# Learning from Perfection

## A Data Mining Approach to Evaluation Function Learning in Awari

Jack van Rijswijck

Department of Computing Science,
University of Alberta,
Edmonton, Alberta, Canada T6G 2H1
javhar@cs.ualberta.ca

**Abstract.** Automatic tuning of evaluation function parameters for game playing programs, and automatic discovery of the very features that these parameters refer to, are challenging but potentially very powerful tools. While some advances have been made in parameter tuning, the field of feature discovery is still in its infancy. The game of Awari offers the possibility to achieve both goals. This paper describes the efforts to design an evaluation function without any human expertise as part of the Awari playing program *Bambam*, as being developed by the Awari team[1] at the University of Alberta.

**Keywords:** Machine learning, heuristic search, game playing, alpha-beta, Awari.

## 1 Introduction

An evaluation in a game playing program typically takes as input a vector of *feature values* that have been extracted from the game position. These features, to be called the *base features*, are typically not those that form the actual description of the game position. The features that describe the game state are the *atomic features*. For instance, the atomic features in chess are the states of the 64 squares; examples of states are "empty", "white pawn", and "black rook". The base features may include concepts such as mobility, passed pawns, and king safety.

Since the choice of base features impacts both the speed of the evaluation function and its ability to approximate the game theoretic value of a position, the automatic discovery of base features, if it is possible, is a potentially very powerful tool. The notion of automatically discovering features "from scratch" is very attractive, but unfortunately seems to be too far fetched for most games. Humans have typically already identified key features in a given game. A language that allows these features to be expressed is usually either so high level that it is specialized and biased towards these features, or it is low level and the feature descriptions are discouragingly complex. In neither case is there much hope of discovering genuinely new high level features.

The ideological objection against adding human defined base features often leads to machine learning systems that are indeed able to make progress from their initial

[1] See http://www.cs.ualberta.ca/~awari.

random state, but whose performance falls far short of that of existing world class programs. For our purposes, the main concern is to come up with a program that plays as strongly as possible. Whether or not it includes human knowledge, the performance of the program should in this context be compared to that of existing strong programs.

One situation in which feature discovery from scratch is relevant is when there is little or no human strategic knowledge available for the game at hand. This is the case in the game of Awari. While developing the Awari playing program *Bambam*, the Awari team at the University of Alberta has been able to come up with only a handful of high level features that may or may not be important in Awari. Perhaps better features can be constructed.

The outline of this paper is as follows. Section 2 introduces the game of Awari and its rules. Section 3 describes the problem of evaluation function learning and lists some previous work that has been done in this area. Base features and evaluation functions were learned automatically via methods explained in Section 4, as well as hand-developed by humans as reported in Section 5 to enable a comparison. Results are presented in Section 6, followed by conclusions and a discussion.

## 2 Awari

Awari is one of a family of ancient African games, played with seeds or pebbles that are distributed over rows of pits. The variant described here is the one used in the Computer Olympiad. The Awari board consists of two rows, each containing six pits. Figure 1 shows the starting position of Awari, where each pit contains four pebbles. The two players are commonly called *North* and *South*. The six pits on the bottom row, named '$a$' through '$f$', form South's *home area*. Pits '$A$' through '$F$' are North's home area. The pits 'a' and 'A' are the *first pit* of the respective players.

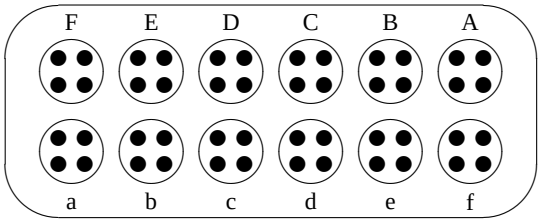


**Fig. 1.** The starting position in Awari

A move consists of taking all of the pebbles from one of the pits in the home area, and "sowing" them counterclockwise around the other eleven pits of the board by dropping one pebble in each consecutive pit. If more than eleven pebbles are sown, then the starting pit is skipped. There is no limit on how many pebbles a single pit can contain. Pebbles can be captured, and the object of the game is to capture more pebbles than the opponent.

Capturing occurs in *vulnerable pits*. A pit is vulnerable if it contains two or three pebbles after the sowing is completed. If the last pebble lands in a vulnerable pit, then the pebbles in that pit are captured by the sowing player. A multiple capture occurs when the preceding pit is also vulnerable; capturing then continues until a non-vulnerable pit is reached. Captures only occur in the opponent's zone; a multiple capture cannot continue beyond the opponent's first pit.[2]

Since the number of pebbles on the board decreases throughout the game, it is possible to build endgame databases that contain the result with perfect play for each possible configuration up to a given number of pebbles. The Awari team at the University of Alberta has built databases that contain all positions containing up to 35 pebbles. This is only 13 pebbles away from the starting position, which is seemingly quite close. However, it can take many moves before 13 pebbles are captured; experiments suggest that this may take on the order of 50 ply with perfect play. Before the stage of the game is reached where the endgame databases can be used, there still is the need for a good evaluation function.

## 3 Evaluation function learning

The biggest problem that evaluation function learners face is getting feedback on how good any given instance of the function is. This is usually solved by playing many games and using reinforcement learning techniques. However, in Awari we have a radically different situation. The quality of an evaluation function can be measured directly against the perfect information in the endgame databases.

### 3.1 Previous work

Previous attempts to mine endgame databases include research done by Terry van Belle [9] and Paul Utgoff et al. [7, 8]. Van Belle attempted to apply genetic algorithms to checkers endgame databases, which proved to be unsuccessful. Utgoff developed the ELF learning algorithm, which was able to make progress in identifying features in checkers by using reinforcement learning techniques rather than learning from endgame databases. However, the program did not reach a high level of play when compared to other checkers playing programs.

Human efforts to gain understanding from endgame databases include work by Michie, Roycroft and Bratko [2], as well as John Nunn [4], all using chess databases. Most attempts have had limited success, with Nunn's impressive work [3, 5, 6] being a favourable exception. All of these results have only come with a considerable investment of time.

Automatic feature discovery and evaluation function construction without using perfect information was done by Buro, with experimental results for the game of Othello [1]. Buro introduced GLEM, a Generalized Linear Evaluation Model, and was able to extract useful features that led to an improvement in playing strength. Each feature

[2] There are several additional rules that deal with specific situations; see `http://www.cs.ualberta.ca/~awari/rules.html` for a detailed explanation.

corresponds to a set of board squares, and weights are assigned to every possible instantiation of the squares question. This leads to a table containing on the order of 1.5 million weights, which were tuned using a hill climbing approach.

The Awari application described in this paper sidesteps several issues that are relevant in Buro's approach, since in Buro's case learning is driven by game play, not by databases. The Awari training data covers all possible board positions up to a given number of pebbles, and contains perfect evaluation information. This eliminates the problems of acquiring representative training data and possible overfitting, and enables the accuracy of any evaluation function to be measured directly. Unlike in GLEM, the Awari evaluation functions described in this paper are not linear combinations of features; instead, each board position is mapped to one unique class whose evaluation values are not learned but extracted directly from the databases. The class mapping is done by calculating a very small number of Boolean comparisons, which allows the evaluation value to be computed quickly.

When using a linear evaluation function, each evaluation feature is to apply in every possible position. One can however imagine that some particular feature $f_0$ may only be relevant in certain types of positions, perhaps only when another feature $f_1$ is present. The weight of feature $f_0$ then depends on the value of $f_1$, which makes the function nonlinear. To cope with this and still keep the function linear, one can introduce the new feature $f_0 \wedge f_1$. GLEM uses this approach on a large scale, generating many such features. Another way of coping with nonlinearity is to have two evaluation functions, where the value of $f_1$ decides which of the two functions to use. The two evaluation functions can have different weights, and possibly even different features; feature $f_0$ could appear in one but not in the other. The method in this paper takes this approach to the extreme; all of the work is done by the feature decisions, and the functions that are used after all the decisions are made are simple value lookups. The evaluation function is iteratively refined by targeting its weakest spot and automatically generating the single feature that reinforces the accuracy the most. Thus, the number of generated features is kept small.

### 3.2 Data mining in Awari

There are two reasons why an evaluation function is still needed when perfect information is available. In game playing programs that rely on search, there is a tradeoff between the quality of the information returned by the evaluation function and the speed with which it can be computed. A program with a "fast and dumb" evaluation can reach a larger search depth than a program with "slow and smart" evaluation, which may enable it to play stronger. For positions in the large databases that do not fit into memory it may therefore be better to approximate the value rather than going to disk to retrieve the correct value, since a disk lookup is relatively very slow. Experiments show that disk retrieval can be on the order of a thousand times slower than calculation. For positions outside the databases, an evaluation function is needed regardless.

This introduces two criteria for success for an evaluation function: its ability to *interpolate*, i.e. approximate an existing database while using less memory, and its ability to *extrapolate*, i.e. generalize to positions with larger numbers of pebbles. Evaluation

functions can be tested according to these criteria, as well as in actual game play when combined with an Awari playing program.

Exploiting the symmetry of the Awari board, the database values are all computed from the viewpoint of South to move. In the remainder of the text, the player to move will always be South.

## 4 Framework

Consider a board position $p$ whose *value*, defined as the difference in the number of captured pebbles when both players play perfectly from $p$ onward, is $v(p)$. If $v(p)$ is positive, South will capture more pebbles; if it is negative then North will capture more. The *material difference* $m$ indicates the difference between the number of pebbles already captured by South and North, respectively.[3] The material difference at the end of the game with perfect play will be $m + v(p)$, and South wins if $m + v(p) > 0$. South is therefore interested in knowing whether $v(p) > -m$.

The Awari endgame databases contain the values for all positions with at most 35 pebbles. From these databases, statistical information can be gathered that may be used to estimate the values of positions that are not in the database, or for situations in which retrieving the database value may be too costly. The databases also enable an accurate assessment of the quality of a given evaluation function. There is no danger of overfitting, since the information in each database covers the entire space of board positions with a certain number of pebbles, rather than just a sample of it.

The automatic construction of an evaluation function consists of two subtasks. The first subtask involves discovering base features, starting with a collection of atomic features. The second subtask involves synthesizing the base features into an evaluation function.

### 4.1 High-level features

The starting point for building high-level features is a set of *atomic features*. In this framework we use only Boolean features, of the following two types:

- pit $i$ contains exactly $j$ pebbles, denoted for example by $C_{=3}$ where '$C$' refers to the pit name and '$= 3$' refers to the target value $j$;
- pit $i$ contains more than $j$ pebbles, denoted for example by $d_{>3}$ where 'd' is the pit name and '$> 3$' is the threshold value $j$.

A high level feature consists of a Boolean operator and two features, each of which can itself be an atomic or a high level feature. If the set of Boolean operators is complete, then *any* Boolean function can be expressed in this way. The set of functions {`and`, `not`, `or`} is complete, as is the set {`nand`, `or`}, but these have the disadvantage that some of the 16 possible Boolean functions of two variables can only be expressed by chaining together functions, which for the feature discovery would essentially mean

[3] The board position $p$ itself does *not* contain any information about the number of pebbles already captured by either player.

one or more extra steps of inference. This complicates the task, and makes it potentially much less feasible. Consider for example the exclusive-or function '$\oplus$'. It can be built by combining the and-function '$\wedge$' and the or-function '$\vee$', for example by $x \oplus y = (x \wedge \neg y) \vee (\neg x \wedge y)$. However, the two terms $(x \wedge \neg y)$ and $(\neg x \wedge y)$ might themselves not be of any value. Unless the system keeps track of *all* previously generated features, this greatly reduces the probability of $x \oplus y$ being discovered.

It would therefore be beneficial to have all 16 different Boolean functions of two variables available. These functions form eight pairs of mutually complementary functions. Since the expressive power of the system does not change when a function is added that is the complement of an existing function, eight of the functions can be omitted. Of the remaining eight functions, there are the superfluous functions $f(x, y) =$ `true`, $f(x, y) = x$, and $f(x, y) = y$. The five remaining functions are the Boolean $\wedge$, $\vee$, $\oplus$, $\Rightarrow$, and $\Leftarrow$.

In practice, the set of possible features quickly becomes too large. The feature discovery system must therefore do a heuristic search of the space of features. Some metric must be defined by which the quality of a feature can be measured, according to the overall goal of building a good evaluation function. Consistent with the evaluation function fitness measure described in Section 4.3, the "fitness" of a feature can be measured by computing the statistical correlation between the feature value $x$ and the database value $y$ over all positions in the database. This correlation $r(x, y)$ is defined as

$$r(x,y) = \frac{\mathrm{cov}(x,y)}{\sqrt{\mathrm{var}(x)}\sqrt{\mathrm{var}(y)}} = \frac{\mathcal{E}(xy) - \mathcal{E}(x)\mathcal{E}(y)}{\sqrt{\mathcal{E}(x^2) - \mathcal{E}^2(x)}\sqrt{\mathcal{E}(y^2) - \mathcal{E}^2(y)}}$$

where $\mathcal{E}$ is the expected-value operator. The correlation $r$ will be in the interval $[-1, +1]$, where $-1$ indicates strong negative correlation and $+1$ indicates strong positive correlation. Since these are both desirable, $r^2(x, y)$ can be used as a fitness measure for a Boolean feature.

### 4.2 Approximating values

Let $M_i$ be the set of all possible board positions containing exactly $i$ pebbles, and let $H_i$ be a partitioning of $M_i$ into subsets. Suppose that the statistical average $\mu_S$ and deviation $\sigma_S$ of the values in each subset $S \in H_i$ be known, and that the database values follow a Gaussian distribution.[4] If $v(p)$ is not accessible, but it is known that $p \in S(p) \in H_i$, then the probability that $v(p) > -m$ is equal to

$$\Phi\left(\frac{\mu_{S(p)} + m}{\sigma_{S(p)}}\right),$$

where $\Phi$ is the cumulative normal distribution

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}\xi^2} \mathrm{d}\xi.$$

[4] Plots of the distributions indicate that this is indeed the case.

This probability can be used as the heuristic evaluation of $p$. Since an evaluation function only needs to impose an ordering on the set of game positions and the function $\Phi$ is strictly monotonically increasing, we can equivalently use just

$$\text{eval} = \frac{\mu_{S(p)} + m}{\sigma_{S(p)}}$$

and omit the $\Phi$ function. If $\sigma_{S(p)} = 0$ then the value $v(p)$ is known, and the evaluation function will assume an infinite value corresponding to a proved win or proved loss. The potentially arbitrarily large evaluation value does not present a problem, since the observed values for $\sigma_{S(p)}$ do not get arbitrarily small in practice.

### 4.3 Measuring fitness

It is clear that the quality of this evaluation depends on the value of $\sigma^2_{S(p)}$, which is the mean squared difference between $\mu_{S(p')}$ and $v(p')$ over all $p' \in S(p)$. If $\sigma_{S(p)} = 0$, then the evaluation function perfectly predicts the database value. The mean squared difference between $\mu_{S(p')}$ and $v(p')$ over the entire database $M_i$ is equal to

$$\sigma^2(H_i) = \frac{1}{|M_i|} \sum_{S \in H_i} |S| \sigma^2_S.$$

The number $\sigma(H_i)$ may be called the *fitness* of the partitioning $H_i$, where $\frac{|S|}{|M_i|}\sigma^2_S$ is the *contribution* that a particular subset $S$ has to $\sigma^2(H_i)$.

The fitness is a direct measure of the accuracy of $H_i$ when used as an evaluation function. The fitness of the partition $H_i = M_i$ is equal to the standard deviation of the entire database itself, which corresponds to an evaluation function that only looks at material difference and ignores the position. This may be used as a baseline comparison for evaluation functions; the *relative fitness* $\sigma_{\text{rel}}(H_i)$ is thus defined as $\sigma(H_i)/\sigma(M_i)$. The relative fitness of the "material-only" evaluation function is 1; a lower relative fitness implies a better evaluation function.

### 4.4 Constructing a partition

The partition $H_i$ can be refined by targeting one of its subsets $S$ and dividing it further according to some criterion. To do this, we consider a set of features and simply choose the feature that reduces the contribution of $S$ the most. The maximum reduction is achieved by the feature with the highest fitness within $S$. This fitness, as mentioned before, is measured by the statistical correlation between the feature value and the database value over all positions $p \in S$.

For each of the subsets $S \in H_i$, we can define the *potential* as the difference between its contribution before and after splitting $S$ according to its highest correlating feature. An evaluation function can now be built up by starting with the initial partition $H_i = M_i$ and iteratively refining it by splitting the subset with the highest potential. The resulting structure is a *decision tree*. Notice that this method will eventually construct a perfect evaluation function by partitioning $M_i$ into subsets of size one, but that partition cannot feasibly be used as an evaluation function since the evaluation function is a lookup table that contains the $\mu$ and $\sigma$ values for each subset.

## 5 Baseline comparison

The main goal of this project is the automatic discovery of evaluation functions that *do better than hand-crafted ones*. It is therefore important to establish a baseline for comparison. For the features, the baseline is formed by the straightforward atomic features of a position. For the evaluation functions, human-defined concepts such as mobility and safety are used.

### 5.1 Comparison for features

The atomic features with the highest correlation $r^2$ are listed in Table 1. The collection of atomic features consists of the set of equality features and the set of inequality features, each of which is capable of uniquely encoding all board states. Using both sets, the $p$ pebble database is described by $2 \cdot 12 \cdot p$ atomic features.

| 10 pebbles | | 15 pebbles | | 20 pebbles | |
|---|---|---|---|---|---|
| atom | $r^2$ | atom | $r^2$ | atom | $r^2$ |
| $a_{=0}$ | 0.09017 | $b_{=0}$ | 0.04320 | $b_{=0}$ | 0.04294 |
| $b_{=0}$ | 0.07374 | $a_{=0}$ | 0.03669 | $a_{=0}$ | 0.04234 |
| $B_{=0}$ | 0.05579 | $c_{=0}$ | 0.02989 | $c_{=0}$ | 0.03467 |
| $A_{=3}$ | 0.04815 | $A_{=3}$ | 0.02494 | $A_{=3}$ | 0.02193 |
| $C_{=0}$ | 0.03565 | $B_{=3}$ | 0.02386 | $d_{=0}$ | 0.02143 |
| $A_{>2}$ | 0.09178 | $A_{>2}$ | 0.07800 | $A_{>2}$ | 0.09017 |
| $a_{>1}$ | 0.06976 | $B_{>2}$ | 0.06154 | $B_{>2}$ | 0.07126 |
| $a_{>2}$ | 0.05290 | $A_{>3}$ | 0.04688 | $a_{>2}$ | 0.06235 |
| $B_{>1}$ | 0.05048 | $a_{>2}$ | 0.04415 | $A_{>3}$ | 0.05727 |
| $A_{>1}$ | 0.04685 | $a_{>1}$ | 0.04139 | $a_{>1}$ | 0.05611 |

**Table 1.** Highest scoring atomic features

The correlations as listed in the Table 1 are relatively weak; perhaps better correlating features can be discovered automatically. It is possible to get an indication of the maximum possible correlation of a binary feature. This can actually be measured from the database, by setting a threshold $k$ and defining the feature value to be 1 if and only if the database value exceeds $k$. This artificially produces a perfect separation of the database, which is the best that a binary feature could possibly do. Table 2 lists the $r^2$ scores of these maximum features; the best score is achieved in each case when $k = 2$.

These experiments establish a score range for useful high-level features. An useful feature should have a correlation $r^2$ exceeding that of the best atoms, as listed in Table 1. At the same time, it is known that its $r^2$ cannot exceed that of the theoretical maximum as listed in Table 2. It should be noted that the latter results do not impose a maximum attainable accuracy on evaluation functions, since an evaluation function can achieve better scores by combining multiple binary features.

| $k$ | number of pebbles | | |
|---|---|---|---|
| | 10 | 15 | 20 |
| 0 | 0.61149 | 0.52256 | 0.56437 |
| 1 | 0.61222 | 0.55392 | 0.57530 |
| 2 | 0.61641 | 0.56141 | 0.57677 |
| 3 | 0.61175 | 0.54993 | 0.56821 |

**Table 2.** Maximum possible feature scores

### 5.2 Comparison for evaluation functions

To assess the quality of an evaluation function, it can be compared to the best evaluation based on human constructed features. The human members of the Bambam team have been able to come up with the following features:

- *mobility*: the number of available moves;
- *safety*: the number of "safe" pits, ie. the number of pits containing three or more pebbles;
- *attack*: the number of enemy pits under attack;
- *balance*: the difference between the number of pebbles in the two home areas.

The first three features can be computed either for South, for North, or the difference between the two. Table 3 lists the relative fitness scores $\sigma_{\mathrm{i.e}}$ for these features on the 10, 15, and 20 pebble databases. Recall that lower relative scores correspond to better evaluation functions. The absolute fitness values of the entire databases are listed on the bottom row.

| feature | number of pebbles | | |
|---|---|---|---|
| | 10 | 15 | 20 |
| mobility difference | 0.85485 | 0.88392 | 0.86392 |
| mobility South | 0.86620 | 0.89240 | 0.89088 |
| mobility North | 0.92840 | 0.95363 | 0.94150 |
| safety difference | 0.97414 | 0.91441 | 0.85615 |
| safety North | 0.97186 | 0.91666 | 0.87136 |
| safety South | 0.99230 | 0.95167 | 0.91030 |
| attack difference | 0.99266 | 0.95284 | 0.93025 |
| attack South | 0.98705 | 0.95901 | 0.95385 |
| attack North | 0.99930 | 0.97697 | 0.96124 |
| balance | 0.90811 | 0.89057 | 0.86243 |
| none | 1.00000 | 1.00000 | 1.00000 |
| none (absolute) | 5.20733 | 4.41549 | 5.02372 |

**Table 3.** Performance of hand crafted features

The evaluation functions from the feature discovery will be constructed by combining several features. To compare these with what can be gained by combining the

human defined features, Table 4 lists the scores for the combinations of human defined features. The best score overall is achieved by a combination of safety difference and mobility difference.

| feature 1 | feature 2 | number of pebbles | | |
|---|---|---|---|---|
| | | 10 | 15 | 20 |
| mobility South | mobility North | 0.84209 | 0.87195 | 0.85521 |
| safety South | safety North | 0.97109 | 0.90897 | 0.85175 |
| attack South | attack North | 0.98680 | 0.94649 | 0.92906 |
| mobility difference | safety difference | 0.84843 | 0.85077 | 0.79654 |
| mobility difference | balance | 0.83790 | 0.85905 | 0.82903 |
| mobility difference | attack difference | 0.83073 | 0.87541 | 0.84633 |
| safety difference | balance | 0.89245 | 0.88615 | 0.84126 |
| attack difference | balance | 0.88296 | 0.88690 | 0.85534 |
| safety difference | attack difference | 0.96568 | 0.90757 | 0.85121 |

**Table 4.** Combining two hand crafted features

The scores correspond to an evaluation function that contains a lookup table for $\mu$ and $\sigma$ for each of the possible input values. Since both mobility difference $E_{\mathrm{mob}}$ and safety difference $E_{\mathrm{saf}}$ can assume thirteen different values, the function based on both $E_{\mathrm{mob}}$ and $E_{\mathrm{saf}}$ can take on 169 values.

The numbers indicate that the amount of "noise" in the evaluation function can be reduced by some 15% to 20% by using human-defined features, when using the databases to tune the parameters. Evaluations that were hand-tuned by the Bambam team members turned out to have a relative fitness of at best 0.95. Section 6.4 contains results that indicate how much increase in playing strength can be gained by improving the fitness.

## 6 Results

This section contains the results of three experiments. In the first experiment, high-level features were discovered by a genetic search algorithm. In the second experiment, an evaluation function corresponding to a decision tree was generated. The decisions in the decision tree are based on atomic features only; in a future experiment, the two approaches can be combined by using high-level evolved features in the decision tree. Both the features and the evaluation functions were subsequently tested for generalization to larger databases. In the third experiment, the playing strength of hand-crafted and of automatically constructed evaluation functions was tested by playing matches.

### 6.1 High-level features

Starting with the atomic features, higher level features were constructed by an evolutionary approach with two different recombination rules. The first rule takes two features at random and combines them with one of the five Boolean functions. The second

method randomly replaces a subtree of one feature by a subtree of another. The lowest scoring features are removed from the population, where the score is equal to the correlation $r^2$ with the database value with a small penalty for size. This penalty, in combination with the second method, is necessary to prune out "evolutionary junk" that would otherwise accumulate in the features. Without it, features proved to grow in size very quickly.

It should be noted that the actual details of the evolutionary scheme are not of main interest in this case. The main question is whether high-level features exist and can be discovered relatively quickly. Several heuristic search approaches can be considered; the genetic algorithm used here is merely one example.

For the 10 pebble database, the average score $r^2$ of the population was 0.19 with an average feature size of 10.0 after 2000 generations.[5] The size of a feature is defined as the number of atoms it contains. If a smaller feature has a higher score than another feature, then the former *dominates* the latter. The best non-dominated features that were discovered are listed in Table 5.

| size | $r^2$ | feature |
|---|---|---|
| 1 | 0.09178 | $A_{>2}$ |
| 2 | 0.14977 | $(b_{>0} \Leftarrow a_{=0})$ |
| 3 | 0.15118 | $(a_{=0} \wedge (b_{>0} \Rightarrow f_{=5}))$ |
| 7 | 0.22012 | $(A_{>2} \vee ((B_{>2} \vee ((a_{=0} \wedge (b_{>0} \Rightarrow f_{=5})) \vee a_{=8})) \oplus A_{>7}))$ |

**Table 5.** Best discovered features for 10 pebbles

The feature $(b_{>0} \Leftarrow a_{=0})$ can be explained in human terms: if pit 'a' is empty, then 'b' should not be empty.[6] However, the larger features have no concept that is understandable to humans. But the feature "$(A_{>2} \vee ((B_{>2} \vee ((a_{=0} \wedge (b_{>0} \Rightarrow f_{=5})) \vee a_{=8})) \oplus A_{>7}))$" is as transparent to computers as the feature of a "passed pawn" is to humans, and conversely the former is as opaque to humans as the latter is to computers. The feature has good correlation and is easy to compute. If the goal is to produce a high level evaluation function, rather than to teach humans something about Awari, then the correlation is the important issue.

Note that the 7-atoms feature, though it seems big, can be computed very quickly as it consists entirely of Boolean operations. If a feature is kept as a tree structure, it can be updated incrementally during an alpha-beta search by only computing the value of a subtree if one of its two children changes its value.

For the 5 pebble database, features with an average score of 0.329 and average size of 12.9 were discovered within half an hour on a 400 MHz PC. But the 5 and 10 pebble databases are too small to be useful for our purposes, as the information contained in

[5] This evolutionary run is quite feasible computationally; it took about an hour on a 400 MHz PC.

[6] This can be either good or bad; in this case, since $r$ is greater than zero, it is apparently good not to have both 'a' and 'b' empty.

them likely does not generalize well to higher databases. For the 15 pebble database, it took about a day on the same PC to reach an average score of 0.12 and average size of 13.8. Some of the best non-dominated features are listed in Table 6. The 5 atoms feature is much smaller and yet reaches almost the same score as the 22 atoms feature, so it may be more useful under the time constraints of actual game play.

| size | $r^2$ | feature |
|---|---|---|
| 1 | 0.07800 | $A_{>2}$ |
| 2 | 0.13577 | $(A_{>2} \vee B_{>2})$ |
| 5 | 0.15191 | $(((B_{>2} \oplus e_{=10}) \oplus d_{=8}) \vee (A_{>2} \vee C_{>2}))$ |
| 22 | 0.15650 | $(C_{=4} \Leftarrow (((D_{>8} \Leftarrow (e_{=13} \vee B_{>2})) \wedge (c_{=13} \oplus (e_{=12} \oplus (E_{>8} \Leftarrow (B_{=9} \vee A_{>2}))))) \wedge (e_{=3} \Leftarrow ((((b_{=2} \Leftarrow A_{>2}) \Rightarrow D_{>6}) \vee (A_{=8} \oplus b_{=0})) \wedge (C_{>4} \vee ((D_{=14} \Leftarrow (B_{>1} \vee (A_{>12} \Leftarrow (C_{>6} \oplus a_{>0})))) \Rightarrow D_{>8})))))$ |
| 34 | 0.16423 | $(A_{>2} \Leftarrow (b_{=3} \Leftarrow ((((((a_{>1} \Leftarrow a_{=0}) \Rightarrow (A_{>2} \vee ((c_{>13} \Leftarrow (D_{=12} \Leftarrow a_{>0})) \Rightarrow (c_{=0} \vee B_{>4})))) \Rightarrow ((((D_{=10} \Leftarrow a_{>2}) \wedge (A_{=10} \oplus ((c_{>11} \vee ((B_{=7} \oplus a_{>2}) \vee B_{=0})) \Rightarrow b_{=0}))) \Rightarrow (b_{>0} \wedge ((F_{=0} \vee f_{>1}) \vee (e_{=0} \Rightarrow B_{=9})))) \vee b_{=8})) \Rightarrow (C_{>10} \oplus (((c_{>11} \vee ((B_{=7} \oplus a_{>2}) \vee B_{=0})) \Leftarrow A_{>3}) \Rightarrow (A_{=3} \vee (e_{=13} \vee B_{>2}))))) \vee c_{=9})))$ |

**Table 6.** Best discovered features for 15 pebbles

It appears that evolved features of relatively small size correlate about twice as well as the best atomic features. For the 10 pebble database, the best evolved feature mentioned achieved a correlation of about 36% that of the theoretical maximum, while the best atom scored about 15%. For the 15 pebble database, these numbers are 28% and 14%, respectively.

### 6.2 Decision tree

An evaluation function corresponding to a decision tree can be generated automatically via the methods described in Section 4.4. In this case, only atomic features were used to split the subsets.

Figure 2 shows the resulting decision tree when the 15 pebble database is split into nine segments, numbered 0 through 8, according to the learning mechanism described above. Vertical arrows correspond to "yes", horizontal arrows correspond to "no". Note that most of the decisions involve the leftmost pits for both players, and are based on "$> 2$", which corresponds to safety, and "$= 0$", which corresponds to mobility. When the tree is expanded further, other features will naturally appear; for example, segment 6 is fourth in line to be expanded and its split involves the feature $e_{>4}$.

Figure 3 lists the relative fitness score $\sigma_{\mathrm{rel}}$ of the decision tree evaluation as it acquires more and more leaves. By the time it contains nine leaves, the score is better than that of the human-defined evaluation function $E_{\mathrm{saf}}$, as indicated by the upper dashed line. The safety difference score can take on thirteen different values, so the decision tree achieves a better accuracy with fewer nodes. The best human defined feature

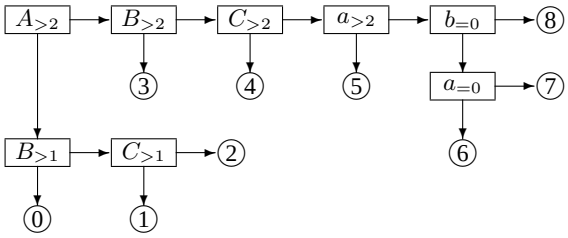


**Fig. 2.** A nine-valued decision tree

$E_{\mathrm{mob}}\&E_{\mathrm{saf}}$, which is the combination of mobility and safety, corresponds to the lower dashed line. Its score is better still, but it needs 169 values to achieve this. When compared to the best hand-tuned evaluation function[7] whose $\sigma_{\mathrm{rel}}$ is about 0.95, the decision tree already does better when it contains three leaves.

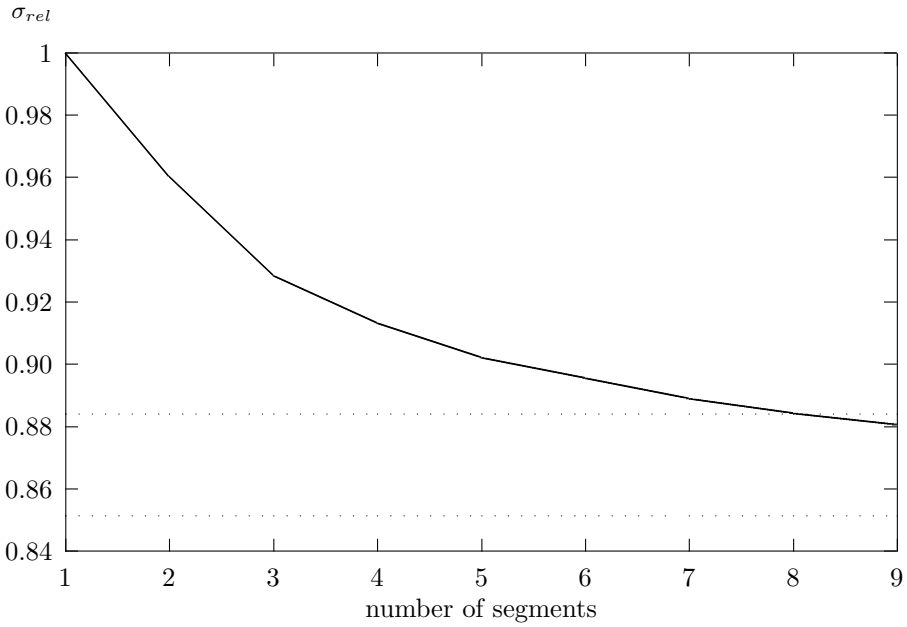


**Fig. 3.** Progressive scores for the decision tree

[7] See Section 5.2.

### 6.3 Generalization

An important issue is generalization. The high-level features and the decision tree were obtained by probing the 15 pebble database, but how well do they generalize to positions with more pebbles on the board? The features and decision tree were tested against the databases up to 24 pebbles. Table 4 shows the correlation scores $r^2$ for two features. The first feature, $f_7$, is the 7-atom feature discovered in the 10 pebble database, as listed in Table 5. The second feature, $f_5$, is the 5-atom feature evolved in the 15-pebble database and shown in Table 6.

As can be seen, the scores of the features are quite stable from 15 pebbles onward. The feature $f_7$ actually does better than $f_5$. These findings suggest that it may be possible to mine the smaller databases for high level features that can be used in the larger databases. An evaluation function based on $f_5$ alone would achieve $\sigma_{\mathrm{rel}} = 0.91642$ on the 15 pebble database. According to Figure 3 this is comparable to the decision tree of Figure 2 when based on four atomic decisions.

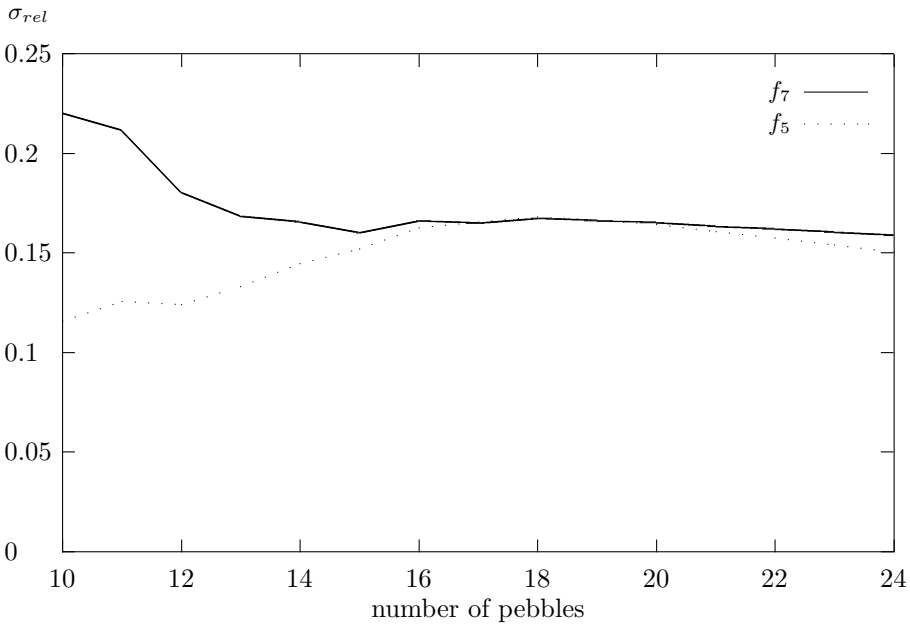


**Fig. 4.** Progressive scores for two high-level features

Figure 5 plots the relative fitness score for two evaluation functions, $E_M$ and $E_D$. The lower line corresponds to $E_D$, a data mined evaluation function containing five parameters. The upper line represents $E_M$, a simple hand-tuned evaluation function based only on mobility. The fitness score appears stable for positions with twenty or more pebbles. The numbers were obtained from the databases containing up to 35 pebbles, which are currently the largest databases built for Awari.

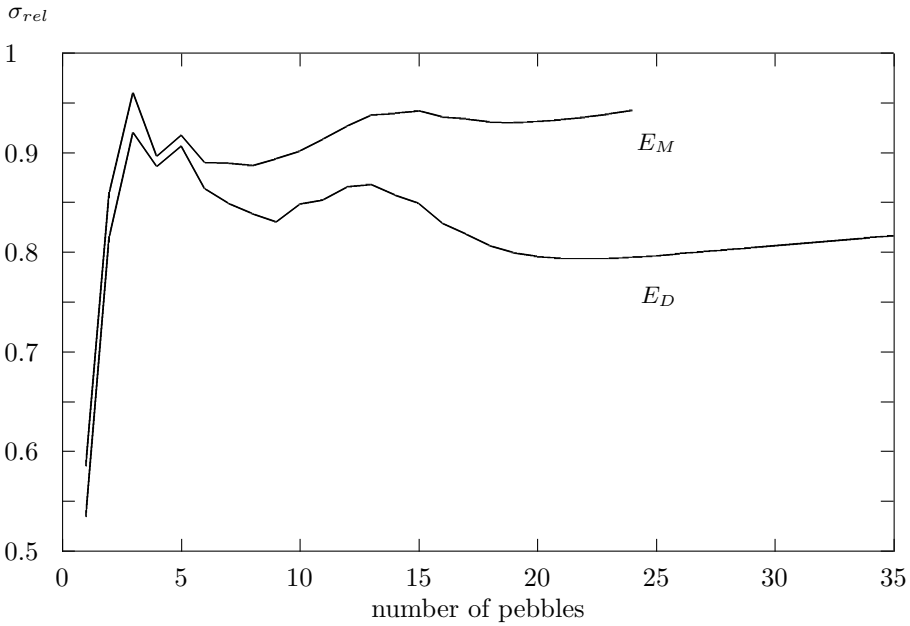


**Fig. 5.** Progressive scores for two evaluation functions

Finally, to test the stability of the evaluation parameters themselves, Figure 6 displays the $\mu$ values of $E_D$. The values also appear to be stable, and extrapolation of the values to positions with more than 35 pebbles appears straightforward.

### 6.4 Playing strength

The fitness measure used to test evaluation features and functions indicates the correlation with perfect information. However, the goal of producing an evaluation function is ultimately to maximize the playing strength. To test this, a round-robin match was played between four evaluation functions:

- $E_P$**:** "pebble count", material only;
- $E_T$**:** mobility used only as tie-breaker;
- $E_M$**:** mobility;
- $E_D$**:** data-mined evaluation.

The pebble count function corresponds to a program that only looks at the number of captured pebbles, and effectively ignores the position. The fitness score $\sigma_{rel}$ of this evaluation function is by definition equal to 1. The functions $E_T$ and $E_M$ both consider mobility, but $E_T$ uses it only as a tie-breaker between positions that have equal material advantage, while $E_M$ counts each point of mobility as equal to one captured pebble.

The evaluation functions were tested with Bambam, a program containing state-of-the-art search techniques. During the tests, the databases themselves were not used.

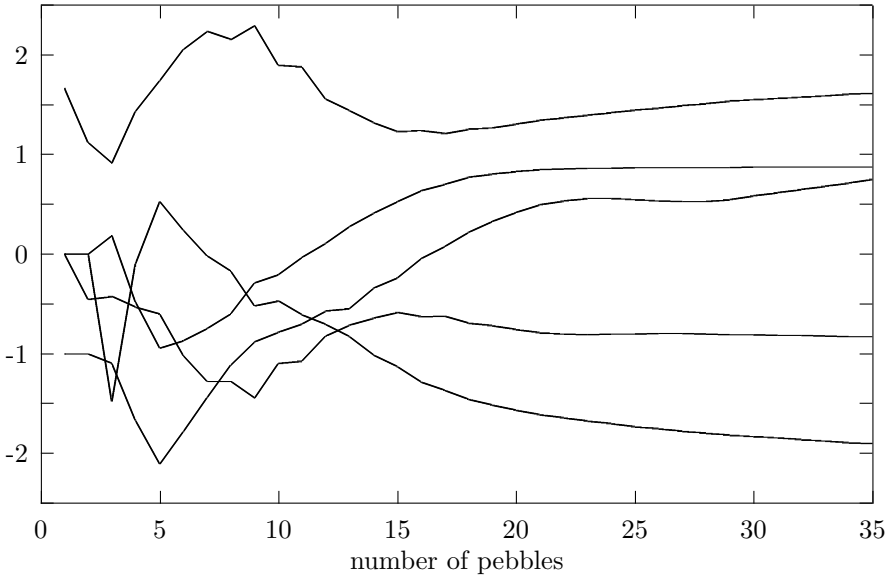


**Fig. 6.** Progressive parameter values for $E_D$

Each encounter between two evaluation functions consisted of 72 games, corresponding to all 36 possible two-move openings in Awari played once from the north side and once from the south side by each program. The results are listed in Table 7. Each box lists the match score in percentage of games won, and the average difference in number of pebbles captured. The data mined evaluation $E_D$ did not exploit the two weaker evaluations better than $E_M$ did, but it did win each of its matches convincingly, including the match against $E_M$ itself.

| | $E_D$ | $E_M$ | $E_T$ | $E_P$ | final result |
|---|---|---|---|---|---|
| $E_D$ | | 65% +2.7 | 60% +3.4 | 73% +4.9 | 66% +3.7 |
| $E_M$ | 35% -2.7 | | 71% +3.6 | 81% +7.4 | 63% +2.8 |
| $E_T$ | 40% -3.4 | 29% -3.6 | | 78% +4.4 | 49% -0.9 |
| $E_P$ | 27% -4.9 | 19% -7.4 | 22% -4.4 | | 23% -5.6 |

**Table 7.** Match results

These experiments test the playing strength of evaluation functions in the absence of databases. When one considers a hybrid program that uses large databases to look up values for all positions containing up to a certain number of pebbles, and a heuristic evaluation function for all other positions, then a game of Awari consists of two phases. During the first phase, the value of the game position cannot be proved yet, although

some positions in the search tree may contain known values. During the second phase, the value of the root position can be proved, and the program can play perfectly.

The average length of the games in Table 7 was on the order of 50 to 60 moves per side. This allows the stronger evaluation sufficient time to gain an advantage. On the one hand one would expect a hybrid program to achieve a smaller gain in playing strength when enhanced with a data mined evaluation function, since the evaluation function can only make a difference in the first phase of the game. On the other hand, experiments with the 35 pebble database have indicated that database positions are reached during the search right from the start of the game, and the presence of these large databases tends to extend the length of the first phase to where it is typically half as long as an entire game without databases.

## 7 Conclusions

The evaluation features and functions obtained by data mining compare favourably with those defined by humans. The evaluation function is a decision tree, in which each node represents a binary decision based on a Boolean feature. The evaluation function contains a lookup table for the $\mu$ and $\sigma$ values for each of the leaves. The values in the lookup table are specific to a given number of pebbles on the board; one table is needed for every number of pebbles whose database does not fit into memory. Building a high performance evaluation function would thus involve three steps:

1. identify features;
2. build a decision tree;
3. extrapolate to positions with more than 35 pebbles.

The first two steps have been demonstrated; the first step may even be skipped. The methods used were relatively simple genetic evolution and decision tree learning, respectively. The third step is a matter of regression, which is feasible since the values follow a stable trend from one database to the next.

The evolution and decision tree learning techniques are computationally feasible when applied to the 15 pebble database. On the 20 pebble database it would take about ten times as long; on the very large databases it is not feasible. However, the generalization results indicate that this is not strictly necessary, since the fitness of the discovered features and evaluations appear to remain stable for larger numbers of pebbles. Taking advantage of this, only the third step needs to involve the largest databases.

Thus, the conclusion is that it is possible in the game of Awari to build a high performance evaluation function entirely from scratch, without any human expertise. The strength of the evaluation functions has been measured both in terms of correlation with the perfect information from end game databases, and in terms of actual playing strength when used in a state-of-the-art game playing program.

## 8 Discussion

The performance of the evaluation functions discussed are of the order of a 15% reduction in the amount of noise relative to a material-only evaluation function. Compared to

other games, this may be quite low. In our experience, Awari seems to be a game that exhibits a relatively high amount of "randomness", where clustering of good and bad positions hardly seems to occur, unless there are powerful strategic concepts that have as yet escaped us. Consulting various sources on Awari strategy has not convinced us that there are such concepts.

While this may explain why the noise in Awari is difficult to suppress, it may also be what enables automatic evaluation function design and learning to work well in Awari. In other games we may get better correlation, but those games may have more strategic "depth" enabling humans to do much better as well at analyzing the game positionally.

Any game of sufficient complexity allows for a very large set of possible features and evaluation functions, of which the ones that can be explained in human terms, and thus discovered by humans, are only a vanishingly small subset. There can be countless good features that do not mean anything to humans, and countless ways to combine them into good evaluation functions. It is generally unrealistic to expect automatically built evaluation functions to discover or rediscover human semantics in a game, but that does not mean "computer semantics" cannot do just as well or better, as mentioned in Section 6.1.

On the other hand, in many games there may be a favourable bias towards human-understandable features, for the reason that the game itself was designed by humans, and the rules have semantics that humans can relate to. Since the strategy of a game is more or less related to its rules, it is not surprising if powerful strategic concepts thus also have human semantics. This is perhaps where the game of Awari is relatively well suited to computers relative to humans. The strategy may be sufficiently chaotic and random that computers can both do better in search *and* in "understanding" the game.

## References

1. Michael Buro. From simple features to sophisticated evaluation functions. In *Computers and Games '98*, pages 126–145. Springer, 1998.
2. Donald Michie and Ivan Bratko. Ideas on Knowledge Synthesis stemming from the KBBKN Endgame. *Journal of the International Computer Chess Association*, 10(1):3–13, 1987.
3. John Nunn. *Secrets of Rook Endings*. Batsford, 1992.
4. John Nunn. Extracting Information from Endgame Databases. *Journal of the International Computer Chess Association*, 16(4):191–200, 1993.
5. John Nunn. *Secrets of Pawnless Endings*. Batsford, 1994.
6. John Nunn. *Secrets of Minor Piece Endings*. Batsford, 1995.
7. Paul E. Utgoff and Doina Precup. Constructive Function Approximation. Technical Report 97-04, Department of Computer Science, University of Massachusetts, Amherst, MA, 1997.
8. Paul E. Utgoff and Doina Precup. Constructive Function Approximation. In H. Liu and H. Motoda, editors, *Feature Extraction, Construction and Selection: A Data Mining Perspective*, volume 453 of *The Kluwer International Series in Engineering and Computer Science*, chapter 14. Kluwer Academic Publishers, 1998.
9. Terry van Belle. A New Approach to Genetic-Based Automatic Feature Discovery. Master's thesis, University of Alberta, 1995.