A Novel Approach to Solving Goal-Achieving Problems for Board Games

Chung-Chin Shih\textsuperscript{1,2*}, Ti-Rong Wu\textsuperscript{1*}, Ting Han Wei\textsuperscript{3}, and I-Chen Wu\textsuperscript{1,2†}

\textsuperscript{1}Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan
\textsuperscript{2}Research Center for Information Technology Innovation, Academia Sinica, Taipei, Taiwan
\textsuperscript{3}Department of Computing Science, University of Alberta, Edmonton, Canada

\{rockmanray,kds285\}@aigames.nctu.edu.tw, tinghan@ualberta.ca, icwu@cs.nctu.edu.tw

Abstract

Goal-achieving problems are puzzles that set up a specific situation with a clear objective. An example that is well-studied is the category of life-and-death (L&D) problems for Go, which helps players hone their skill of identifying region safety. Many previous methods like lambda search try null moves first, then derive so-called relevance zones (RZs), outside of which the opponent does not need to search. This paper first proposes a novel RZ-based approach, called the RZ-Based Search (RZS), to solving L&D problems for Go. RZS tries moves before determining whether they are null moves post-hoc. This means we do not need to rely on null move heuristics, resulting in a more elegant algorithm, so that it can also be seamlessly incorporated into AlphaZero’s superhuman level play in our solver. To repurpose AlphaZero for solving, we also propose a new training method called Faster to Life (FTL), which modifies AlphaZero to entice it to win more quickly. We use RZS and FTL to solve L&D problems on Go, namely solving 68 among 106 problems from a professional L&D book while a previous state-of-the-art program TSUMEGO-EXPLORER solves 11 only. Finally, we discuss that the approach is generic in the sense that RZS is applicable to solving many other goal-achieving problems for board games.

Introduction

Traditional board games such as Go and Hex have played an important role in the development of artificial intelligence. Goal-achieving problems are puzzles that challenge players to achieve specific goals under given board configurations.

Life-and-death (L&D) problems in Go is a typical goal-achieving problem. In Go, where the goal for both players is to hold more territory on the game board than their opponent, a critical skill is to identify safety for stone groups. A safe (live) stone group holds territory on the board indefinitely, thereby giving an advantage to the player that owns it. To build this skill, Go players have been creating and solving L&D problems, also called tsuonego, for centuries. Many of these problems are challenging even for professional players. A straightforward method of solving L&D problems using computer programs involves finding a solution tree (Stockman 1979; Pijls and de Bruin 2001) that represents a full strategy of answering moves. As a computational problem, one of the main challenges is that the large branching factor in Go tends to lead to a prohibitively large solution tree. To cut down on the search size, Kishimoto and Müller (2003; 2005a; 2005b) manually designated a specific search space to prevent searching irrelevant spaces.

An elegant way to reduce the branching factor is to take advantage of threats (Allis 1994), which are moves that force the opponent to respond in a specific way. When threats are involved, not all opponent moves need to be explored to ensure correctness when solving L&D problems, especially moves that do not answer the threat. In this vein, Thomsen (2000) proposed a threat-based search algorithm named lambda search, by trying null moves first (usually by passing), then deriving a so-called relevance zone (RZ), outside of which the opponent does not need to search.

This paper proposes a novel approach called RZ-Based Search (RZS) to solving L&D problems. We illustrate the basic concepts of RZS with examples in Go and Hex. RZS itself is then described in detail with Go examples. With RZS, moves are searched first, before they are determined to be null moves post-hoc. This allows our approach to be seamlessly incorporated into most search algorithms. Specifically for this paper, we applied RZS into an AlphaZero-like program (Silver et al. 2018), well-known for playing at superhuman levels.

A straightforward way of leveraging AlphaZero methods for goal-achieving problems is to use it in the construction of solution trees, by adding Boolean AND-OR tree logic for wins and losses into the AlphaZero MCTS procedure. However, AlphaZero is fundamentally trained for the task of winning the game, rather than proving it exhaustively. This leads to a discrepancy in behaviour between a strong player and a solver. Namely, AlphaZero picks any winning move, rather than the move that leads to the shortest distance to win, as also mentioned in (Agostinelli et al. 2019).

In order to reduce the solution tree size (with shorter paths to prove), we also propose a new AlphaZero-like training method called Faster to Life (FTL), by modifying the goal of the AlphaZero algorithm so that it prefers winning in fewer moves. In our experiments, among a collection of 20 7x7 L&D problems, all are solved with RZS integrated...
into AlphaZero with FTL; with RZS without FTL, 5 can be solved; without RZS nor FTL, none can be solved. Furthermore, among 106 19x19 Go problems selected from a well-known L&D book, written by a Go master, we solve 68 with both FTL and RZS, while solving 36 with RZS but without FTL. In contrast, a previous solver by Kishimoto and Müller (2005b) that does not use FTL and RZS solves 11 problems.

**Background**

**Solution Trees for Achieving Goals**

For two-player games, a solution tree (Stockman 1979; Pijls and de Bruin 2001) is a kind of AND-OR search tree representing a full strategy of answering moves to achieve a given goal, e.g., safety of pieces for Go or connectivity of pieces for Hex, as described in next subsections. Both AND and OR nodes represent game positions, and edges represent moves from one position to another. In this paper, for a position \( p \), let \( \beta(p) \) denote the corresponding board configuration and \( \pi(p) \) the player to play. The player to play at an OR node is called the OR-player, denoted by \( \pi_\text{OR} \), whose objective is to achieve the given goal, while the other player, called the AND-player, denoted by \( \pi_\text{AND} \), tries to prevent the same goal from happening. Achieving the goal in this context is equivalent to winning from the OR-player’s perspective.

A solution tree, rooted at \( n \), is a part of a search tree that satisfies the following properties.

1. If \( n \) is a leaf, the goal is achieved for \( \pi_\text{OR} \) at \( n \).
2. If \( n \) is an internal OR-node, there exists at least one child whose subtree is a solution tree. For simplicity, we usually consider the case where there is exactly one child.
3. If \( n \) is an internal AND-node, all legal moves from \( n \) must be included in the solution tree, and all subtrees rooted from \( n \)’s children are solution trees.

**L&D Problems for Go**

We first quickly review the game of Go and then define the goal for L&D problems. The game of Go is a game played by two players, Black and White, usually on an \( n \times n \) square board, where at most one stone, either black or white, can be placed on each intersection of the board, called a cell in this paper. A set of stones of the same color that are connected via adjacency to one another in four directions is called a block. Unoccupied cells adjacent to a block are called liberties of that block. A block is captured if an opponent stone is placed in the block’s last liberty. Following the rules of Go, each block has at least one liberty, and players are not allowed to make suicidal moves that deprives the last liberty of their own blocks, unless that move also captures opponent stones. Safety refers to a situation where blocks cannot be captured by the opponent under optimal play.

An L&D problem is defined as follows. Given a position and some stones of \( \pi_\text{OR} \) designated as crucial, as (Kishimoto and Müller 2005b), \( \pi_\text{OR} \) is said to achieve the goal if safety can be guaranteed for any of these crucial stones, or fail if all crucial stones are captured.

A set of blocks is said to be unconditionally alive (UCA) by Benson (1976) if the opponent cannot capture it even when unlimited consecutive moves are allowed. Namely, each of these blocks can sustain at least two liberties for unlimited consecutive moves by the opponent. By illustrating the seven positions in Figure 1, the white blocks satisfy UCA in \( p_0, p_c, \) and \( p_d \), illustrated in the bottom row of (b), (c), and (d), but not for others. The details of UCA are specified by Benson (1976). Achieving UCA for any crucial stones is a valid solution to a L&D problem.

In addition to UCA, safety can be achieved through other ways, such as seki (Niu, Kishimoto, and Müller 2005), ko, or situational super-ko (SSK) (van der Werf, Van Den Herik, and Uiterwijk 2003). Coexistence in seki is also considered a win for \( \pi_\text{OR} \), since the crucial stones are safe from being captured. Lastly, the rules of ko or SSK are used to prohibit position repetition, i.e., any move that repeats a previous board position with the same player to play is illegal. In this paper, we limit our goal to achieving safety via UCA only for simplicity. L&D problems in which the only solution is via seki and ko (SSK) are left as open problems. For simplicity, achieving safety via UCA is referred to as a win, or winning for the rest of this paper.

A specific instance of the more general L&D definition above is the so-called 7x7 kill-all Go game, in which Black, the first player, plays two stones initially, then both players play one stone per turn alternately on a 7x7 Go board. The goal of White, playing as \( \pi_\text{OR} \), is to achieve safety for any set of white stones via UCA, while Black (playing as \( \pi_\text{AND} \)) must kill all white stones. Note that for this specific problem, there is no need to mark crucial stones for White; any safe white stone is sufficient. For simplicity, the rest of this paper is
written with 7x7 kill-all Go in mind, i.e., \( \pi_0 \) wins when any white stones are UCA, unless specified otherwise.

**Connectivity for Hex**

For generality, we also discuss another goal-achieving problem for Hex. We first review Hex, then define the goal of achieving connectivity. Hex is a two-player game commonly played on a board with \( n \times n \) hexagonal cells in a parallelogram shape as illustrated in Figure 3 below. The two players, Black and White, are each assigned two opposite sides of the four boundaries of the board, and take turns to place stones of their own color. Similarly, a set of stones of the same color that are connected via adjacency to one another in a hexagonal way is called a block, and each side of the board can be viewed as a block of the same color for simplicity. A board configuration consists of four sides and all cells, each of which is unoccupied or occupied by either Black or White. The player who has a block of their own color on the boundary wins the game. The rules of Hex are relatively simple in the sense that the player who has a block of their own color connecting the two correspondingly colored sides wins the game. The player who has a block of their own color connecting the two correspondingly colored sides wins the game.

A goal-achieving problem for Hex is defined as follows. Given a position \( p \) and some stones of \( \pi_0 \) designated as crucial, \( \pi_0 \) is said to achieve the goal if they can connect these crucial stones together, or fail otherwise. For example, in Figure 3 (a), a problem may specify that with Black (\( \pi_0 \)) playing first, C3 is a crucial stone and White must connect it to the lower side of the board. Figure 3 (c) shows a case of achieving the goal.

**Our Approach**

This section first defines relevance zones (RZs), and then reviews null moves and must-play regions (Hayward et al. 2003; Hayward and Van Rijswijck 2006; Hayward 2009) with illustrations. Next, we define RZ-based solution trees (RZSTs) and present Consistent Replay Conditions, under which \( \pi_0 \) can replay and therefore reuse previously searched solution trees to win. The final subsection proposes a novel approach, called RZ-based Search (RZS), which can be incorporated with other depth-first or best-first search for solving goal-achieving problems.

**Relevance Zones**

A zone \( z \) is a set of cells on the board. Given a position \( p \) and a zone \( z \), let \( \beta(p) \cap z \) denote the board configuration \( \beta(p) \) inside the zone \( z \), also called a zone pattern. Let \( \bar{z} \) denote the zone outside of \( z \). Thus, \( \beta(p) \cap \bar{z} \) is the zone pattern outside the mask \( z \).

A zone \( z \) is called a relevance-zone (RZ) with respect to a winning position \( p \), if the following property is satisfied.

**RZ-1**

For all positions \( p_a \) with \( \beta(p_a) \cap z = \beta(p) \cap z \) (the same zone pattern) and \( \pi(p_a) = \pi(p) \) (the same player turn), \( p_a \) are also wins.

In general, there exists some RZ with respect to a winning position. For positions that achieve UCA for Go, their RZs include all cells of the safe blocks and their regions (Benson 1976), since these blocks are safe regardless of any moves or changes outside the zone. Examples are given in Figure 1. Thus, a solution tree for \( p_a \) is shown in Figure 2 (a), and the zone \( z_a \) is clearly an RZ for \( p_a \). From RZ-1, we can easily obtain the following property.

**ZX-1**

If \( z \) is an RZ with respect to \( p \), any bigger zone \( z' \supseteq z \) is an RZ for \( p \) as well.

For example, the RZ for \( p_a \) in Figure 1 (a) can be expanded, say, by adding the black blocks or any number of unoccupied cells. Taken to the extreme, as long as \( p \) is a win, the entire board can serve as an RZ.

**Null Moves and Must-Play Regions**

Relevance zones can be derived based on null moves, as in lambda search (Thomsen 2000; Wu and Lin 2010), and must-play regions (Hayward et al. 2003; Hayward and Van Rijswijck 2006; Hayward 2009) with illustrations. Next, we define RZ-based solution trees (RZSTs) and present Consistent Replay Conditions, under which \( \pi_0 \) can replay and therefore reuse previously searched solution trees to win. The final subsection proposes a novel approach, called RZ-based Search (RZS), which can be incorporated with other depth-first or best-first search for solving goal-achieving problems.

For RZ pruning, previous lambda-based search methods tried null moves first, then derived RZs as above (Thomsen 2000; Yoshizoe, Kishimoto, and Müller 2007; Wu and Lin 2010). However, the decision on when and how to play null moves is itself a heuristic that requires deliberate thought. Usually, a null move consists of passing with the hope of reusing previously searched solution trees to win. The final subsection proposes a novel approach, called RZ-based Search (RZS), which can be incorporated with other depth-first or best-first search for solving goal-achieving problems.
In this paper, we propose searching promising moves directly without needing to try null moves first, then determining which moves are null moves post-hoc. For example, in Figure 1 (a), let Black attempt to kill White’s blocks by playing at D1 as in (b), leading to White responding at E2 with UCA achieved and the must-play region for Black is reduced to include only the unoccupied cells, C2, B1, and C1. Next, if Black plays at C2 as in (d), White replies at B2 and wins due to symmetry to (b). Again, since Black C2 is a null move, the RZ is the shaded cells as in (d) and the must-play region is reduced to B1 only, from the intersection of the two RZs in (b) and (d). Now, we only need to search Black B1, in which case White replies at D2 for a win as in (e), since C3 is guaranteed to be connected to D2, and D2 is guaranteed to be connected to the lower side independently. Since all unoccupied cells in the must-play region have been searched, the goal can be achieved, and the RZ for the position in (a) is the shaded area in (f), which is the union of the above RZs, denoted by \( z_f \). Namely, C3 can be connected to the lower side, regardless of any moves or changes outside of \( z_f \).

In this case, we achieve UCA at G5. This search continues with Black being forced to consider G5, and again White at F6 forces Black to play at G6 as in (d). Finally, White plays at F7 to achieve UCA. Thus, for the Black move at D2, we can conclude that White wins and the RZ \( z_e \) is the union of the RZs as in (e). Interestingly, since D2 is outside of \( z_e \), it is a null move and all of Black’s moves outside \( z_e \) can be disregarded. The must-play region is therefore the unoccupied cells in \( z_e \). Now, let Black choose to play F4 (inside \( z_e \)) as in (f). Similarly, White wins with an RZ \( z_f \) as in (f) due to symmetry to (e). Since we can disregard Black moves outside \( z_e \) and \( z_f \), the must-play region becomes the intersection of the two RZs as shown in (g), which contains no legal moves for Black. Thus we have proved White wins in (a), with large reductions to the search space. Note that the corresponding RZ for the position in (a) is the union of the two RZs, shaded in (h).

**Illustrations** We first illustrate an example for the game of Hex to facilitate to understand. Previously, (Hayward et al. 2003) has used the above notion of null moves and must-play regions to solve Hex problems. An example for Hex is given in Figure 3 (a), modified from (Hayward et al. 2003), where its goal is to connect C3 to the lower side, and can be achieved through Figure 3 (b)-(f).

Now, we want to illustrate another example to demonstrate the power of RZ pruning for the game of Go using the above approach. The position in Figure 4 (a) can be derived as a win for White as shown in (b)-(h). Thus, we can prove White wins in (a) by only searching two moves at D2 and F4 as shown in (e) and (f) respectively.

**Comparison with Previous Null Move Methods** Other than Hex, most traditional RZ approaches need to search...
null moves in advance to facilitate pruning, e.g. null-move pruning in chess (Donninger 1993; Heinz 1999; Campbell, Hoane Jr, and Hsu 2002) and lambda search (Thomsen 2000; Yoshizoe, Kishimoto, and Müller 2007; Wu and Lin 2010). For example, for the position in Figure 4 (a), if we want to make another region, say at G3, two consecutive null moves for Black need to be searched first (say, two passes) for White to play consecutively at F4 and G4, as in second order lambda search (Thomsen 2000). Null moves tend to be weak choices for πn (Black in this case), which runs counter to the intuition that strong moves should be searched first. Consequently, heuristics are usually used to determine whether and how πn makes null moves. During this process, incorrect guesses for null moves may incur additional overhead.

In contrast, with our approach, πn searches ahead, then retroactively handles null moves after solving them. In most cases of winning, πn gradually search those moves far away from the fighting regions, that is, it is likely to play null moves which are located outside of RZs. This allows our method to be seamlessly and elegantly incorporated into search algorithms such as MCTS (Coulom 2006) or alpha-beta search (Knuth and Moore 1975); namely, the algorithm can prioritize strong moves as usual, and exploit RZ pruning simultaneously.

Relevance-Zone Based Solution Trees (RZSTs)

The previous subsection illustrates the power of using relevance zones. We now incorporate the notion of relevance zones into solution trees recursively, which we refer to as Relevance-Zone Based Solution Trees (RZSTs). Since Hex is relatively straightforward compared to Go, we will focus on Go in the rest of this section.

For a node n, let p(n) be its corresponding position of n. For simplicity, let β(p(n)) be simplified as β(n), and π(p(n)) as π(n). If p(n) is a win, let z be an RZ for n, where z is guaranteed to exist as described above. From the definition of RZs, since the conditions of the win are satisfied entirely by the zone pattern within z, and the pattern in z are irrelevant, the solution tree rooted at n does not need to include any children in z if n is an AND-node. As an example, the solution tree in Figure 2 (a) can be reduced to include the first three children only, as shown in (b), where the thicker arrow for the move D1 indicates a null move whose RZ z0 is used to prune all moves other than E2 and F2. More specifically, all the moves outside z0 lead node a to b'.

The above definition for RZSTs leaves out how RZs are derived for the winning node n. A key to justifying the correctness of RZs is to replay RZSTs rooted at null moves. For the example in Figure 1 (a), if Black plays anywhere outside z0, White can win by simply "replaying" the strategy depicted by the solution tree at b', which is simply E2. Similarly, for another example in Figure 4 (e): if Black plays anywhere outside the shaded RZ (say C2), White can win by simply "replaying" F4, as shown in (b)-(d).

Before discussing this in more detail in later subsections, we first describe a common principle of deriving RZs.

ZZ-1 For parent-child pairs n and n' in RZSTs, let z and z' be their RZs respectively. We derive both RZs such that z' ⊆ z.

Intuitively, since the cells in z are irrelevant for n, moves in z should also be irrelevant to n’s descendants. We can see this from many examples in the previous subsection, where the RZ is a union of their children’s RZs.

Consistent-Replay (CR) Conditions

This subsection presents some sufficient conditions for RZs such that replay is always legal, which in turn justifies the correctness for RZSTs. An important notion of RZs, which originates from threat-space search (Thomsen 2000; Wu and Lin 2010), is that πn moves played outside the zone do not interfere with the winning strategy of πn inside the zone, which is represented by RZSTs.

Let a winning node n be the root of an RZST, and z be an RZ for n. From Property ZZ-1, all positions p, with β(p) ∩ z = β(n) ∩ z and π(p) = π(n) are wins too. In addition to UCA, one way to justify the correctness for πn is to verify that the winning strategy of πn consists of simply following (or replaying) the RZST to win. In order to replay consistently for all pn based on the RZST, we propose the following sufficient conditions, called Consistent Replay (CR) Conditions.

CR-1 Suppose it is πn’s turn. Consider all πn’s legal moves at unoccupied cells m inside z on pn. Then, playing at m on p(n) must be legal for πn as well, and must lead to the same zone pattern. Namely, β(p') ∩ z = β(p) ∩ z, where p' and p' are the next positions of p(n) and pn respectively after the move at m.

CR-2 Suppose it is πn’s turn. For any πn’s move outside z on pn, the zone pattern inside z remains unchanged, namely, β(p) ∩ z = β(p') ∩ z, where p' and p' are the next positions of p(n) and pn respectively after the move.

CR-3 Suppose it is πn’s turn and the winning move is at an unoccupied cell m inside z on p(n). Then, the move at m must be legal on pn as well, and must lead to the same zone pattern. Namely, like CR-1, β(p') ∩ z = β(p') ∩ z, where p' and p' are the next positions of p(n) and pn respectively after the move at m.

The conditions CR-1 and CR-3 ensure that πn can replay the same winning responses to all πn’s moves inside z on pn, while maintaining identical RZ patterns. This can be illustrated by the winning position pn with the RZ za in Figure 1 (a). For all positions p, with the same RZ pattern as p, all πn legal moves inside the zone on p (e.g., D1, E2, and F2) are also legal on pn, so πn can play winning responses in p as those in Figure 1 (b), (c) and (d), and the resulting zone patterns remains the same.

CR-2 ensures that for πn moves outside z, the zone pattern inside z remains unchanged. In this case, πn simply follows the same winning strategy as that for n. From ZZ-1, there must exist some null move with a zone z' ⊆ z in n, so White can simply respond by replaying the same strategy as for n'. For example, in Figure 1 (a), for all moves outside the RZ za, πn simply chooses a null move, D1 in this case, and then uses its reply at E2 to reply in pn. From the above, we obtain the following lemma.

Lemma 1. Assume that node n is a win with an RZ z, and that all RZs in the RZST rooted at n are derived following the
three CR conditions. Then, for all positions \( p_n \) with \( \beta(p_n) \odot z = \beta(n) \odot z \) (the same zone pattern) and \( \pi(p_n) = \pi(n) \) (the same player turn), \( p_n \) is a win by following the winning strategy of the RZST.

Proof. It suffices to show that \( \pi \) can replay the winning strategy of the RZST rooted at \( n \) by induction. If the goal is reached for \( p(n) \) with an RZ \( z \), then the goal is reached for \( p_n \) too.

Now, assume that the goal has not been reached for \( p(n) \) yet. Consider the case of \( \pi \) to play. Let \( \pi \) win by playing a winning move \( m \), leading to a new node \( n' \). From CR-3, \( \pi \) can play the same winning move at \( m \) on \( p_n \). From CR-3, the resulting zone patterns are identical, i.e., \( \beta(p'_n) \odot z = \beta(n') \odot z \). Since \( n' \) is also a win, there exists an RZ \( z' \) for it, where \( z' \subseteq z \) from ZZ-1. This implies that \( \beta(n') \odot z' = \beta(p'_n) \odot z' \). From RZ-1, it is a win for \( p'_n \) too.

For the case of \( \pi \) to play, we need to consider all legal moves \( m \) on \( p_n \) by \( \pi \). First, assume \( m \) to be inside \( z \). From CR-1, \( \pi \) can play at the same \( m \) on \( p(n) \), resulting in the same zone pattern, i.e., \( \beta(n') \odot z = \beta(p'_n) \odot z \) where \( n' \) is the next node of \( n \) and \( p' \) is the next position of \( p_n \). Since \( n' \) is also a win, there exists an RZ \( z' \), with \( z' \subseteq z \) from ZZ-1, such that \( \beta(n') \odot z' = \beta(p'_n) \odot z' \). Thus, \( \pi \) can win for \( p'_n \) by following the RZST rooted at \( n' \).

Second, assume \( m \) to be outside of \( z \). From CR-2, the zone pattern inside \( z \) remains unchanged, namely, \( \beta(n) \odot z = \beta(p_n) \odot z \) where \( p_n \) is the next position of \( p_n \) after the move. \( \pi \) can win by simply following the RZST rooted at \( n \).

In order to satisfy the above CR conditions, we first derive RZs from leaf nodes (e.g. by including all cells that contain the UCA blocks). For internal nodes, we dilate (expand) children RZs based on ZX-1 such that all CR conditions hold. In the worst case \( z \) is dilated to include the whole board, in which case \( z \) is empty. Note that smaller RZs lead to smaller trees, so for a winning position, it is preferable to choose as small an RZ as possible for more RZ pruning. For the game of Go, dilation requires some thought for capturing stones and prohibiting suicide moves. A heuristic zone dilation method for Go, referred to as the function \( \text{dilate}(z) \) below, is presented in (Shih et al. 2021). In contrast, the game of Hex needs no dilation, since there are no stone capturing (or moving) and no prohibited moves like suicides.

Relevance Zone Based Search

In this subsection, we propose a goal-achieving search approach, called RZ-based Search (RZS), which can be embedded into other search algorithms using best-first search or depth-first search. In this approach, we propose a method to derive more efficient RZSTs, called replayable RZSTs, which are defined recursively. Namely, replayable RZSTs rooted at \( n \) as well as their RZs \( z \) will be constructed and derived, starting from leaf nodes, in a bottom-up manner. The three CR conditions need to be satisfied if the node \( n \) is a win with an RZ \( z \). Our approach RZS can be incorporated into best-first search, such as MCTS, as described in the rest of this subsection, which is used in our experiments later. Actually, RZS can also be incorporated into depth-first search (e.g., alpha-beta search) as presented in (Shih et al. 2021), which behaves similarly to the must-play-based Hex solver by Hayward (2009).

Leaf Nodes Suppose that position \( p(n) \) for generated node \( n \) achieves the goal, e.g., UCA, and wins. Then, \( n \) is an RZST (consisting of a single leaf node), and their RZ is derived accordingly.

Internal Nodes Suppose in \( p(n) \), no \( \pi \) blocks are UCA yet (i.e., the position is not immediately winning). These internal nodes \( n \) will be recursively updated from their children \( n' \) (in a bottom-up manner), e.g., during the backpropagation phase of MCTS. Since \( n \) is an internal node, it can either be an OR-node or an AND-node, described as follows.

Internal OR-Nodes Suppose it is \( \pi \)’s turn (\( n \) is an OR-node). Assume that the child \( n' \) via a legal move \( m \) is proven to be a win, and that the subtree rooted at \( n' \) is a replayable RZST. Let \( z' \) be the associated RZ for \( n' \). The RZ \( z \) associated with \( n \) is derived from \( \text{dilate}(z \cup \{m\}) \).

Thus, \( n \) together with the RZST rooted at \( n' \) forms a replayable RZST. From Lemma 1, for all positions \( p_n \) with \( \beta(p_n) \odot z = \beta(n) \odot z \), \( \pi \) is allowed to replay \( m \) with the RZST rooted at \( n' \) to win \( p_n \) with the same next RZ \( z' \).

Internal AND-Nodes Suppose it is \( \pi \)’s turn. Let \( n \) maintain a must-play region \( M \) that is initialized to all the legal moves in \( p(n) \). Assume that a child \( n' \) via a legal move \( m \in M \) is proven as a win with the associated RZ \( z' \), and that the subtree rooted at \( n' \) is a replayable RZST. The node \( n \) is updated as follows.

1. Consider the case that \( m \notin z' \) and the move \( m \) does not change the zone pattern inside \( z' \), that is, \( m \) is a null move. Then, shrink the must-play region by \( M = M \setminus z' \).

2. Consider the case where \( m \in z' \), that is, \( m \) is not a null move. Then, simply remove \( m \) from \( M \).

When the must-play region becomes empty, the node \( n \) is a win with an RZ \( z \), derived by \( \text{dilate}(z \cup \{z\}) \), where \( z \) is the union of all the child’s RZs. Then, \( n \) together with all of its child RZSTs, forms a replayable RZST. From Lemma 1, for all positions \( p_n \) with \( \beta(p_n) \odot z = \beta(n) \odot z \), \( \pi \) is allowed to replay with the same winning strategy based on these child RZSTs.

Faster to Life

This section proposes an AlphaZero-like method, named Faster to Life (FTL), so that the search prefers choosing moves that win with the least number of moves. Consider 7x7 kill-all Go, where White wins as long as any white stones are UCA. When using unaltered AlphaZero, we can simply set a komi\(^2\) of 48.

\(^1\)In this context, \( m \) is both a conceptual construct representing an edge in the tree, and also a cell point signifying where the player places their stone. In this way we may add \( m \) into some zone \( z \), which is itself a set of cells.

\(^2\)In Go, komi is a number of compensation points for White since Black has the advantage of playing the first move.
<table>
<thead>
<tr>
<th></th>
<th>w/o RZS</th>
<th>w/ RZS</th>
</tr>
</thead>
<tbody>
<tr>
<td>7x7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/o FTL</td>
<td>0/20</td>
<td>5/20</td>
</tr>
<tr>
<td>w/ FTL</td>
<td>3/20</td>
<td>20/20</td>
</tr>
<tr>
<td>19x19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/o FTL (ELF)</td>
<td>0/106</td>
<td>36/106</td>
</tr>
<tr>
<td>w/ FTL</td>
<td>6/106</td>
<td>68/106</td>
</tr>
<tr>
<td>T-EXP</td>
<td>11/106</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: Number of solved problems under different settings.

With FTL, we defined the winning condition to be White achieving UCA for any number of stones within \( d \) moves from the current position. For example, with \( d = 20 \), White wins only if it is able to reach UCA within 20 moves. Given multiple values of \( d \), the problem setting is similar to determining win/loss with multiple komi values, which can be handled with multi-labelled value networks (Wu et al. 2018).

Given a position \( p \), the value network head outputs a set of additional \( d \)-win rates (the rates of winning within \( d \) moves), where \( d \) ranges from 1 to a sufficiently large number (set to 30 in our experiments). Namely, for all positions \( p \) during the self-play portion of AlphaZero training, if White lives by UCA at the \( d \)-th move from the current position, we count one more \( d \)-win for all \( d' \geq d \), but not for all \( d' < d \). These \( d' \)-wins are used to update the set of \( d \)-win rates of \( p \). Hence, White tends to win faster if their \( d \)-win rates are high for low values of \( d \).

To ensure self-play plays reasonably well under multiple values of \( d \), we apply a concept similar to dynamic komi for multiple win rates (Baudiš 2011; Wu et al. 2018), where we set the winning condition to \( d_w \)-win, and \( d_w \) is adjusted dynamically during self-play. Usually, \( d_w \) is adjusted such that the win rate is close to 50% for balancing (Baudiš 2011). The above can be applied to 19x19 L&D problems, with the difference that a win is defined as achieving UCA for any crucial stones.

Experiments

To demonstrate our RZ solver, we solve a collection of L&D problems on 7x7 kill-all Go and 19x19 Go.

7x7 Kill-All Go L&D Problems

First, based on our program CGI (Wu, Wei, and Wu 2020), we train two AlphaZero programs for 7x7 kill-all Go, one with FTL and the other without FTL method, and chose 20 problems for 7x7 kill-all Go for analysis. A problem is marked as proven if a program can prove it within 500,000 simulations. In addition, a transposition table is used to record proven positions to prevent redundant search.

In this experiment, we compare versions with and without FTL and RZS. The results, shown in Table 1, indicate that RZS together with FTL solves all 20 problems.

19x19 Go L&D Problems

Next, we apply the RZS method to 19x19 Go L&D problems, which are widely-studied in the Go community. We select problems from a well-known L&D problems book, aptly named the “Life and Death Dictionary”, written by Cho Chikun [1987], a Go grandmaster; problem difficulties range from beginner to professional levels. A total of 106 problems, where the goal is to keep any crucial stones alive with UCA, were chosen. For simplicity, these problems do not require seki or ko (SSK) to achieve safety.

We first incorporated our approach into a 19x19 Go MCTS program that uses a pre-trained network with 20 residual blocks from the open-source program ELF OpenGo (Tian et al. 2019). Second, we trained a 19x19 AlphaZero Go program with the FTL method by using the problems from the tsumego book The Training of Life and Death Problems in Go (Shao, Ding, and Liu 1991). The game ends immediately if Black/White solves the L&D problems. We also added two additional feature planes to represent the crucial stones of each problem for both players.

We set up programs in a similar manner to those in 7x7 kill-all Go. The baseline is the program TSUMEGO EXPLORER (Kishimoto and Müller 2005b) (abbr. T-EXP), which is a DF-PN based program with Go specific knowledge, for which the search space needs to be manually designated. Since the problems cannot be used directly for ELF OpenGo (the network is trained to play with a komi of 7.5) and T-EXP (it requires limited regions surrounded by \( \pi_0 \)'s stones), we modified the problems such that they satisfy the requirements of these two programs. The time budget for each move is limited to 5 minutes, the same as the setting in (Kishimoto and Müller 2005b). Note that in 5 minutes, about 150,000 simulations are performed using ELF and the one trained with FTL. Results are shown in Table 1. Most notably, our program can solve 68 problems with RZS and FTL, 36 with RZS only (not FTL), and none without these methods (ELF). In comparison, T-EXP can only solve 11 problems due to the large search space in the 19x19 board; it outperforms ELF (without RZS nor FTL), most likely due to its stronger domain knowledge.

The above L&D problems, statistics, and training details can be accessed via the Github repository (Shih et al. 2021).

Discussion

Our approach RZS is general in the sense that it is applicable to many other goal-achieving problems, in addition to Go and Hex. A list of open issues worthy of further investigation is as follows.

- Extend RZS to other goal-achieving problems for Go, such as seki and SSK, to solve more L&D problems. Incidentally, many experts expect a win for White for 7x7 kill-all Go. If so, the success of this work will make it more likely to solve 7x7 kill-all Go entirely.
- Apply RZS to other goal-achieving problems for other games, such as Gomoku, Connect6 and Slither (Bonnet, Jamain, and Saffidine 2015), and even other domain of applications.
- Incorporate RZS into other search algorithms, such as proof-number search (Allis 1994; Nagai 2002) or alpha-beta search (Knuth and Moore 1975).
- Finally, it is worth investigating whether zone patterns can act as features in pattern recognition to facilitate explainability in deep neural network training.
Acknowledgements

This research is partially supported by the Ministry of Science and Technology (MOST) of Taiwan under Grant Numbers 110-2634-F-009-022, 110-2634-F-A49-004 and 110-2221-E-A49-067-MY3, and the computing resources are partially supported by National Center for High-performance Computing (NCHC) of Taiwan. The authors also thank Professor Martin Müller and anonymous reviewers for their valuable comments.

References


