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

Chung-Chin Shih[1,2*], Ti-Rong Wu[1*], Ting Han Wei[3], and I-Chen Wu[1,2†]

[1]National Yang Ming Chiao Tung University, Hsinchu, Taiwan
[2]Research Center for Information Technology Innovation, Academia Sinica, Taiwan
[3]Department of Computing Science, University of Alberta, Edmonton, Canada
{rockmanray,kds285}@aigames.nctu.edu.com, 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 *tsumego*, 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 playing and 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

*These authors contributed equally.
†Corresponding author.

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 $\pi_{\square}$, whose objective is to achieve the given goal, while the other player, called the AND-player, denoted by $\pi_{\bigcirc}$, 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 $\pi_{\square}$ 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 Problem 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 a $n \times n$ square board, where at most one stone, either black or white, can be placed on each intersection of the board, called a *grid* 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 grids 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.

A L&D problem is defined as follows. Given a position and some stones of $\pi_{\square}$ designated as crucial, as (Kishimoto and Müller 2005b), $\pi_{\square}$ 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_b, 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 $\pi_{\square}$, 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 $\pi_{\square}$, is to achieve safety for any set of white stones via UCA, while Black (playing as $\pi_{\bigcirc}$) 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., $\pi_{\square}$ 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, also called grids for simplicity in this paper, 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 grids, each of which is unoccupied or occupied by either Black or White. The player who has a block of their own color connecting the two correspondingly colored sides wins the game. The rules of Hex are relatively simple in the sense that there are no stone capturing (i.e., any changes to stones that are already on the board) and no prohibited moves like suicides.

A goal-achieving problem for Hex is defined as follows. Given a position p and some stones of $\pi_{\square}$ designated as crucial, $\pi_{\square}$ 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 ($\pi_{\bigcirc}$) playing first, C3 is a crucial stone and White must connect it

to the lower side of the board. Figure 3 (b) 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 $\pi_{\square}$ 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 grids on the board. Given a position p and a zone z, let $\beta(p) \odot z$ 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, $\beta(p) \odot \bar{z}$ 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_*, where $\beta(p_*) \odot z = \beta(p) \odot z$, p_* is also a win.

In general, there exists some RZ with respect to a winning position. For positions that achieve UCA for Go, their RZs include all grids of the safe blocks and their regions[1], 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.

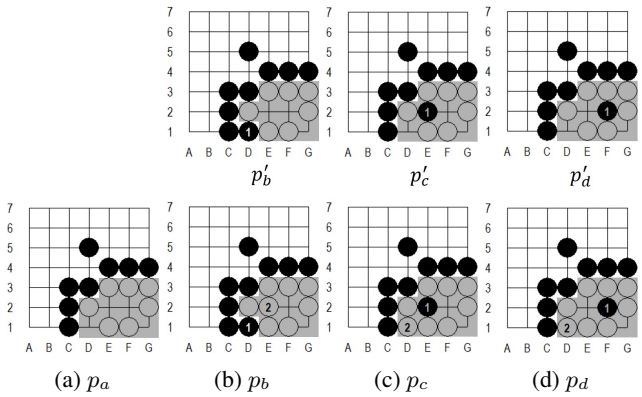
ZX-1 If z is an RZ with respect to p, any bigger zone $z' \supseteq z$ is an RZ for p as well.

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 grids. 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

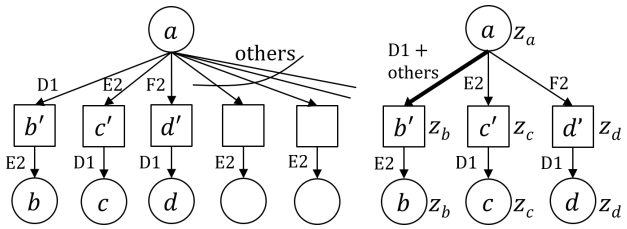
Relevance zones can be derived based on null moves, as in lambda search (Thomsen 2000; Wu and Lin 2010), and must-play regions (Hayward et al. 2003; Hayward and Van Rijswijck 2006; Hayward 2009). For example, in Figure 1, if Black makes a null move on p_a (in this case any move outside of z_a, including passes), White simply plays at E2 to reach UCA, obtaining in the process an RZ, z_b, like p_b. That is, it is a win for White regardless of any moves or changes outside z_b. Thus, back to p_a, to prevent White from winning directly, Black *must play* (or *try*) at unoccupied grids inside z_b, including moves E2, F2 and G1; z_b is conceptually equivalent to *must-play regions* in Hex by Hayward et al. (2003).

[1]Refer to Benson (1976) for the definition of regions in more detail.



(a) p_a (b) p_b (c) p_c (d) p_d

Figure 1: An example for Go. For position p_a, if Black plays at 1 as in p'_b, p'_c, and p'_d, White replies to win at 2 as in p_b, p_c, and p_d respectively. Since positions p_b, p_c, and p_d satisfy UCA, we obtain RZs as the shaded grids, denoted by z_b, z_c, and z_d respectively. For their previous positions p'_b, p'_c, and p'_d, White wins by playing 2 and it is not hard to see that their corresponding RZs are also z_b, z_c, and z_d respectively. Furthermore, consider p_a. For all moves outside z_a, denoting the shaded zone of p_a, White can simply play at E2 to win. Thus, p_a wins. Note that G1 is illegal for Black since suicide is prohibited. In addition, z_a is its RZ, a union of z_b, z_c, and z_d, since White follows the same strategy as above to win for all Black moves outside of it.



(a) The solution tree of Figure 1 (a). (b) The RZST for (a).

Figure 2: Solution trees.

By ignoring moves outside z_b, the search space is greatly reduced; we refer to this branch reduction as *relevance-zone pruning*. Since G1 is illegal in Go, only E2 and F2 actually need to be tried. For Hex, all the unoccupied grids inside an RZ are legal and are required to be tried.

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 reducing the branching factor. Since giving up one's turn tends to be one of the weakest moves to consider in most games, this runs counter to the general intuition that strong moves should be searched first. In addition, when the null move does not result in pruning, the extra effort spent to search the null move may be entirely wasted. Note that many previous researches employed some lightweight heuristics for

forced moves to prevent from incurring too much overhead.

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 RZ z_b, which encompasses the grids with UCA white stones. If we now look back at Black's decision of playing at D1, it is outside the RZ z_b, which makes it in effect a null move. According to Property RZ-1, White wins regardless of any moves and changes (including the move D1) outside z_b.

In this case, the must-play region that initially includes all unoccupied grids is reduced to E2 and F2 (note that G1 is illegal). If Black plays at E2 as in (c), it is a win with RZ z_c for White's reply at D1. Since E2 is inside z_c, it is not a null move and only E2 can be removed from the must-play region. It is similar for Black at F2 in (d) with an RZ z_d. After searching E2 and F2, the must-play region becomes empty, which implies a win for White for the position p_a in (a). The RZ z_a for p_a is therefore the union of z_b, z_c, and z_d. The power of relevance zones and must-play regions are illustrated in more examples for Hex and Go as follows.

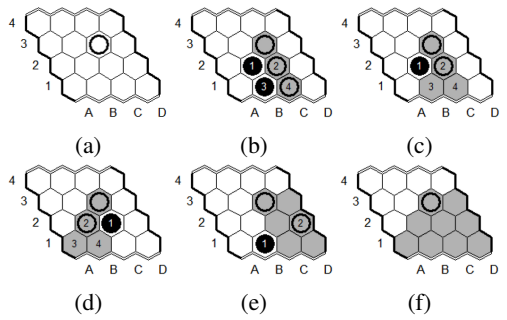


Figure 3: A goal-achieving problem in Hex: connecting C3 to the bottom side.

Illustration for Hex Previously, Hayward et al. (2003) has used the notion of null moves and must-play regions to solve Hex problems. Consider the example in Figure 3 (a), modified from (Hayward et al. 2003), and its goal is to connect C3 to the lower side. For Black B2 in (c), White replies at C2, then wins by either B1 or C1. The RZ is the shaded grids, since the goal can be achieved regardless of any move outside of the RZ. Similar to the example in the previous subsection, B2 is viewed as a null move, and the must-play region for Black is reduced to include only the unoccupied grids, C2, B1, and C1. Next, if Black plays at C2 as in (d), White replies at B2 and wins due to symmetry to (c). Again, since Black C2 is a null move, the RZ is the shaded grids as in (d) and the must-play region is reduced to B1 only, from the intersection of the two RZs in (c) 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 grids 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.

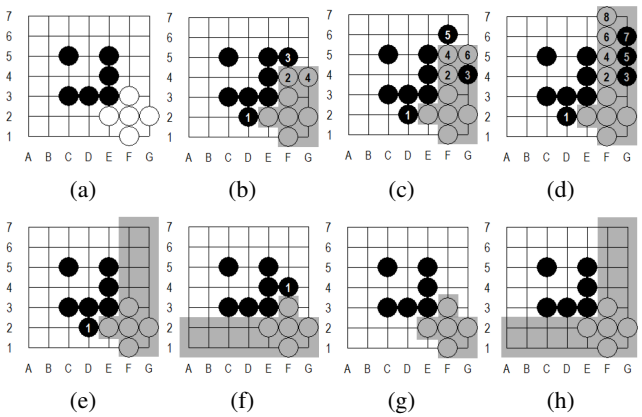


Figure 4: A 7x7 killall-Go example.

Illustration for Go Now, we examine an example in Go to demonstrate the power of RZ pruning using the above approach. The position in Figure 4 (a) can be derived as a win for White as follows. First, Black tries to prevent White from achieving UCA by playing at, say, D2 as in (b). In this case, White replies at F4 in order to achieve UCA by making a second region at G3 (the first is at G1). If Black counters with F5, then White achieves UCA with G4, where the set of shaded grids is its RZ, denoted by z_b. Since F5 is outside of z_b, it is a null-move, so the must-play region includes the two legal moves G3 and G4 that Black can try. For G3, it is trivial to win at G4. Black now considers playing at G4 instead, leading to the position in (c). In this variation, White plays at F5 in response to Black's move at G4. Again, if Black counters with F6, then White achieves 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 Figure 4 (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 grids 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), which is used to determine RZs for (a)'s ancestors.

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 $\pi_{\bigcirc}$ (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 $\pi_{\bigcirc}$ makes null moves. During this process, incorrect guesses for null moves may incur additional overhead.

In contrast, with our approach, $\pi_{\bigcirc}$ searches ahead, then retroactively handles null moves after solving them. In most cases of winning, $\pi_{\bigcirc}$ 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 straight-forward 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 $\beta(p(n))$ be simplified as $\beta(n)$, and $\pi(p(n))$ as $\pi(n)$. 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 $\bar{z}$ are irrelevant, the solution tree rooted at n does not need to include any children in $\bar{z}$ 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 z_b is used to prune all moves other than E2 and F2. More specifically, all the moves outside z_b lead node a to b' as in Figure 2.

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 z_a, 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 $z' \subseteq z$.

Intuitively, since the grids in $\bar{z}$ are irrelevant for n, moves in $\bar{z}$ should also be irrelevant to n's descendants. We can see this from many examples in the previous subsection, where the RZ is an 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 $\pi_{\bigcirc}$ moves played outside the zone do not interfere with the winning strategy of $\pi_{\square}$ 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 RZ-1, all positions p_* with $\beta(p_*) \odot z = \beta(n) \odot z$ are wins too. In addition to UCA, one way to justify the correctness for p_* is to verify that the winning strategy of p_* consists of simply following (or replaying) the RZST to win. In order to replay consistently for all p_* based on the RZST, we propose the following sufficient conditions, called *Consistent Replay (CR) Conditions*.

CR-1 Suppose it is $\pi_{\bigcirc}$'s turn. Consider all $\pi_{\bigcirc}$'s legal moves at unoccupied grids g inside z on p_*. Then, playing at g on $p(n)$ must be legal for $\pi_{\bigcirc}$ as well, and must lead to the same zone pattern. Namely, $\beta(p') \odot z = \beta(p'_*) \odot z$, where p' and p'_* are the next positions of $p(n)$ and p_* respectively after the move at g.

CR-2 Suppose it is $\pi_{\bigcirc}$'s turn. For any $\pi_{\bigcirc}$'s move outside z on p_*, the zone pattern inside z remains unchanged, namely, $\beta(p) \odot z = \beta(p_*) \odot z = \beta(p'_*) \odot z$, where p'_* is the next position of p_* after the move.

CR-3 Suppose it is $\pi_{\square}$'s turn and the winning move (by $\pi_{\square}$) is at an unoccupied grid g inside z on $p(n)$. Then, the move at g must be legal on p_* for $\pi_{\square}$ as well, and must lead to the same zone pattern. Namely, like CR-1, $\beta(p') \odot z = \beta(p'_*) \odot z$, where p' and p'_* are the next positions of $p(n)$ and p_* respectively after the move at g.

The conditions CR-1 and CR-3 ensure that $\pi_{\square}$ can replay the same winning responses to all $\pi_{\bigcirc}$ moves inside z on p_*, while maintaining identical RZ patterns. This can be illustrated by the winning position p_a with the RZ z_a in Figure 1 (a). For all positions p_* with the same RZ pattern as p_a, all $\pi_{\bigcirc}$ legal moves inside the zone on p_* (e.g., D1, E2, and F2) are also legal on p_a, so $\pi_{\square}$ 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 $\pi_{\bigcirc}$ moves outside z, the zone pattern inside z remains unchanged. In this case, $\pi_{\square}$ simply follows the same winning strategy as that for n. From ZZ-1, there must exist some null move with a zone $z' \subseteq z$ in n, so White can simply respond by replaying the same strategy as that for n'. For example, in Figure 1 (a), for all moves outside the RZ z_a, $\pi_{\square}$ simply chooses a null move, D1 in this case, and then uses its reply at E2 to reply in p_*. 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_* with the same zone pattern, i.e., $\beta(p_*) \odot z = \beta(n) \odot z$, p_* is a win by following the winning strategy of the RZST.

Proof. It suffices to show that $\pi_\square$ 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_* too.

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

For the case of $\pi_\bigcirc$ to play, we need to consider all legal moves m on p_* by $\pi_\bigcirc$. First, assume m to be inside z. From CR-1, $\pi_\bigcirc$ can play at the same m on $p(n)$, resulting in the same zone pattern, i.e., $\beta(n') \odot z = \beta(p'_*) \odot z$ where n' is the next node of n and p'_* is the next position of p_*. Since n' is also a win, there exists an RZ z', with $z' \subseteq z$ from ZZ-1, such that $\beta(n') \odot z' = \beta(p'_*) \odot z'$. Thus, it is a win for p'_* 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, $\beta(n) \odot z = \beta(p_*) \odot z = \beta(p'_*) \odot z$, where p'_* is the next position of p_* after the move. $\pi_\square$ 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 grids 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 $\bar{z}$ 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 suicidal moves. A heuristic *zone dilation* method for Go, referred to as the function $dilate(z)$ 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 section, 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 $\pi_\square$ 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 $\pi_\square$'s turn (n is an OR-node). Assume that the child n' via a legal move m[2] 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 $dilate(z \cup \{m\})$. Thus, n together with the RZST rooted at n' forms a replayable RZST. From Lemma 1, for all positions p_* with $p_* \odot z = p(n) \odot z$, $\pi_\square$ is allowed to replay m with the RZST rooted at n' to win p_* with the same next RZ z'.

Internal AND-Nodes Suppose it is $\pi_\bigcirc$'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 $m \notin z'$, that is, m is a null move. Then, shrink the must-play region by $M = M \cap z'$.
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 $dilate(z_{union})$, where z_{union} is the union of all the children' RZs. Then, n, together with all of its child RZSTs, forms a replayable RZST. From Lemma 1, for all positions p_* with $p_* \odot z = p(n) \odot z$, $\pi_\square$ 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[3] of 48.

[2] In this context, m is both a conceptual construct representing an edge in the tree, and also a grid 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 grids.

[3] In Go, komi is a number of compensation points for White since Black has the advantage of playing the first move.

		w/o RZS	w/ RZS
7x7	w/o FTL	0/20	5/20
	w/ FTL	3/20	**20/20**
19x19	w/o FTL (ELF)	0/106	36/106
	w/ FTL	6/106	**68/106**
	T-EXP	11/106	-

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, 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, indicates 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 $\pi_{\bigcirc}$'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 7x7 and 19x19 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.

Acknowledgement

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

References

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8): 356–363.

Allis, L. V. 1994. *Searching for Solutions in Games and Artificial Intelligence*. Ph.D. thesis, University of Limburg, Maastricht, The Netherlands.

Baudiš, P. 2011. Balancing MCTS by dynamically adjusting the komi value. *ICGA Journal*, 34(3): 131–139.

Benson, D. B. 1976. Life in the game of Go. *Information Sciences*, 10(2): 17–29.

Bonnet, É.; Jamain, F.; and Saffidine, A. 2015. Draws, zugzwangs, and PSPACE-completeness in the slither connection game. In *Advances in Computer Games*, 160–176. Springer.

Campbell, M.; Hoane Jr, A. J.; and Hsu, F.-h. 2002. Deep Blue. *Artificial Intelligence*, 134(1-2): 57–83.

Cho, C. 1987. *Life and Death Dictionary*, volume 1-2. Mercury Publishing House. ISBN 9575611411,957561142X.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *5th International Conference on Computers and Games*, 72–83.

Donninger, C. 1993. Null move and deep search: Selective search heuristics for obtuse chess programs. *ICGA Journal*, 16(3): 137–143.

Hayward, R.; Björnsson, Y.; Johanson, M.; Kan, M.; Po, N.; and van Rijswijck, J. 2003. Solving 7× 7 Hex: Virtual connections and game-state reduction. In *Advances in Computer Games*, 261–278. Kluwer Academic Publishers.

Hayward, R. B. 2009. A puzzling Hex primer. In *Games of No Chance 3, Proc. BIRS Workshop on Combinatorial Games*, 151–161.

Hayward, R. B.; and Van Rijswijck, J. 2006. Hex and combinatorics. *Discrete Mathematics*, 306(19-20): 2515–2528.

Heinz, E. A. 1999. Adaptive null-move pruning. *ICCA Journal*, 22(3): 123–132.

Kishimoto, A.; and Müller, M. 2003. Df-pn in Go: An application to the one-eye problem. In *Advances in Computer Games 10*, 125–141. Kluwer Academic Publishers.

Kishimoto, A.; and Müller, M. 2005a. Dynamic Decomposition Search: A Divide and Conquer Approach and its Application to the One-Eye Problem in Go. In *IEEE Symposium on Computational Intelligence and Games (CIG'05)*, 164–170. IEEE Press.

Kishimoto, A.; and Müller, M. 2005b. Search versus knowledge for solving life and death problems in Go. In *Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 1374–1379. AAAI Press.

Knuth, D. E.; and Moore, R. W. 1975. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4): 293–326.

Nagai, A. 2002. *Df-pn Algorithm for Searching AND/OR Trees and Its Applications*. Ph.D. thesis, University of Tokyo, Tokyo, Japan.

Niu, X.; Kishimoto, A.; and Müller, M. 2005. Recognizing seki in computer Go. In *Advances in Computer Games*, 88–103. Springer.

Pijls, W.; and de Bruin, A. 2001. Game tree algorithms and solution trees. *Theoretical Computer Science*, 252(1-2): 197–215.

Shao, Z.-Z.; Ding, B.; and Liu, Q.-Q. 1991. *The Training of Life and Death Problems in Go*, volume 1-3. Chengdu Times Press. ISBN 7805488444,7805488525,7805488533.

Shih, C.-C.; Wu, T.-R.; Wei, T. H.; and Wu, I.-C. 2021. https://github.com/rockmanray/rzone.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Stockman, G. C. 1979. A minimax algorithm better than alpha-beta? *Artificial Intelligence*, 12(2): 179–196.

Thomsen, T. 2000. Lambda-Search in Game Trees—With Application to Go. *ICGA Journal*, 23(4): 203–217.

Tian, Y.; Ma, J.; Gong, Q.; Sengupta, S.; Chen, Z.; Pinkerton, J.; and Zitnick, L. 2019. ELF OpenGo: an analysis and open reimplementation of AlphaZero. In *Proceedings of the 36th International Conference on Machine Learning*, 6244–6253.

van der Werf, E. C.; Van Den Herik, H. J.; and Uiterwijk, J. W. 2003. Solving Go on small boards. *ICGA Journal*, 26(2): 92–107.

Wu, I.-C.; and Lin, P.-H. 2010. Relevance-Zone-Oriented Proof Search for Connect6. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(3): 191–207.

Wu, T.-R.; Wu, I.-C.; Chen, G.-W.; Wei, T.-h.; Wu, H.-C.; Lai, T.-Y.; and Lan, L.-C. 2018. Multi-labelled value networks for computer Go. *IEEE Transactions on Games*, 10(4): 378–389.

Yoshizoe, K.; Kishimoto, A.; and Müller, M. 2007. Lambda Depth-First Proof Number Search and Its Application to Go. In *Proc. of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2404–2409.