Learning a Game Strategy Using Pattern-Weights and Self-Play

Ari Shapiro[1], Gil Fuchs[2], and Robert Levinson[2]

[1] Computer Science Department
University of California, Los Angeles
Westwood, CA 90024, USA
ashapiro@cs.ucla.edu
[2] Board of Studies in Computer and Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064, USA
levinson@cse.ucsc.edu

Abstract. This paper demonstrates the use of pattern-weights in order to develop a strategy for an automated player of a non-cooperative version of the game of Diplomacy. Diplomacy is a multi-player, zero-sum and simultaneous move game with imperfect information. Pattern-weights represent stored knowledge of various aspects of a game that are learned through experience. An automated computer player is developed without any initial strategy and is able to learn important strategic aspects of the game through self-play by storing pattern-weights and using temporal difference learning.

1 Introduction

Automated computer players have been designed for many games including chess, backgammon, Go and card games such as bridge. For games such as chess and backgammon, the best computer programs can play at the level of the best humans. However, programs for games such as Go can be beaten handily by human players [1]. Typically, computer players can beat human players in games where the branching factor of the search space is small. The approaches for creating computerized players vary and can be categorized as follows: brute force searching approaches, learning approaches, knowledge approaches.

The game of Diplomacy was chosen due to its unique combination of features and difficulty of computerized play. Diplomacy is a multi-player, zero-sum and simultaneous move game with imperfect information. Past attempts at developing an automated player have applied the searching approach and the expert knowledge approach in order to develop a game strategy. In this paper, a strategy for game play is developed automatically through self-play of a non-cooperative version of Diplomacy. A learning-based approach was taken rather than a search-based approach because the tremendous search space makes a purely search-based approach intractable. Rules of the game and descriptions of the state of the game were added as knowledge without assigning any value to

these features. The automated player then extracted the important features of the game and generated a strategy for playing. This strategy was then used to generate starting moves in a game, which were then compared to a human-expert created opening book based on tournament play.

Games were played utilizing a database of knowledge that was consulted for decision-making on every turn. Partial state information was stored in a structure called a pattern-weight pair [2]. For each move, the automated player searched the database for partial state information that best matched the current state, then calculated the best move from that state. Once a game had been completed, temporal difference learning was applied to the pattern-weights and the process was repeated.

We chose to implement a non-cooperative version of the game where players do not form alliances with each other. Real games of Diplomacy typically involve cooperative strategies and negotiation tactics between players. Our simplified version of the game focusses on the strategic aspects of position and movement. Ultimately, a negotiating agent would need to be coupled with our automated player in order for our automated player to compete in real tournament play. However, a negotiating agent would utilize the information from this automated player in order to determine the benefits of a proposed cooperative strategy.

1.1 Rules of Diplomacy

Diplomacy is a seven-player board game that is based on the struggle of the major European powers during World War I. The countries played are: England, France, Germany, Russia, Italy, Austria-Hungary and Turkey [3].

Game Board. The board consists of seventy-three adjacent provinces and each player starts with pieces representing military units in their respective home countries. Thirty-five of the seventy provinces on the board are called supply centers (see Appendix A, Fig. 2). The object of the game is to control eighteen of the thirty-five supply centers. A player will have as many pieces on the board as he owns supply centers. A supply center is acquired by moving a piece from an adjacent province to the supply center. The game advances by years, starting in 1901. There are three different turns every year: spring, fall and winter. Supply centers are owned if they are occupied after the fall turn. During the winter turn, additional units are built in the players' home country and then the turns advance to the subsequent year. No moves are made during the winter turn.

Pieces. Pieces are represented as one of two types of military units: armies and fleets. Fleets are allowed to move across bodies of water and coastal provinces, while armies can move onto any adjacent land province. Both types of units have equal strength in the game.

Actions and Orders. On each spring or fall turn, a piece can be ordered to perform one of the following actions: *move*, *hold* or *support*. A *move* order will

transport a piece from one province to an adjacent province. A *hold* order will keep the piece in its current location. A *support* order will assist another piece that is moving from one province to a province adjacent to the piece executing the support order. This assistance allows the moving piece to take control of a province during conflict situations as described below.

Fleets are also allowed to order a convoy action, which transports the army piece across the body of water that is occupied by the fleet to any coastal province adjacent to that body of water. This is the only exception to the rule that states that pieces may only move one province at a time.

Game Play. A face-to-face game of Diplomacy involves a negotiation period during each turn of approximately 30 minutes where players attempt to gain the trust and cooperation of the other players. Players meet with each other in private rooms to discuss collaborative strategy and to establish alliances. Alliances can be made so as to coordinate the moves of each player's countries, as well as to define 'demilitarized zones' (DMZs) which are provinces deliberately kept unoccupied by both players. Once the 30 minutes has expired, each player secretly writes their actions on a piece of paper and the actions are resolved simultaneously. The following is an example of a set of orders for the French player for the initial turn:

```
Fleet in Brest to Mid-Atlantic Ocean
Army in Marseilles to Piedmont
Army in Paris to Burgundy
```

Simultaneous Moves and Standoffs. Since the moves are recorded in secret, no player knows the moves of the other players until after they have committed to their moves for the turn. This differs from turn-based games where the current state of the game and subsequent state after applying the operators is known. In addition, the simultaneous resolution of moves creates potential conflicts, as two pieces may move into the same province. In order to resolve conflicts, pieces that move into the same province create a standoff, where both pieces remain in their original provinces and do not move. This, in turn, could create additional standoffs, as pieces moving into the provinces of the pieces that were involved in a standoff would also generate a standoff, thus a single stand-off could create a chain of multiple standoffs.

Supports and Cooperation. A standoff may be overcome with the aid of a third piece that is not involved in the standoff. This piece issues a *support* action which effectively aids the movement of one of the pieces into the contested province. This *support* action gives the moving piece the additional strength necessary to usurp an occupied province, or to win a contest between another piece moving into the same province. However, if each piece moving into a province is subsequently supported by another piece, then another standoff is created.

Table 1. Average number of moves

Type of Units	Avg. No. of Moves	Avg. No. of Supports	Avg. No. of Holds	Avg. No. of Convoys
Army	3.22	1.11	1	.56
Fleet	4.15	1.58	1	.79

Whichever piece retains a greater amount of support will be allowed to move into the province. This aspect leads to cooperation among players, since the initial positions of the players leave them very little room to move without coming into conflict with another player. Often, two players will cooperate by supporting each other's pieces against a third player. In the example below, the Russian and Turkish players are coordinating their moves against an enemy piece in Budapest:

Russia: `Army in Ukraine to Galicia`
Turkey: `Army in Rumania Support Army Ukraine to Galicia`

The effect of supporting a piece will be nullified if another piece moves into the province occupied by the piece giving the support. In effect, the piece performing the support must 'defend itself' against its attacker and will be unable to aid another piece.

Popularity. Diplomacy is played by tens of thousands of people worldwide in both face-to-face, as well as email and web-based versions. Email and web-based versions typically take one week between turns.

2 Analysis of the Search Space

Search-based approaches to game playing utilize a search tree where each node represents the state of the game and each edge represents an operator available from that node [1]. An evaluation function is applied to the leaves on the tree to determine the value of choosing the operators. The state of the game can be represented by the locations of each player's pieces on the board and the supply centers controlled by each country.

The Diplomacy game board can be viewed as two undirected graphs, one that indicates army adjacencies and one that indicates fleet adjacencies. The average degree of each node in the fleet adjacency graph is 4.15 and for the army adjacency 3.22.

The board consists of 73 provinces, so assuming an even distribution of pieces on the board, each piece will have the following number of average moves per turn:

Since the game is a seven-player simultaneous move game, an operator is the combination of all moves by all players during a turn. On average, each piece

has approximately six moves each turn. During most of the game, all thirty-five supply centers are occupied and there are thirty-five pieces on the board. Since each piece can have six different orders, there are approximately 6^{35} move combinations, which is approximately 2^{91}. In the beginning of the game, there are fewer pieces on the board, since most countries start with three pieces (Russia starts with four). It has been shown that there are over four trillion initial move combinations [4]. This huge search space makes a brute-force search impossible without using heuristics. Clearly, this search space must be reduced in order to perform a search in a reasonable amount of time.

Simultaneous move games, such as the Prisoner's Dilemma or Chicken can be modeled with a payoff matrix. Small matrices can be solved by finding dominant strategies in order to generate Nash Equilibra. However, a payoff matrix in Diplomacy would be a seven-dimensional matrix with 6^{35} entries, which would require an extremely long time to search fully. In addition, the payoff matrix would only represent the next move in the game and not the entire game.

A simple way to reduce the search space is to only consider other pieces that could affect the results of an order. Since pieces can only move one province at a time, each piece needs only to evaluate the moves of the other pieces within a distance of two. However, the nature of the heuristic evaluation function will have a direct effect on the size of the search space. Heuristics that include all pieces on the board will require a search of all combinations. Heuristic evaluation functions that examine only one player's pieces will require a smaller subset of the board.

2.1 Use of Knowledge in a Heuristic Evaluation

In order to determine a good heuristic evaluation function, knowledge of the game is needed [5]. A simple evaluation function would be to generate the number of supply centers occupied by the player. Loeb [6] utilizes a heuristic evaluation function based on rating the provinces according to their 'height'. The 'height' of a province is based on a mountain metaphor where the peaks of the mountain reside in the player's home country and the mountain descends into the provinces of the other players. The height of a province is reduced when other player's pieces are adjacent to the province and increased when the player's own pieces are adjacent. The orders are then made based on the provinces with the greatest height.

This heuristic clearly utilizes knowledge about the game. By rating the home provinces higher than neighboring provinces, the heuristic values protection of the homeland. Also, by reducing the value of the provinces due to other player's pieces, the heuristic encodes the probability of success of a move due to the presence of other pieces that could stop the move. However, the effectiveness of Loeb's heuristic is not clear.

Experts often spend a great deal of time working in conjunction with the software designer in order to obtain the proper heuristic evaluation. A self-learning approach removes the need for an expert in order to develop a game-playing strategy.

Table 2. Features used

Feature	Values
Province Occupation	*supply center, non-supply center*
Supply Center Control	*you, enemy, no one*
Piece Type	*army, fleet*
Order Type	*hold, move, support, convoy*
Adjacent Province Occupation	*none, you, enemy*

3 Pattern-Weights

Levinson describes patterns in [7] as representations of previous experience during game play. These patterns represent partial positions on the game board. The pattern-weights were used for a chess program called Morph as a way to remember game experiences. Instead of saving the entire chess board, a reduced board was saved to a database. This reduced board showed only the attack patterns in chess, for example. In this representation, a white queen that is capable of attacking a black knight is represented as a directed graph from a node representing the white queen to a node representing the black knight. Thus, this pattern can represent any number of game states that reflect that attack situation.

Each pattern is associated with a weight that is used as a heuristic evaluation of the actual state. The advantage of using pattern-weights is that through self-play and learning mechanisms, the weights are tuned so as to develop a game strategy without any initial value of the various board positions. The learning mechanism will evaluate all the knowledge in the game and choose only the most important ones. Similarly, strategic aspects of the game can be individually analyzed by examining the pattern-weight database.

Pattern-weights can only be ordered in a generalization-specialization hierarchy where the least specific patterns lead to more specific ones [8]. Thus general patterns are applicable to many states but are less accurate than more specific ones that apply to fewer states. Pattern-weights that consist of n features will be called n-pattern-weights.

The pattern-weights used in Levinson's chess program were of two types: material patterns and attack patterns. Material patterns represented the relative assessment of black and white pieces remaining in the game. Thus, "down one bishop" received a pattern assessment. Attack patterns are the subsets of the game graph as described above.

Diplomacy utilizes a board that can be viewed as an undirected graph. Pieces can only move one space at a time, so while a pattern-weight for chess represents attack formations, the pattern-weight for Diplomacy uses the adjacent nodes in the graph and examines both the adjacent pieces and the aspects of the provinces themselves.

Fig. 1. Feature example

Distinct features of the state were chosen that describe the state of the board and the influences on the specific orders given to each piece. Each of the features that are represented in the pattern-weight database are shown in Table 2.

These features can be combined and are independent of each other. Thus, by examining the pattern-weight database we can see the relative values of each of these features. For example, the Province Occupation, Order Type and Piece Type pattern can be combined to determine the value placed on moving into a supply center that is occupied by an enemy army. Additional features of the game can be added. Each feature will produce a pattern that will be stored as a pattern-weight whose value can be learned.

For example, in Fig. 1, the Russian player in Silesia, represented by the white tank, could move to Berlin, a German supply center. The order would be written as follows:

```
Army in Silesia to Berlin
```

Since the German player (represented in black) is considered to be an enemy by the Russian player, the following features are utilized from the Russian pattern-weight database: *supply center*, *owned by an enemy*, *unoccupied*, *army*, *move*, *adjacent to two enemies*.

Similarly, the German player could move from Munich to Silesia, yielding the following patterns from its pattern-weight database: *not a supply center*, *occupied by enemy*, *army*, *move*, *adjacent to no one*.

3.1 Feature-based vs. Featureless Patterns

While a pattern is a proper subset of a state, a feature can be viewed as a directed abstraction of the state. A state indicates a board position, a turn, ownership of supply centers and so on, while the features were chosen as representations of perceived useful aspects of the state. The inclusion of the features listed above to define a pattern-weight has both benefits and drawbacks. Features will help guide the automated player make decisions about game play. No value was initially attached to any feature, so it is up to the learning process to decide which ones to utilize and which ones to avoid. We attempted to choose the features of the game neutrally, to avoid biasing the computer player towards preselecting one feature over another.

The initial attempts to create patterns utilized a featureless approach, where the pattern-weights did not include any features, but rather stored specific moves for each piece. Since the moves were stored, information about the state of the game was included, such as the locations of the units on the board.

A computer player utilizing a feature-based approach learns a strategy more quickly than one who uses a featureless approach. However, by restricting the search domain to that of specified features, we leave open the possibility that the features chosen do not describe the semantics of the game with the proper amount of detail. As a result, an automated player might miss some important aspect of strategy. For example, by indicating the presence of an enemy by a single value, the computer player is unable to distinguish between enemies of two different countries. Another example is a province that is surrounded by three enemies (as noted by a feature) but the actual positions of those enemies are not known as it is not recorded in the pattern.

Given long enough game play, a featureless approach has the potential of producing more surprising strategy, since the preconceived notions of its creator do not confine it.

3.2 Temporal Difference Learning

Temporal difference learning has been showed to be an effective technique for learning game play when feedback occurs only after an extended period of time [9]. All pattern-weights start with an initial weight of .5, indicating that the pattern is neither good nor bad, and ranged from zero to one. The pattern-weights also utilize an age field that indicates the frequency of occurrence of that pattern. Since patterns can occur frequently in one game, the age of the pattern for some patterns will be greater than the number of games played. The age is not used during the learning process, but rather when evaluating a pattern-weight during game play. Older pattern-weights are given slightly higher consideration than newer patterns.

Temporal difference learning is applied using a method similar to Morph [7] as follows:

$$Weight_n := \frac{Weight_{n-1} * (n-1) + k * v}{n + k - 1} \tag{1}$$

$Weight_i$ is the weight after the ith update, v is final value of the game and k is the learning rate. v is 1 when the winning conditions were met, and 0 otherwise. When $k = 0$, only past experiences are considered. When $k = 1$, a new experience is averaged with all previous ones. With a larger k, the system values the latest experience more than previous ones. An external counter varied k from 1 to 5, thus simulated annealing was utilized by keeping a *global temperature* that varied the intensity of the learned experience. Levinson [7] showed that the effect of simulated annealing on the development of pattern-weights was to hone in on specific values. Without simulated annealing, the weights were shown to fluctuate between certain ranges, while the effect in Morph with simulated annealing was a smooth transition to the final value of a material pattern-weight.

Credit assignment within the same game was handled by assigning credit equally among all patterns found by their frequency. The pattern-weights were updated at the end of each game, which typically lasted 20 moves. The pattern-weights that were updated corresponded to patterns that occurred in the game.

4 Tests

A separate pattern-weight database was generated for each player through self-play. Since the game graph is asymmetric, a separate pattern-weight database for each player was necessary. However, all the pattern-weight databases could be generated simultaneously.

The first test was done using featureless pattern-weights, where moves were explicitly stored as 1-pattern-weights and combinations of two moves were stored as 2-pattern-weights. The goal of the game was to possess the greatest amount of supply centers. The game ended when one of the players reached seven supply centers.

The second test utilized feature-based pattern-weights. No cooperation was assumed in the game; each automated player was playing against all the other automated players. The goal of the game was the same as above.

The game was run utilizing the FreeDip software, a web-based version of Diplomacy written in Java [10]. The software was modified by adding the logic for an automated player, pattern databases and self-play.

4.1 Game Play

The following sequence of actions was taken for each automated player of the game. Each step is examined in more detail below:

1. Generate all individual piece legal orders.
2. Generate all order combinations.
3. Match the moves against a pattern-weight database.
4. Weigh the moves according to the pattern-weights.
5. Choose among the top-rated moves.
6. Submit the moves and advance the turn.

1. Generate all individual piece legal orders. All possible moves, holds, supports and convoys were entered for each individual piece. At this stage, the move combinations are not generated, but only the potential orders for each piece. No cooperation was performed; so orders that involved other player's pieces were not generated. For example, it is possible to support another player's piece movement into an adjacent province, but this option was not considered.

2. Generate all move combinations. The cross product of all the orders for all the pieces was applied to generate a move set. Since each piece has approximately six orders per turn, a move set consisted of approximate 6^N move combinations where N is the number of pieces. This number becomes very large as N increases. Once N reached six, the number of move combinations was approximately 50,000, which took approximately 5 minutes to analyze per move. In order to reduce this number and maximize the number of games played, fewer orders for each piece were generated in (1) above, which yielded fewer move combinations. Randomness was used to decide which orders to generate.

3. Match the moves against a pattern-weight database. Each move was compared against the pattern-weight database in search of pattern-matches. Features of the individual orders were independently compared to the 1-pattern-weights. Next, pairs of features were combined together and compared to the 2-pattern-weights and so forth.

4. Weigh the moves according to the pattern-weights. Each pattern-weight that matched a move is associated with a value. The value of each pattern-weight found w_1 was then compared to current total value for that move w_2 by averaging them and weighing them by their size. Thus, a 1-pattern-weight contributed half of its value as did a 2-pattern-weight. This value was then averaged with the weights of all the individual moves. The final value was between 0 and 1. Any move that did not match a pattern in the pattern-weight database was assigned the value of .5.

The pattern-weights can be seen as advisors. Each advisor suggests a result for the pattern. The smaller pattern-weights, representing less specific information, were valued less than the more specific patterns. Also, the averaging of pattern-moves was done to avoid the dominance of any one order for a single piece to the pattern.

5. Choose among the top-rated moves. The weighted moves were sorted by the highest value. The top five moves were selected and one of them was chosen randomly based on their weighted value. Thus, a move with twice as much weight as another was chosen twice as often.

Table 3. 2-pattern-seight featureless test (1042 games)

Player	Number of 1-Pattern-Weights	Number of 2-Pattern-Weights
England	434	6,681
France	502	8,744
Germany	588	9,761
Italy	530	8,073
Austria-Hungary	580	6,898
Russia	589	12,397
Turkey	359	5,031

4.2 Self-Learning and Pattern-Weight Updates

Temporal difference learning was applied to the pattern-weight database by examining the relative success of the sequence of moves and then updating the weights of all of those moves. Success was determined by retaining all the home supply centers and by capturing the greatest, second greatest or third greatest number of supply centers.

If the player's move sequence was considered a success, then the sequences of moves were decomposed into 1-pattern-weights, 2-patterns-weights and n-pattern-weights containing all the features of the game state. Pattern-weights higher than 2-pattern-weights and lower than n-pattern-weights were not considered. The weights of all the pattern-weights that matched the move sequences were reinforced according to temporal difference learning as indicated above.

4.3 Test 1 Results - Featureless Pattern-Weights

Game strategies were learned via self-play as follows:

Table 3 shows the number of 1- and 2-pattern-weights generated for each country in the featureless test of 1042 games of self-play. The featureless players were also tested against random players and beat them in nearly all of the games tested.

The number of 1-pattern-weights differed between the countries due to the asymmetry of the game board. Initially, Turkey has very few moves and is confined to a corner of the board, so it tended to produce fewer patterns than other countries. Russia, on the other hand, starts with four pieces instead of three, thus generated a greater number of patterns. The size of the 1-pattern-weights is expected to grow only as large as the number of possible moves for the pieces in the game. The size of the 2-pattern-weights are limited by the number of pairs of individual moves.

Initially, moves were chosen at random, so *support*, *hold* and *move* orders were frequently seen. As more games were played, fewer support moves were chosen, perhaps due to the fact that success involved attaining supply centers quickly. The players learned to move as quickly as possible to the other provinces in order to claim the supply centers.

Competition between the automated featureless pattern-weight player showed that it learned basic strategies. Moves tended to alternate with hold orders to prevent other units from entering key locations.

There were many shortcomings to this approach. Units tended to move in line behind each other to the same locations as their predecessors. This is logical since their movements are stored and valued and units are indistinguishable from each other. Also, the units had no spatial concept of the game board and essentially played a blind game against each other and ignored the positions of other units. This would be easily fixed by storing larger than $n = 2$ n-patterns.

The games played with the pattern-weight databases revealed interesting biases in the game. Different countries tended to choose different directions to move and thus different players to move against. For example, Italy tended to ignore its neighbor, Austria-Hungary, and tended to move its fleets to Greece and Turkey in order to capitalize on those open supply centers. Similarly, France tended to invade Germany, while England found its benefits by invading France. Russia and Germany were constantly at battle with each other, particularly over Silesia, which borders both German and Russian homes. The moves chosen by the automated player seemed to reveal the relative benefit of attacking another country due to the asymmetry of the players and the board.

4.4 Test 2 Results - Feature-based Pattern-Weights

Feature-based pattern-weight tests developed a strategy much more quickly than the featureless tests. The strategy of moving quickly towards their opponents and then retaining their positions was adopted in the same way, but in a shorter amount of time. In addition, the feature-based test players did not tend to repeat the same actions, since their knowledge was based on relative positioning of the units on the board. Like the automated player in the first test, initial versions of this player suffered from a lack of a spatial concept of the game board. Since only units in immediately surrounding provinces were analyzed, the automated player was at a particular disadvantage when another unit was two spaces away,

In early tests of featureless pattern-weights, the feature that indicated the presence of units in adjacent provinces was left out. As a result, some moves were misguided. Since the pattern-weights represented only the locations of pieces, not the positions of the other pieces. In the absence of this information, a piece could not distinguish between a state where it was surrounded by enemy pieces and one where it was far away from any enemies. Like the earlier featureless tests, supports were used only rarely. Once the feature of adjacent units was added, the automated players began to implement more complicated strategies, as described below.

In general, the feature-based patterns were able to conceptually abstract large states into relatively small data sets. This had a distinct advantage over the featureless patterns that required a great amount of storage space.

Table 4. Highest weighted 1-pattern-weights

Num	1-Pattern-Weights	Weight
1.1	Supporting a piece into a supply center owned by an enemy	.9872
1.2	Supporting a piece into province that has no adjacent units	.9377
1.3	Supporting a piece into province that is not a supply center	.9028
1.4	Convoy from a supply center that is owned by an enemy	.8807
1.5	Supporting a piece into a province that is only adjacent to your own units	.8665
1.6	Supply center currently occupied is controlled by an enemy	.8618
1.7	Province moving to is occupied by an enemy	.8371
1.8	Province occupied is not adjacent to any units	.8347
1.9	Province moving to is occupied by a another one of your own pieces	.8264
1.10	Province moving to is unoccupied	.8206

4.5 1-Pattern-Weights

Table 4 shows the highest valued 1-pattern-weights in the pattern-weight database for the Turkish player after 252 games.

The highest valued moves are related to moves and supports. This is perhaps because supported moves will have a tendency to displace other units by breaking standoffs. In fact, the highest rated move for four of the seven players was *1.1* - Supporting a piece into a supply center owned by an enemy. It appeared as one of the top ten pattern-weights for the other three players.

The 1-pattern-weight *1.2* in Table 4 can be seen as a wasted move. The piece performing the support is helping its own piece move into a province that cannot possibly be occupied by an enemy on the next turn. Perhaps this is a by-product of valuing all support moves very highly, and the player has not experienced enough situations to understand that this is unnecessary, since the same move without support is guaranteed to succeed. In fact, this is a classical example of how features may produce a biased strategy. A featureless pattern would not 'over assume' the need for support. The philosophical goal of our method is to emphasize knowledge storing as opposed to the power of search. Of course a real system would benefit from doing both. However we want to see how far we can get without doing any search and relying solely on stored knowledge. To this end, we avoid the *explore* all together and concentrate on the *exploit*. So, like any young system, which has not yet had enough exposure to experience naïve mistakes, our agent would not yet know many useful facts, although more experience would eventually teach it.

The 1-pattern-weight *1.4* in Table 4 is clearly wrong. This shows a willingness to leave valuable spaces before taking control of them. Convoys are much rarer in the game, since they require a fleet in an ocean province and an army on a coastal province.

Table 5. Highest weighted 2-pattern-weights

Num	2-Pattern-Weights	Weight
2.1	Piece has no neighbors AND	1.000
	Piece it is supporting will move into province adjacent to an enemy	
2.2	Piece has no neighbors AND	.9546
	Piece it is supporting will move into province adjacent to no one	
2.3	Piece has no neighbors AND	.9500
	Piece it is supporting is a fleet	
2.4	Piece has no neighbors AND	.9207
	Piece supported will move into a location occupied by your own piece	
2.5	Piece has no neighbors AND	.9115
	Piece supported will move into a location that is not a supply center	
2.6	Piece supported will move into a supply center controlled by an enemy	.9009
	AND	
	Piece supported is a fleet	
2.7	Piece has no neighbors AND	.8895
	Piece it is supporting will move into supply center owned by an enemy	
2.8	Piece has no neighbors AND	.8864
	Piece it is supporting will move into supply center	
2.9	Piece is a fleet AND	.8551
	Piece supported is adjacent to no one	
2.10	Piece is an army AND	.8478
	Piece it is supporting will move into a supply center owned by an enemy	

4.6 2-Pattern-Weights

In the 2-pattern-weights of Table 5, there is a dominance of *Piece has no neighbors* in conjunction with a support move. This demonstrates the strategy of using support move by pieces whose support cannot be disabled by the enemy. Since the supporting pieces have no adjacent neighbors, their support cannot be cut by another piece, thus potentially leading to a better success rate with the supported move.

The top weighted n-pattern-weights represented the values of pattern-weights that matched features of the entire order. Although the analysis of these features would be complicated, the results of the opening moves can be generated and compared to an opening book of moves that has been generated by human experts through tournament play [11].

4.7 n-Pattern-Weights

Since all moves have approximately equal weight, they will be chosen approximately equally. Table 6 shows a comparison of the top ten pattern-weight moves to a human-developed opening book and to random play. The leftmost column indicates the Turkish pieces in their original positions. The right columns show the frequency of moving to a specific province or performing a different action, such as a *hold*.

Table 6. Comparison of human expert opening book to pattern-weight moves for Turkey

Piece	Human Expert Opening Book	Pattern-Weight Player	Random Player
Army Constantinople	Bulgaria 98.1% Holds 1.9%	Bulgaria 40% Holds 10% Supports 50%	Bulgaria 24.3% Holds 25.6% Supports 27.0% Smyrna 13.7% Ankara 9.6%
Fleet Ankara	Constan 36.5% Black Sea 57.7% Holds 5.8%	Constan 10% Black Sea 20% Holds 20% Supports 50%	Constan 9.6% Black Sea 25.6% Holds 27.0% Supports 17.6% Armenia 20.2%
Army Smyrna	Ankara 26.9% Holds 9.6% Armenia 32.7% Constan 26.9% Syria 1.9% Supports 1.9%	Ankara 0% Holds 20% Armenia 20% Constan 10% Syria 10% Supports 40%	Ankara 10.8% Holds 21.6% Armenia 16.2% Constan 6.8% Syria 21.6% Supports 23.0%

The random player does not choose equally among the adjacent provinces since the orders are selected from all legal moves which include *holds* and *supports*.

The support strategy seems to dominate the opening moves for the Turkish player. However, if we ignore the supports, we can see that the proportions of moves do correspond reasonably well with the opening book. For example, the army in Constantinople moves to Bulgaria only 40% of the time if supports are included, but 80% of the time if we ignore the support moves. In addition, although the piece in Constantinople has three choices of moves (Smyrna, Ankara, Bulgaria) it never chooses to move deeper into its own province. Again, this is an independently discovered strategy.

Clearly, the automated player has not yet learned that supporting a unit when it cannot be attacked is useless. This simple description requires at least a 3-patten-weight to be cognizant of the three elements: the self, a friendly support and an enemy. For this reason, any 2-pattern-weight is unable to ever learn that one need not defend against an absent attacker. This can be shown by demonstrating four possible scenarios after a foreign supply center has been conquered:

Case 1- Enemy is present to dislodge us AND we have defenses.

Case 2- Enemy is present to dislodge us AND we do not have defenses.

Case 3- Enemy is not present to dislodge us AND we have defenses.

Case 4- Enemy is not present do dislodge us AND we do not have defenses.

Case 1 would get a good positive score for the defense of our new supply center, while a bad score would be given to *Case 2* since we would be unable to

defend against the attack. Thus, we have the ability to learn the need for defense in the presence of attack. However, the situation is different for learning whether a defense was necessary in the absence of an attack. Since an enemy is not present in either *Case 3* or *Case 4* regardless of our choice of defending the province or not, we would obtain a good resulting score for either pattern. Thus either action would be sanctioned as a good one. It would appear that our temporal differencing approach to learning would only learn what is a punishment. In the absence of punishment, we would not learn what is not useful, merely what is not harmful. However, this would lead us to a wrong conclusion, since the fact that an action yields a good score whether it is chosen or not would mean that this action is unnecessary. Furthermore, if a defending unit could be deployed more effectively elsewhere, then some other 3-pattern-weights (or even higher order ones) would warrant that a defending piece could be used with a higher affinity elsewhere. So in other words, the sum of all the patterns is greater the individual values. Knowledge is not left alone in individual patterns (of course kernels of knowledge are stored in individual patterns) but it is the global mutual maximization of all patterns in concert that would yield the highest score. So, with sufficient experience, the total knowledge of the database would cause the right event to occur, even if any one of the patterns alone might the point in the wrong direction.

5 Conclusions and Future Work

The discovery of strategic use of supports by the automated player seems to indicate the potential for using pattern-weights for the game of Diplomacy. In addition, the strategy was refined upon inclusion of 2-pattern-weights, specifically the use of supports only when the supporting piece is not accessible to enemy players. This seems to indicate that more specific patterns lead to better strategy. In addition, feature-based patterns seemed to acquire this higher level strategy, whereas featureless patterns did not.

Comparisons to a human-expert generated opening book of moves show that the pattern-weights recognize certain strategies in opening moves, such as moving to unoccupied supply centers. However, the strategy of using supports seemed to dominate the opening moves, indicating the shallow understanding of the use of *support* moves.

Since the inclusion of features added to the discovery of strategic knowledge, it would seem that adding additional features would allow the system to discover even more complex strategies. One such experiment is to include recognition of both allies in addition to enemies. Thus the automated player could react differently, depending upon whether or not they are cooperating with another country.

As previously mentioned, this method serves as a partial strategy since it only models the non-cooperative aspects of the game. For this reason, it cannot be accurately compared with other complete systems or compete in tournament play. Past attempts to create an automated computer player have separated

the task into two aspects: the negotiating agent and the strategy finder. The negotiating agent communicates with other players and develops alliances. The negotiating agent determines the value of a cooperative strategy by consulting the strategy finder, which indicates the value of various moves in the game. Our system can be coupled with a negotiating agent, thus acting as a strategy finder, by adding an *ally* feature (in addition to the *enemy* feature) and then receiving a list of enemies and allies from the negotiating agent. These features in turn would be added to the pattern-weight database and utilized in the same way as the other features.

One important consideration in these experiments was the choice to model pattern-weights after the moves, rather than the state of the board. Levinson's original work on Morph utilized the position of the chess pieces on the board after choosing one particular move. This is more difficult to implement in Diplomacy since the game is not turn-based, so the final positioning of the pieces on the board is not known due to the uncertainty in predicting the opposing player's moves. Modeling the pattern-weights after moves instead of board states also reduced the size of the pattern-weight database significantly. All the pattern-weight databases were less than 100k in size and no pruning or removal of pattern-weights was necessary.

Many temporal difference learning experiments with games, such as TD-Gammon [9] have required a large number of games to be played before developing complex strategies. For example, Tesauro observed that TD-Gammon learned basic strategies early on after only a few hundred games and complicated strategies over the course of 300,000 self-played games. Clearly, more games need to be played in order to evaluate the effectiveness of pattern-weights and features on game play.

5.1 Future Directions

The following list potential future directions that can be leverages from the work described in this paper:

Future 1. Provide a performance comparison of a random player against another learning agent, an agent using Loeb's heuristic and a human player.

Future 2. Perform temporal difference learning of games played played by humans and store that knowledge in a pattern-weight database. Compare the pattern-weight database learned from human players against the pattern-weight database obtained from playing a random player.

Future 3. Analyze how two different 3-pattern-weights differ greatly in weight, even though the common 2-pattern-weights to both of them would not be able to capture that nuance. Generate an English description of a pattern which would explain why it is a good one or not a good one, based on this $n+1$ child pattern-weight analysis.

Future 4. The current patterns do not take into account specific locations of other pieces. These should be incorporated as part of the pattern-weight (larger patterns-weights would be captured without size limit) and the resulting strategy would be compared to the results of this experiment.

6 Related Work

Kraus [4] creates an automated negotiating agent that collaborates with other players, makes alliances, assesses the values of the various alliances and determines which ones to keep and which ones to break. The computer is able to negotiate with the other players though the use of a negotiation language that contains semantics for starting an alliance, creating DMZs and proposing collaborative moves. Once the negotiations are finished, the agent passes this information to the strategy finder module that finds the moves that best satisfy the various alliances made during the last negotiation period. The strategy finder was written by E. Ephrati and published in Hebrew.

Loeb [6] created a negotiating agent and a strategy finder that searched for the best moves for a country during a turn. The strategy finder would execute seven searches in parallel, one for each player. Each search process would periodically poll the other six search processes to determine the current best move for the other players. All seven moves would then be combined and evaluated using a heuristic evaluation.

Lorber [12] has created a negotiating agent that uses an opening book, expert knowledge and statistical analysis in order to determine the best move.

Microprose has developed a computerized version of Diplomacy. It utilizes both a negotiating agent and an opening book. It widely regarded as a poor strategist and easy to fool through negotiations.

7 Acknowledgements

Special thanks to Guy Tsafnat who toiled many long hours in the co-development of the FreeDip software which provided the foundation and inspiration for developing an automated computer player.

References

1. Korf, R.: Problem solving and search (2002) CS261A Course Reader, UCLA.
2. Gould, J., Levinson, R.: Experience-based adaptive search. In Michalski, R., Tecuci, G., eds.: Machine Learning 4: A Multi-Strategy Approach, Morgan Kaufman Publishers (1994) 579–604
3. Hasbro: Rules of diplomacy, 4th edition. http://www.hasbro.com/instruct/Diplomacy.PDF (2000)
4. Kraus, S., Lehmann, D.: Designing and building a negotiating automated agent. Computational Intelligence **11** (1995) 132–171
5. Kraus, S., Lehmann, D., Ephrati, E.: An automated diplomacy player. In Levy, D., Beal, D., eds.: Heuristic Programming in Artificial Intelligence, Ellis Horwood Limited (1989) 136–153
6. Hall, M., Loeb, D.: Thoughts on programming a diplomat. In van den Herik, H., Allis, V., eds.: Heuristic Programming in Artificial Intelligence 3, Chinester, England, Ellis Horwood Limited (1992) 123–145

7. Levinson, R.: Experience-based creativity. In Dartnall, T., ed.: Artificial Intelligence and Creativity: An Interdisciplinary Approach, Kluwer Academic Press (1994) 161–179
8. Levinson, R., Fuchs., G.: A pattern-weight formulation of search knowledge. Computational Intelligence **17** (2001)
9. Tesauro, G.: Temporal difference learning and td-gammon. Communications of the ACM **38** (1995)
10. Shapiro, A., Tsafnat, G.: FreeDip, open source software. http://freedip.sourceforge.net (2002)
11. Nelson, M.: Opening's custodian report for 1995. http://devel.diplom.org/DipPouch/Zine/F1996M/Nelson/Part3.html (1996)
12. Lorber, S.: Diplomacy AI software. http://www.stud.uni-bayreuth.de/ a0011/dip/ai/ (2002)

A Appendix - Diplomacy Game Board

The following is the board and initial positions of the pieces of the standard Diplomacy game. Lines delineate the provinces and the supply centers are indicated by the presence of a colored dot in the interior of the province.

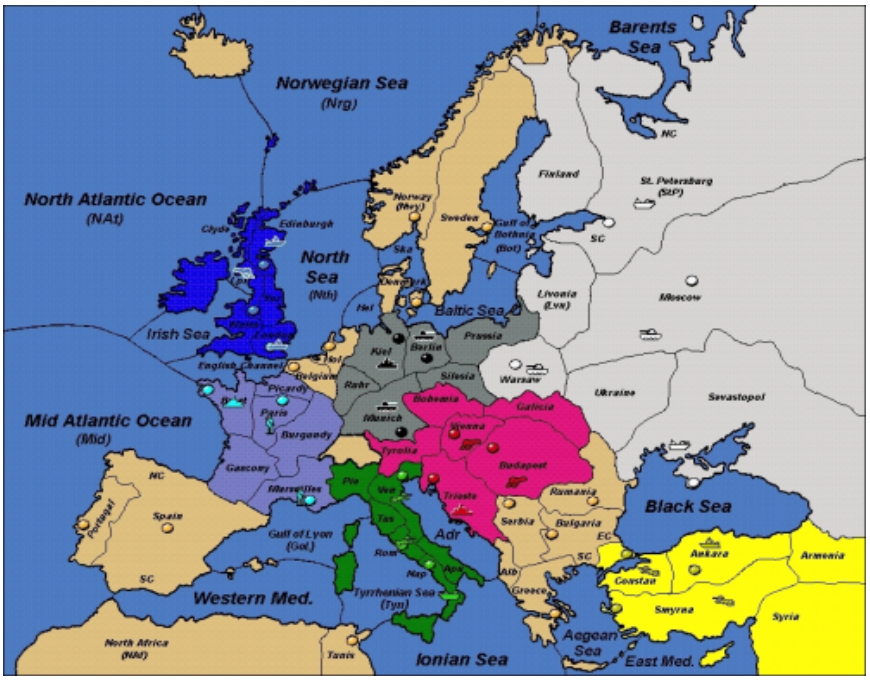

Fig. 2. Diplomacy game board