Recognizing Seki in Computer Go

Xiaozhen Niu[1], Akihiro Kishimoto[2], and Martin Müller[1]

[1] Department of Computing Science, University of Alberta, Edmonton, Canada, T6G 2E8
{xiaozhen, mmueller}@cs.ualberta.ca
[2] Department of Media Architecture, Future University-Hakodate 116-2, Kamedanakano-cho, Hakodate, Hokkaido, 041-8655, Japan
kishi@fun.ac.jp

Abstract. Seki is a situation of coexistence in the game of Go, where neither player can profitably capture the opponent's stones. This paper presents a new method for deciding whether an enclosed area is or can become a seki. The method combines local search with global-level static analysis. Local search is used to identify possible seki, and reasoning on the global level is applied to determine which stones are safe with territory, which coexist in a seki and which are dead. Experimental results show that a safety-of-territory solver enhanced by this method can successfully recognize a large variety of local and global scale test positions related to seki. In contrast, the well-known program *GNU Go* can solve only easier problems from a test collection.

1 Introduction

In the game of Go, the player who has the larger territory wins the game. It is therefore a fundamental requirement for programs to always assess the safety of territories correctly. Current computer Go programs use a combination of exact and heuristic techniques, including eye-space analysis [4], search and heuristic rules based on influence. Heuristic approaches do not guarantee correctness. In order to improve the accuracy of the territory evaluation, they need to be replaced by exact techniques, using search.

An exact, state of the art search-based safety-of-territory solver is described in [12]. However, this solver could not recognize safe stones in seki. The new solver described in this paper extends the previous one and is able to recognize many, even complex seki situations.

Seki is a position where neither player can capture the opponent's stones, so coexistence is the best result. There are two major issues about how to recognize seki. First, a pass is often the only good move for both players. Consecutive passes can cause the Graph History Interaction (GHI) problem [13], which leads to incorrect search results. As a second issue, even if a seki position is recognized locally, this recognition is done in a specific context. The most pessimistic context for one player assumes that the outside is completely filled by safe opponent stones. The result of a local search depends on such assumptions and might need to be updated once its surrounding information has been changed by searches in other regions.

This paper presents search-based methods that correctly deal with enclosed areas involving seki. The contributions of the paper are summarized as follows: The status of

local seki is correctly determined by using the df-pn(r) algorithm [5]. An efficient research method distinguishes a seki from a win for one player by capturing the opponent. An algorithm for solving global seki problems combines the outcome of local seki searches.

The paper is organized as follows: Section 2 describes the terminology. Section 3 briefly reviews related work. Section 4 explains recognizing seki and its importance to territory evaluation. Section 5 introduces the safety solver and the search algorithms for solving the local and global seki problems. Section 6 presents and describes experimental results and Section 7 provides conclusions and discusses future work.

2 Terminology

Terminology follows [12]: A *block* is a maximal connected set of stones with the same color on the Go board. The adjacent empty points of a block are called *liberties*. A block that loses all its liberties is *captured* and removed from the board. Stones of one color divide the rest of the board into *basic regions*. A *merged region* is the union of two or more basic regions of the same color. The term *region* refers to either a basic or a merged region. If a block is adjacent to only one region of its color, it is called an *interior block*, otherwise it is called a *boundary block* of the region.

A liberty of a boundary block of a region r is called *internal liberty* if it is in r, and *external liberty* otherwise. The color of boundary blocks is called the *defender*, the opponent is called the *attacker*. A *shared liberty* is an empty point that is adjacent to blocks of attacker and defender. A defender's region is called *safe* if all its boundary blocks can be proved safe and the attacker can not live inside the region.

In a region, simple seki are positions that falls into either of following two types:

1. The defender cannot capture attacker's stones, neither can the attacker. They both do not have two clear eyes. The best result for both players is to pass.
2. The defender and the attacker are allowed to capture each other's stones. However it will lead to repetitions of the board.

For a discussion of complex, strange cases, see http://senseis.xmp.net/?StrangeSekis. In simple cases, a region can be recognized as a seki by using static rules. However, verifying complicated seki may require deep search. A seki is often related to a *semeai*, a race to capture between two adjacent groups that cannot both live. For a detailed discussion of semeai, and static methods for evaluation of semeai classes including many seki, see [10].

Figure 1 shows two seki examples. The left Black region is a *static seki*. Both the defender and the attacker must pass locally, to ensure that their blocks △ and △ are not captured. They share two liberties in A and B. In the right Black region, after 3 moves, with White as the attacker playing first, the Black region becomes a seki. Even though the region is not safe, the boundary block △ is safe by seki. This kind of seki that must be discovered by search will be called *dynamic seki*.

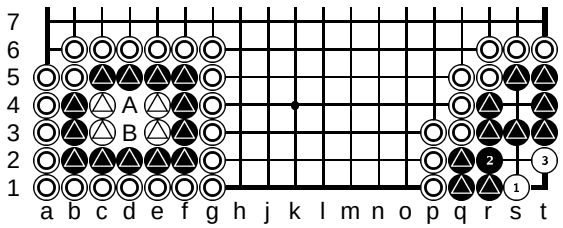

Fig. 1. Left: a static seki. Right: a dynamic seki.

3 Related Work

Many approaches have been proposed for recognizing safe stones and territories, including both static and search-based methods.

Unconditionally alive blocks [2] are safe even when the attacker can play an unlimited number of moves and the defender always passes. Related work on seki analysis also includes Müller's static analysis and search-based semeai algorithms [10], and Vilà and Cazenave's static classification rules to recognize safe blocks containing regions up to a size of 7 points [15], including some seki. Among these static methods, only [10, 15] can detect seki.

Search-based methods are used in life and death (tsume Go) and safety-of-territory solvers. Müller introduces local search methods for identifying the safety of regions by *alternating play*, where the defender is allowed to reply to each attacker move [9]. Van der Werf extends Müller's static rules and uses them in his program to score Go positions [16]. The methods in [9, 12] prove territories safe by using static rules and search.

Although some of the static approaches above can handle seki, they do not provide a general solution for recognizing seki in enclosed areas. A practically successful heuristic way of dealing with seki is based on tactical search. Most strong Go programs contain tactical search engines, typically with a threshold of 5 liberties, to recognize blocks that seem safe from capture. This approach does not work for complicated seki cases. In addition, most programs fail to detect vulnerable territories which do not contain any attacker stones yet, where a seki can be reached after an invasion sequence. Even in the tiny example in Figure 1 on the right, most current programs will pass as both black and white in the starting position.

4 Recognizing Seki

Recognizing seki is strongly related to both the safety of territory and life and death problems. When evaluating the safety of a territory, seki can be viewed as a win for the attacker, since the defender's territory is destroyed. However when viewed as a life and death problem, the roles are switched since the defender can survive by achieving coexistence in seki.

The search methods to recognize seki in this paper use the most pessimistic assumptions about unproven outside properties for recognizing local seki. The worst-case assumption is that for a given region, all its boundary blocks are completely surrounded by safe attacker stones and have no external liberties. However, the alogirthm can take external eyes that have been detected before into account. Any seki recognized under worst-case assumptions will be called as a *local seki*.

The result of local seki can be modified when information about the surroundings changes. For example, external liberties of boundary blocks of a region might affect the result of a local seki. An example is shown in Figure 2. The Black region is a local seki. However, since the boundary block of Black has one external liberty at A, the status of this Black region is unsettled. If White plays first at A, then the Black region is a seki. Black playing first can capture four White stones inside the region and the Black region becomes territory.

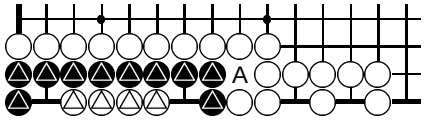

Fig. 2. External liberties can change seki status.

Recognizing a global seki is much more complicated because the result of a local seki might be affected when its surrounding conditions are changed. For example in Figure 1, if the surrounding White blocks ◎ , which were assumed safe, are proved to be dead later by a search on the outside, then in both examples the Black region changes from seki to safe, since the local seki collapses.

This paper describes a more general region-based approach to recognize local and global seki positions efficiently and correctly. A detailed discussion of the algorithms is provided in the following section.

5 Safety Solver and Algorithms

5.1 Safety Solver

The safety solver described in this paper is based on the ones in [8, 12]. It has been integrated into the Go program *Explorer* [8]. The search-based solver in [12] sequentially processes single regions and tries to prove their safety. To prove a region safe, all boundary blocks must remain safe - none of them can be captured by the attacker, and the attacker may not live inside the region surrounded by safe blocks. The solver uses Alpha-Beta search with enhancements including iterative deepening, a transposition table, move ordering and heuristic evaluation functions.

The safety solver used in this paper utilizes a more powerful df-pn(r) search algorithm [6, 5]. It includes the techniques of [12], such as the solution of strongly and weakly dependent regions, as well as static and heuristic region evaluation functions [12]. The following new features will be described in detail in later sections:

– Solutions to local and global seki.
– Solutions to basic ko situations and the GHI problem.
– The ability to switch goals between attacker and defender. The solver can be used both to find successful invasions, and to defend against them. For seki detection, the solver is used to prove that stones are safe while territories are not.

5.2 The Df-pn(r) Search Algorithm

Df-pn(r) [6, 5] is an extension of Nagai's depth-first proof-number (df-pn) search algorithm [11]. Df-pn modifies Allis' best-first proof-number search (PNS) [1] algorithm to use depth-first search. It can expand fewer interior nodes and use a smaller amount of memory than PNS. Df-pn utilizes local thresholds for both proof and disproof numbers, selects the most promising node, and performs iterative deepening until exceeding either one of the thresholds. Because df-pn is an iterative deepening method that expands interior nodes again and again, the heart of the algorithm is its use of the transposition table. Whenever a node is explored, the transposition table is used to cache previous search efforts (i.e., proof and disproof numbers). Df-pn(r) solves a problem of df-pn on computing proof and disproof numbers in the presence of repetitions, while inheriting the good properties of df-pn. Df-pn(r) contributed to the strength of a one-eye solver and the currently best tsume-Go solver [5].

5.3 The Local Search Algorithm

The local search algorithm is region-based. It takes a region r as an input and generates all legal moves in r as well as a pass move. It searches the region until either the result is decided as safe/safe by seki, or unsafe, or a time limit is exceeded. r is evaluated as safe if and only if all boundary blocks of r are proved to be safe, and no attacker's stones can live inside. For details, see [12]. r is evaluated as unsafe if and only if any boundary block b of r is captured, or b has only one liberty left and it is the attacker's turn to make a move, or the attacker lives inside the region.

Since the outcome of the search sometimes depends on ko, the local search algorithm needs to model ko threats. The current model assumes that the attacker has the unlimited number of ko threats to win the ko. Therefore, the attacker is always allowed to immediately recapture ko. To deal with more complicated ko such as double or triple ko, the algorithm uses the *situational super-ko (SSK) rule*. Under the SSK rule, any move that repeats a previous board position, with the same color to play, is illegal. However, assuming an unlimited number of ko threats is often unrealistic, and in some special cases the safety solver fails in proving the safety. See Section 6.1 for an example and further discussion.

5.4 Seki and the Graph History Interaction Problem

Let P be a position where player p_1 is to play and Q be the position after p_1 passes in P. If p_1 passes in P and the opponent p_2 passes in Q, the second pass leads back to P. In the local search algorithm, this repetition is allowed, and p_2's pass leads to a

local seki. The two passes mean that neither player can win this position. Becasue of the unlimited ko threat model, ko is not affected by this.

Since df-pn(r) uses a transposition table, in the presence of repetitions the Graph History Interaction (GHI) problem must be addressed [13, 3]. GHI is a notorious problem that may cause search algorithms to incorrectly solve positions. A typical transposition table implementation ignores paths that df-pn(r) takes to cache search results. However, if a search result that depends on a path is saved in the transposition table, an incorrect cached result may be retrieved from the transposition table.

Figure 3, adapted from [7], shows an example of the GHI problem. Assume that G is a win for the attacker. Let $E \rightarrow H$ and $H \rightarrow E$ be pass moves. If H is searched via $A \rightarrow B \rightarrow E \rightarrow H$, seki is saved in the transposition table entry for H. Two consecutive passes lead back to E. However, this is incorrect if H is reached via $A \rightarrow C \rightarrow F \rightarrow H$, since $A \rightarrow C \rightarrow F \rightarrow H \rightarrow E \rightarrow G$ leads to an attacker win at H.

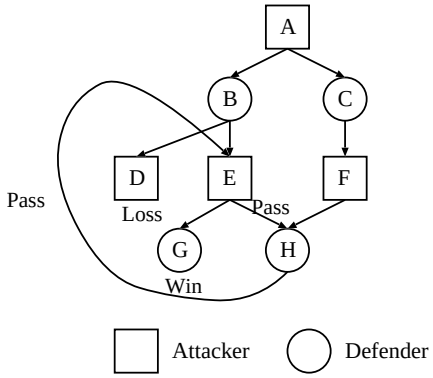

Fig. 3. The GHI problem related to seki.

The GHI problem related to seki is solved by the techniques in [7]. This approach uses an additional field in each transposition table entry to hash the path leading to a position.

5.5 Algorithm for Recognizing Local Seki

Since df-pn(r) can only answer binary questions, the algorithm for recognizing local seki may have to perform two search to determine the outcome as win, seki, or loss. Assume that the attacker plays first, and the first search evaluates terminal seki positions as a win for the defender. If the root position r is a loss for the defender, the result is established. However if the result of this search is a win, it could be either a defender win or a seki. In this case, for the second search all seki terminal positions are considered attacker wins. If the second search is still a defender win, then r is a defender win. However, if the second result is an attacker win, then r is a seki since neither player can win without using seki terminal positions.

The two searches with different seki winners are often quite similar. A method similar to speeding up re-search for ko in [6] decreases the overhead of the second search as follows: Each transposition table entry contains a *seki flag* f. In the first search, f is set if and only if a position's disproof tree contains seki.

Consider the second search, when df-pn(r) looks up the transposition table entry for a position n. If n's table entry contains either a proof, or the proof and disproof numbers of an unproven node, the information can be used safely. However, if n's table entry contains a disproof from the first search, df-pn(r) checks n's seki flag. If the flag is not set, the position is an unconditional win for the defender and this result can be reused. However, if the seki flag is set, the position is unsolved for this search. The proof and disproof numbers are initialized to 1 to perform a re-search.

The second search often has a low overhead because of the reuse of previous search results in the transposition table. If n's disproof tree does not involve seki, no search is performed. In the extreme case, the second search consists only of a single table lookup, if disproving the root did not involve seki in the first search. Even if a transposition table entry does not contain a disproof, using the proof and disproof numbers in the transposition table results in better move ordering. However, if the solution changes dramatically by seki, it might need a high overhead. One example is the test position 15 from Test Set 1 (see Section 6). The first search returns a loss for the attacker in 0.02 seconds, and the second search returns a seki win for the attacker in 0.93 seconds. In this example, the loss in the first search is obtained quickly by static rules. There is not much useful information in the transposition table that the second search could utilize.

5.6 Algorithm for Recognizing Global Seki

The local result of seki for a region can be proved by using the algorithm discussed in Section 5.5. However, this result might need to be updated once the assumptions about surrounding conditions are modified. Figure 4 provides an example. In the beginning, White region A is considered to be unsafe due to the following two reasons: The external liberties for block △ are not used during a local search. The block also has no internal liberties. White region B is a local seki because 1. the White block ⊗ is already proved to be safe with two eyes elsewhere, and 2. White's boundary block △ and Black block ▲ share two liberties. The White block △ can be marked as "at least seki", because the *worst* result for this block is safe-by-seki. It might be proved to be unconditionally safe in the future. The *best* result for the Black block ▲ is safe-by-seki, therefore it is marked as "at most seki". The information about safe "at least seki" blocks is used in a re-search for White region A, which will become safe. The black Block ■ is now dead. The white boundary block △ changes from "at least seki" to safe. Finally, by using this updated information region B is proved to be not a seki, but safe for White, and the status of the Black block ▲ changes from "at most seki" to dead.

In Figure 4, the observation that region A affects region B is trivial because they are adjacent. However, in a real board position the situation can be much more complex. For example, there might be multiple regions that are local seki and form a chain of seki. If the region in one end of the chain are proved to be not a seki later, then most likely the local result of every region in the chain has to be updated. In addition, care

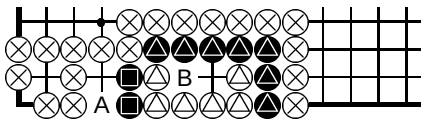

Fig. 4. A local seki that collapses on the global level.

should be taken for using the information about "at least seki" blocks. Clearly any "at least seki" block can only be used to prove one of its adjacent region that has the same color with the block. A very important question is: given an "at least seki" block, when it is safe to use it to prove its adjacent regions and when is it not? As described in the previous example, region B can be proved to be a local seki by search in the first round. If the "at least seki" block △ is used to perform the second search to B itself right away, then B will be proved to be safe instantly. However, this result is not correct. Region B can only be confirmed as safe once region A is confirmed as safe (so block △ becomes safe indeed). The solution used in this implementation uses the following condition. For a "at least seki" boundary block b and a region r to be proved:

- If b has no liberties inside r, then it is safe to be used as "safe" to prove r because it will not affect the liberties of any opponent's blocks that are inside r.
- If b does have internal liberties in r, then it should not be used as "safe" to prove r because it will affect the liberties of opponent's blocks that are in r.

By using this condition, it is obvious that in the previous example in Figure 4 the "at least seki" block △ should never be used to prove White region B directly because △ does has two internal liberties in B.

The algorithm for global recognition first statically recognizes safe regions using static rules in [2, 9]. Then it calls the local search algorithm for proving the safety of the remaining unsolved regions. When processing regions globally, the information about "at least seki" blocks achieved by local searches is stored and possibly used for updating the status of other regions. Eventually, if no further updates can be made, because the status of every region becomes stable, either safe/safe by seki, unsafe, or unknown due to time out, then the global search terminates. At this moment all "at least seki" and "at most seki" blocks are marked to safe, and the results for all local regions are guaranteed to be correct. The pseudo code in Figure 5 gives the global region processing algorithm for recognizing seki.

The two full-board positions from Test Set 2 shown in Figure 6 and Figure 7) are used to illustrate details of the global processing algorithm. In Figure 6, the solver processes unsafe regions one at a time for both colors in several rounds. At the beginning, the three White regions in the left side of the board have already been proven safe by static rules. When processing a region, the attacker is the opponent and plays first. Assume that all unsafe Black regions are processed first, followed by all unsafe White regions in the order A, B, C and D. The order of regions and colors does not affect the final result, but it may influence the efficiency. In the example, A, B, C and D are used to refer to different regions for Black and for White. For example, the white region A

```
List at-least-seki = ∅;
List at-most-seki = ∅;
R = all remaining unsafe regions of both colors;

bool updated = true;
while ( updated ) // The main loop terminates when there is no update can be made
    updated = false;
    for each region r in R perform a first local search
        if ( the first search result is proved safe )
            updated = true;
            mark all points in region r and its boundary blocks as safe;
        else if ( the first result is a local seki )
            Add attacker's blocks inside r to at-most-seki;
            Add unsafe defender's boundary blocks of r to at-least-seki;
            if ( any new boundary block was added to at-least-seki )
                updated = true;
        else if ( the first result is proved loss )
            Perform a re-search in region r by using information in at-least-seki;
                if (the second search result is proved safe )
                    updated = true;
                    Mark all points in region r and its boundary blocks as safe;
                if (the second search result is a local seki )
                    Add attacker's blocks inside r to at-most-seki;
                    Add unsafe defender's boundary blocks of r to at-least-seki;
                    if ( any new boundary block was added to at-least-seki )
                        updated = true;

Mark all blocks in both at-least-seki and at-most-seki as safe;
```

Fig. 5. Global region processing algorithm for determining seki.

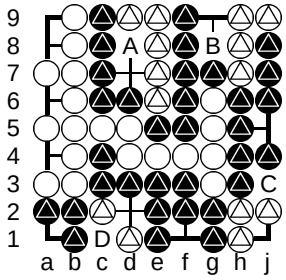

Fig. 6. Global search confirms local regions to be seki.

contains the black block to the left, while the black region A contains the white block to the right but not the black block.

1. Following the worst-case assumption for the outside, black Region A is unsafe because one of its boundary blocks does not have any internal liberties. Similarly black regions B and C are proved to be unsafe. Blocks $j8$ and $g1$ both have an outside eye, but it only provides one liberty. Region D is proved to be a local seki. Therefore its two boundary blocks at $b1$ and $g1$ are added to the"at least seki" list L. For White, region A is proved a local seki, and block $e9$ is added to L. Using this information, White region B is also a local seki and block $h9$ is added to L. Similarly, White region C is a local seki and block $h1$ is added to L, and then the merged region at $b1$ and $e1$ is also a local seki and blocks $c2$ and $d1$ are added to L. Techniques for merging strongly related regions from [12] are used here. After the first round of processing for both colors, L contains 2 Black blocks at $b1, g1$ and 5 White blocks at $d9, h9, h1, c2, d1$.
2. Re-process unsolved regions in the same order. Black regions A and B are still unsafe. Using the "at least seki" Black block at $g1$, now Black region C can be proved to be a local seki, and block $j8$ is added to L. For White, no progress is made.
3. Black region B is proven to be seki by using the "at least seki" block at $j8$, and block at $f9$ is added to L. Again, no progress for White can be made.
4. Black region A is proven seki by using the "at least seki" block at $f9$ and block $c9$ is added to L.
5. No update can be made in the fifth round. The main loop terminates and the remaining "at least seki" and "at most seki" blocks are marked as safe.

In this example, 10 "at least seki" blocks are marked as safe, and no "at most seki" blocks. In the end, all blocks are proved to be safe. A total of 70 points are marked as safe. The remaining empty points are neutral points in seki in Japanese rules. In Chinese rules, the points surrounded by a single player such as $a1$ are counted for that player.

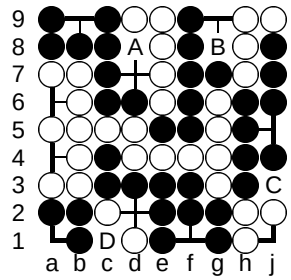

Fig. 7. Local seki collapse in global search.

Figure 7 illustrates how local seki can collapse globally. The computation proceeds as follows:

1. In the first round, as in the previous example, assume Black regions are processed first in the order A, B, C and D. Region D is proved a local seki and blocks $b1, g1$

are added to L. For White, region A is not a local seki because Black can capture White by playing at $d8$. Therefore White $d9$ is not added to L. Regions B, C and the merged region at $b1, e1$ are also not local seki. After the first round, L contains only the two Black blocks $b1$ and $e1$.

2. Using the "at least seki" block at $g1$ Black region C is a local seki, and $j8$ is added to L. There is no progress for White.
3. Similarly, using the "at least seki" block $j8$, Black region B becomes a local seki, $f9$ is added to L, and there is no progress for White.
4. Using the "at least seki" block at $f9$, Black region A is proven to be safe! Region A and block $f9$ are updated to safe and $f9$ is removed from L. Similarly, using safe block $f9$ and "at least seki" block at $j8$ region B is proven safe, and block $j8$ becomes safe and is removed from L. In the same way, Black regions C and D are proven to be safe, blocks $e1$ and $b1$ are safe, and they are removed from L. All Black regions A, B, C, D are safe and L is empty. Since no further progress can be made for either Black or White, the main loop terminates. All 81 points on this board are proven to be safe for one of the players.

6 Experimental Results

Two seki test sets were created by combining examples from several resources, including [14, 17], and positions from professional games. Test Set 1 is used for local seki testing and Test Set 2 for global seki testing. Both sets contain a mix of easy, moderate and hard problems. They are available at: http://www.cs.ualberta.ca/games/go/seki. All experiments were performed on a Pentium IV/1.6GHz machine with 512 Mb memory.

6.1 Experiment 1: Local Seki Tests

Test Set 1 contains 45 positions, some of them seki and some not when external liberties are considered. Among them, 23 positions are classified as easy, 16 positions as moderate and 6 positions as hard. The seki-enhanced safety solver solves 42 positions within a time limit of 120 seconds per position.

Table 1 summarizes the cost of seki re-search on all 45 positions. The total execution time for phase two search is 14.5% of phase one, and the total number of nodes expanded in phase two is 17.7% of phase one. The cost is relatively small compared to the version that does not detect seki.

Phase One Search		Phase Two Search	
Total nodes expanded	Total time (sec)	Total nodes expanded	Total time (sec)
1,101,733	403.12	195,039	58.47

Table 1. Overhead of seki re-search.

Figure 1 showed two easy positions from Test Set 1. Figure 8 provides another four positions with different difficulty levels. The Black regions in the top corners are two

moderate cases, solved in 16 seconds (left) and 31 seconds (right) total execution time for the two phases. The bottom right corner is a hard case, solved in 116 seconds. In these examples, seki is reached dynamically through a sequence of moves from the starting position. The Black region in the bottom left corner is safe with best defense. The attacker cannot achieve a seki. This hard case is solved in 44 seconds. Strongest move sequences for both players are shown in Figure 8.

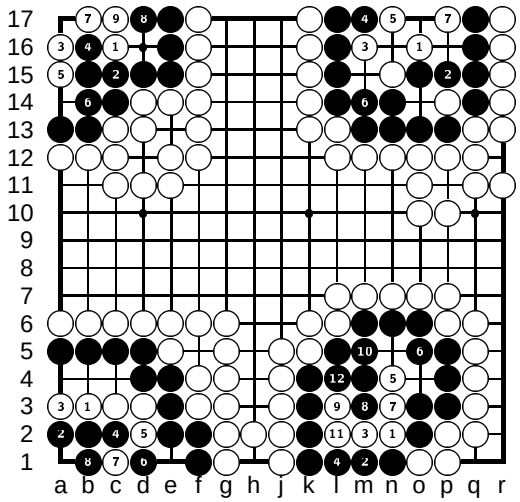

Fig. 8. Four examples from Test Set 1. White to play.

Sometimes seki depends on ko. It is important to model ko in the search of seki. The main technical limitation of the current solver is a class of ko positions called *Moonshine Life*, described at http://senseis.xmp.net/?MoonshineLife. The current solver defines the ko winner to be the attacker, and allows the ko winner get unlimited ko-threats to win the ko. In these situations, of which there are three in the test set, the solver produces a doubtful result. Figure 9 shows an example. Since White is considered to have infinite ko threats, there is no way for Black to capture White. So block △ is identified as "at least seki" and △ is identified as "at most seki" in the local search. If no update can be made after the global search, both of them will be identified as safe by seki. This result is dubious because in a real game ko threats are only unlimited when there is a multiple ko elsewhere on the board. In this example, static rules can be added to solve the problem. However, a more general solution that takes a finite number of available ko threats into account remains as future work.

The correctness of global search is based on the correctness of local searches. Still, even if Moonshine Life occurs in a local search, it will not invalidate the result of the global search. The reason is that only "at least seki" blocks are used to update other regions in the global search. In the example, only block △ is marked as "at least seki" and it can safely be used when searching other regions.

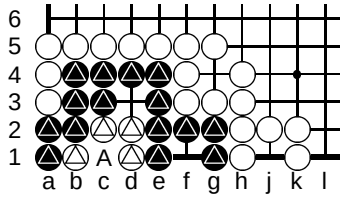

Fig. 9. An unsolved position.

A comparison is made between the safety solver in the program *Explorer* and one of the strongest programs, GNU Go 3.6, available at http://www.gnu.org/software/gnugo/download.html. Table 2 shows the number of test positions that are solved correctly by each program.

Program	Easy (total 23)	Moderate (total 16)	Hard (total 6)	Solved / Total
Explorer	20	16	6	93.3%
GNU Go	19	7	0	57.8%

Table 2. Comparison with *GNU Go* in Test Set 1.

The seki-enhanced safety solver solved 42 positions with correct results while *GNU Go* solved 26. Both *Explorer* and *GNU Go* can solve most easy positions. The safety solver in *Explorer* does not use any static rules to recognize seki, while it seems that *GNU Go* use mainly static rules. *GNU Go* solves less than half of the moderate positions and none of the hard positions.

6.2 Experiment 2: Global Seki Tests

Test Set 2 contains 20 global seki problems. In each of these problem, the results of local seki search needs to be resolved on the global level. 19 of them are solved within a time limit of 200 seconds. *GNU Go* was not tested on these global positions because the correctness of global search is based on the correctness of local search. Two full-board positions shown in Figure 6 and Figure 7 have been used in Section 5.6 to illustrate the global processing steps. The total execution time for these examples was 192 and 121 seconds respectively.

Figure 10 shows the only unsolved global position in Test Set 2. Currently there is no semeai search used to compute how many liberties the white blocks can get from the big eyes in regions A and C, and how many liberties the black block can get from region B. By using the most pessimistic assumption for the defender (White), the solver believes that the blocks $e5$ and $m5$ have only 2 internal liberties and one external liberty each (one liberty per eye). Therefore it cannot solve this problem. In this example, seki can be confirmed only if the semeai status of several related regions is resolved first.

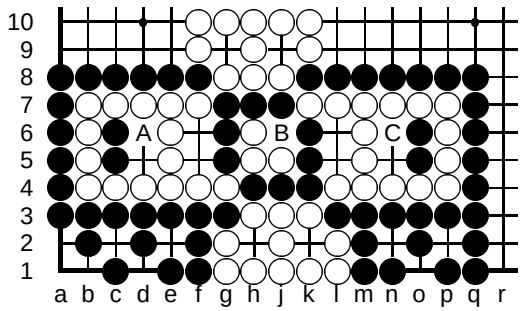

Fig. 10. Unsolved seki example.

7 Conclusions and Future Work

This paper presents two methods for recognizing seki positions locally and globally. Although the results are very encouraging, there are still numerous ideas to improve the performance. The limitations of the current solver and other possible topics for future work include:

- Handling of ko situations. Instead of allowing the ko winner unlimited ko threats, handling a finite number of threats must be implemented.
- The current solver is purely search-based. Static rules could improve the efficiency in many simple cases.
- The solver is region-based, and does not work for more open-ended areas that occur in games.
- Integrate the functionality of tactical solvers and semeai search into a safety solver.
- In the current global method, there is no priority for selecting which region to process in the main loop of the global search. Ordering regions by considering their adjacency should increase the efficiency of the global search.
- Seki spanning multiple regions such as in Figure 10 need a different approach.
- When to call the safety solver to recognize seki during a game?

References

1. L. V. Allis, M. Meulen, and H. J. Herik. Proof-number search. *Artificial Intelligence*, 66(1):91–124, 1994.
2. D. B. Benson. Life in the game of Go. *Information Sciences*, 10:17–29, 1976. Reprinted in Computer Games, Levy, D.N.L. (Editor), Vol. II, pp. 203-213, Springer Verlag, New York 1988.
3. M. Campbell. The graph-history interaction: On ignoring position history. In *Association for Computing Machinery Annual Conference*, pages 278–280, 1985.
4. Ken Chen and Zhixing Chen. Static analysis of life and death in the game of Go. *Information Science*, 121:113–134, 1999.
5. A. Kishimoto. *Correct and Efficient Search Algorithms in the Presence of Repetitions*. PhD thesis, Department of Computing Science, University of Alberta, 2005.

6. A. Kishimoto and M. Müller. Df-pn in Go: Application to the one-eye problem. In *Advances in Computer Games. Many Games, Many Challenges*, pages 125–141. Kluwer Academic Publishers, 2003.
7. A. Kishimoto and M. Müller. A general solution to the graph history interaction problem. In *19th National Conference on Artificial Intelligence (AAAI'04)*, pages 644–649. AAAI Press, 2004.
8. M. Müller. *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory*. PhD thesis, ETH Zürich, 1995. Diss. ETH Nr. 11.006.
9. M. Müller. Playing it safe: Recognizing secure territories in computer Go by using static rules and search. In H. Matsubara, editor, *Game Programming Workshop in Japan '97*, pages 80–86, Computer Shogi Association, Tokyo, Japan, 1997.
10. M. Müller. Race to capture: Analyzing semeai in Go. In *Game Programming Workshop in Japan, Vol.99(14) of IPSJ Symposium Series*, pages 61–68, 1999.
11. A. Nagai. *Df-pn Algorithm for Searching AND/OR Trees and Its Applications*. PhD thesis, Department of Information Science, University of Tokyo, 2002.
12. Xiaozhen Niu and Martin Müller. An improved safety solver for computer Go, 2004. To appear in Springer Verlag Lecture Notes in Computer Science (LNCS).
13. A. J. Palay. *Searching with Probabilities*. PhD thesis, Carnegie Mellon University, 1983.
14. ShiYu Tao. *Guan Zi Pu*. 1689. Reprinted in Wei Qi Ji Qiao Da Quan, Jiang MingJiu, Jiang ZhuJiu (Editors), Shu Rong Qi Yi Chu Ban She, Cheng Du, China, 1996.
15. R. Vilà and T. Cazenave. When one eye is sufficient: a static classification. In J. van den Herik, H. Iida, and E. Heinz, editors, *Advances in Computer Games 10*, pages 109 – 124. Kluwer, 2004.
16. E. van der Werf, J. van den Herik, and J. W. H. M. Uiterwijk. Learning to score final positions in the game of Go. In J. van den Herik, H. Iida, and E. Heinz, editors, *Advances in Computer Games 10*, pages 143 – 158. Kluwer, 2004.
17. Eric van der Werf. *AI techniques for the game of Go*. PhD thesis, University of Maastricht, 2005.