An Analysis of Polynomial and Generalized Learning as Applied to the Game of Score-Four

Stephen Alan Broeker, M.S.
School of Information Science
Aurora University
Aurora, Illinois
Dennis F. Cudia, Advisor

January 12, 1989

Abstract

This thesis applies Findler's version of polynomial learning (with aggression) and a modified version of Findler's generalization learning to the game of Score-Four. Score-Four is similar to tic-tac-toe except that it is three-dimensional, with a win achieved by gaining for-in-a-row in a single plane. Findler's generalization technique built up a table of all possible board positions and respective moves. Move selection then, consisted of table lookup and thus did not use an Alpha-Beta look-ahead procedure. Even though the table was reduced by removing board positions that were symmetrically equivalent, table size was still excessive. Even though Findler's generalization technique does not rely on skilled opponents to achieve significant learning, it does depened on polynomial learning to achieve significant results.

This thesis, on the other hand, only stores board positions that directly lead to wins. This reduction in table size is great enough to allow table lookup to be employed by the Alpha-Beta look-ahead heuristic. The result is a generalization technique that does not rely on skilled opponents to achive significant learning, and does not use polynomial learning to achieve significant results. A statistical analysis is applied to the results to determine the relative merits of polynomial learning (with and without aggresion) and generalization (with and without polynomial).

1 Background

1.1 Introduction

This thesis is submitted in partial fullfillment of the requirements for the degree of Master of Science in Computer Science in the School of Information Science of Aurora University.

Machine learning has received increased attention in the artificial intelligence community in the last two decades. The motives for this increased attention range from understanding human learning to supporting automation construction. Different types of learning have been studied and implemented on special and general purpose computers. Some of these learning techniques have been applied to games, since they represent a conceptually rich universe with well defined boundaries. Many computer games make use of a look-ahead procedure such as Alpha-Beta mini-maxing technique [5,6]. When look-ahead procedures are applied to games of enough complexity, generating all possible paths is not possible in real time and evaluation functions or heuristics become paramount. This project focuses on two learning techniques that fall under these criteria: polynomial and generalization.

1.2 Samuel - Polynomial Learning

Polynomial learning was first investigated and applied to the game of Checkers by A. L. Samuel [5, 6]. For a brief synopsis of Samuel's Polynomial learning technique refer to Griffith [3]. Samuel proposed defining a set of heuristic parameters and assigning each parameter to a term in a linear polynomial. This polynomial was used in the heuristic evaluation of board configurations. The computer program's task was to determine the value of the coefficients in the polynomial. The coefficients were adujsted as follows: the scoring polyomial was computed for the board and saved in its entirety at each move by Alpha. At the same time, Alpha computed the backed-up score for all board positions using a look-ahead procedure. The polynomial was evaluated by comparing the saved value of the polynomial from the previous move to the result of the look-ahead procedure. The difference of these two values was defined as delta. If delta was significantly positive, then Samuel assumed that the initial board evaluation was in error and terms that contributed positively should have been given more weight, while those that contributed negatively should have been given less weight. A converse statement was made for the case when delta was significantly negative. The change to the polynomial terms was not made directly, but was brought about in an involved way. A record was kept of the correlation between the signs of the invidvidual terms in the initial scoring polynomial and the sign of delta. After each play, the correlation coefficients were determined by setting the term with the largest correlation coefficient to a prescribed maximum value and scaling down the rest of the term coefficients accordingly. The goal then, was for term coefficients to stabalize over time as an indication of the learning process.

Samuel defined a total of 39 parameters for his Checkers polynomial. This large number of terms resulted in a heuristic evaluation function that in his opinion was to slow. So, he limited the number of terms used in evaluation to 16 at any one time. During play, if a coefficient fell below a prescribed value, then that term was replaced by a term that was currently unused.

After the term coefficients were picked arbitrarily, a series of games was held by engaging in self-play, play against many different individuals (several of

these being checker masters) and play against book games. Self-play, primarily used during the early stages of learning, involved playing against a polynomial with fixed coefficients. At the conclusion of the learning period, the program was judged to be a better-than-average-player. Samuel stated that a detailed analysis of the results of the games showed that the learning procedure did work and that the rate of learning was surprisingly high, but that the learning was erratic and unstable.

Samuel pointed out a few deficiences in his learning technique. During the remembered play, use was made of Alpha's current scoring polynomial to determine Alpha's moves, but not to determine the opponent's moves. During the anticipation play, the moves for both sides were made by using Alpha's scoring polynomial. In other words, the polynomial assumed that both sides were of equal ability. Samuel did not propose a solution to this problem. Another problem was that the program was fooled by bad play by its opponent. The program could learn bad or good technique depending on the ability of the opponent. His solution to this was to change the correlation coefficients less drastically when delta was positive.

1.3 Samuel - Signature Tables

As previously noted, Samuel felt that the rate of learning was erratic and unstable. He thought that this was due, in large part, to the use of a subset of the terms in the polynomial evaluation function. The evaluation function thus did not take into account all possible inter-parameter effects. A compromise solution to this problem was to change the parameters to output only 3 values 0, 1, and 2 corresponding to high positive, near zero, and low negative values of the parameter. The modified parameters were then grouped into sets (of three, five, and eight) called signature tables. A set of five thus consisted of 3^5 (243) entries corresponding to all possible combinations of the values of the five parameters comprising that signature table. Board evaluation thus consisted of looking up a series of tuples from each of the signature tables (a signature table consisting of three parameters would give a 3-tuple, a five parameter table would give a 5-tuple, and an eight parameter table would give an 8-tuple). The process of learning then, involved the acquisition of the signature table entries. Each table entry was calculated as the quotient of the number of strong board positions to which the entry corresponds, divided by the total number (strong and weak) to which it corresponds. The positions (both strong and weak) were derived from book games.

Obviously, it was not practical for the evaluation function to use all possible signature tables for the 38 parameters in Checkers. Samuel chose to limit the number of signature tables to 48. After processing over 180,000 book moves, Samuel concluded that the signature table method seemed to be superior to the polynomial procedure. But, he also noted that the chief defect of the program was a failure to maintain a fixed strategy during play. This was caused by a bad choice of signature tables during play.

1.4 Findler - Polynomial Learning

Nicholas V. Findler [1] modified Samuel's Polynomial technique and applied it to Go-Moku, a two-dimensional game in which the object is to obtain a chain of five consecutive stones. He simplified the algorithm by limiting the number of different coefficient terms as follows: every chain on the board had a value to each of the two players. If a chain had one or more stones of one player, then it was of no interest to the other player. The polynomial was limited to four terms, with the first term representing one stone in an interesting chain, the second term representing two stones in an interesting chain, and so on. The scoring polynomial was only applied to the moves that had been made during the look-ahead procedure, not to all the stones on the board. Thus, if the procedure used a tree of depth four, then the scoring polynomial was applied to two moves of Alpha and two moves of Beta during each look-ahead.

Findler addressed the problem of Samuel's program applying the polynomial equally to both sides by multiplying Alpha's polynomial score by an aggression factor. The aggression factor was the ratio of the Alpha move with the maximum scoring polynomial value to the Beta move with the maximum scoring polynomial. This factor was calculated before each application of the look-ahead procedure. Thus, if Alpha was ahead, it would play more aggressively than if Beta was ahead.

Findler also simplified the way that the term coefficients were modified. The program learned strictly by self-play. The opponent was assigned an exact copy of Alpha's current scoring polynomial, except that a specific term coefficient was multiplied by .95. A game was played with Alpha going first, then a game was played with Beta going first. The same term coefficient was then changed from being .95 times Alpha's coefficient to 1.05 times Alpha's coefficient. Two more games were then played with Alpha and Beta alternating the first move. At the completion of the series of games, Alpha's term coefficient was increased if the outcome of the four games was one of (A, A, B, B), (A, D, B, B), (D, A, B, B), (A, A, B, D), and (A, D, D, B). The coefficient was decreased if the outcome of the four games was one of (B, B, A, A), (B, D, A, A), (D, B, A, A), (B, B, D, A), (B, B, A, D), and (B, D, D, A). Here A shows that Alpha won, B shows that Beta won, and D shows a draw game. The program repeated this procedure several times for all terms in the scoring polynomial. The result, according to Findler, was a program that played reasonably better than the average human player.

When compared to Samuel's Polynomial learning technique, Findler's Polynomial learning algorithm has the advantage of not depending on the skill of the opponent to achieve significant results. Samuel depended largely on book games to develop experience. Findler's Polynomial, on the other hand, uses self-play to develop experience. Findler's Polynomial though was in no way complete, since a number of possible terms were not included in the polynomial (number of stones in corners, number of stones in the middle positions, and number stones on the edges, to name a few). Thus, like Samuel's Polynomial, Findler's Polynomial is dependent on a wise choice of polynomial terms and

term combinations.

1.5 Findler - Generalized Learning

Findler also designed and implemented a generalization technique in a program that played Go-Moku. The program was built in several stages and was named NIP (Novice In Playing). The author implied that the stages could be said to reflect different stages in human learning. In the first stage, called imitation, NIP played against an experienced player and built up a table of board positions and moves made in the respective positions. Findler use an experienced polynomial version of the game (referred to as Old Wise Logician or OWL for short) as the experienced player. The table consisted of local patterns, small subsets of the (19 x 19) board. Whenever NIP was to make a move, it compared board patterns to entries in the table (Alpha-Beta mini-maxing was not employed). If a match was found, NIP imitated OWL. For multiple matches, NIP used a heuristic rule to pick the best move. If a match was not found, then NIP asked OWL to make the move for him and recorded the result in the table.

The next stage, called reversed Socrates type learning, was employed when the table grew to an excessive size. It consisted of NIP analyzing the table and recognizing patterns that were similar, that is, reflections or rotations of each other. These patterns were then replaced by a single pattern with a code to indicate valid permutations. The patterns were simplified further by removing a stone and passing the new pattern to OWL for evaluation. If the result was similar, then the stone was identified as being inconsequential to the pattern and thrown away. Otherwise, the stone was replaced. This stage resulted in the table being reduced from a maximum of several million entries to about 9,000 entries.

The last stage allowed NIP to improve on OWL's strategy. Whenever NIP lost a game, it would change one of its last moves of the game and restart the game from that point. If it won, then the new pattern would be entered into the table. NIP repeated this procedure several times for each lost game. The result was that NIP was able to learn from its mistakes and out-perform OWL. Clearly, at this point, NIP was generalizing.

At first glance NIP, to some degree, might seem to depend on the skill of its opponent to achieve significant results. Even though NIP depends on OWL to evaluate moves, OWL relies on self-play and thus NIP's ability does not depend on skilled opponents to achieve experience. Therefore Findler's Generalization technique, when compared to Samuel's Polynomial and Signature Table techniques, has the advantage of not depending on skilled opponents. But since OWL does rely on a wise choice of polynomial terms and term combinations, so does NIP. Therefore in this respect, Findler's Generalization technique is similar to Samuel's techniques.

2 Results

This thesis applies Findler's version of polynomial learning and a modified version of Findler's generalization learning to the game of Score-Four. This thesis stores only those configurations that directly lead to wins. The reduction in table size is great enough to allow table lookup to be employed by the Alpha-Beta look-ahead heuristic. The result is a generalization technique that does not rely on skilled opponents to achieve significant learning, and does not use polynomial learning to achieve significant results.

Score-Four is similar to tic-tac-toe except that it is three-dimensional, with a win achieved by gaining four-in-a-row in a single plane. Therefore, Score-Four is inherently more complex than the planer, two-dimensional game of Checkers. The board is thus (4 x 4 x 4), which gives a maximum of 64 different moves at one time, or an average of 32. Thus the generation of all possible paths (or search tree) through the game involves a maximum of 64! (approximately 10^{89}) moves and an average of 32! (approximately 10^{35}) moves. At 12 moves per millisecond (see the Programming Details - Section 5.1), the generation of the maximum search tree would require 10^{75} centuries and the generation of the average search tree would require 10^{21} centuries. If the Alpha-Beta look-ahead factor is limited to 8 (4 moves by Alpha and 4 moves by Beta), then the number of nodes in the maximum search tree is given by:

$$64 \text{ x } 63 \text{ x } 62 \text{ x } 61 \text{ x } 60 \text{ x } 59 \text{ x } 58 \text{ x } 57 \approx 10^{14}$$

and the number of nodes in the average search tree is given by:

$$32 \text{ x } 31 \text{ x } 30 \text{ x } 29 \text{ x } 28 \text{ x } 27 \text{ x } 26 \text{ x } 25 \approx 10^{11}$$

The generation of the maximum search tree would thus require 50 centuries and the generation of the average search tree would require 10 years.

Samuel calculated the number of moves in the Checkers search tree to be approximately 10^{40}. He also stated that his checker playing program could select a move in 15 milliseconds. Thus, the generation of the Checkers search tree, by Samuel, would require 10^{28} centuries. Cleary then, search tree generation in Score-Four is of greater computational complexity than search tree generation in Checkers. Score-Four is thus unsolvable in real time when using Alpha-Beta look-ahead to completion and thus lends itself nicely to learning algorithms.

3 Procedure

3.1 Introduction

The first task in the research was to create goals and guidelines. It was decided that the design of the programs would stress ease of implementation and avoid hardware or operating system's dependencies. This guideline was chosen to ease porting to general purpose machines. Another requirement was that the Alpha-Beta mini-maxing procedure to be fast enough so that the choice of a single move

did not take more than 60 seconds. This goal was chosen to ensure real time analysis. A final requirement was that memory usage be kept to a minimum (i.e., 8 Megabytes). These goals and guidelines led to the following decisions. C was chosen as the programming language, because of its wide acceptance and great power. The project was tested and analyzed on an AT&T 3B2 600, that has been rated at 5.0 MIPS, running UNIX System V Release 3.2.1.

The next task in the research was to create a program (called SCORE4) that used Alpha-Beta mini-maxing to play the game against both humans and other computer programs. The Alpha-Beta look-ahead factor was limited to two (one move by Alpha and move move by Beta) so that SCORE4 was forced to make use of learning techniques, rather than relying on a depth-first tree search (rote learning). The Alpha-Beta evaluation heuristic was designed to use any combination of polynomial, aggression, and generalization learning techniques when playing games. Particular combinations were chosen by use of command line options or other external means.

3.2 Stage 1 Generalized Learning

The generalization technique used by SCORE4 (here after referred to as GENERAL) was built in two stages. Stage 1 GENERAL was similar to NIP in that it used a table of local patterns. In Stage 1 GENERAL, a local pattern was defined to encompass a plane (i.e., 4 x 4). Since there are 18 planes in a cube, there were 18 distinct (but not disjoint) local patterns in the board at any one time. Since each cell in a plane had three possible values ('X', 'O', or blank), there are a total of 3^{16} (= 43,046,721) possible configurations to be stored in the table. Of this number, 9,228,284 contain four-in-a-row for 'X' or 'O' and will thus not be included in the table. That leaves a total of 33,818,437 patterns. The table can be reduced further by replacing patterns, that are symmetrical, with a single pattern and a code to indicate the valid transformations. So, there are 3,646 patterns that are symmetrical with respect to a single rotation. This number of patterns can thus be replaced by 1,823 (= 3,646/2) patterns. There are 45,248 patterns that are symmetrical with respect to three rotations. This number of patterns can thus be replaced by 11,312 (= 45,248/4) patterns. There are 33,769,488 patterns that are symmetrical with respect to size rotations and a single reflection. This number of patterns can thus be replaced by 4,221,186 (= 33,769,488/8) patterns. So instead of 33,818,437 patterns, the number of patterns is:

```
4,234,376 = 33,818,437 -
            (3,646 - 1,823) -
            (45,248 - 11,312) -
            (33,769,488 - 4,221,186)
```

This number of patterns can be cut in half, since for each pattern in the table, there exists another pattern that is its mirror image (replace the 'X's with 'O's and the 'O's with 'X's). The result is 4,234,376 patterns. If two words

(eight bytes) are needed to store a single pattern (two bytes for 'X's plane, two bytes for 'O's plane, two bytes for the resulting move, and two bytes for the transformation code), then approximately 32 Megabytes are needed for the table. This exceeds the previously stated limit of 8 Megabytes. Thus Findler's method of Generalization would result in excessive memory requirements when applied to Score-Four.

This led to the decision that Stage 1 GENERAL would only store a reduced set of patterns in the table. This reduced set consisted of patterns that contained traps, that is, patterns in which Alpha would win no matter what move was chosen by Beta. Such patterns were referred to as "trap patterns". These patterns contained at least two interesting chains, each of which contained three stones. For example, in the pattern in Figure 1, 'X' will win no matter what move is chosen by 'O'.

Figure 1

X			
X	X	X	
X			

This table did not actually consist of trap patterns, but rather patterns that could result in traps. For example, in the pattern in Figure 2, 'X' is one move away from a trap.

Figure 2

X			
	O		
		X	X
X	O		

The pattern table was thus broken up into two levels. Level one contained patterns that were one move away from a trap. Level two contained patterns that were two moves away from a trap.

Since the tables only contained patterns that lead directly to traps, pattern table lookup had to be performed in an Alpha-Beta look-ahead heruistic. Stage 1 GENERAL's evaluation technique consisted of first comparing all local planes in the board to level one patterns. If a match was found, then that move (board configuration) was given a high value (TRAP1_VALUE). If a match was not found, then all local planes in the board were compared to level two patterns. If a match was found, then that move was given a value that was less than (TRAP1_VALUE - TRAP2_VALUE). If a level two match was not found, then polynomial evaluation (POLY) was applied. Since there were 18 local patterns for each board configuration, care was taken to ensure that TRAP1_VALUE, TRAP2_VALUE, and POLY values were significantly different. A board configuration that did not contain a level one trap could not have an Alpha-Beta value

greater than or equal to TRAP1_VALUE, and a board configuration that did not contain a level two trap could not have an Alpha-Beta value greater than or equal to TRAP2_VALUE. The heuristic evaluation function was thus designed to detect such error states. In addition, the contraints in Figure 3 ensure that TRAP1_VALUE, TRAP2_VALUE, and POLY values are significantly unequal.

Figure 3

```
TRAP2_VALUE = (TRAP1_VALUE - 1) / (NUM_TRAP2 x PLANES)
MAX_POLY_VALUE < (TRAP2_VALUE - 1) / CHAINS

where:

NUM_TRAP2 = number of possble level 2 traps in a plane = 1,152.
PLANES = number of planes in a cube = 18.
MAX_POLY_VALUE = the maximum value returned by POLY.
CHAINS = number of interesting chains in a cube = 76.
```

The time spent by Stage 1 GENERAL in heuristic evaluation was much greater than the time spent by POLY in similar evaluation (the implementation of POLY is discussed in a later section). The data structures used to represent the board were thus chosen to minimize Stage 1 GENERAL's evaluation time. In particular, the board was represented by two sub-boards - one each for Alpha and Beta. Each sub-board was an array of 18 planes. Planes then, contained 16 elements or cells. The storage requirements for each cell were one bit since each cell assumed a value of "move taken" or "move not taken". Thus, each sub-board was an array of 18 short integers. This obviously assumes a 32 bit machine word length. This data structure is erroneous for a machine word with less than 32 bits and not optimal for a machine words with more than 32 bits. It was felt that this restriction was not limiting because of the wide use of the 32 bit word in today's architectures.

Since the pattern table did not contain all possible patterns, hashing could not be used as a look-up technique. Care then, had to be taken to ensure that the pattern table was of minimal size so that sequential search could be used. This resulted in a generalization similar to that of NIP and was achieved as follows. Additions were made to the pattern table whenever Stage 1 GENERAL lost. In such a case, Stage 1 GENERAL referred to the second-to-the-last move made by the opponent (the one just before the win) and examined all local planes that contained that move. If a local plane contained a trap, then it was considered to be a candidate for entry into the pattern table. A plane was considered to contain a trap if Stage 1 GENERAL was not able to stop the opponent from winning in the plane. That is, Stage 1 GENERAL restarted the game. If the opponent was able to win using a different move, then the plane was judged to contain a trap. Local trap planes were then simplified by eliminating extraneous stones and restarting the game from that point. If the opponent still won, then the local plane had been reduced. Otherwise, stones were replaced.

The resulting pattern was represented by two short integers. The first short showed all positions that had to be filled. Positions that were filled signified the moves required for that trap. The second short indicated positions that had to be empty. Empty positions signified moves that must not be made. With this representation, pattern table comparison consisted of comparing local and table planes. The C code fragment in Figure 4 performs the comparison.

Figure 4

```
short a_local;  /* Alpha local plane */
short b_local;  /* Beta local plane */
short a_table;  /* Alpha table plane */
short b_table;  /* Beta table plane */
int f_equal;    /* are the planes equal? */


if ((a_local & a_table) == a_table && ((a_local | b_local) & b_table) == 0)
    f_equal = TRUE;
else
    f_equal == FALSE;
```

If the pattern was already in the table, then the win was the result of a trap that encompassed more than one plane. Stage 1 GENERAL did not attempt to recognize this type of trap. If the pattern was not in the table, then Stage 1 GENERAL could conclude that the pattern represented a trap that should be added to the table. At this point, Stage 1 GENERAL generalized by permuting, revolving, and rotating the trap. The pattern was rotated six times and revolved once. If the pattern was asymmetrical, then the result was eight distinct patterns. Otherwise the pattern was symmetrical about one axis and only four of the patterns were unique. Whatever the case, the resulting patterns were each permuted nine times. Thus, from one trap, Stage 1 GENERAL was able to add either thirty-size or seventy-two level one patterns to the table. The alternative was to add the pattern to the table and have Stage 1 GENERAL do the rotations, reflections, and permutations for each pattern table comparison. Cleary, this would have resulted in a slower comparison. An example of a permutation is as follows. Assume Stage 1 General discovered that the level 2 trap in Figure 5 was not in the table. (This trap would be stored in the table with the first short integer containing the 'X's and the second short integer containing the 'O's).

Figure 5

			X
	X		X
		O	O
X			O

Then the trap in Figure 6 would be a valid permutation and could also be added to the table.

Figure 6			
O			X
O	O		
X		X	
X			

It turns out that there were five different traps in a plane. Three of these traps were asymmetrical and two were symmetrical about one axis. When these traps were permuted and rotated, there was thus a total of 288 level one traps. Each level one trap resulted in four level two traps, so there were a total of 1,152 level two traps. Now, the storage requirements for each trap was four bytes (two short integers), so the total space requirement for the table was:

(288 X 4) + (1,152 X 4) = 5,760 bytes <6 KB

The pattern table was stored in a disk file between games. So one of SCORE4's first tasks, when invoked, was to read in the table from the disk file to memory. For simplicity's sake (user readability), only the level one permutations (72 in number) were stored in the disk file. Thus, when the file was read into memory, SCORE4 generated the remaining level one patterns, by rotating each pattern three times. The level two traps then, were obtained by copying in each level one pattern along with its three rotations.

The learing phase for Stage 1 GENERAL, consisted of matching Stage 1 GENERAL, without Polynomial evaluation, against an unexperienced version of POLY with arbitrary coefficients. This enabled Stage 1 GENERAL to lose quickly and thus learn quickly. Theoretically, in order for Stage 1 GENERAL to recognize the five different traps, it would need to lose a game for each of the five traps. But in actuality, Stage 1 GENERAL lost one game and then it was a simple matter for the author to generate the rest of the table by hand. Stage 1 GENERAL was able then, to become highly experienced in a short amount of time without relying on a skilled opponent.

3.3 Polynomial Learning

As previously stated, SCORE4 also applied Findler's version of the polynomial technique, with aggression, to the game. The resulting polynomial learning technique used a polynomial with three terms: term one represented one stone in an interesting chain; term two represented two stones in an interesting chain, and term three represented three stones in an interesting chain. Four stones in a row were not represented by a term since such a combination was a win and would thus be found by the mini-maxing algorithm before heuristics were applied.

Unlike Findler's polynomial version, the heuristic function (POLY and Stage 1 GENERAL) was applied to all positions on the board during each look-ahead procedure. The reasoning behind this is as follows. Consider the scenario where the opponent had two disjoint level two traps on the board at one time. And assume that the heuristic was only applied to the two moves made during the look-ahead procedure. Then the backed-up mini-max value would be (-1 x TRAP2_VALUE), no matter what move was chosen by SCORE4, since the opponent would always have a level two trap move available. This would result in SCORE4 choosing the first available move on the board, regardless if the move blocked one of the opponent's level two traps. Now assume that the heuristic was applied to all positions that were currently filled. Then two of the moves open to SCORE4 would have the value (-1 x TRAP2_VALUE) and the remaining open moves would have the value (-2 x TRAP2_VALUE). SCORE4 would thus choose to block one of the opponent's level two traps.

Once POLY and Stage 1 GENERAL had been completed, the polynomial learning technique could be implemented. As previously noted, Findler's polynomial learning technique (here after referred to as Version A) defined a tournament to consist of four games. In each tournament, Beta was assigned a copy of Alpha's coefficients, except that a particular coefficient was multiplied by .95 for the first two games and 1.05 for the last two games. Alpha and Beta took turns going first. Version A also defined a series to be four tournaments, one tournament for each coefficient. At the completion of a tournament, Alpha's term coefficient was increased if the outcome of the four games was one of (A, A, B, B), (A, D, B, B), (D, A, B, B), (A, A, B, D), and (A, D, D, B). The coefficient was decreased if the outcome of the four games was one of (B, B, A, A), (B, D, A, A), (D, B, A, A), (B, B, D, A), (B, B, A, D), and (B, D, D, A). Version A was implemented in its entirety. The learning process was stopped when either a series did not result in a change to a coefficient or a total of 125 series had been played.

A variation of Version A, called Version B, was also implemented. Version B was different in that a tournament consisted of two games. In each tournament Beta was assigned a copy of Alpha's coefficients, except that a particular coefficient was multiplied by a factor. This factor differed for each series. It was initially .95 and decreased by increments of .05. If one of Alpha's coefficients changed during a series, then the factor was reset to .95. Alpha and Beta took turns going first in each tournament. At the completion of a tournament, Beta's coefficient was assigned to Alpha's coefficient if Beta won both games. The learning process was stopped when the factor reached zero or a total of 125 series had been played.

3.4 Stage 2 Generalized Learning

Even though Stage 1 GENERAL did not rely on skilled opponents to become highly experienced, it did have the shortcoming of using polynomial learning to achieve significant results. Stage 2 GENERAL was developed to overcome this limitation by changing the pattern table to consist of only one level. That is,

the pattern table only contained patterns that were one move away from a trap. The amount of table space was thus decreased from 5,760 bytes to 1,152 bytes. This reduction in table size decreased the time required for table lookup and thus sped up the heuristic evaluation.

Stage 2 GENERAL's evaluation technique consisted of comparing all local planes in the board to patterns in the table. Local planes were assigned values depending on how many moves were required to turn a local plane into a table pattern. That is, if a local pattern was equal to a table pattern, then TRAP1_VALUE was added to the local plane's value. If a local pattern was one move away from a table pattern, then TRAP2_VALUE was added to the local plane's value. If a local pattern was three moves away from a table pattern, then TRAP3_VALUE was added to the local plane's value. If a local pattern was four moves away from a table pattern, then TRAP4_VALUE was added to the local plane's value. If a local pattern was more than four moves away from a table pattern, then the local plane's value was not changed. A maximum of four moves were used to change local patterns into level one traps since a level one trap requires four stones. Polynomial evaluation was not used by Stage 2 GENERAL. The C code fragment in Figure 7 performs the heuristic evaluation for Stage 2 GENERAL.

Figure 7

```
short a_local;  /* Alpha local plane */
short b_local;  /* Beta local plane */
short a_table;  /* Alpha table plane */
short b_table;  /* Beta table plane */
int f_equal;    /* are the planes equal? */


if (((a_local | b_local) & b_table) == 0)
{
    switch (bit_count(a_local & a_table))
    {
    case 4:     /* 1 move away from a trap */
        b_value = TRAP1_VALUE;
        break;

    case 3:     /* 2 moves away from a trap */
        b_value = TRAP2_VALUE;
        break;

    case 2:     /* 3 moves away from a trap */
        b_value = TRAP3_VALUE;
        break;

    case 1:     /* 4 moves away from a trap */
        b_value = TRAP4_VALUE;
        break;

    defaultl:   /* no trap */
        b_value = 0;
        break;
    }
}
```

4 Analysis

4.1 Introduction

The final task in the research was to determine the relative capabilities of each learning technique. This was done by conducting contests between different combinations of heuristics. Each contest consisted of 125 tournaments. Each tournament consisted of two games, such that each player took turns going first. The players were assigned different coefficients for each tournament. The coefficients were uniformnly distributed as follows. As previously shown, the max-

imum value returned by POLY could not exceed MAX_POLY_VALUE. Thus the maximum allowable value of a single coefficient was given by:

```
MAX_COEFF = MAX_POLY_VALUE < NUM_COEFF

where:

MAX_COEFF = 3
```

The number of different coefficient values was fixed at 5. Thus there were (5 x 5 x 5 = 125) different coefficient combinations. Five different coefficient values also gave a coefficient interval size of:

```
INVERVAL = MAX_COEFF < 5
```

The coefficient values used were thus INVERVAL, (2 x INTERVAL), (3 x INTERVAL), (4 x INTERVAL), and (5 x INTERVAL). Each player was assigned scores as indicated in Figure 8.

Figure 8	
Player didn't start the game and won.	5
Player started the game and won.	4
Player didn't start the game and tied.	3
Player started the game and tied.	2
Player didn't start the game and lost.	1
Player started the game and lost.	0

Each tournament's score was obtained by subtracting the second player's scores from the first player's scores and was thus tallied as follows:

A_1 - B_1 + A_2 - B_2

where:

A_1 = first player's score for game 1.

B_1 = second player's score for game 1.

A_2 = first player's score for game 2.

B_2 = second player's score for game 2.

Freund and Walpole [2] have shown that for the null hypothesis *($H_0 : \delta = 0$)* the estimated variance is given by:

$$(\hat{S}_{\bar{d}})^2 = \frac{\sum d^2 - n\bar{d}^2}{(n-1)n}$$

where:

d = tournament score.

$\bar{d}$ = average tournament score.

n = number of tournaments in a contest.

Plugging in the value of n we have:

$$(\hat{S}_{\bar{d}})^2 = \frac{\sum d^2 - (125 \times \bar{d}^2)}{15{,}500}$$

Freund and Walpole also defined the test statistic to be:

$$\frac{|\hat{d} - \delta|}{\hat{s}_{\bar{d}}}$$

Plugging in the value of δ we have:

$$\frac{|\hat{d}|}{\hat{s}_{\bar{d}}}$$

The critical value of t was defined (by Freund and Walpole) to be:

$$t_{(1-\sigma)/2, n-1}$$

where:

t_σ = the significance level of the test.

If we use a significance level of 95%, then we have (by Freund and Walpole):

$$t_{.475,124} = 1.96$$

If we use a significance level of 99%, then we have (by Freund and Walpole):

$$t_{.495,124} = 2.58$$

Freund and Walpole also showed that *($H_0 : \delta = 0$)* could be rejected if the test statistic was greater than 1.96. If the test statistic was greater than 1.96 and not greater than 2.58, then the probability of rejecting a true hypothesis was .05. And if the test statistic was greater than 2.58, then the probability of rejecting a true hypothesis was .01. Kmenta [4] adopted the following terminology. If the test statistic is less than 1.96, then results are significantly equal. If the test statistic is between 1.96 and 2.59, then results are significantly unequal. And if the test statistic is greater than 2.58, then the results indicate that the difference is highly significant.

4.2 Playoff Results

The first set of contests were held to determine if polynomial learning (versions A and B) resulted in a significantly increased winning ability. Both versions of polynomial learning were applied to various heuristic combinations. As previously stated, coefficient values were uniformly distributed for each learning series. Once a particular series was complete, a contest was held between the original (or unexperienced) heuristic and the resulting (or experienced) heuristic. The difference was that the original heuristic used the original coefficient values and the resulting heuristic used the coefficient values that were the result of the learning series. For example, assume that the heuristic combination was polynomial with aggression and we were testing polynomial version A. Then a series, using learning technique A, was applied to the heuristic combination - polynomial with aggression. A contest was then held between player one, which used polynomial with aggression and the original coefficient values, and player two, which used polynomial with aggression and the coefficient values that were output from the learning series. Table 1 gives the resulting average tournament scores and test statistics for contests held between experienced and unexperienced polynomial heuristics. The rows are in order of increasing test statistic.

The second set of contests were held to compare polynomial learning version A to polynomial learning version B. Both versions were given the same initial coefficient values. The contests were held using coefficient values obtained from the learning series. Table 2 gives the resulting average tournament scores and test statistics. The rows are in order of increasing test statistic.

The third set of contests were held to compare polynomial learing to polynomial learning with aggression. Table 3 gives the resulting average tournament scores and test statistics. The rows are in increasing test statistic.

The fourth through ninth set of contests were held to compare polynomial learning to Stage 1 pattern table lookup. The following six tables give the resulting average tournament scores and test statistics. The rows are in order of increasing test statistic.

The tenth set of contests were held to compare Stage 2 Generalization to all other heuristic combinations. Table 10 gives the resulting average tournament scores and test statistics.

The final set of contests were held to compare Stage 2 Generalization without aggression to Stage 2 Generalization with aggression. Table 11 gives the resulting average tournament scores and test statistics.

For all tables, Figure 9 gives the definitions for the test symbols.

5 Conclusions

5.1 Acknowledgements

I would like to thank Dennis F. Cudia for his assistance and quidance in writing this thesis. It was Dr. Cudia who first presented the idea of applying a statistical

Figure 9
P = polynomial without aggression.
A = polynomial with aggression.
P1 = level one of pattern table and polynomial without aggression.
A1 = level one of pattern table and polynomial with aggression.
P2 = level one and two of pattern table and polynomial without aggression.
A2 = level one and two of pattern table and polynomial with aggression.
*.A = polynomial version A.
*.B = polynomial version B.

Polynomial Learning			
Contest	Unexperienced	Experienced	Average Score / Test Statistic
1	P2	P2.B	0.13/0.282
2	A2	A2.A	-0.06/1.000
3	P1	P1.A	0.16/1.679
4	A1	A1.A	0.32/2.555
5	P2	P2.A	-0.64/3.284
6	P	P.A	-0.74/4.002
7	A1	A1.B	-2.56/4.610
8	A2	A2.B	3.14/6.420
9	A	A.A	-4.38/10.319
10	A	A.B	-4.38/10.319
11	P	P.B	-4.70/12.281
12	P1	P1.B	-4.32/13.236

Table 1:

analysis to learning techniques. Therefore, his contribution to this research was invaluable.

5.2 Programming Details

Table 12 and Table 13 give approximate sizes of the routines used in this project and by Samuel's Checkers programs.

The Alpha-Beta routine, used by Score-Four, was capable of making 12 moves per millisecond. Table 14 and Table 15 give the approximate computation times for the Score-Four routines.

There are a couple of interesting points to made about the results in table 14. First, Stage 1 Generalization with Polynomial was quicker than simple Stage 1 Generalization. This is the result of simple Stage 1 Generalization only returning values for planes that are 1 and 2 moves away from a trap. When Polynomial is used with Stage 1 Generalization, then the Polynomial heuristic is applied when the Stage 1 Generalization heuristic does not find a trap. Stage 1 Generalization

Polynomial A vs. Polynomial B			
Contest	Version A	Version B	Average Score / Test Statistic
1	P2	P2	0.64/1.366
2	A1	A1	-2.69/4.934
3	A2	A2	3.20/6.539
4	A	A	-4.19/9.843
5	P1	P1	-4.22/13.240
6	P	P	-4.83/13.583

Table 2:

Polynomial vs. Aggression			
Contest	Polynomial Version	Agression Version	Average Score / Test Statistic
1	P.B	A.B	0.64/1.353
2	P	A	-0.51/1.782
3	P.A	A.A	-0.77/2.351
4	P1.A	A1.A	1.79/5.072
5	P1	A1	1.79/5.203
6	P2.A	A2.A	-1.47/5.288
7	P2	A2	-1.79/5.983
8	P1.B	A1.B	3.10/6.188
9	P2.B	A2.B	-3.33/7.775

Table 3:

Stage 1 Generalization vs. Polynomial (unexperienced)		
Contest	Generalization Heuristic	Average Score / Test Statistic
1	2	0.93/1.580
2	1	-0.83/2.232
3	A2.A	1.01/2.596
4	A2	1.12/2.729
5	P2	1.98/4.375
6	P2.A	2.11/4.612
7	A2.B	2.08/5.230
8	A1.A	2.18/5.272
9	A1	2.21/5.384
10	P1	2.05/6.214
11	P1.A	2.21/6.981
12	P2.B	3.33/7.732
13	A1.B	4.22/10.573
14	P1.B	5.63/17.011

Table 4:

Stage 1 Generalization vs. Aggression (unexperienced)		
Contest	Generalization Heuristic	Average Score / Test Statistic
1	A2	0.74/1.452
2	A2.A	0.74/1.452
3	2	-1.25/2.503
4	P2	1.57/3.885
5	P2.A	1.57/3.885
6	1	2.62/4.968
7	A2.B	2.66/6.418
8	A1	2.18/7.261
9	A1.A	2.21/7.458
10	P2.B	3.84/9.092
11	P1.A	3.65/9.332
12	P1	3.74/10.162
13	A1.B	4.64/10.567
14	P1.B	6.30/21.865

Table 5:

Stage 1 Generalization vs. Polynomial (Version A)		
Contest	Generalization Heuristic	Average Score / Test Statistic
1	2	0.42/0.685
2	1	-0.90/2.615
3	A2.A	1.25/2.857
4	A2	1.28/2.960
5	P2	1.95/4.048
6	A1.A	1.86/4.391
7	A1	1.89/4.492
8	P2.A	2.14/4.615
9	P1	1.92/5.617
10	A2.B	2.34/5.706
11	P1.A	1.98/6.262
12	P2.B	3.65/8.708
13	A1.B	4.32/10.793
14	P1.B	5.60/17.789

Table 6:

Stage 1 Generalization vs. Polynomial (Version B)		
Contest	Generalization Heuristic	Average Score / Test Statistic
1	A1.B	0.19/0.423
2	P2.B	0.35/1.045
3	P2.A	-0.48/1.199
4	P2	-0.58/1.457
5	A2	-0.74/1.617
6	A2.A	-0.77/1.676
7	P1.A	-1.34/2.474
8	P1	-1.44/2.656
9	A2.B	1.82/3.965
10	A1.A	-2.78/6.002
11	A1	-2.82/6.197
12	P1.B	2.69/7.294
13	1	-4.22/10.007
14	2	-5.92/14.001

Table 7:

Stage 1 Generalization vs. Polynomial with Aggression (Version A)		
Contest	Generalization Heuristic	Average Score / Test Statistic
1	A2	0.77/1.514
2	A2.A	0.80/1.569
3	2	-1.38/2.729
4	P2	1.66/4.196
5	P2.A	1.66/4.196
6	1	2.62/4.969
7	A1	1.79/6.201
8	A1.A	1.79/6.444
9	A2.B	2.69/6.502
10	P1.A	3.20/8.079
11	P1	3.30/8.925
12	P2.B	4.00/9.722
13	A1.B	4.61/10.402
14	P1.B	6.18/20.150

Table 8:

Stage 1 Generalization vs. Polynomial wht Aggression (Version B)		
Contest	Generalization Heuristic	Average Score / Test Statistic
1	1	-0.38/0.646
2	A2	-0.86/1.684
3	A2.A	-0.86/1.684
4	P2.B	0.74/1.685
5	A2.B	1.15/2.044
6	P1	-1.28/2.203
7	A1.B	0.67/2.408
8	P1.A	-1.60/2.793
9	A1	-1.60/3.031
10	A1.A	-1.86/3.522
11	P2	-1.86/3.589
12	P2.A	-1.98/3.806
13	P1.B	2.69/5.193
14	2	-4.99/12.889

Table 9:

with Polynomial thus returns values for planes that are 1, 2, 3, and 4 moves away from a trap. The result is that Stage 1 Generalization with Polynomial experiences more pruning in the Alpha-Beta search tree, thus requiring less moves to be made by the Alpha-Beta look-ahead procedure.

Another interesing point is that Aggression is quicker than Polynomial no matter what the heuristic combination. This is a result of the aggression factor forcing the Polynomial heuristic not to be applied equally to Alpha and Beta. This results in the player that is ahead playing more aggressively - which leads to an increase in search tree pruning.

Samuel's Checkers program usually requires 30 seconds to select a move. Table 16 gives the approximate computation times for Samuel's checkers playing program.

It is difficult to compare the sizes of the two games since Score-Four was written in C and Checkers was written in assembler. But the Score-Four executable is significantly larger than the Checkers executable. This size difference can be partially accounted for by the use of assembler in the Checkers game. Most of the difference though, is due to the greater complexity of Score-Four. This complexity is a result of the Score-Four board consisting of three dimensions as apposed to the two dimensional Checkers board. Thus, checking for a win in Score-Four requires the examination of as many as seven planes. In Checkers, a single counter can be used to indicate a win.

Even though Score-Four is more complex than Checkers, the Score-Four Alpha-Beta routine is about 18 times as fast as the Checkers Alpha-Beta routine (12 moves per millisecond vs. 1 move in 1.5 milliseconds). And the Score-

Stage 2 Generalization vs. All Other Heuristic Combinations			
Contest	Type 1 Heuristic	Type 2 without Aggression	Type 2 with Aggression
		Average / Test Statistic	Average / Test Statistic
1	P	-0.75/8.955	-0.62/5.176
2	P.A	-0.79/9.376	-0.63/5.292
3	P.B	-0.87/9.048	-0.87/8.280
4	A	-1.42/15.208	-1.03/11.638
5	A.A	-1.42/15.208	-1.02/11.453
6	A.B	-0.92/7.667	-1.10/11.801
7	1	-2.00/∞	-2.00/∞
8	2	-2.00/∞	-2.00/∞
9	P1	-0.75/8.955	-0.62/5.176
10	P1.A	-0.79/9.376	-0.63/5.292
11	P1.B	-0.87/9.048	-0.87/8.279
12	A1	-1.42/15.208	-1.03/11.638
13	A1.A	-1.42/15.208	-1.02/11.453
14	A1.B	-0.92/7.667	-1.10/11.801
15	P2	-0.75/8.955	-0.62/5.176
16	P2.A	-0.79/9.376	-0.63/5.292
17	P2.B	-0.87/9.048	-0.87/8.279
18	A2	-1.42/15.208	-1.03/11.638
19	A2.A	-1.42/15.208	-1.02/11.453
20	A2.B	-0.92/7.667	-1.10/11.801

Table 10:

Type 2 Generalization vs. Type 2 Generalization with Aggresion	
average score	-0.46
test statistic	4.391

Table 11:

Approximate Size of Score-Four Routines		
Basic Score-Four playing routines.	260	C instructions
Alpha-Beta move routines.	1,650	C instructions
Polynomial heuristic routines.	100	C instructions
Aggression routines.	460	C instructions
Other heuristic routines.	250	C instructions
Polynomial learning routines.	230	C instructions
Statistical routines.	130	C instructions
Playoff routines.	170	C instructions
Total for Score-Four using polynomial heuristic.	3,250	C instructions
Generalization heuristic routines.	30	C instructions
Generalization learning routines.	220	C instructions
Total for Score-Four using polynomial and generalization heuristics.	3,500	C instructions
Tables and constants for basic play.	60	words
Generalization table space.	1,440	words
Size of Score-Four executable.	18,700	words

Table 12:

Approximate Size of Checkers Routines		
Basic Checker playing routine.	1,100	assembler instructions
Input, move verification, and output.	1,400	assembler instructions
Game starting and terminating routines.	600	assembler instructions
Loaders, table generators, dumping, etc.	850	assembler instructions
Statistical and analytical routines.	700	assembler instructions
Polynomial learning routine.	650	assembler instructions
Total for Checkers using polynomial heuristic.	5,350	assembler instructions
Rote learning routines.	1,500	assembler instructions
Total for Checkers using polynomial and rote heuristics.	6,850	assembler instructions
Tables and constants for basic play.	700	words
Working space for basic play.	2,000	words
Working space for polynomial learning.	500	words
Working space for rote learning.		balance of memory

Table 13:

Approximate Computation Times for Score-Four Heuristics (in milliseconds)	
Polynomial.	3.1
Polynomial with Aggression.	3.1
Stage 1 Generalization (Level 1).	23.7
Stage 1 Generalization with Polynomial (Level 1).	25.9
Stage 1 Generalization with Polynomial and Aggression (Level 1).	25.9
Stage 1 Generalization (Level 2).	130.9
Stage 1 Generalization with Polynomial (Level 2).	131.9
Stage 1 Generalization with Polynomial and Aggression (Level 2).	125.7
Stage 2 Generalization.	123.8
Stage 2 Generalization with Aggression.	131.9

Table 14:

Approximate Computation Times for Score-Four Move Selection, Average and Maximum (in milliseconds)		
Polynomial.	1.4	4.8
Polynomial with Aggression.	1.2	4.2
Stage 1 Generalization (Level 1).	8.7	77.7
Stage 1 Generalization with Polynomial (Level 1).	8.1	52.9
Stage 1 Generalization with Polynomial and Aggression (Level 1).	6.2	34.2
Stage 1 Generalization (Level 2).	44.8	164.0
Stage 1 Generalization with Polynomial (Level 2).	53.8	344.2
Stage 1 Generalization with Polynomial and Aggression (Level 2).	53.2	293.7
Stage 2 Generalization.	28.3	95.3
Stage 2 Generalization with Aggression.	31.5	147.3

Table 15:

Approximate Computation Times for Checkers (in milliseconds)	
To find all available moves from a given board position.	2.6
To make a single move and find the resulting board position.	1.5
To evaluate a board position (4 terms).	2.4
To find the score for a saved board position (rote learning).	2.3
To evaluate a position (with 16 terms for polynomial learning).	7.5

Table 16:

Four Polynomial heuristic is roughly twice as fast as the Checkers Polynomial heuristic (3.1 vs. 7.5 milliseconds). This speed difference is due to the 3B2 600 being much faster than the machine that Samuel used for his Checkers project. The 3B2 600 used in this project has a 18 MHz clock and a Math Acceleration Unit. Samuel used an early IBM uniprocessor (704).

The Score-Four Polynomial and Generalization methods do have advantages over Samuel's Polynomial and Signature Table methods. Both POLY and GENERAL do not rely on skilled opponents to gain significant learning. But POLY and Stage 1 GENERAL use a minimal number of polynomial terms when compared to the 38 used by Samuel's methods. Stage 2 GENERAL, on the other hand, has the additional advantage of not using polynomial evaluation.

GENERAL differs from Findler's Generalization method in that GENERAL is able to keep the pattern table small enough so that table lookup can be used in an Alpha-Beta look-ahead heuristic. Stage 2 GENERAL is an improvement oever Findler's Generalization method since it does not rely on polynomial evaluation.

5.3 Heuristics

The purpose of this section is to draw conclusions from the eleven playoffs whose results were presented in Tables 1 through 11 in Section 4 - Analysis. The following paragraphs are devoted to the results of each playoff.

The first playoff was held to determine if Polynomial (versions A and B) resulted in a significantly increased winning ability. The results in Table 1 indicate the Polynomial learning did not significantly change the heuristic ability in the first three contests, since their test statistics were less than 1.96. The results also show the Polynomial learning resulted in a highly significant winning ability in contests 5 - 7 and 9 - 12, since their test statistics were greater than 2.58 and their average tournament scores were negative. Table 1 also indicates that Polynomial learning significantly decreased ability in contest 4, since the test statistic was between 1.96 and 2.59 and the average tournament score was positive. Finally, Table 1 indicates that Polynomial learning resulted in a highly significant decrease in ability on contest 8, since the test statistic was greater than 2.58 and the average tournament score was positive.

Table 1 shows that Version B resulted in a significant increase in ability when not used with Stage 1 Generalization (contests 10 and 11) and when used with Stage 1, Level 1 Generalization (contests 7 and 12). The table also indicates that Version B did not result in an increased ability when used with Stage 1, Level 2 Generalization (contest 1 and 8). The seemingly erratic performance of Version B is explained by remembering that in Stage 1, Level 1 Generalization, Polynomial is used if there are no Trap 1 matches and in Stage 1, Level 2 Generalization, Polynomial is used when there are no Trap 1 or Trap 2 matches. Version B thus performed well when Polynomial evaluation played a greater role in the heuristic.

Table 1 also shows that Version A resulted in a significant increase in ability when not used with Stage 1 Generalization (contests 6 and 9). The table also

indicates that Version A did not result in an increased ability when used with Stage 1, Level 1 Generalization (contests 3 and 4) and when used with Stage 1, Level 2 Generalization with Aggression (contest 2). Thus Version A also performed well when Polynomial evaluation played a greater role in the heuristic.

The results from Table 1 thus indicate that Versions A and B differ in their performance when used with Stage 1, Level 1 Generalization (B did well, A did not). It can thus be said that Version A was more adversely affected by Stage 1 Generalization than was Version B.

The second playoff was held to compare Polynomial learning Version A with Polynomial learning Version B. The results in Table 2 indicate that Version A was significantly similar to Version B in the first contest, since the test statistic was less than 1.96. The results also show that Version B was highly superior to Version A in contests 2 and 4 - 6, since their test statistics were greater than 2.58 and their average tournament scores were negative. And finally Table 2 indicates that Version A was hightly superior to Version B in contest 3, since the test statistic was greater than 2.58 and the average tournament score was positive.

Table 2 shows that Version B was significantly superior to Version A when used with Polynomial learning (contests 4 and 6) and Stage 1, Level 1 Generalization (contests 2 and 5). The table also indicates that Version B was not significantly superior when used with Stage 1, Level 2 Generalization (contests 1 and 3). These results agree with the results from Table 1. Table 1 indicates that Version A worked well with Stage 1, Level 2 Generalization (Table 1 contests 2 and 5) and that Version B did not work well when used with Stage 1, Level 2 Generalization (Table 1 contests 1 and 8). Table 1 also shows that Version A did not work well with Stage 1, Level 1 Generalization (Table 1 contests 3 and 4) and that Version B worked well with Stage 1, Level 1 Generalization (Table 1 contests 7 and 12).

The third playoff was held to compare Polynomial learning without Aggression to Polynomial learning with Aggression. The results in Table 3 indicate that unexperienced Polynomial was significantly identical to Aggression in the first two contests, since their test statistics were less than 1.96. The results also show that Aggression was significantly superior to Polynomial in contest 3, since the test statistic was between 1.96 and 2.59 and the average tournament score was negative. The table also shows that Aggression was highly superior to Polynomial in contests 6, 7, and 9, since their test statistics were greater than 2.58 and their average tournament scores were negative. And finally, Table 3 indicates that Polynomial was highly superior to Aggression in contests 4, 5, and 8, since their test statistics were greater than 2.58 and their average tournament scores were positive.

Table 3 thus shows that Aggression was signifcantly superior to Polynomial when used with any version of Stage 1, Level 2 Generalization (contests 6, 7, and 9). The table also indicates that Aggression was superior to Polynomial when used with Version A (contest 3). And Table 2 shows that Aggression is identical to Polynomial when used with unexperienced Polynomial or Version B (contests 1 and 2). Finally, the table indicates that Polynomial was superior

when used with any version of Stage 1, Level 1 Generalization (contests 4, 5, and 8).

From these results we can conclude that Aggression has a positive affect on several heuristic combinations and that it only has an adverse affect when used with Stage 1, Level 1 Generalization. One reason for this might be as follows. The aggression factor is calculated by applying Polynomial evaluation and not Stage 1 Generalization. When Polynomial is used with Stage 1 Generalization, the idea is for Polynomial to lead up to trap patterns, at which point Stage 1 Generalization would kick in. Clearly it is more difficult for Polynomial to lead to Trap 1 patterns than to Trap 2 patterns. Thus, if Polynomial doesn't blend well with Stage 1, Level 1 Generalization, then Aggression would only exaggerate the problem since Aggression has nothing to do with grap patterns.

The fourth playoff was held to compare unexperienced Polynomial learning to Stage 1 Generalization. The results in Table 4 indicate that Stage 1 Generalization was significantly equal to Polynomial in the first contest, since the test statistic was less than 1.96. The results also show that Polynomial was significantly superior to Stage 1 Generalization in the second contest, since the test statistic was between 1.96 and 2.59 and the average tournament score was negative. And finally, Table 4 indicates that Stage 1 Generalization was highly superior to Polynomial in the remaining twelve contests, since their test statistics were greater than 2.58 and their average tournament scores were positive.

Table 4 thus shows that Polynomial was significanlty superior to simple Stage 1, Level 1 Generalization (contest 2). The table also indicates that Polynomial was identical to simple Stage 1, Level 2 Generalization (contest 1). And Table 4 shows that Polynomial is inferior to all forms of Stage 1 Generalization when used with any version of Polynomial or Aggression (contests 3 through 15). We can thus conclude that Stage 1, Levels 1 and 2 Generalization are an improvement on unexperienced Polynomial learning as long as some form of Polynomial is used with the Stage 1 Generalization heurisitic.

The fifth playoff was held to compare unexperienced Polynomial learning with Aggression to Stage 1 Generalization. The results in Table 5 indicate that Stage 1 Generalization was significantly equal to Aggression in the first two contests, since their test statistics were less than 1.96. The results also show that Aggression was significantly superior to Stage 1 Generalization in the third contest, since the test statistic was between 1.96 and 2.59 and the average tournament score was negative. And finally, Table 5 indicates that Stage 1 Generalization was highly superior to Aggression in the remaining eleven contests, since their test statistics were greater than 2.58 and their average tournament scores were positive.

Table 5 thus shows that Aggression was significantly superior to simple Stage 1, Level 2 Generalization (contest 3). The table also indicates that Aggression was identical to simple Stage 1, Level 2 Generalization (contest 1) and Version A with Stage 1, Level 2 Generalization (contest 2). And Table 5 shows that Aggression is inferior to all other forms of Stage 1 Generalization (contests 4 through 14). We can thus conclude that Stage 1 Generalization is not inferior to Aggression as long as some form of Polynomial is used with the Stage 1, Level

2 Generalization heuristic. We can also conclude that Stage 1 Generalization is superior to Aggression as long as Aggression is not used with the Stage 1, Level 2 Generalization heuristic.

The sixth playoff was held to compare Version A of Polynomial learning to Stage 1 Generalization. The results in Table 6 indicate that Stage 1 Generalization was significantly equal to Version A in the first contest, since the test statistic was less than 1.96. The results also show that Version A was highly superior to Stage 1 Generalization in the second contest, since the test statistic was greater than 2.58 and the average tournament score was negative. And finally, Table 6 indicates that Stage 1 Generalization was highly superior to Version A in the remaining twelve contests, since their test statistics were greater than 2.58 and their average tournament scores were positive.

Table 6 thus shows that Version A was significantly superior to simple Stage 1, Level 1 Generalization (contest 2). The table also indicates that Version A was identical to simple Stage 1, Level 2 Generalization (contest 1). And Table 6 shows that Version A is inferior to all forms of Stage 1 Generalization when used with any version of Polynomial or Aggression (contests 3 - 14). We can thus conclude that Stage 1 Generalization is an improvement on Vesion A as long as some form of Polynomial learning is used with the Stage 1 Generalization heuristic.

The seventh playoff was held to compare Version B of Polynomial learning to Stage 1 Generalization. The results in Table 7 indicate that Stage 1 Generalization was significantly equal to Version B in the first six contests, since their test statistics were less than 1.96. The results also show that Version B was significantly superior to Stage 1 Generalization in contest 7, since the test statistics were between 1.96 and 2.59 and the average tournament score was negative. The table also shows that Version B was highly superior to Stage 1 Generalization in contests 8, 10, 11, 13, and 14, since their test statistics were greater than 2.58 and the average tournament scores were negative. And finally, Table 7 indicates that Stage 1 Generalization was highly superior to Version B in contests 9 and 12, since their test statistics were greater than 2.58 and their average tournament scores were positive.

Table 7 thus shows that only Version B with Stage 1, Level 2 Generalization and Aggression (contest 9) and Version B with Stage 1, Level 1 Generalization (contest 12) were significantly superior to Version B. All other forms of Stage 1 Generalization were identical to or inferior to Version B (contests 1 - 8, 10, 11, 13, and 14). From these results we can conclude that Version B is not inferior to unexperienced Stage 1 Generalization, Stage 1, Level 2 Generalization, and Stage 1, Level 1 Generalization with Aggression.

The eighth playoff was held to compare Version A of Polynomial learning with Aggression to Stage 1 Generalization. The results in Table 8 indicate that Stage 1 Generalization was significantly equal to Version A with Aggression in the first two contests, since their test statistics were less than 1.96. The results also show that Version A with Aggression was highly superior to Stage 1 Generalization in the third contest, since the test statistic was greater than 2.58 and the average tournament score was negative. And finally, Table 8 indicates

that Stage 1 Generalization was highly superior to Version A with Aggression in the remaining eleven contests, since their test statistics were greater than 2.58 and their average tournament scores were positive.

Table 8 thus shows that Version A with Aggression was significantly superior to simple Stage 1, Level 2 Generalization (contest 3). The table also indicates that Version A with Aggression was identical to unexperienced Stage 1, Level 2 Generalization with Aggression (contest 1) and to Version A with Stage 1, Level 2 Generalization and Aggression (contest 2). And Table 8 shows that Version A with Aggression is inferior to all other forms of Stage 1 Generalization (contests 4 - 14). We can thus conclude that all forms of Stage 1, Level 1 Generalization are superior to Version A with Aggression. We can also conclude that Stage 1, Level 2 Generalization is superior to Version A with Aggression as long as some form of Polynomial without Aggression is used with the Stage 1 Generalization heuristic.

The ninth playoff was held to compare Version B of Polynomial learning with Aggression to Stage 1 Generalization. The results in Table 9 indicate that Stage 1 Generalization was significantly equal to Version B with Aggression in the first four contests, since their test statistics were less than 1.96. The results also show that Version B with Aggression was significantly superior to Stage 1 Generalization in contest 6, since the test statistic was between 1.96 and 2.59 and the average tournament score was negative. The table also shows that Version B with Aggression was highly superior to Stage 1 Generalization in contests 8 - 12, and 14, since their test statistics were greater than 2.58 and the average tournament scores were negative. Table 9 also indicates that Stage 1 Generalization was significantly superior to Version B with Aggression in contests 5 and7, since their test statistics were between 1.96 and 2.59 and their average tournament scores were positive. And finally, the table indicates that Stage 1 Generalization was highly superior to Version B with Aggression in contest 13, since the test statistic was greater than 2.58 and the average tournament score was positive.

Table 9 thus shows that Version B with Aggression was significantly inferior to Version B with Stage 1, Level 2 Generalization and Aggression (contest 5) and Version B of Stage 1, Level 1 Generalization with or without Aggression (contests 7 and 13). The table also indicates that Version B with Aggression was identical to simple Stage 1, Level 1 Generalization (contest 1), Stage 1, Level 2 Generalization with Aggression (both unexperienced and Version A) (contests 2 and 3) and Version B with Stage 1, Level 2 Generalization (contest 4). And Table 9 shows that Version B with Aggression is superior to all other forms of Stage 1 Generalization (contests 6, 8 - 12, and 14).

We can thus conclude that Version B with Aggression is not inferior to unexperienced Stage 1 Generalization, simple Stage 1 Generalization, and Stage 1, Level 2 Generalization. We can also conclude that Version B with Aggression is not superior to Version B with Stage 1 Generalization.

The tenth playoff was held to compare Stage 2 Generalization to all other heuristic combinations. The results in Table 10 indicate that Stage 2 Generalization was highly significantly superior to all other heuristic combinations since

the test statistics were greater than 2.58 and the average tournament scores were positive.

The eleventh playoff was held to compare Stage 2 Generalization without aggression to Stage 2 Generalization with aggression. The results in Table 11 indicate that Stage 2 Generalization with aggression was highly significantly superior to Stage 2 Generalization without aggression since the test statistic was greater than 2.58 and the average tournament score was negative.

The following are my conclusions from the results of eleven of the playoffs. First, Tables 1 and 2 show that Version A of Polynomial learning only works well when Stage 1 Generalization is not used in the heuristic. Version B, on the other hand, provides an improvement both without Stage 1 Generalization and with Stage 1, Level 1 Generalization. We can thus conclude that Version B provides a better learning mechanism than Version A. From Table 3 we can conclude that Aggression provides better results than Polynomial without Aggression as long as Stage 1, Level 1 Generalization is not used. Tables 4 - 14 show us that Stage 1 Generalization works well with most heuristic combinations and poorly with some others. Specifically, the combinations - Version B with Trap 1 and Aggression, and Version B with Traps 1 and 2, are superior to all combinations not using Stage 1 Generalization. The fact that all three of these combinations involve Version B gives added weight to the superiority of Version B over Version A. From Table 10 we can conclude that Stage 2 Generalization is superior to all other heuristic combinations. From Table 11 we can conclude that Stage 2 Generalization with aggression is superior to Stage 2 Generalization without aggression.

6 Open Problems

Samuel presented the idea of dividing the game into stages. The idea is that there are different strategies for the beginning, middle, and end of the game. Actually, there could be any number of stages in a game. This concept could be implemented by using multiple sets of coefficients and pattern tables. Each set would correspond to a stage in the game. This idea could be implemented by POLY and GENERAL.

Samuel terminated his learning algorithm after a given number of moves had been made. That is, he assumed that if a learning strategy was not good enough to win in say 20 moves, then the game was assumed to end in a draw. This has the result of choosing strategies that are aggressive and not defensive. This idea could be implemented by both Versions A and B.

Stage 2 Generalization should be applied to checkers and statistically compared to Samuel's Table technique. Stage 2 Generalization should be also applied to Go-Moku and statistically compared to Findler's Generalization technique.

References

[1] N. V. Findler, "Some new approaches to machine learning," in *IEEE Transactions on Systems Science and Cyberbetics*, vol. SSC-5, pp. 173–182, July 1969.

[2] J. E. Freund and R. E. Walpole, *Mathematical Statistics*. Prentice Hall, 1980.

[3] A. K. Griffith, "A comparison and evaluation of three machine learning procedures as applied to the game of checkers," in *Artifical Intelligence*, pp. 137–148, Summer 1974.

[4] J. Kmenta, *Elements of Econometrics*. MacMillian Co., 1986.

[5] A. L. Samuel, "Some studies in machine learning using the game of checkers," in *IBM Journal*, vol. 3, pp. 40–44, 1960.

[6] A. L. Samuel, "Some studies in machine learning using the game of checkers ii - recent progress," in *IBM Journal*, pp. 601–617, November 1967.