $\mathbf{W}^T$ is the transpose of $\mathbf{W}$. Then one would compare the value of a state $C$ to a state $B$ by evaluating the expression $H(C) > H(B)$. In general, one can define a predicate $P(x, y)$ that is true if and only if $H(x) > H(y)$, similar to Huberman's (1968) hand-crafted *better* and *worse* predicates. One can convert each instance of state preference to a constraint on the evaluation function by expanding its definitions. For example, as shown in Figure 1, if state $C$ is identified as best, one would infer constraints $P(C, B)$ and $P(C, D)$. Expanding $P(C, B)$, for example, leads to:

$$P(C, B)$$
$$H(C) > H(B)$$
$$\mathbf{W}^T\mathbf{F}(C) > \mathbf{W}^T\mathbf{F}(B)$$
$$\mathbf{W}^T(\mathbf{F}(C) - \mathbf{F}(B)) > 0$$

By expanding all instances of state preference in this way, one obtains a system of linear inequalities, which is a standard form of learning task for a variety of pattern recognition methods, including perceptron learning (Duda & Hart, 1973) and other more recent connectionist learning methods. Note that these instances of state preference are expressed as $d$-dimensional vectors, meaning that learning from pairs of states is no more complex than

learning from single states. This is in contrast to Tesauro (1989), where both states are given as input to a network learner.

It is worth noting that Samuel's (1967) method for learning from book moves is an SP method. When learning from book moves, Samuel computed a correlation coefficient as a function of the number of times $L$ ($H$) that the feature value in a nonpreferred move was lower (higher) than the feature value of the preferred move. The correlation coefficient for each feature was $\frac{L-H}{L+H}$, and was used directly as the weight in his evaluation function. The divisor $L+H$ is constant for all features, serving as a normalizing scalar, and can be ignored. Thus the total $L-H$ is a crude measure of how important the feature is in identifying a state preferred by an expert. Tesauro (1988) also uses a relative scoring of positions for training.

# 4 Illustration of a TD and an SP Method

This section illustrates a TD method and an SP method applied individually to the same problem. The purpose is to ground the discussion of the previous sections, and to get some indication of which source of training information leads to faster learning. However, the comparison is not a contest because we are not advocating picking one. Each method learns from a different kind of training information, so it should be no surprise that one learns more quickly than the other. Instead, in Section 5, we observe that one can and should learn from all the training information, regardless of its source.

## 4.1 The Task Domain

Although we are experimenting with TD and SP methods in larger domains, discussed in the final section, we have selected the smaller Towers of Hanoi domain for pedagogical purposes. The main reason for this choice is that the domain is characterized by a small state space, which can be controlled by varying the number of disks. The small domain makes it possible to implement a simple expert, which is needed as a means of providing state preference information for the SP method.

The semantics of this problem makes it more natural to think of the value of a state as a measure of remaining cost. Thus, the goal state would have the lowest cost. The problem-solving program will be looking for a state with a low cost, and one state will be preferred to another if its cost is lower. Expanding $P(x,y) \leftrightarrow H(x) < H(y)$ leads to a constraint of the form

$$\mathbf{W}^T(\mathbf{F}(x) - \mathbf{F}(y)) < 0.$$

## 4.2 A Temporal Difference Method

The temporal difference method uses a hill climbing search strategy for problem solving. At each step, the algorithm generates the successors of the current state and evaluates them. The value backed-up to the current state is the value of the lowest cost successor plus 1. This backed-up value is the desired value of the current state, and the learning mechanism adjusts the weights $\mathbf{W}$ so that the evaluation for the parent state is equal to the backed-up value. Because the value of the goal state is defined to be 0, the evaluation function is trained to predict the distance remaining from the state to the goal. After the training occurs, the problem solver moves to the chosen state. A training episode consists of picking a start state, and proceeding in this manner until the goal state is reached. The error correction rule is

Table 1. Number of Training Instances Needed.

| Method | 3 disks | 4 disks | 5 disks | 6 disks |
|---|---|---|---|---|
| TD | 129 | 13,203 | 66,044 | >415,282 |
| SP | 32 | 256 | 512 | 6,144 |

similar to the well known absolute error correction rule (Nilsson, 1965; Duda & Fossum, 1966), but the error is corrected exactly. One solves

$$(\mathbf{W} + c\mathbf{F}(x))^T\mathbf{F}(x) = \textit{backed-up value}$$

for $c$ and then adjusts $\mathbf{W}$ by

$$\mathbf{W} \leftarrow \mathbf{W} + c\mathbf{F}(x)$$

so that $\mathbf{W}^T\mathbf{F}(x)$ has the intended value.

### 4.3 A State Preference Method

The state preference method uses a best-first search strategy. Before the training begins, the optimal solution path from the chosen start state to the goal state is identified via a brute-force search. This step is included solely for the purpose of obtaining state preference training information, which would ordinarily come from an expert. At each step, the state preference method consults the expert for the correct node to select from the frontier and expand. From this choice, the algorithm infers that the selected state is to be preferred to each of the nonselected states. From each such pair of states, the SP method infers a constraint on the weight vector $\mathbf{W}$ expressed as a linear inequality, as described above. If the constraint is not satisfied, then the weight vector $\mathbf{W}$ is adjusted. The correction rule is a form of the absolute error correction rule. One solves

$$(\mathbf{W} + c(\mathbf{F}(x) - \mathbf{F}(y)))^T(\mathbf{F}(x) - \mathbf{F}(y)) = -1$$

for $c$ and then adjusts $\mathbf{W}$ by

$$\mathbf{W} \leftarrow \mathbf{W} + c(\mathbf{F}(x) - \mathbf{F}(y)).$$

Adjusting the weights so that the weighted difference is $-1$ corresponds to wanting the selected state to evaluate to one less than a nonselected state. One could pick any negative value in order to become correct for the inequality. After the training is complete, the successors of the chosen state are placed on the frontier, and one training step is complete. The algorithm repeats this process until the goal state is reached.

### 4.4 Discussion

The TD method and the SP method were each trained repeatedly until each was able to solve problems optimally. The cost of training is measured as the total number of adjustments to the weight vector $\mathbf{W}$. Table 1 shows that the state preference method is much cheaper than the temporal difference method. This is in accord with the intuition that one can learn

Table 2. Features for the 3-Disk Problem.

| Feature |
|---|
| Is Disk 3 on Peg 3? |
| Is Disk 2 at its desired location? |
| Is Disk 1 at its desired location? |
| Is Disk 2 on Disk 3? |
| Is Disk 1 on Disk 3? |
| Is Disk 1 on Disk 2? |
| Is Disk 2 on Peg 3? |
| Is Disk 1 on Peg 3? |
| Is Disk 3 clear? |
| Is Peg 3 empty? |
| Distance from class mean |
| Threshold |

more quickly from state preference information than temporal difference information. As mentioned above, given that the methods make use of distinctly different sources of training information, there is no need to pick one method instead of the other. One would like to pick them both, and this is discussed below in Section 5. For the 6-disk problem, the TD method terminated prior completion due to a floating point error. supplied).

The features for describing a state are a function of the number of disks. For the 3-disk problem, there are 27 possible puzzle states and 11 features. Table 2 shows the features for the 3-disk problem. All of these features are binary-valued, except for distance-from-class-mean. In general, the number of states and features for the $n$-disk problem is $O(3^n)$ and $O(n^2)$ respectively. The constant for the number of features is 0.5. As the number of disks increases, the number of states outpaces the number of features.

# 5 Integrating TD and SP Methods

This section discusses the relationship between TD and SP methods, and shows that both kinds of methods can work together in learning one evaluation function.

## 5.1 Relationship of TD and SP Methods

TD methods learn to predict future values, whereas SP methods learn to identify preferred states. For TD methods, training information is propagating vertically up the search tree. For SP methods, the training information is propagating horizontally among siblings. A TD method is unsupervised because backed-up values are inferrable from search and evaluation; no external teacher is needed in order to assign a value to a state. An SP method is supervised because a teacher or other external expert is needed in order to identify which state from a set of successors is most preferred.

Semantically, the two kinds of methods are compatible because the evaluation function is of the same form, and serves the same purpose of allowing identification of a best state. One can adjust the weights $\mathbf{W}$ so that the value of a state is predictive of its eventual payoff, and

Table 3. Number of Training Instances Needed.

| Method | 3 disks | 4 disks | 5 disks | 6 disks |
|---|---|---|---|---|
| TD | 129 | 13,203 | 66,044 | >415,282 |
| TDSP | 34 | 622 | 34,400 | 84,348 |
| SP | 32 | 256 | 512 | 6,144 |

one can also adjust $\mathbf{W}$ so that the relative values among the states become correct. Thus, in terms of the semantics of the learning, one can simply apply both kinds of error correction to the same evaluation function without fear that they are incongruous. However, a practical problem that can arise is that the expert might be fallible, putting the two sources of training information in conflict to some degree. This issue is discussed below.

### 5.2 An Integrated Method

In the same way that TD and SP are each a class of methods, there are many combinations of methods that would produce an integrated TDSP method. We present one such method here.

Given that TD is an unsupervised method, we assume that a learning program will train on such information whenever it is available. In particular, whenever a node is expanded, one can train the evaluation function on the difference between the local evaluation of the parent, and the backed-up value of the child with the best value. In the context of problem solving, the TD rule for computing correction $c$ is

$$(\mathbf{W} + c\mathbf{F}(x))^T\mathbf{F}(x) = \textit{backed-up value} + \delta.$$

The $\delta$ is a positive constant, e.g. 1, so that a more expensive state will have a higher value than a less expensive state.

The SP method relies on an expert, which can be a human or a search procedure. We assume that a learning system will want to query the expert as little as possible, or that the expert will only respond to a portion of the queries it receives. As an extreme, one could avoid querying the expert altogether, and learn only from the TD information. However, expert preferences provide strong training information and should be used when available. In the TDSP method described here, one queries the expert whenever a node is expanded and the TD error is above 0.1, where the error is $\frac{|backedup-local|}{|backedup|}$. The effect is that the expert exerts great influence early in the training, but is progressively ignored as the evaluation function becomes more accurate. The SP rule for computing correction $c$ is $(\mathbf{W} + c(\mathbf{F}(x) - \mathbf{F}(y)))^T(\mathbf{F}(x) - \mathbf{F}(y)) = -\delta$. The rule is applied only if the weighted vector difference is greater than $-\delta$.

Table 3 shows the number of instances that the TDSP method required to learn various-sized Tower of Hanoi tasks. To facilitate comparison, the results from the TD-only and SP-only methods are repeated. As expected, learning is faster for TDSP than plain TD. Table 4 shows how far the number of queries to the expert declined during training, but does not show that the number of queries drops rapidly. The table shows the number of queries made while training on the first problem, and the last problem, but not for any of

Table 4. Number of Queries to Expert.

| | 3 disks | 4 disks | 5 disks | 6 disks |
|---|---|---|---|---|
| Queries First Problem | 7 | 50 | 138 | 359 |
| Queries Final Problem | 7 | 17 | 17 | 21 |
| Total Problems | 1 | 4 | 269 | 242 |

the intermediate problems. In the 3-disk case, learning was complete after the first problem, so it is both first and last.

The TDSP method described here increasingly ignores the expert as the evaluation function is learned. This is a desirable characteristic in terms of gaining autonomy, but it is also desirable if the expert is imperfect, e.g. human. One can learn rapidly from the expert, and then let TD training correct any flaws that may have crept in from believing the expert. We are in the process of experimenting with various kinds and degrees of imperfection in the expert. Too much error would overwhelm the TD contribution to learning, and we would like to be able to detect this situation.

## 6 Conclusion

We have identified two different kinds of training information for learning evaluation functions, and described their relationship. For state preference, we have shown that one can convert instances of state preference to constraints on an evaluation function, and that one can learn an evaluation function from such information alone. We have pointed out that one should be able to learn from all sources of training information, and not be diverted by arguments that one should be favored over another. We have observed that it is semantically correct to apply a TD method and an SP method simultaneously to the learning of one evaluation function. Finally, we presented a specific method that integrates both approaches, and demonstrated that the two can indeed work together.

Although we have chosen a simple problem for illustration, the issues that motivated this work arose while studying the effectiveness of TD and SP methods in the game of Othello. The program was able to learn from either source of information, but it was unclear whether or how one could learn simultaneously from both sources. We are in the process of finishing the integration of the methods in Othello, and are in the early stages of experimenting with an integerated approach in learning to control air traffic.

### Acknowledgments

This material is based upon work supported by the National Aeronautics and Space Administration under Grant No. NCC 2-658, and by the Office of Naval Research through a University Research Initiative Program, under contract number N00014-86-K-0764. We thank Rich Sutton for a useful discussion, and Sharad Saxena, Jamie Callan, Tom Fawcett, Carla Brodley, and Margie Connell for helpful comments.

## References

Duda, R. O., & Fossum, H. (1966). Pattern classification by iteratively determined linear and piecewise linear discriminant functions. *IEEE Transactions on Electronic Computers, EC-15*, 220-232.

Duda, R. O., & Hart, P. E. (1973). *Pattern Classification and Scene Analysis.* New York: Wiley & Sons.

Huberman, B. J. (1968). *A Program to Play Chess End Games.* Doctoral dissertation, Department of Computer Sciences, Stanford University.

Nilsson, N. J. (1965). *Learning Machines.* New York: McGraw-Hill.

Samuel, A. (1963). Some studies in machine learning using the game of checkers. In E. A. Feigenbaum, & J. Feldman (Eds.), *Computers and Thought.* New York: McGraw-Hill.

Samuel, A. (1967). Some studies in machine learning using the game of checkers II: Recent progress. *IBM Journal of Research and Development, 11*, 601-617.

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

Tesauro, G., & Sejnowski, T. J. (1988). A parallel network that learns to play backgammon: Recent results. *Proceedings of the AAAI Symposium on Computer Game Playing* (pp. 41-45). Palo Alto, CA.

Tesauro, G. (1989). Connectionist learning of expert preferences by comparison training. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems.* Morgan Kaufmann.

Utgoff, P. E., & Saxena, S. (1987). Learning a preference predicate. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 115-121). Irvine, CA: Morgan Kaufmann.

Utgoff, P. E., & Heitman, P. S. (1988). Learning and generalizing move selection preferences. *Proceedings of the AAAI Symposium on Computer Game Playing* (pp. 36-40). Palo Alto, CA.