Fuzzy Patterns Based Approach for Environment Analysis in Adversarial Games

Stephany Coffman-Wolph and Dionysios Kountanis

Department of Computer Science, Western Michigan University
1903 West Michigan Avenue, Kalamazoo, Michigan 49008
stephany.s.coffman-wolph@wmich.edu and dionysios.kountanis@wmich.edu

Abstract

Adversarial game-playing situations have been commonly studied in both game theory and artificial intelligence since the 1950s. However, developing strategies for game playing has been a challenge due to large search tree size – especially as the environment size increases. In this work, we introduce a new concept entitled Fuzzy Patterns that is inspired from Stilman's zone concept and utilizes a combination of fuzzy logic and division of the environment into mini-goals to prune a large search tree. Our proposed approach provides an essential tool to break a large environment into smaller interest areas to focus, study, and select possible moves. Our proposed Fuzzy Patterns approach improves upon the existing zone concept from Stilman's Linguistic Geometry algorithm in two ways: quicker to compute and producing significantly smaller number of patterns. Even though our proposed approach is generic and can be applied to any adversarial game-playing scenario, in this paper we use a standard chessboard as the environment with standard chess pieces and movement to demonstrate the potential benefits of our proposed approach (up to 80% reduction in the number of fuzzy patterns that need to be evaluated over the non-fuzzy based approach).

Analysis of the environment for any adversarial game-playing situation can be a daunting task. The larger the environment, the greater the number of potential sequences of moves and the harder the task of choosing among various strategies. The game of chess has a fairly restricted environment of only 8 by 8. Chess is known to have an average branching factor of 35 with 50 moves per game for each side resulting in searching trees with 35100 nodes (Russell and Norvig 1995). Even the extremely simple game of tic-tac-toe contains a branching factor of 9 at the first level alone – the same size as the environment (Russell and Norvig 1995). Therefore, it is essential to develop techniques to eliminate search branches in order to improve any viable adversarial game-playing algorithm.

A pattern is used to analyze and identify potential moves for various pieces within the environment. Patterns are used primarily to sub-divide the environment into smaller and easier to analyze areas. Patterns contain five essential elements: a starting piece (location), an ending piece (location), a shortest path between the starting and ending location (the main path), the attacking or blocking pieces to this shortest path, and the shortest paths of any attacking or blocking pieces. The shortest path contains a candidate next move for the starting piece and the information to evaluate the potential impact the move has on the end goal of the system. It is important to note that each pattern only contains one valid shortest path and that all larger paths are not considered for either the main path or any attacking/blocking paths.

There are several key terms that need to be defined or clarified including: step, reach-ability, attack-ability, and shortest path. A step, in this context, is a single move made by a piece during a turn in the game (assumed to be a valid move via the rules of the game in question). Reach-ability is the ability of a piece to reach a given location within the environment given the rules or physical limitations of the piece. Attack-ability is the ability of a piece to attack/remove a piece in a given location within the environment given the rules or physical limitations of the piece. In some situations the reach-ability and attack-ability are the same – a good example is the queen piece in chess. In the case of the pawn, the reach-ability and the attack-ability are not the same. The shortest path is defined as the minimum number of moves or steps a piece needs in order to each an ending location assuming that the location is reachable or attack-able. In many game situations, including chess, there are often multiple shortest paths between two locations.

The Linguistic Geometry (LG) Search Strategy introduced by Stilman contains 4 layers (from lowest to highest): trajectories, zones, translations, and searches (Stilman 2000). The first two layers of the LG are part of the inspiration to the patterns based approach presented in this paper. The lowest level, trajectories, finds all shortest paths between a starting location and ending location (Stilman 1994). The second lowest level, zones, finds all reachable starting and ending piece combinations and any intersections. The zone is used to analyze the environment in small focused areas (Stilman 2000).

There are several key differences between the zone and the pattern concepts. The LG algorithm is written as a series of grammars that are designed to be used with older

rule-based or recursive computer programming languages (Stilman 2000). Our proposed pattern based algorithm is designed to be used with modern object-oriented programming languages (java, C#, etc). The LG grammars are recursive in nature (Stilman 2000) while the algorithms for patterns are not. The storage of a pattern is an object while a zone is a complex string that requires complex processing for interpretation. The pattern is also a computationally less complex algorithm due to restricting the depth of examination of intersections. Patterns only examine the first level of pieces that can attack or block the main path while the zones look at all levels in a recursive manner. A major difference between zones and patterns is that patterns differentiate between reach-ability and attack-ability for each given piece. The grammar of trajectory, a method for calculating all the shortest paths between two elements, is used extensively in the pattern generation algorithm but it has been modified from the recursive grammar format to a straightforward procedural algorithm.

This paper presents the algorithm to generate fuzzy patterns. A detailed example is used to illustrate the applicability of our proposed approach to an adversarial game. Additionally, an in-depth analysis of the efficacy of using our proposed fuzzy approach is presented.

The Problem Description

The environment is the area to which the patterns (fuzzy or non-fuzzy) are applied. For the examples in this paper, the environment is a chessboard. A standard chessboard is 8 rows by 8 columns for a total of 64 squares. Each chessboard square is shaded by one of two contrasting colors in an alternating dark/light color scheme (the color scheme is commonly referred to as black and white, sometimes white and red or even black and red). A standard chess set comes with 8 pawns, 2 rooks, 2 bishops, 2 knights, 1 queen, and 1 king of each of two colors usually white (attack) and black (defense).

Reach-ability and Attack-ability

The reach-ability is the set of locations within the chessboard that a given piece at a specific location on the board can reach and the minimum number of steps it takes to get there (ignoring the positions of any other pieces on the board). The attack-ability is the locations within the chessboard that a given piece (at a specific location) can remove a piece of the opposing side (assuming that a piece is in place to be taken). Each category of pieces has a unique reach-ability and/or attack-ability within the chessboard determined by the rules of the game. Pieces could take longer paths to reach a given location. However, in this algorithm the focus will exclusively be on shortest paths.

The obtain-ability matrix is found using a similar process to the reach-ability for the starting piece, except it is centered on the ending piece. In other words, the calculation is similar to that of the reach-ability of the starting piece, but from the point of view of the ending location. Like reach-ability, obtain-ability is based on the rules or laws of the piece. Basically the obtain-ability is the reverse of the reach-ability starting at the end location.

Rooks, bishops, and the queen can move any number of spaces across the chessboard within a single turn. Rooks can only move north, south, east, or west. Therefore, a rook can reach any space within the chessboard in either 1 or 2 steps. Every location on the board that is not directly north, south, east, or west of the rook has a shortest path of 2 steps. All locations that are directly north, south, east, and west of the rook are 1 step. The reach-ability matrix of a rook (R) currently located at (4,3) is seen in figure 1.

2	2	2	2	1	2	2	2	*8*
2	2	2	2	1	2	2	2	*7*
2	2	2	2	1	2	2	2	*6*
2	2	2	2	1	2	2	2	*5*
2	2	2	2	1	2	2	2	*4*
1	1	1	1	R	1	1	1	*3*
2	2	2	2	1	2	2	2	*2*
2	2	2	2	1	2	2	2	*1*
8	*7*	*6*	*5*	*4*	*3*	*2*	*1*	

Figure 1: Rook

The number within each square represents the smallest number of steps it would take the rook to make it to the given location. For example, it takes the rook 2 steps to reach location (8,1) and only 1 step to reach location (4,7).

A similar reach-ability matrix can be developed for all of the other chess pieces. Pawns are a special case because pawns have different attack-ability and reach-ability. Pawns can only move in a "forward" direction (i.e., towards the opposing side). Pawns are also unique in their ability to move one or two spaces on their first move but only one space on subsequent moves. The pawns have the most complex reach-ability and attack-ability patterns. Pawns have the most restricted movement and cannot reach all locations within the chessboard.

Finding the Shortest Paths

A key element in the pattern generation algorithm is finding the shortest path between two pieces. The LG Search Strategy introduces an algorithm entitled "Grammar of Trajectory", which finds all the shortest paths between two elements within a defined environment (Stilman 2000). This recursive algorithm extensively uses the overlap of reach-ability matrices to identify the shortest path locations.

A modified non-recursive algorithm inspired by the trajectory approach to find the shortest paths is given as follows:

1. Determine the reach-ability matrix for the starting piece

2. Determine the obtain-ability matrix for the ending piece
3. Add the reach-ability from step 1 and the obtain-ability from step 2
4. Find the minimum values within the combined matrix and eliminate all non-minimum values
5. Generate the list of shortest paths using the locations identified in previous step

Definition of Fuzzy Pattern

A pattern is defined by a starting element of one side being able to reach an ending element of the opposing side, a single shortest path between these two elements and all elements from either side that could interact (positively by blocking or negatively by intersecting). Each pattern has a calculated rank, which combines several elements to determine goodness. The rank is based on the distances between the elements, the value of the elements involved, and the number of intersecting or blocking pieces and the values of these pieces.

A fuzzy pattern is a fuzzified version of the pattern where (1) distances are replaced with fuzzy values, (2) exact shortest paths are replaced with the environment locations that could be part of any shortest path between these two elements, and (3) the calculated rank is also replaced by a fuzzy value (Coffman-Wolph and Kountanis 2013b). The pattern is fully contained within the fuzzy pattern.

The Algorithm for Finding Patterns

The calculation of the shortest path begins by finding the reach-abilities and/or attack-abilities for each possible combination of starting and ending elements within the environment. The starting and the ending elements are always from different sides. In chess, there are two sides (generally white and black). Therefore, if the starting element were white, then the ending element would be the opposite - black.

Our patterns extraction algorithm generates all the patterns for all possible combinations of starting and ending pieces where the starting and ending pieces are from opposing sides. The pattern also finds any intersecting or blocking pieces. After identification of these pieces, the shortest paths from these pieces to the shortest main path are found. Any time multiple shortest paths are found, a new pattern is created for each of these shortest paths.

Pseudocode for Finding Patterns

```
1 Find the reach-ability/attack-ability for each piece
2 For each piece x in the environment {
3  For each piece y in the environment {
4    If(( x != y) && (x.side != y.side)) {
5     Find the shortest paths S between (x, y)
6       For each shortest path s in S  {
7         Create a pattern
8         Find all intersecting/blocking pieces (I)
9         For each i piece in I {
10          Find shortest paths B between (i, locations in s{
11            If B > 1, for each shortest path b in B {
12              Create new pattern w/same main path
13              Add shortest path b to pattern }
14            Else:  Add shortest path b to pattern }}}}}}
```

The Algorithm for Finding Fuzzy Patterns

Fuzzification of a concept provides for a more abstract view of information (Yen and Langari 1998). It can also be thought of as a tool that trades abstraction for a less precise representation of information (Zadeh 1965). A fuzzy pattern is simply an abstract view of the pattern. The introduction of abstraction to the pattern increases the overall speed of the algorithm by decreasing the algorithm's complexity and storage requirements. The less information there is to analyze, the faster the information can be interpreted and used.

The fuzzy pattern generation algorithm is similar to the pattern generation algorithm. There are two major differences between the algorithms – the removal of two loops. The first change is the removal of the for-each loop that walks through each of the shortest path choices generated for the main path and generates a new pattern (line 6). The second change is the removal of the for-each loop that walks through each of the shortest path choices generated from the blocking/attacking pieces (line 11).

The fuzzy pattern stores the shortest paths together as a group. In other words, the fuzzy pattern is concerned with the ability of a piece to get from the starting location to the end piece's location, but not with the exact path that is taken. The shortest paths within the fuzzy pattern will, therefore, be analyzed and evaluated as a group, enabling us to achieve a significant speedup factor.

The removal of these two loops decreases the complexity of the algorithm. Each fuzzy pattern may contain a larger amount of information than the equivalent non-fuzzy pattern because all the shortest paths are stored in a group. Further, since there does not need to be a pattern for each shortest path or intersecting/attacking shortest path the overall number of fuzzy patterns is smaller.

Pseudocode for Finding Fuzzy Patterns

```
1 Find the reach-ability/attack-ability for each piece
2 For each piece x in the environment {
3  For each piece y in the environment {
4    If(( x != y) && (x.side != y.side)) {
5     Find the shortest paths S between (x, y)
6     Create a fuzzy pattern f (store all shortest paths)
7     Find any intersecting/blocking pieces (I)
8     For each i piece in I {
9       Find the shortest paths B between (i, locations in s)
10      Add to fuzzy pattern f  }}}}
```

Finding Patterns and Fuzzy Patterns

Figure 2 shows an example scenario of a chessboard containing a total of 7 pieces: black king (Bkg), black queen (BQ), black knight (Bkn), white king (Wkg), white bishop (WB), white rook (WR), and a white pawn (WP). This scenario is used to demonstrate the generation of both the pattern and the fuzzy pattern. It is also used to illustrate the differences between the resulting patterns and the fuzzy patterns. For comparison, pattern and fuzzy pattern generation algorithms will be reviewed step-by-step. For simplicity, the pattern and fuzzy pattern generation will only be for the white pieces going to the location of the black pieces. In other words, this is from the white side's point of view.

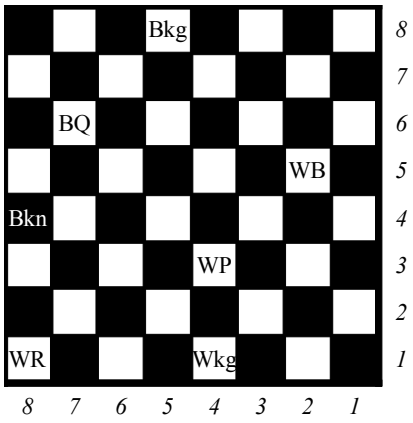

Figure 2: Example Scenario

The algorithms for both the pattern and fuzzy pattern begin with finding the reach-ability and attack-ability for all pieces. Figure 3 illustrates the reach-ability of the bishop.

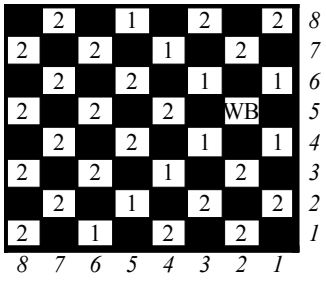

Figure 3: Bishop Reach-Ability

The next step is to find the obtain-abilities. Since these patterns and fuzzy patterns are from the perspective of the white side, we will only need to find the obtain-abilities for the black pieces. Figures 4 and 5 illustrate the obtain-abilities for the Black Queen using the King and Rook reach-abilities.

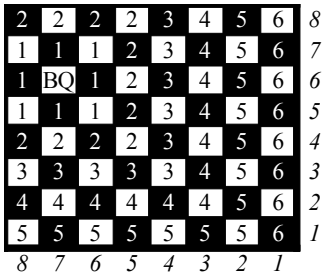

Figure 4: Black Queen using King Reach-ability

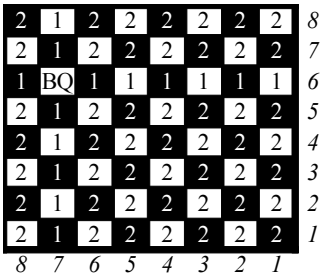

Figure 5: Black Queen using Rook Reach-ability

Finding the Shortest Paths

The current list of potential combinations to check after eliminating the unreachable pairs are as follows: (white rook, black knight), (white rook, black queen), (white rook, black king), (white pawn, black queen), (white pawn, black king), (white bishop, black queen), (white bishop, black king), (white king, black knight), (white king, black queen), and (white king, black king).

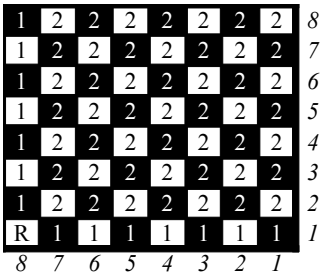

Figure 6: White Rook Reach-Ability

Shortest Path(s) for (white rook, black king)

Using the calculations in step 1 of the algorithm, combine (i.e., add the values at) each location and get the resulting combined reach-ability. Figure 6 illustrates the

reach-ability for the white rook and figure 7 illustrates the obtain-ability for the black king. Figure 8 illustrates the combined reach-ability.

1	1	1	K	1	1	1	1	8
2	2	2	1	2	2	2	2	7
2	2	2	1	2	2	2	2	6
2	2	2	1	2	2	2	2	5
2	2	2	1	2	2	2	2	4
2	2	2	1	2	2	2	2	3
2	2	2	1	2	2	2	2	2
2	2	2	1	2	2	2	2	1
8	7	6	5	4	3	2	1	

Figure 7: Black King using Rook Reach-ability

2	3	3	K,2	3	3	3	3	8
3	4	4	3	4	4	4	4	7
3	4	4	3	4	4	4	4	6
3	4	4	3	4	4	4	4	5
3	4	4	3	4	4	4	4	4
3	4	4	3	4	4	4	4	3
3	4	4	3	4	4	4	4	2
R,2	3	3	2	3	3	3	3	1
8	7	6	5	4	3	2	1	

Figure 8: Combined Reach-ability for White Rook and Black King

As seen in figure 8, the minimum cost path is 2. Looking at only the spaces with a 2 in them, it can be seen that there are two shortest paths:

white rook → (5,1) → black king

white rook → (8.8) → black king

Therefore, there will be at least two patterns, one for each of the shortest paths. The black queen could intersect the path in 1 step by means of multiple shortest paths. The white bishop could also become involved in 1 step, but only one shortest path. The black queen would take different shortest paths depending on the main path and there are multiple choices for each. Each of these choices generates a different path.

- White rook (8,1) → (5,1) → Black king (5,8), length = 2, attack: Black queen (7,6) → (5,4), block: White bishop (2,5) → (5,8)
- White rook (8,1) → (5,1) → Black king (5,8), length = 2, attack: Black queen (7,6) → (7,1), block: White bishop (2,5) → (5,8)
- White rook (8,1) → (5,1) → Black king (5,8), length = 2, attack: Black queen (7,6) → (5,6), block: White bishop (2,5) → (5,8)
- White rook (8,1) → (8,8) → Black king (5,8), length = 2, attack: Black queen (7,6) → (7,8), block: White bishop (2,5) → (5, 8)
- White rook (8,1) → (8,8) → Black king (5,8), length = 2, attack: Black queen (7,6) → (8,7), block: White bishop (2,5) → (5,8)
- White rook (8,1) → (8,8) → Black king (5,8), length = 2, attack: Black queen (7,6) → (8,6), block: White bishop (2,5) → (5,8)
- White rook (8,1) → (8,8) → Black king (5,8), length = 2, attack: Black queen (7,6) → (8,5), block: White bishop (2,5) → (5,8)

The fuzzy pattern, as stated before, combines the multiple paths of information. Also, the fuzzy pattern ignores the specifics of the attack/block locations to decrease computational complexity. Additionally, the lengths have been fuzzified (see figure 9 for the definitions of the fuzzy lengths used for this example). The above seven (non-fuzzy) patterns are reduced to the following single fuzzy pattern:

White rook (8,1) → (5,1) or (8,8) → Black king (5,8), length = very short, attack: Black queen, block: White bishop

Values	Fuzzy Variable
0-2	Very Short
2-4	Short
4-6	Long
6-8	Very Long

Figure 9: Definition of Fuzzy Length Variables

Analysis of Patterns vs. Fuzzy Patterns

The following is an analysis of the computational advantages obtained by using the fuzzy pattern representation over the pattern representation. Informally from the example scenario in the last section, it can be observed that the number of patterns is often greater than the number of fuzzy patterns. This section quantifies this observation. An approximation of the computational savings for each piece can be calculated by looking at the average number of patterns required. There is always only one fuzzy pattern for each starting and ending pair of pieces. In the absolute worst case (i.e., only one shortest path between a pair of pieces), the number of fuzzy patterns would be equal to the number of patterns. As mentioned before, the patterns and the fuzzy patterns are similar in structure. An individual fuzzy pattern may require a slightly larger amount of storage space, as noted earlier, but there are fewer fuzzy patterns overall.

Continuing using the standard chess game as an example, the following paragraphs demonstrate that statistically there are significantly fewer fuzzy patterns needed than patterns. Each side in a chess game has 8 pawns, 2 rooks, 2 bishops, 2 knights, 1 queen, and 1 king. Pawns are the most restricted in movement and the lowest value pieces within the game. Pawns are extremely restrictive in movement

and, therefore, usually have only one shortest path to a destination. The bishops are the next most restricted as they can only move on the diagonal and each can only reach 50 percent of the board. Knights and rooks, unlike bishops, can travel to any location on the board. Knights are restricted to moving in an "L" pattern while rooks are restricted to moving north, south, east, and west. Kings can move in any direction, but can only move one space at a time. The queens are the least restricted pieces as they can move in any direction and for any number of spaces.

Special Case: Pawns

As stated earlier, the pawns are greatly restricted in movement and rarely have multiple shortest paths. Therefore, when calculating the approximation, all patterns and fuzzy patterns with a pawn starting piece are assumed to be equivalent. For analysis purposes it is assumed that the pawns will not increase (or decrease) the total number of patterns or fuzzy patterns.

Rooks, Bishops, Knights, Queens, and Kings

Calculating an approximate number of patterns for a given chess piece is not an easy task. The task begins by examining the number of spaces that are reachable from a given starting location by a specific piece, how many moves are required to reach each of these locations, and most importantly the number of shortest paths. To do these complex calculations, each piece (rook, bishop, knight, queen, and king) must be examined individually with respect to its reach-ability. To further complicate the calculations for some pieces, the destination location can play a significant role in the total number of shortest paths. (The greater the number of moves required to reach the ending location, the greater the number of shortest paths).

General Formula

The general formula for calculating the number of patterns needed for a single starting point is as follows:

$[(L_1/r * 1) + (L_2/r * 2) + (L_3/r * 3) + \ldots (L_n/r * n)]$

L_j = number of locations reach-able in j steps
r = number of locations reach-able for a give piece
n = maximum number of steps a piece could take

Several starting locations are equivalent to each other (in terms of the reach-ability). So the general formula for calculating the number of patterns needed for a given chess piece considering all 64 spaces as a starting point is as follows:

$\{[(L_1/r * 1) + (L_2/r * 2) + (L_3/r * 3) + \ldots (L_n/r * n)] * z_1/t\} + \ldots + \{[(L_1/r * 1) + (L_2/r * 2) + (L_3/r * 3) + \ldots (L_n/r * n)] * z_k/t\}$

L_j and r are the same as above
k = number of different situations for a given piece
z_k = number of equivalent locations for the kth situation
t = total number of locations a piece can be placed
n = maximum number of steps a piece could take

Most pieces will have an r-value of 63. (There are 64 locations on the chessboard and the piece is sitting on 1 location so there are 63 locations the piece could visit). The bishop is the only exception with r being equal to 31 since it can only visit half of the locations. The value of k is specific to each piece and is partly dependent on the movement of the piece. The formulas given above are best explained by use of examples.

Example: Rook

Due to the unique properties of the rook chess piece, it is the easiest to analyze. (For the purposes of this analysis we ignore the unique move involving the rook and king known as castling). The calculation for the rook begins with the examination of locations on the chessboard reach-able with only one move. Because rooks can move north, south, east, or west as many spaces as desired, anything along the north, south, east, or west of the rook is reachable in one move. In figure 10, a rook (R) is located at (6,5) and the chessboard locations marked with an "X" are the locations reachable in only one move.

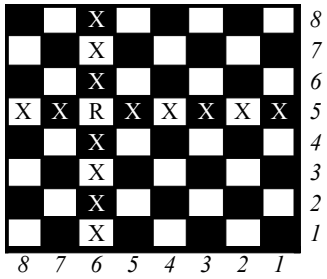

Figure 10: Reachable Locations for Rook at (6,5) in 1 Move

A chessboard is 8 by 8 and, therefore, has 64 locations. The rook is sitting on one location leaving 63 locations that the rook could move. There are 14 total locations that a rook can reach in only one move from location (6,5). Therefore, for these 14 locations there is only one shortest path. The other 49 locations within the chessboard are reach-able in two moves. There are always only two shortest paths that can be used to reach any of these two-move locations. Regardless of the starting position of the rook, there are always 14 locations reach-able in 1 step and 49 locations reach-able in 2 steps.

From this information and using the general formula (provided earlier), the value of r equals 63 (total number of reachable locations), the value of n equals 2, the value of k equals 1, and the value of t equals 64. The average number of patterns for a given rook piece to a location on the chessboard is calculated using the following general formula:

$[(L_1/63 * 1) + (L_2/63 * 2)] * z_1/64$

Plugging in the numbers for the rook (the value of L_1 equals 14, the value of L_2 equals 49, and the value of z_1 equals 64):

$[(14/63 * 1) + (49/63 * 2)] * (64/64) = (14/63) + (98/63) = 112/63 = 1.778$

Therefore, there are on average 1.778 patterns for each rook piece using the non-fuzzy pattern generation algorithm instead of only one fuzzy pattern. Using a fuzzy pattern provides computational savings of over 40%.

Further Analysis of the Other Pieces:

A similar analysis can be done for all other chess pieces. However, the logic is more complex. For starters, most pieces (except rook and bishop) have the property that the number of shortest paths is not constant and depends greatly on location within the chessboard environment. Calculations for both the king and the knight are significantly more complex because reaching any location does not happen in one or two moves (a property of the rook, bishop, and queen). Consequently, the king introduces a new level of complexity to the analysis: the number of shortest paths for a king depends on both the starting location and the distance to the end location. When the king is farther away from the end location, the number of shortest paths increases dramatically.

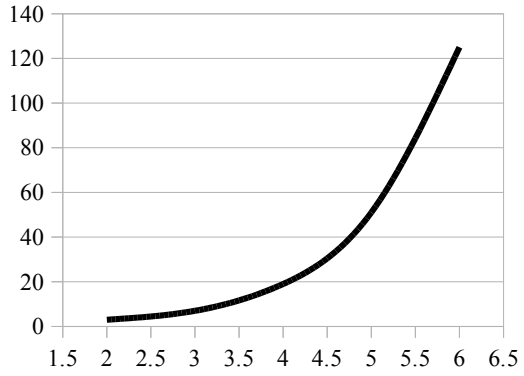

Figure 11: Number of Shortest Paths for King as Distance From End Piece Increases

From figure 11 it can be seen that, as the king and the end piece get farther apart, the number of possible shortest paths increases dramatically (nearly exponentially). To calculate the approximate number of patterns, it was decided to use the extremely conservative number of 5. This number is the average number of shortest paths for the king piece and an end piece at 2 or 3 steps away. It was selected to reflect the usage of the king in more tight-cornered situations, because the king pieces are rarely used strategically in long-term attack strategies.

Figure 12 contains a table that displays the average number of patterns for each chess piece. The value of 5 is used in the estimate for both the queen and the king, while the value of 6 is used as the estimate for the knight. In both cases 5 and 6 are an underestimate given the ranges for each piece and selected for consistency purposes (i.e., always selecting the lower bound). Unlike the king, both the queen and knight are used in long range planning and are actively used throughout a game of chess.

Piece	Number on Board	Range of Upper Bound of Shortest Paths	Number used for Estimate	Average Number of Patterns
Pawn	8	1-2	1	1
Rook	2	2	2	1.778
Bishop	2	2	2	1.718
Knight	2	6-12	6	5.167
Queen	1	5-12	5	3.681
King	1	3-125	5	4.475

Figure 12: Average Number of Patterns for Each Chess Piece

As can be seen from the table in figure 12, the greatest amount of computational savings involves the knight. There are on average 5.167 patterns in the non-fuzzy algorithm and only one fuzzy pattern. The knights are active chess pieces and two per side. By using a fuzzy pattern there is a computational savings of slightly over 80%. Figure 13 and previous analysis clearly demonstrates the savings of using fuzzy patterns over patterns.

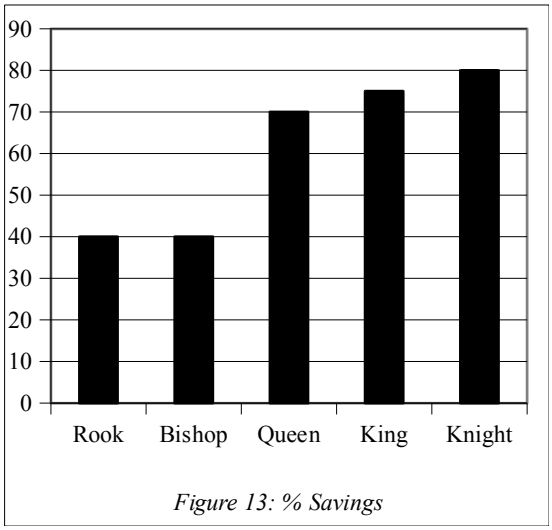

Figure 13: % Savings

Conclusions

Our proposed fuzzy patterns approach for adversarial game playing provides a far superior time advantage over the pattern or zones (from the LG algorithm) approaches. As has been shown, the fuzzy patterns eliminate two loops in the algorithm. Thus, reducing the complexity of the algorithm by $O(n^2)$. Each fuzzy pattern is slightly larger in storage size, but there are significantly fewer for any given scenario. The computational savings are greatest any time a knight or king is involved (average of 5 patterns to every

1 fuzzy pattern for knight and 4.5 patterns to every 1 fuzzy pattern for king). The rooks and bishops save close to half with the usage of fuzzy patterns. The queen pieces have around a 3.5 to 1 savings.

Furthermore, the computational saving discussed has been on the main path (i.e., the path between the starting and ending pieces). There is an additional multiplicative computational saving for all attacking or blocking pieces. In the pattern generation algorithm, whenever there is an attacking or blocking piece discovered that had multiple shortest paths, separate patterns are generated for each of these paths. The fuzzy pattern generation algorithm considers the multiple shortest paths as a set (same as the main path) providing even further savings than shown in the above analysis.

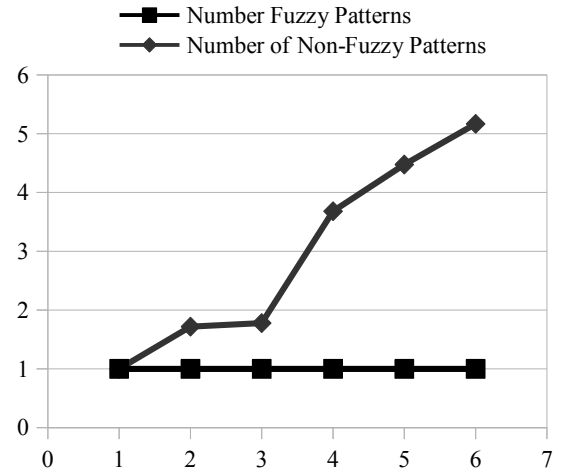

Figure 14: Number of Fuzzy vs Non-Fuzzy Patterns

The fuzzy patterns are just one part of a larger search algorithm (Coffman-Wolph and Kountanis 2013a). The patterns and their fuzzy counterparts are used to determine the possible moves for each chess piece. By representing several related moves as a single entity, the fuzzy patterns have a much smaller branching factor and, thus, a significantly smaller search tree. Specifically using the values calculated in the results section and weighting for the number of each chess piece, the search tree for a chess game would be cut at least in half.

Acknowledgments

The authors would like to thank Dr. Ala Al-Fuqaha for his careful review and helpful suggestions that were incorporated into this paper. The authors would like to thank the reviewers for their suggestions and comments.

References

1. Coffman-Wolph, S. and Kountanis, D. 2013. Fuzzy Process Particle Swarm Optimization. In the Proceedings of the 43rd Southeastern Conference on Combinatorics, Graph Theory, & Computing. Winnipeg: Utilitas Mathematica Pub., Inc. Forthcoming.
2. Coffman-Wolph, S. and Kountanis, D. 2013. A General Framework for the Fuzzification of Algorithms. In the Proceedings of MICWIC 2013. Forthcoming.
3. Cox, E. 2005. Fuzzy Modeling and Genetic Algorithms for Data Mining and Exploration. San Francisco, California: Morgan Kauffman Publishing.
4. David-Tabibi, O., Koppel, M., and Netanyahi, N. S. 2010. Optimizing Selective Search in Chess. In the Proceedings of the ICML Workshop on Machine Learning and Games. Madison, Wisconsin: ICML.
5. Groen, F. C. A., den Boer, G. A., van Inge, A., and Stam, R. 1992. A Chess Playing Robot: Lab Course in Robot Sensor Integration, IEEE Transactions on Instrumentation and Measurement 41: 911-914.
6. Ross, T. J. 2004. Fuzzy Logic with Engineering Applications. West Sussex, England: John Wiley & Sons Ltd.
7. Russell, S., and Norvig, P. 1995. Artificial Intelligence: A Modern Approach. Saddle River, New Jersey: Prentice Hall.
8. Si, J., and Tang, R. 1999. Trained Neural Network Play Chess Endgames. In the Proceedings of the International Joint Conference on Neural Networks (IJCNN '99), 3730-3733. Institute of Electrical and Electronics Engineers, Inc.
9. Stilman, B. 1994. A Formal Model for Heuristic Search. In the Proceedings of the 22nd Annual ACM Computer Science Conference, 380-389. New York, New York: Association for Computing Machinery.
10. Stilman, B. 2000. Lingustic Geometry: From Search to Construction, Norwell, Massachusetts: Kluwer Academic Publishers.
11. Wang, J., Dong, L., Gao, X., Xu, C., Wang, F., and Zhang, C. 2009. The Research and Development of Super-Master-Like Chess Playing Robot. In the Proceedings of the Chinese Control and Decision Conference (CCDC '09), 1332-1335. Institute of Electrical and Electronics Engineers, Inc.
12. Yen, J., and Langari, R. 1998. Fuzzy Logic Intelligence, Control, and Information. Upper Saddle River, New Jersey: Pearson Education.
13. Zadeh, L.A. 1965. Fuzzy Sets. Information and Control 8: 338-353.