Memory Efficient Algorithms and Implementations for Solving Small-board-sized Go

Hung-Cheng Lin

National Taiwan University

Advisor: Chih-Wen Hsueh, Ph.D.
Coadvisor: Tsan-sheng Hsu, Ph.D.

Motivation

- Determine all possible states for a small rectangular Go board
- Determine the fair Komi and opening
- Huge number of state information is needed to keep
- Memory-efficient method with acceptable performance is required

Previous Work: Small-board-sized Go

- Alpha-Beta search with optimization to weakly solve small rectangular board with intersections less than 30[1]
- Using Meta-MonteCarlo-Tree-Search to build a huge opening book for 7×7 Go, and can defeat professional Go players[2]
- Variation of small-board-sized Go
  1. Solve 5×5 Atari-Go: the winner is the player that first captures the stone(s), and playing pass is prohibited[3]
  2. Determine 7×7 kill-all Go opening positions: Black plays two stones first, and White wins if there's a white live string, Black wins if there's no legal move for both players[4]
- Legal states for square Go boards are calculated for size up to 17×17 and give the boundary of the legal state count for 19×19 Go[5]

---

[1] van der Werf, et al., 2009
[2] Chou, et al., 2011
[3] van der Werf, et al., 2002
[4] Chang, Wei and Wu, 2016
[5] Tromp and Farnebäck, 2006

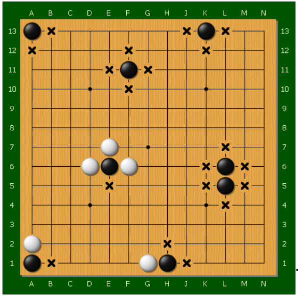
Related Work: Retrograde Analysis

- Retrograde analysis is widely used for searching endgames in chess-like game programming, such as Chinese chess and shogi
- Or solve the full games when the state-space complexity of the game is small, like awari
- Previous study [6] categorizes four types of retrograde analysis algorithms by their implementations
- This study use the fourth method, which is the refined version that reduce propagate count

[6] Ping-hsun Wu, Ping-Yi Liu, and Tsan-sheng Hsu. An external-memory retrograde analysis algorithm. 2004.

Definition

- $R \times C$ Go: R horizontal lines and C vertical lines on the board
  - $R \times C$ intersections
- String: connected set of stones in the same color
- Liberty of a String: the number of connected empty intersections



[7]

[7] https://upload.wikimedia.org/wikipedia/commons/3/39/Weiqi_qi.png?1530850936595

Go Rules

- There are many different Go rules, mainly different between rules of Ko and scoring
- General Go Rule:
  1. The black player plays first
  2. Black and white players place stones with the corresponding color in order
  3. If the opponent's string is out of liberty, it is captured and moved off the board
  4. A player cannot play a Ko move
  5. A player cannot play a suicidal move, which is the move that makes his or her string's liberty become 0, unless this move involves the capture of an opponent's string
  6. A player can pass a move. If two players pass continuously, the game is ended and the score is calculated

Ko Rules

- Definition
  - board position: positions of stones on the board
  - board configuration: board position and player's turn
- Ko is the rule that prevents a loop in the game
- Some commonly used Ko rules are:
  1. **Basic Ko**
     - Prevent the move that recreates the position from two moves before, but allows a longer cycle
  2. Positional Superko
     - Prevent the repeat of a board position
  3. Situational Superko
     - Prevent the repeat of a board configuration

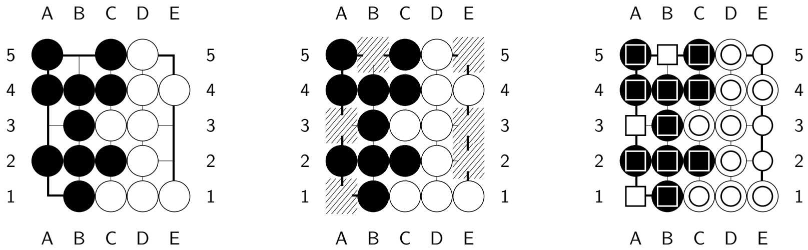
Scoring Rules

The two main scoring methods involve area scoring and territory scoring:

1. **Area Scoring**
   - The player's score is the number of empty intersections only his color surround and the number of its stones on the board
   - Used in Chinese rule
   - Easier to implement in computer Go
2. Territory Scoring
   - Dead stones and stones that are captured are looked as another color's player's prisoners
   - The player's score is calculated in terms of his or her territory and the number of prisoners
   - Territory is the empty intersections that are controlled by one color
   - Used in Japanese rule and Korean rule

Scoring Rules

- In general, area scoring and territory scoring give the same result or one or two points difference
  - If there's no stone captured in the game
  - Territory scoring : draw (Black 3 points, White 3 points)
  - Area scoring : Black win one point (Black 13 points, White 12 points)



Problem Definition

The Rules that are used for this study:

1. Basic Ko
    - When the game falls into a loop, the result of the game is considered to be draw
2. Area scoring
3. No komi

Goal

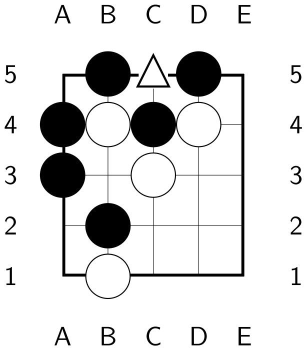
- Strongly solved small-board-sized Go
- For every posible state, get
  - the game result (which player wins the game)
  - the score (the difference in points between the winner and the loser)

Feature	Maximum Different Values	Bit Used	Description
Board Position	$3^{R \times C}$	40	
Ko and Pass	$R \times C + 3$	5	Pass is 0 with all intersections, pass is 1 with Ko is 0, pass is 2 with Ko is 0
Turn	2	0	Compressed
Degree	$R \times C + 1$	5	All possible move in the board and pass
Game Result	4	2	(black)win, draw, lose, and undetermined
Score	$2R \times C + 2$	6	$-R \times C$ to $R \times C$, and undetermined

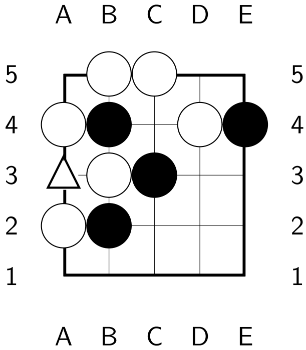
State Compression

- Only consider legal states
- Terminal state: state that the result can be directly determined
  - Not save in the memory because the result can be calculated when it is needed
- States with symmetrical board configuration
  - Have the same result and score
  - Can be compressed into one state

State Compression



White's turn



Black's turn

State Compression in 4×5

Property	Value
Possible States	$3^{20} \times 21 \times 3 = 219,667,417,263$
Legal States	$1,840,058,693$
Legal States Ratio in Possible States	0.837%
Compressed States	$460,114,319$
Compressed States Ratio in Legal States	25.01%

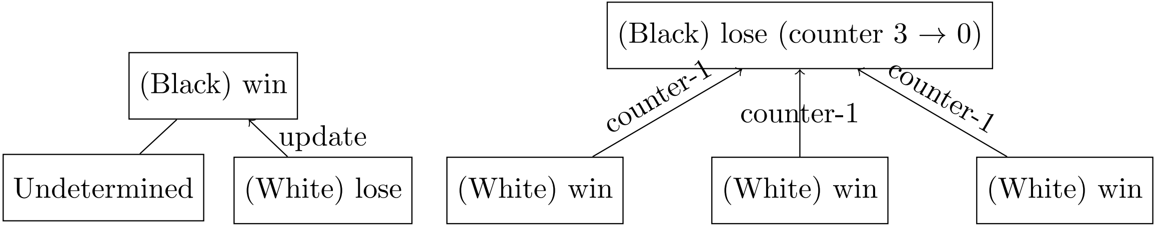
Search Algorithm

1. Preprocessing
2. Search Game Result
3. Search Score
4. Save Search Result
5. Validation

Search Game Result

Retrograde Analysis Algorithm

- All states are undetermined except terminal states
- If a state is not undetermined, propagate to its previous states
- remaining states are draw

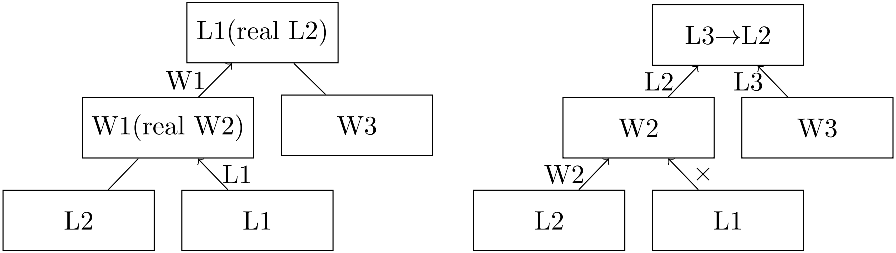


Search Game Result

- For every state S, all of its next states S' are propagated to S at most once
- There is a positive correlation between edge counts and search time

Search Score

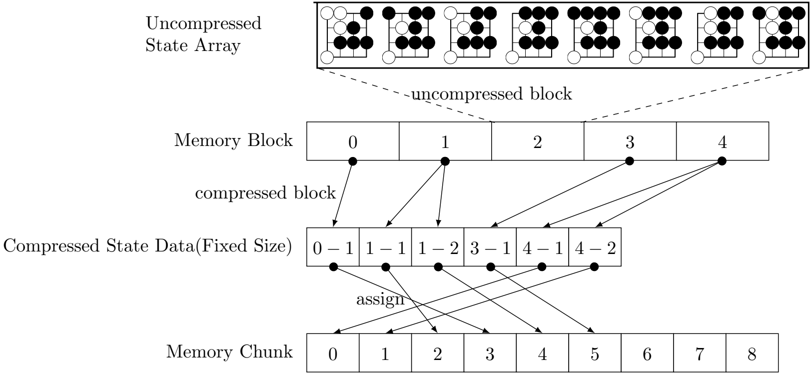
- Retrograde analysis is repeated to search for the states in decreasing order of the absolute value of the score
  - Let $R \times C = N$
  - Retrograde Analysis(win N and lose N) $\rightarrow$ Retrograde Analysis(win $N-1$ and lose $N-1$) $\rightarrow \cdots\cdots$
  - In this order, for every state S, all of its next states S' are propagated to S at most once



Save Result into File

- For each state, save all possible next states' result
  - A state has at most $R \times C + 1$ next states, including pass
  - Each score is saved using 5 bits
  - The size of the result file is $5 \times N \times (R \times C + 1)$ bits
- When there is a query
  1. The state is transformed to the compressed state
  2. Get index of the state for the legal state array
  3. Access result from the result file using the index

- Divide the legal state array into blocks, we called it memory blocks
- Use zlib library to compress memory blocks
- Memory block is compressed if they are not in use



Memory Issue

- A state is represented by its index in the legal state array
- Flags are saved in the memory as Boolean arrays to allow quick access
  - States are already searched or not
  - States that are currently visited
- The size of uncompressed state number is flexiable to adjust, in order to reduce memory block compression and decompression counts

Performance Issue

Time-consuming operations

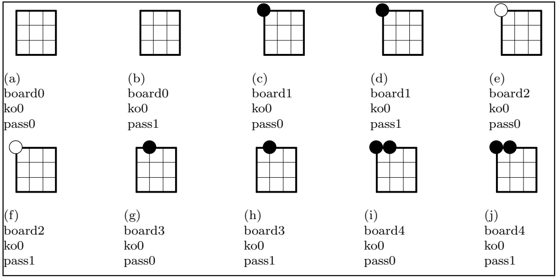
- If the state is in the compressed memory block, it must be decompressed before it can be used
- Obtain the index of a state in the state array
- Find previous states in memory blocks

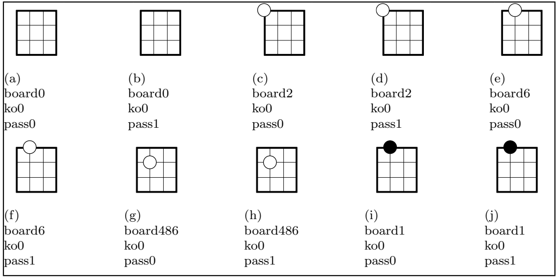
Performance Consideration - Memory Block Size

- If the memory block is small
  - Less states would be compressed and decompressed during an iteration
- If the memory block is large
  - Greater compression ratio
- Optimize the search process
  1. Determine maximum number of uncompressed states
     - Can not exceed the memory limit
  2. Determine memory block size
     - To minimize the running time

Sort Order

1. Serial Order
   - Sort order: board position $>$ Ko position $>$ pass count
2. Piece Order
   - Sort order: total number of stones on the board $>$ number of black stones on the board $>$ board position $>$ Ko position $>$ pass count

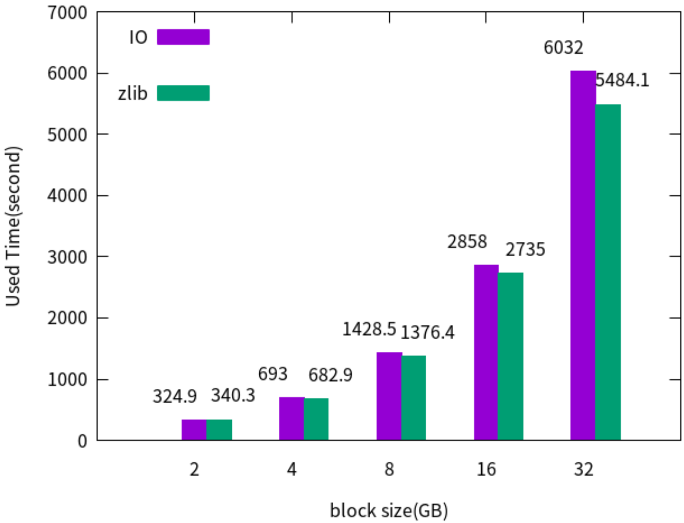




Performance Consideration - Sort Order

- Piece ordering achieves better data locality
- But much more time is required for sorting
  - The additional time to calculate the number of stones, compared to serial order

Performance Consideration - Data Saving Method



Result

- The biggest board searched is 2×11
- Total 6,941,794,698 states, use about 80 GB memory

Size	Depth	Compressed State Number	Time	Best Result	Best First Move
1×1	–	–	–	*draw*	pass
1×2	–	–	–	*draw*	pass
1×3	–	–	–	$B+03$	B1
1×4	–	–	–	$B+04$	B1
$1 \times n$ $n \leq 20$ and $n \geq 5$	–	–	–	*draw*	–

Result

Size	Depth	Compressed State Number	Time	Best Result	Best First Move
2×2	2	26	0.164 second	*draw*	A1
2×3	11	293	0.167 second	*draw*	B1
2×4	18	$2,169$	0.234 second	$B + 08$	B1
2×5	30	$18,205$	0.791 second	$B + 10$	C1
2×6	32	$152,887$	3.944 second	$B + 12$	C1
2×7	35	$1,304,472$	1.1 minute	$B + 14$	D1
2×8	41	$11,122,653$	10.61 minute	$B + 16$	D1
2×9	49	$141,646,333$	107.43 minute	$B + 18$	E1
2×10	54	$1,206,719,025$	18.0 hour	$B + 04$	E1
2×11	63	$6,941,794,698$	32.6 day	$B + 06$	F1

Result

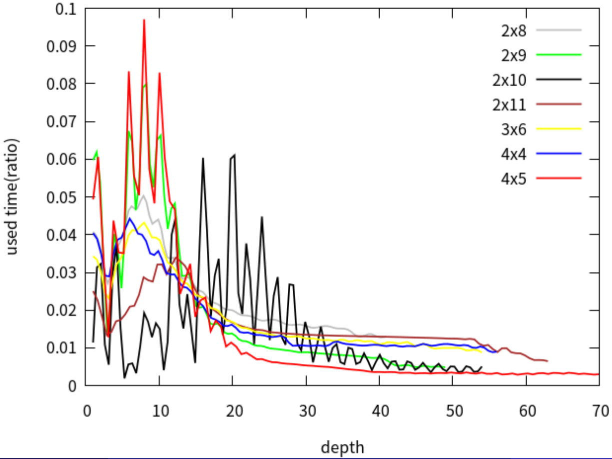
Size	Depth	Compressed State Number	Time	Best Result	Best First Move
3×3	26	$3,696$	0.462 second	$B+09$	B2
3×4	46	$166,358$	4.121 second	$B+04$	B2
3×5	46	$4,200,206$	3.8 minute	$B+15$	B2
3×6	54	$106,590,386$	152.2 minute	$B+18$	B3
3×7	59	$2,715,285,034$	2781.0 minute	$B+21$	B3
4×4	56	$9,276,006$	5.9 minute	$B+01$	B2
4×5	70	$1,402,761,648$	951.4 minute	$B+20$	C2

Strategy for 1 × n Go

```
Input  : A state S that is in S_w
Output: A legal move that can play at S which generates optimal result
Determine the position by the fact that board[2] = white, or board[2] ≠ black and
  board[3] ≠ black
firstblack ← minimum position that board[firstblack] = black
if firstblack not exist then
    firstblack ← n + 1
if firstblack > 3 and board[firstblack − 1] = white and board[firstblack − 2] = empty then
    // Step 1
    return firstblack-2
else
    firstempty ← first empty intersection from position 2 to position n which is legal
    if firstempty exist then
        // Step 2:  if there's no suitable move in step 1
        return firstempty
// Step 3:  if there's no suitable move in step 1 and 2
return pass move
```

Time and Search Depth

- Black can fully win and other boards have different distribution in time and search depth



Conclusion

- Retrograde analysis requires a vast amount of memory
  - Previous approach solves this problem by either using parallelism, storing on disk or advanced indexing method
  - We use in-memory method with state compression methods
- Refine the algorithm, change the sort order and memory block size to make performance acceptable

Future Work

- Small-board-sized Go may have rules to find
  - Optimal move generating method of $2 \times N$, $3 \times N \cdots$
- Better sorting criteria that can improve performance to access data

The End