

Using Filters to Improve the Efficiency of Game-playing Learning Procedures

R. A. Hasson* S. Markovitch† Y. Sella‡
Department of Computer Science
Technion - Israel Institute of Technology
Haifa 32000, Israel

Abstract

In this paper we describe a learning system which acquires knowledge in an attempt to improve the performance of a problem solver that plays the game of *fives*. We present experimental results indicating that the performance of the problem solver is improved at the beginning, but degrades afterwards down to almost its initial level. We introduce a selection mechanism that tests the system performance with and without the acquired knowledge and allows its addition to the knowledge base only if it proves to be beneficial. We introduce additional selection mechanisms to reduce the number of tests that need to be performed by the above filter and save learning resources. Experiments show that the addition of the filters eliminates the deterioration in performance and improves the learning outcome significantly.

1 Introduction

Most of the efforts in the machine learning research have been concentrated in finding methods for acquiring knowledge. The possibility that the acquired knowledge can be harmful to the system was paid only little attention. Many systems employed selection mechanisms to filter out knowledge that was estimated to be harmful to the system, but the selection mechanisms (or filters) have never played a major role in those system. A clear manifestation of that approach is the lack of experimental results that compare the performance of the learning systems with and without the filters.

Minton's research [4] was one of the first to concentrate on the problem of harmful knowledge (which he calls *the utility problem*) and on building selection procedures to eliminate the problem. Markovitch and Scott [2, 3] define knowledge to be harmful if its addition to the knowledge base reduces the system's performance according to a given criterion. They also define a framework that unifies the research efforts concerned with the problem of eliminating harmful knowledge. The framework is based on a view of learning systems as information processing systems where information flows from the experience space through some attention procedure then through the acquisition procedure and the knowledge base and finally to the problem solver. The framework defines five types of filters, according to their location within

*e-mail: rhasson@cs.technion.ac.il

†e-mail: shaulm@cs.technion.ac.il

‡e-mail: sella@cs.technion.ac.il

the information flow: selective experience, selective attention, selective acquisition, selective retention and selective utilization.

We have encountered the problem of harmful knowledge in the context of a learning program that tries to improve its playing performance in a game called *fives*. The program learns by playing against a teacher (an instance of the program that is given more time resources) and modifying coefficients of the evaluation function. Unfortunately, experiments show that some of the modifications can be harmful. We have studied the implementation of information filters to avoid the problem of acquiring harmful knowledge in the context of game playing programs.

Game playing programs have the advantage of a very clear evaluation criterion associated with them: two programs can be easily compared by letting them play against each other. The availability of a good evaluation criterion enables us to create acquisition filters in the following way. Before making a modification to the knowledge base, compare the performance of two instances of the program: one that uses the knowledge base in its current state, and another that uses the modified knowledge base. Only if the modified state is better than the current state the modification takes place.

Such a test-based acquisition filter was implemented and proved to be useful. The problem with this method of filtering is the high cost associated with its application. Another acquisition selector uses domain specific rules to filter out some of acquired knowledge before it arrives the high cost filter. In addition, an experience filter was inserted before the acquisition procedure in order to allow only learning instances that are likely to be highly informative. These two additional filters reduce the number of times that the test-based filter is applied and thus save learning resources.

Section 2 of this paper describes the learning system before the filters were inserted. Section 3 describes experiments performed with that version of the learning systems and shows that some of the learned knowledge was harmful. Section 4 describes the information filters that were added to the system, and demonstrates their usefulness by experimentation. Section 5 concludes and discusses some potential extensions.

2 Learning to play the game of FIVES

The game of *fives* can be regarded as an extension of the famous game *Tic-tac-toe* or as a reduced version of the game *Go-muku*. The game is played on a square board of 10×10 , i.e. there are 100 places in the board where any player can make his move, provided that the place is not already taken. The number of players is two, one of them plays with the x symbol, and the other with the symbol o . The two players place their symbols alternately. The game's objective is to generate a sequence of five consecutive symbols (x 's or o 's) in any legal direction, where the legal directions are vertical, horizontal and diagonal. The first player who succeeds in creating such a sequence (obviously of his own symbol) is the winner. If none of them achieves that goal the game is tied.

The basic *fives*-playing program consists of an implementation of the *minimax* algorithm with *alpha-beta pruning* [5]. It is a search procedure that generates moves through the problem space until encountering a goal state (i.e. a winning board) or reaching a predefined maximal depth (called *max level*). This depth is naturally a very important parameter, that affects the playing quality of the problem solver.

In order to speed up execution time, i.e. to improve the effectiveness of the search, we

introduced the following variations:

- Improvement of the generation procedure: For a given node (i.e. board), we only generate the children which are inside the *interest zone* of that node. The *interest zone* of a board is defined as the sub-board of that board, which includes every place filled until now, expanded by two columns (one to the left and one to the right) and by two rows (one from above and one from below). The *interest zone* can include the entire board at the most.
- Improvement of the test procedure: From the set of children generated for a given node we select the most promising ones (i.e. best moves) to be further developed in the *minimax* tree.

The size of the set of best children has a major influence on the problem solver's playing quality. Since it actually determines the width of the *minimax* tree, it is designated by a constant named the *branching factor*.

In order to evaluate the leaves (i.e. boards) of the *minimax* tree, and for the purpose of selecting the best children of an internal node, we used a polynomial style evaluation function, that will be described in subsection 2.2. In the next subsection, we will show the architecture of a learning system that tries to improve the performance of the player of *fives*. Finally, in subsection 2.3, we explain the approach taken in the construction of the learning procedure of the system.

2.1 Learning System Architecture

The learning system employs two different instances of the problem solver that play a game of *fives*. One instance of the problem solver is the *student* and the other is the *teacher*. The *student* attempts to adapt its strategy to the teacher's one. Therefore we allocate more computing resources to the *teacher* (greater *max level* and *branching factor* values) than to the *student*.

During a *learning game* (i.e. a game of *fives* played between the *teacher* and the *student*), every time the *teacher* has to play, the *student* tries to predict the next move of the *teacher* (called the *expected move*). The *teacher* makes his move (called the *actual move*), which might be the same as the *expected move* or not. The pair formed by the *expected move* and the *actual move* serves as input for the *learning procedure*. The output of the learning procedure are some coefficient modifications which are fed into the knowledge base that holds the coefficients of the evaluation function used by all the instances of the problem solver. Figure 1 shows the architecture of this learning system.

2.2 The Evaluation Function of the Problem Solver

We start this subsection by defining some useful concepts related to the evaluation function. We will use $s \in \{x, o\}$ to denote a player's symbol, and " \square " to denote an empty place:

1. *five* : five consecutive s .
2. *full open four* : four consecutive s , that can grow to a *five* in two directions. E.g. $\square s s s s \square$.

Figure 1: The architecture of the learning system.

3. *half open four* : four consecutive s , that can grow to a *five* in only one direction. E.g. $\sqcup s s s s$.
4. *full open three* : three consecutive s , that can grow to a *full open four* in two directions. E.g. $\sqcup \sqcup s s s \sqcup \sqcup$.
5. *half open three* : three consecutive s , that can grow to a *full open four* in one direction, and in the other they can grow to a *half open four*. E.g. $\sqcup s s s \sqcup \sqcup$.
6. *quarter open three* : three consecutive s , that can only grow to a *half open four*. E.g. $\sqcup s s s \sqcup$ or $\sqcup \sqcup s s s$.

The evaluation function involves the following board features:

- $five^s$: number of *fives* of the player s .
- $four_1^s$: number of *full open fours* of the player s .
- $four_2^s$: number of *half open fours* of the player s .
- $three_1^s$: number of *full open threes* of the player s .
- $three_2^s$: number of *half open threes* of the player s .
- $three_3^s$: number of *quarter open threes* of the player s .
- $k_i; i = 1, \dots, 5$: coefficients.

The evaluation function is computed in the following manner (for the player whose symbol is x)¹ :

1. If $five^x \geq 1$ then $F = +999999$.
2. If $five^o \geq 1$ then $F = -999999$.
3. Otherwise

$$F = k_1 * (four_1^x - four_1^o) + k_2 * (four_2^x - four_2^o) + k_3 * (three_1^x - three_1^o) + k_4 * (three_2^x - three_2^o) + k_5 * (three_3^x - three_3^o)$$

Note that a legal board created during a legal game can not include a *five* of both x and o . The vector constructed from the 5 differences above (e.g. $four_1^x - four_1^o$, etc...), will be referred to as the *contents* of the board.

In future versions of the system we intend to introduce more board features in order to increase the expressiveness of the evaluation function. For example, the number of $\sqcup s \sqcup s s \sqcup$ for each player.

¹For the opponent the function simply takes the negative sign.

2.3 The Learning Procedure

In this section we describe the approach used in the construction of the learning procedure, i.e. the part of the system that attempts to improve the performance of the problem solver. We used a learning procedure which continuously re-evaluates the coefficients of the evaluation function. Hence the knowledge to be learned is the values of the coefficients (the k_i 's from the previous subsection) which yield a better performance of the problem solver. This kind of learning technique is sometimes called *learning by parameter adjustment* [6].

As stated above, The learner employs two problem solvers that play a game of *fives* (the *student* and the *teacher*). The starting position of every game (the first 5 paired-moves) is predefined by a human operator. Thus it can be stated, that the learning procedure generates its own experiences by a process, which is mostly automated, but does need some initialization from an external agent (i.e. human).

The learning procedure starts by creating two boards:

- $Board_1$ - the board generated by applying the *actual move*.
- $Board_2$ - the board generated by applying the *expected move*.

The learning procedure then computes the *contents* of the two boards (B_1 , B_2 are the *contents* of $Board_1$, $Board_2$, respectively):

1. $B_1 = (t_1, t_2, t_3, t_4, t_5)$
2. $B_2 = (p_1, p_2, p_3, p_4, p_5)$

The difference between these two vectors is then computed. Let us denote this vector by $\Delta = B_1 - B_2 = (\Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5)$. Note that it can include positive integers, negative integers and zeroes. Let us denote by N the negative numbers in Δ , and by P the positive numbers in Δ . The learning process will cause coefficients corresponding to $\Delta_i \in P$ to grow according to the following calculations.

First, the *gain* (GA) is computed by:

$$GA = \frac{\sum_{\Delta_i \in N} |\Delta_i * k_i|}{\sum_{\Delta_i \in P} \Delta_i} * 0.1$$

Second, the *gain* is added to each of the k_i 's corresponding to $\Delta_i \in P$.

For example, consider the coefficients:

$$k_1 = 500, \quad k_2 = 400, \quad k_3 = 300, \quad k_4 = 200, \quad k_5 = 100$$

and the Δ vector:

$$\Delta = (1, -1, 2, -1, 0)$$

The updated coefficients will be:

$$k_1 = 520, \quad k_2 = 400, \quad k_3 = 320, \quad k_4 = 200, \quad k_5 = 100$$

The rationale behind the above formulas is that the components corresponding to $\Delta_i \in P$ outweighed (from the *teacher's* point of view) the components corresponding to $\Delta_i \in N$, and therefore $\Delta_i \in P$ should be increased by a value proportional to the strength of the $\Delta_i \in N$. The division by 10 is done in order to stabilize the learning process, and the other division is simply normalization.

3 The Problem of Learning Harmful Knowledge

The system enables two different operating modes, which have different objectives. They allow the system to use its three different instances of the problem solver (i.e. the *fives game-players*). In the previous section we introduced two of them: the *teacher* and the *student*, the third one is the *tester*. The operating modes are:

- *Test Mode*: *Student* confronts *tester*. Learning is off. Its objective is to evaluate the *student* (vs. another *fives game-player* program, the *tester*).
- *Learn Mode*: *Student* confronts *teacher*. Learning is on. Its goal is to allow the *student* to learn how to play *fives* in a better fashion.

Following we describe the parameters of the three instances of the problem solver used by the operating modes.

1. *Student*:

- Plays with the symbol x
- *Max Level* = 3
- *Branching Factor* = 3
- Coefficients – Initial values read from an external data file. May change during the game.

2. *Teacher*:

- Plays with the symbol o
- *Max Level* = 5
- *Branching Factor* = 10
- Coefficients – Initial values read from an external data file. May change during the game.

3. *Tester*:

- Plays with the symbol o
- *Max Level* = 3
- *Branching Factor* = 3
- Coefficients – Fixed in software.

During the experiments we had the system play games in both *test mode* and *learn mode*. Every game was played until one of the players (*x* or *o*) won, or until 25 paired-moves were taken without a victory. For the latter case the game is announced tied. The first 5 paired-moves of each game (i.e. the first 5 *x*'s and *o*'s in the board) are predefined through data files. This means that the starting position of each game is fixed.

Let us now define the different kinds of games that were played during the experiment:

- *Learning session*: a game between the *student* and the *teacher*, in which the system acquires knowledge. The *student* always moves first.
- *Test game*: a game between the *student* and the *tester*. The decision concerning who moves first, comes from the data file.

We gathered 5 starting positions together with their 5 duals (i.e. *x*'s are replaced by *o*'s and conversely), to form a set of 10 test games. This set is called the *exam*.

For each *test game* the *student* gets a score in $[0,100]$. Before calculating the score we set the value of the variable *g* in the following manner:

- if the game was tied, then $g = 0$
- if the *student* won, then $g = 26 - moves$
- if the *student* lost, then $g = -(26 - moves)$

where *moves* is the rounded up number of paired moves played in the game (e.g. if *x* moved first and won on his 7th move, the number of paired-moves executed is only 6.5, but it is rounded up to 7).

The score is then computed according to the following formula :

$$Score = g * 2.5 + 50$$

The *student's* score in the entire *exam* is the average of the scores he got in the 10 games composing it.

The idea behind this scoring scheme is that a victory achieved after *n* moves worth more than a victory achieved after *n*+1 moves, and a defeat caused after *n*+1 moves worth more than a defeat caused after *n* moves (i.e. better players will win faster and loose slower). The rest is simply normalization of the score to the range $[0,100]$ (consider the fact that we allowed 25 paired moves after which the game is announced tied).

The experiments were conducted in the following manner:

1. The data file from which the coefficients of the evaluation function were read, was initialized with the coefficients that were fixed in software for the *tester* (i.e. 500, 400, 300, 200, 100).
2. A preliminary *exam* (*exam0*) was held. The score of *exam0* was expected to be exactly 50, since at this stage the *student* and the *tester* are identical, and both the *exam* and the scoring-scheme are symmetric.

Figure 2: System performance during learning sessions.

3. The following steps were repeated five times:
 - (a) A *learning session* was held.
 - (b) The coefficients in the data file were updated according to the changes made in them during the *learning session*.
 - (c) an *exam* was held.

The graph in figure 2 shows the scores of the *student* in the *exams* as learning advanced.

The top score, 66.25, was reached after the second *learning session*. *Learning sessions* carried out from this point on, had a deleterious effect on the system performance which degraded to the score of 55 after the last (fifth) session. It is obvious that some of the knowledge acquired by the system was harmful. In the next section we describe our attempts to solve the problem of harmful knowledge by introducing information filters to the system.

4 Selective Learning

The poor performance of the learning program that was demonstrated in the last section raises an important question: is the learning technique that we are using inherently bad? Looking closely at the experiment graph reveals that it is not so. There were some learning sessions where the performance of the learning program had improved. On the other hand, there were some sessions where the learning decreased the performance. But how can we differentiate between the cases? The answer is simple: the tests that were performed as part of our research methodology can also be performed by the learner itself as a part of its learning techniques. The learner can use this self test to decide whether a particular change in the knowledge base is likely to improve its performance.

The test described above is an example of an acquisition filter: it decides whether to acquire a piece of knowledge which is generated by the acquisition procedure. It is different than most of the filters employed in other works. While most of the other filters try to estimate the utility of a newly generated knowledge by employing a relatively simple (and low-cost) heuristics, the filter that we employ here invests substantial resources for that decision. In that sense, it is similar to the hypothesis filter [1] that performs an extensive set of statistical tests before deciding whether to accept a newly generated classifier.

Specifically, the filter operates in the following way: a set of coefficients is preprogrammed into the filter. Whenever a coefficient modification is generated by the acquisition (learning) program, the filter invokes an *exam* to test the performance of the problem solver with the modified coefficients. The score that the program achieved after the last change is stored. The modification takes place only if the score achieved in the exam is higher than the last score.

One major problem with employing such a filter is its high cost in learning time. In order to reduce that cost we have implemented two other filters that reduces the number of times that the self-test filter is being called. One is another acquisition filter which is inserted between the acquisition procedure and the self-test filter. The filter uses a set of domain specific rules to filter out some modifications that do not make sense. For example, there is a rule that specifies that half open four can not be worse than quarter open three, and any modification that suggests so should be filtered out. This filter reduce the number of modifications that needed to be tested by the self-test filter.

The other filter is an experience filter which is inserted between the learning instance generator and the acquisition procedure. The input to the acquisition program are pairs of actual score and expected score. The experience filter allows only instances with significant difference between the actual and expected values to pass through. The architecture of the learning system with the filters is shown in figure 3.

Figure 3: The learning system with its filters.

We have repeated the experiment described in section 3, this time with the system that includes the filters. The results of the experiment is shown in figure 4.

The coefficients that were used to achieve the highest score were

$$k_1 = 560 , k_2 = 450 , k_3 = 300 , k_4 = 222.5 , k_5 = 122.5$$

The first observation that can be made by looking at the results of the experiment with the system that includes the filters is that the filters were extremely beneficial. While the score achieved by the system that did not employ filters was 55 after five learning sessions, the score achieved after the filters were introduced to the system was 74.2. It is worthwhile to notice that during the third and fourth learning sessions the system performance was not changed. This is due the rejection of all the proposed modifications by the various filters.

Figure 4: Results of experiments with and without filters.

5 Discussion

The work presented in this paper concentrated in the problem of harmful knowledge acquired by learning systems that attempt to improve game playing programs. In section 3 we have demonstrated that such a phenomenon indeed occurs in a particular learning system that tries to improve its ability to play the game of *fives*. Some of the acquired knowledge had deleterious effects on the performance of the game playing program.

Using the information filtering model as a guide we were able to build an acquisition filter that only selects acquired knowledge that can pass a performance test. Minton [4] and Markovitch and Scott [2, 3] define the utility of a knowledge element to be the difference in the performance of the system with and without that element. Our filter uses this definition as a functional one and actually tests the system performance under the two conditions.

While the filter proved to be very efficient in blocking harmful knowledge, it has a major drawback of requiring a substantial amount of learning resources. We had overcome this problem by inserting other filters into the system that reduced the number of times that the expensive filter was invoked.

The combination of filters proved to be very useful and the system achieved a much better performance when using them. The filters also made the learning curve behave more consistently. System performance did never decrease when using the filters.

It is interesting to note some of the ideas discussed here were implemented as early as 1959 by Samuel [7]. Unfortunately, Samuel did not report any experiment comparing the system's performance with and without the various selection mechanisms.

The research described in this paper is far from being completed. The most important step to follow is to perform a much more extensive experimentation in order to have a better understanding of the roles of the various filters. Particularly, we need to experiment with each filter separately and with every combination of filters so that the relationships between them can be well understood.

References

- [1] Oren Etzioni. Hypothesis filtering: A practical approach to reliable learning. In *Proceedings of The Fifth International Conference on Machine Learning*, Ann Arbor, MI, 1988. Morgan Kaufmann.
- [2] Shaul Markovitch. *Information Filtering: Selection Mechanisms in Learning Systems*. PhD thesis, University of Michigan, 1989.
- [3] Shaul Markovitch and Paul D. Scott. Information filters and their implementation in the SYLLOG system. In *Proceedings of The Sixth International Workshop on Machine Learning*, Ithaca, New York, 1989. Morgan Kaufmann.
- [4] Steve Minton. *Learning Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, Boston, Mass, 1988.
- [5] Judea Pearl. *Heuristics : intelligent search strategies for computer problem solving*. Addison-Wesley, 1984.
- [6] Elaine Rich and Kevin Knight. *Artificial Intelligence*. McGraw-Hill, New York, 2nd edition, 1991.
- [7] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal*, 3(3):211–229, 1959.