AN EFFICIENT ALGORITHM FOR EYESPACE CLASSIFICATION IN GO

Peter Drake drake@lclark.edu
Niku Schreiner nikus@lclark.edu
Brett Tomlin btomlin@lclark.edu
Loring Veenstra loring@lclark.edu

Lewis & Clark College
0615 SW Palatine Hill Road
Portland, OR 97219-7899
Phone (503) 768-7539
Fax (503) 768-7668

ABSTRACT

Writing programs to play the classical Asian game of Go is considered one of the grand challenges of artificial intelligence. One of the key tactical issues in Go is whether a group of pieces can divide the area it surrounds ("eyespace") into two separate regions, thus rendering the group immune to capture. Human players quickly learn to recognize various eyespace patterns, invariant under translation, rotation, reflection, and distortion. In this paper, we present an efficient canonical form for classifying such patterns. This representation, along with an algorithm for finding the inside of a group, is used to quickly analyze the life and death (capturability) status of all groups on the board.

KEYWORDS

artificial intelligence, algorithms, Go game, graph theory, eyes

1. Introduction

Writing programs to play the classical Asian game of Go is considered one of the grand challenges of artificial intelligence. While the best Chess programs are on a par with the strongest human players, the best Go programs are easily defeated by moderately strong amateurs. Indeed, strong human players can defeat these programs even in the face of an enormous handicap, e.g., allowing the program to make dozens of free moves at the beginning of the game [3].

We present the rules of Go very briefly here. For further information, readers are referred to any of various tutorials available in print or on line [1].

The game is played on a square grid of intersecting lines. In the full game there are 19 horizontal and 19 vertical lines, but this can be changed without altering the other rules. 5x5, 9x9, and 13x13 are common sizes.

The board is initially empty. There are two players, black and white. Each player has a collection of pieces (stones) in his/her color. Black plays first.

On a turn, a player may either place one of his/her stones on an empty intersection or pass. Once a stone is placed, it never moves (unless it is captured—see below).

The game ends when both players pass consecutively. Each player scores one point for each empty intersection surrounded by his/her stones and loses one point for each of his/her stones that was captured. The player with the higher score wins.

The vacant points adjacent to a group of contiguous stones are called *liberties* of that group. If the opponent plays on a group's last liberty, all of the stones in that group are captured and removed from the board. For example, in Figure 1, a white stone played at A would capture the two black stones marked B.

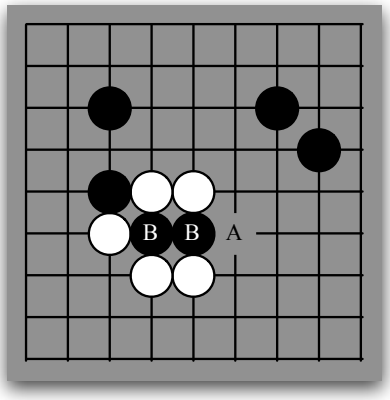


Figure 1: A white stone played at A would capture the two black stones marked B.

It is illegal to play a suicidal move that would cause the played stone to be captured. A move that captures one or more enemy stones is never suicidal, because the played stone gains a liberty where the captured stones used to be.

Finally, to prevent endless games, it is illegal to make a move that results in a repetition of a previous board configuration.

While the rules to Go are very simple, it is a deep and complex game. The search tree is astronomically deep and wide [3], rendering any sort of brute force search intractable. Furthermore, heuristic estimation of the value of a position is extremely difficult. Determining whether a large group of stones can be captured can require very subtle analysis, but any estimate of the score that does not take this into account is nearly useless. This paper addresses this issue of whether a group can be captured.

Section 2 reviews the life and death problem, defining the terms *alive*, *dead*, *unsettled*, and *eyespace*. Section 3 opens the discussion of how eyespace shapes can be classified. Section 4 provides a simple canonical form for eyespace shapes, but explains how this representation is both inefficient and insufficiently detailed. Section 5 improves on the representation, solving both of the problems raised in Section 4. Section 6 explains how to use the improved representation to determine the life status of every group on the board. Sections 7 and 8 provide discussion and conclusions.

2. Background: Life and Death

One of the key tactical concepts in Go is the question of *life and death* [4]. This is the question of whether a given group of stones can be captured and removed from the board. A group is immune to capture if it has two or more internal liberties or *eyes*, as in group A in Figure 2. White would have to play on both of these points to capture the black stones, but either move would be suicide and therefore illegal.

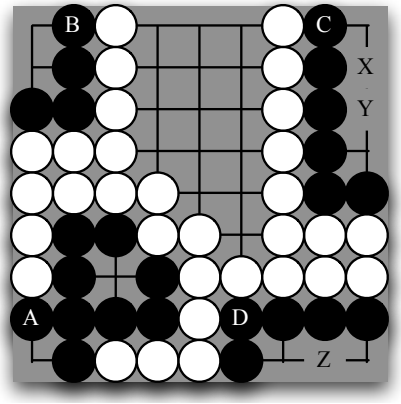


Figure 2: Life depends on the ability to produce two separate internal liberties (eyes).

If a group has a single large, internal space, the group's life hinges on whether this *eyespace* can be divided into two separate eyes. Group B in Figure 2 cannot make two eyes, because the two internal liberties are adjacent. Group C can make two eyes by playing at either X or Y. The fate of group D depends on whose turn it is. Black can make two eyes by playing at Z, but if it's white's turn, white can doom the group by playing at Z. Even though black could eventually capture the invading stone, this would leave black with a single liberty at Z; white could play on this point again and capture the black stones.

We divide groups into three categories:

- A group is *alive* if, even if the attacker moves first, the defender can ensure that the group is never captured.
- A group is *dead* if, even if the defender moves first, the defender cannot ensure that the group is never captured.
- A group which is neither alive nor dead is *unsettled*.

In Figure 2, group A is alive, B is dead, C is alive, and D is unsettled. For readers familiar with Go, we note that in this paper a group that can reach seki is considered alive, while a group that must fight ko to survive is considered dead.

The purpose of the algorithm presented in this paper is to place a group into one of these three categories based on its eyespace. This is enormously important for playing the full game because a Go player is constantly trying to accomplish goals in several places on the board. Since a player only places one stone per turn, one cannot afford to waste a move trying to

change the fate of a group that is already either alive or dead. An unsettled group, on the other hand, calls for immediate action.

A number of features are relevant to the classification of an eyespace:

- The size of the eyespace.
- The shape of the eyespace.
- Which points within the eyespace, if any, are occupied by friendly or enemy stones.
- Which points within the eyespace, if any, are on edge or corner points.

The next section begins by addressing the size and shape of an eyespace.

3. Eyespace Size and Shape

A group with two separate eyespaces is, of course, alive. We could determine the status of a group with a single eyespace by minimax search, but it would be much more efficient to look up the eyespace in a database of stored patterns. This means we must have some way to determine whether a given eyespace is equivalent to one found in the database.

We first might begin classifying eyespaces based on their size. Ignoring internal stones and edge-of-board effects, eyespaces of size 1 or 2 are dead and an eyespace of size 3 is unsettled. Any eyespace of size 7 or more is alive. Unfortunately, the status of a group with an eyespace containing between 4 and 6 points depends on the shapes of the eyespace [4].

We need to store the shape of the eyespace. Representing an eyespace as a set of locations on the board would be vastly inefficient, as we would need to store an entry for each shape at each position where it might appear on the board. In other words, this representation is not invariant under translation.

A slightly more refined approach is to store a local image of the shape translated to the lower left corner of the board. This is invariant under translation, but not under rotation and reflection. Since an eyespace shape results in the same life status reflected or unreflected in each of four rotations, this representation would have to maintain eight copies of each shape.

An even better representation is to treat the shape as a graph [9]. This has the advantage that it is invariant to distortion of the shape. For example, three points in a line are isomorphic to three points in an L shape, so there is no need to store both shapes. Figure 3 shows the twenty different shapes of sizes 1 through 6.

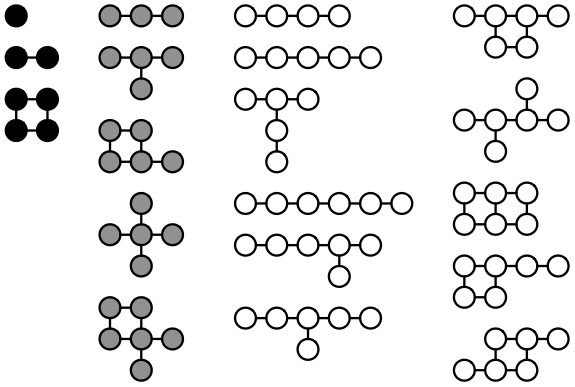


Figure 3: The twenty different eyespace shapes of sizes 1 through 6. Ignoring internal enemy stones and edge-of-board effects, the black shapes are dead, the grey shapes are unsettled, and the white shapes are alive.

A graph is typically represented by an adjacency matrix, as in Figure 4. The entry at position *i*, *j* indicates whether there is an edge connecting vertices *i* and *j*.

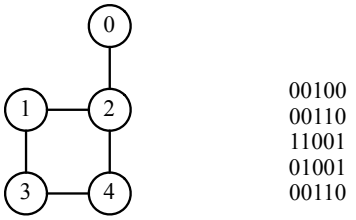


Figure 4: A graph and its adjacency matrix representation. Vertex 1 is adjacent to vertices 2 and 3, so in row 1 there are ones in columns 2 and 3.

This representation is invariant under translation, rotation, reflection, and distortion, but unfortunately there can be more than one adjacency matrix representation for the same graph. If the vertices are numbered differently, a different matrix can result. In order to quickly determine if two graphs (e.g., eyespaces) are the same, we need a canonical form, so that the same graph (or two isomorphic graphs) always has the same representation.

4. Simple Canonical Form

A conceptually simple way to define a canonical form is to consider all possible numberings of the vertices and choose the one that produces the lexicographically largest adjacency matrix (Figure 5). In other words, we choose the matrix whose bits corresponds to the largest binary number.

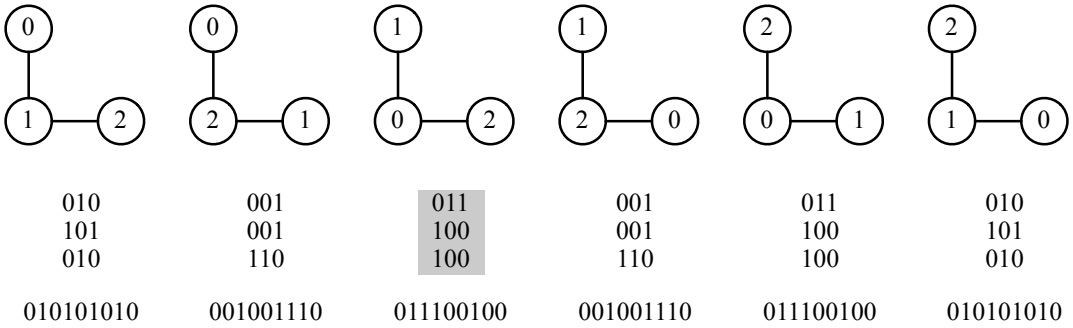


Figure 5: Finding the canonical array matrix representation of a graph. These graphs are all isomorphic, but the vertices are numbered differently. Below each graph is its adjacency matrix representation and the bits of that matrix read as a single binary number. The shaded matrix, being lexicographically largest, is the canonical form. (Because this particular graph is symmetric, there are two different numberings that produce this matrix.)

This is a solid representation, but it presents two major problems for representing eyespaces. First, since there are $n!$ ways of numbering a graph with n vertices, finding the canonical form for a graph with more than a handful of vertices is intractable. Second, there is not enough information here to determine life status in situations involving internal stones and the edge of the board. Since we are only representing the *shape* of the eyespace, we cannot deal with the common situation where there are friendly or enemy stones present within the eyespace. It also sometimes matters whether a particular point within the eyespace is at the edge or corner of the board. The next section addresses these problems.

5. Improved Canonical Form: Labeling the Rows

In Figure 6, the eyespaces of groups A, B, and C are superficially different, but they are all isomorphic to the same linear, four-vertex graph. Furthermore, each has an edge-of-board point at one end of the "line" and the farther of the two points in the middle of the line is occupied by a white stone. In life and death terms, all three black groups have the same status (unsettled) for precisely the same reason: the life of the group hinges on which player first occupies the other middle point.

Handily, the two problems mentioned at the end of the previous section (inefficiency and lack of information about the points) can be used to solve each other. We do this by appending some additional bits to the beginning of each row in the adjacency matrix. These bits provide information about the corresponding point on the board.

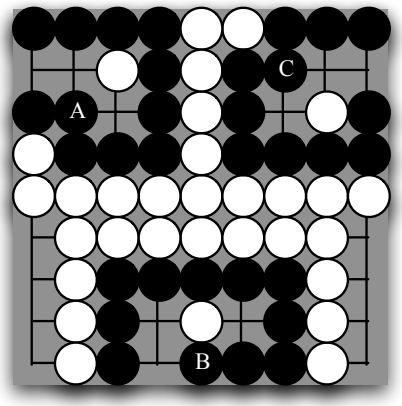
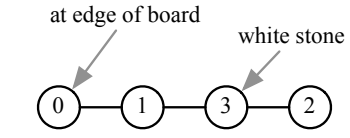


Figure 6: The eyespaces of the black groups A, B, and C are equivalent and should have the same representation.

For example, Figure 7 shows the representation for any of the eyespaces in Figure 6. There are seven additional *label bits* at the left of each row. The first two bits represent the color of the point: 00 for black, 01 for white, and 10 for vacant. The next pair of bits represents the "edginess" of the point: 00 for a point away from the edge of the board, 01 for a point on the edge, and 10 for a point in the corner. The next three bits are a binary number specifying the degree of the vertex, that is, how many neighbors it has within the eyespace.

This clearly solves the lack of information problem. It also solves the intractability problem because we only have to consider the permutations of each set of points *with the same label bits*, rather than the entire eyespace. This drastically reduces the number of permutations to be considered. In this particular case, no two points have the same label bits, so only one permutation has to be considered.



color	edginess	degree	neighbors
10	01	001	0100
10	00	010	1001
10	00	001	0001
01	00	010	0110

Figure 7: Labeled adjacency matrix for any of the eyespaces in Figure 56. The rows are ordered (and the nodes numbered) to give the lexicographically largest array.

We mention two technical details at this point. First, the degree information is technically redundant, as it could be deduced from the adjacency bits. It is worth including nonetheless, because it breaks the graph into smaller identically-labeled sets. Second, when lexicographically sorting the permutations, it is important to read the bits down the columns, rather than across the rows. This insures that, for example, all of the vacant points are listed before the white points.

Using this canonical form in a transposition table, the status of a group can be determined in a reasonable amount of time using minimax search [7]. Once any eyespace configuration has been encountered in a game, the status of any group with the same eyespace configuration can be determined almost instantly by looking up the configuration in the table. Note that pure eyespace analysis is sufficient only when the group consists of a single contiguous group that cannot be cut apart and has no external liberties.

6. Life and Death On the Whole Board

The previous discussion dealt with what happens inside a group's eyespace. This begs the question of where the eyespace is. As seen in Figure 8, this is not a trivial question.

The problem is to find the space that is "inside" the group. We apply a simple heuristic to distinguish inside from outside: the inside is smaller than the outside. A contiguous group partitions the board into one or more disjoint regions. All of them except for the largest one are eyespaces. If a group has two or more eyespaces, it is alive. If it has none, it is dead. (We currently ignore the possibility of the group connecting with an ally or escaping into open space where it can make eyes.) If it has exactly one, we must do search to see what happens inside that eyespace.

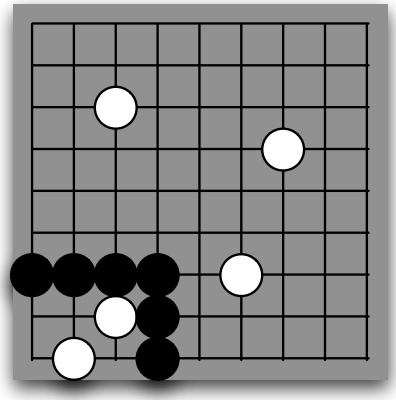


Figure 8: Where is the eyespace of the black group? To a human player, it is immediately obvious that the eyespace is in the lower left. How is a computer to know that the large area on the other side of the group is not eyespace?

There is a further complication. If there is a viable (unsettled or alive) enemy group *inside* one of the partition regions, that region should not be considered eyespace (Figure 9).

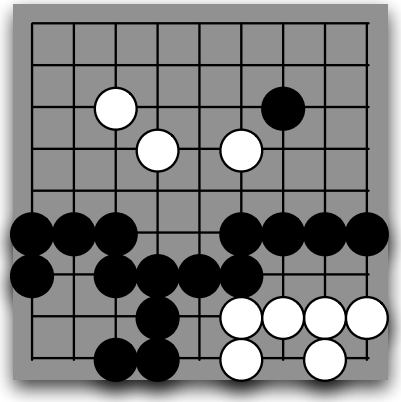


Figure 9: The large black group appears to have two eyespaces, but the one at lower right contains a live white group.

When analyzing the life and death of all groups on the board, we therefore start with the groups with the smallest total eyespace size. If one of these is alive or unsettled, we don't count it as eyespace for an enclosing group of the opposite color.

An example is shown in Figure 10. In analyzing this position, our program first looks at group A, which has only one point of eyespace. It is dead. Group B is unsettled, C is alive, and D is unsettled. Since group A is dead, group E has two separate eyespaces and is alive. This analysis runs in under half a second in our Java implementation on a dual 1 GHz Macintosh PowerPC G4.

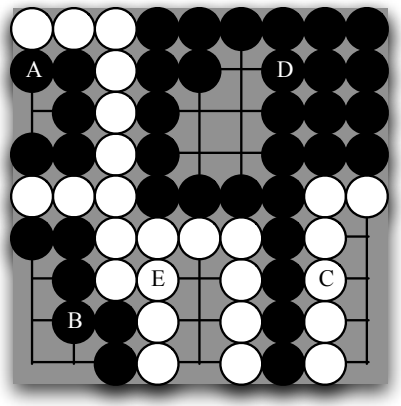


Figure 10: Which groups are alive, unsettled, and dead?

7. Discussion and Related Work

Much previous work on static analysis of life and death in Go has focused on logical inference and heuristic search [2, 5, 6, 10]. These techniques are certainly necessary in the most difficult life-and-death struggles, where the fate of several groups hangs in the balance. The current pattern-recognition approach enhances this work by very quickly classifying simpler situations. These situations sometimes appear directly in games; they very often appear in searches, even if the stones are never actually played because the inevitable outcome becomes clear to the players. Our method for very quickly analyzing relatively simple (but nontrivial) situations can significantly increase the feasible search depth. Where a blind searcher might proceed until the group either is captured or forms two separate eyespaces, a searcher with a table of patterns stored using our representation could stop several moves earlier, thus searching hundreds of times faster. Using the "smallest total eyespace first" algorithm described in the previous section, the table tends to be filled automatically by the searcher, avoiding the need to hand-code domain-specific knowledge.

Vilà and Cazenave [8] take a similar approach to ours, classifying eye shapes into categories of isomorphic graphs. Their paper only examines the shape of the eyespace. While their algorithm is somewhat faster, ours is more powerful, offering more precise life-and-death information in the face of internal stones, edges, and corners.

Our algorithm for analyzing the entire board by starting with the smallest groups is consistent with our introspections on how we, as humans, play the game. When analyzing the life status of a group, we take into account the status of other groups that are in some sense simpler to analyze. This can occasionally lead players astray when these analyses are based on the heuristic that "big groups don't die," which is not always true.

Our technique only applies when the eyespace is strictly enclosed by a single contiguous group of stones that has no external liberties. Further work is needed to overcome these limitations.

Our canonical form for local patterns may also be useful for databases of locally good moves (*tesuji* in Japanese), standard opening sequences in corners (*joseki*), and for other search-intensive parts of the game such as determining whether a group can be cut apart.

Code for the algorithms described in this paper is available on the web page for the Orego project, http://www.lclark.edu/~drake/go/.

8. Conclusions

We have presented a canonical form for classifying eyespace shapes in Go. This representation can be computed quickly and uniquely classifies the surrounding group as alive, unsettled, or dead. It takes into account stones within the eyespace and edge-of-board effects.

We have also presented an algorithm for determining the life status of every group on the board under certain circumstances. This algorithm is capable of finding the inside of a group, so it does not count as "eyespace" a region that contains a live or unsettled enemy group.

Acknowledgments

We thank the WM Keck Foundation for funding this work, and Jens Mache and all the members of the Computer Go Mailing List for their helpful comments.

References:

[1] K. Baker, *The Way to Go* (New York, NY: American Go Association, 1986). Available on line at http://www.usgo.org/usa/waytogo/index.asp.

[2] D. Benson, Life in the game of Go. *Information Sciences*, 10, 1976, 17-29.

[3] B. Bouzy & T. Cazenave, Computer Go: an AI Oriented Survey, *Artificial Intelligence, 132*(1), 2001, 39-103.

[4] J. Davies, *Life and Death* (Tokyo: Kiseido Publishing Company, 1996).

[5] A. Kishimoto & M. Müller, Search versus Knowledge for Solving Life and Death Problems in Go. In *Proceedings of the Twentieth National Conference on Artificial Intelligence* (Pittsburgh, PA: AAAI Press, 2005), 1374-1379.

[6] M. Müller, Playing it Safe: Recognizing Secure Territories in Computer Go by Using Static Rules and Search. In H. Matsubara (ed.), *Game Programming Workshop in Japan '97* (Tokyo: Computer Shogi Association, 1997).

[7] S. Russell & P. Norvig, *Artificial Intelligence: A Modern Approach, 2nd Edition* (Upper Saddle River, NJ: Prentice Hall, 2003).

[8] R. Vilà & T. Cazenave, When one eye is sufficient: a static classification. In *Advances in Computer Games. Many Games, Many Challenges* (Kluwer Academic Publishers, 2003), 109-124.

[9] R.J. Wilson, *Introduction to Graph Theory, 4th Edition* (Boston: Addison Wesley, 1996).

[10] T. Wolf, The program GoTools and its computer-generated tsume go database. In H. Matsubara (ed.), *Proceedings of the First Game Programming Workshop in Japan* (Hakone, Japan, 1994).