논문 2007-44CI-1-7

컴퓨터 바둑에서 계가 알고리즘

(Score-Counting Algorithm for Computer Go)

박 현 수*

(Hyun Soo Park)

요 약

본 논문은 컴퓨터 바둑에서 계가 알고리즘의 방법을 제안한다. 제안된 방법은 안정도의 고려와 죽은 돌에 대한 처리 그리고 계가 알고리즘으로 이루어진다. 저자는 죽은 돌의 처리를 위한 방법과 빈 공배의 채움, 그리고 가일수에 대한 방법을 제안한다. 제안한 계가 알고리즘에서는 공배를 그룹의 영역에 포함되지 않은 빈 점으로 정의하고 가일수를 모든 공배를 채우는 과정에서 삶을 위해 강요되어지는 수로 정의한다. 362개의 종료된 게임을 사용하여 실험하였으며, 그 결과 CGoban, HandTalk과 제안한 방법에서 각각 8.66, 5.96 그리고 4.15의 평균 에러 값을 얻었다. 제안된 방법은 실험을 통해 종료된 게임에 성공적으로 적용됨을 검증한다.

Abstract

This paper presents a method of score counting for computer Go that includes the consideration of stability, management of dead stones, and an algorithm for score counting. Thus, method for managing dead stones, filling all dames, and making additional moves is presented, along with a score-counting algorithm, where dames are defined as empty points that are not included in the area of a group, while additional moves are required for life when filling all the dames. In experiments using the final positions of 362 games, a mean error of 8.66, 5.96, and 4.15 was recorded for the score counting produced by the CGoban, HandTalk, and proposed methods, respectively. The proposed method was confirmed by experiments where it was success fully applied to the final positions.

Keywords : score-counting algorithm, filling all dames, additional moves, final positions, computer Go

I. Introduction

Previous research on artificial intelligence has already included expert systems, written text and voice recognition, intelligence games, robots, and natural language processing. In particular, the intelligence game Go provides one of the biggest challenges for artificial intelligence, as the problem scope for Go is enormous at about 10^{170}. The artificial intelligence techniques related to computerized Go include evaluation functions, heuristic searches, machine learning, automatic knowledge generation, mathematical morphology, and cognitive science. However, this paper suggests a score-counting algorithm for computer Go in the case of games that have been played to completion, yet still require all the dames to be filled and additional moves.

D. Dyer[1] previously presented an algorithm for score counting, and tested it using 2000 games. However, the results only had 33% accuracy, where only 75% had the correct score, while the other 25% were usually off by 1 point due to a fine point in the endgame play. Sometimes, territories were not really final, just determined in the eyes of the pros. Nonetheless, gross errors, where tsume go was judged incorrectly and a wildly inaccurate score calculated, were very rare. In Dyer's algorithm, areas

* 정회원, 경동정보대학 컴퓨터정보기술과
(Dept. of Computer Information Technology, Kyungdong College of Techno-Information)
접수일자: 2006년 9월26일, 수정완료일: 2007년1월11일

are determined to be absolutely alive using a zero-entropy move generator that has the objective to fill all empty spaces, avoid capturing anything, and avoid changing connectivity among groups. Dyer also uses 'a database of eye-shape' outcomes to determine which "big eyes" can actually make two. Each not-absolutely-safe group is processed and classified as alive or dead based on several heuristics (such as "surrounded by a live group") or by invoking a problem solver to actually try to capture it. All the dames, defined as empty areas adjacent to live groups of both colors, are then filled, and the final phase looks for this occurrence by running a tsumego solver on all groups with few liberties.

Meanwhile, E.C.D. van der Werf[2] presented a learning system for scoring the final positions in a game of Go. The system is taught to predict life and death from labeled game records, as a result, 98.9% of the positions are scored correctly, and nearly all incorrectly scored positions are recognized without any human intervention. By providing reliable score information, the system provides a large source of Go knowledge implicitly available from human game records, thereby paving the way for the successful application of machine learning for Go. In the experiments, the game records were obtained from the NNGS archive (NNGS, 2002), and all the games were played on a 99 board between 1995 and 2002. Detecting these games is important because most machine-learning methods require reliable training data for a good performance.

Martin Miller[3] also devised an exchange method attached to unconditional safety, developed chain patterns, and extended the safety notion to the chains. He also heuristically evaluated the whole Go board using his program (EXPLORER). Miller's research focused on stability through the use of static rules and local searches in a game. His paper describes three exact evaluation methods for safe territories, capturing races using static rules, and endgame areas based on the combinatorial game theory used in EXPLORER, plus a zone-based heuristic position evaluation is explained. Zones are

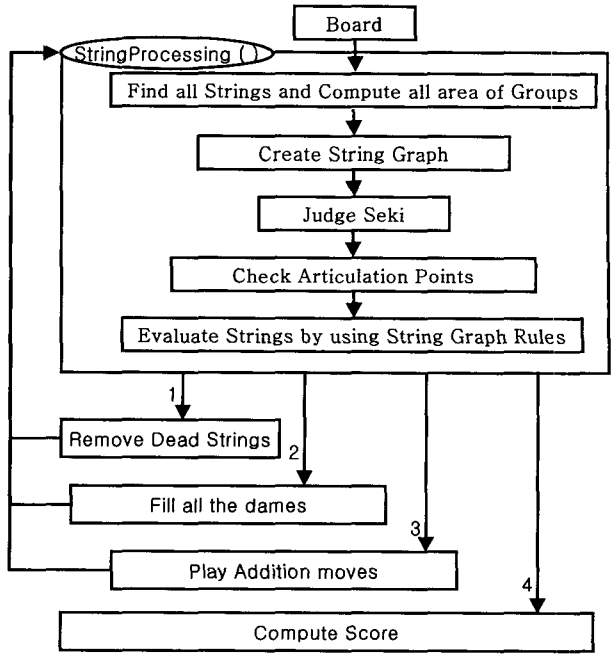


그림 1. 제안된 방법
Fig. 1. Suggested Block Diagram.

classified as safe territory, potential territory, threatened, or unused. Points outside the territories are further classified into near points, junction points, and far-away points. Each point in a safe territory counts as +1 for Black or -1 for White towards the total score. Meanwhile, each point in a potential territory counts as ±1/2.

Each near point is counted as ±0.2. Junction points and far-away points have a weight of zero, and do not contribute to the score. Therefore, the final score is the sum of the board score, plus the Komi (handicap).

However, this article presents an algorithm for score counting that includes string processing based on a string graph, the removal of dead strings, filling all the dames, and additional moves, as represented by the block diagram in Fig. 1. In the remainder of this paper, section 2 outlines the proposed algorithm and provides details on the string processing, removal of dead stones, filling all the dames, and additional moves. Section 3 then demonstrates the effectiveness of the proposed method through experiments and analysis. Finally, section 4 summarizes the proposed method, discusses some limitations, and suggests directions for future studies in this field.

Ⅱ. Score counting

1. String Processing

In this paper, the purpose of the string processing is to evaluate strings and compute all the group areas. This includes finding all the strings, computing all the group areas, creating a string graph, evaluating the strings according to the string graph rules, determining the existence of Seki (stalemate), and checking articulation points.

The stability of a string[4] is a numerical representation of the status of the strings on the board, i.e. whether the stability is close to life or death. Thus, to facilitate a static analysis[5], a string graph (SG) is created, then the life and death of the strings is judged according to specific rules. For example, the Articulation Point Check (APC) rule determines life and death according to the number of junctions, while Seki rules are applied to check the existence of Seki.

These steps are used after removing dead strings, filling all the dames, and making additional moves.

2. Processing Dead Stones

In this paper, the life and death of all strings are determined by string processing. The string and group information is updated after processing the dead stones, along with the stability of the strings and territory of the groups. Strings with a KJ-stability will be captured, so they are removed from the board. The captured stones are calculated by score counting, then the string and group information is updated again. Strings with stability of more than KI are also judged as dead and removed from the board. The captured stones are then calculated using the same method, and the string and group information updated again.

However, problems arise when Atari stones neighbor each other. Thus, a method for processing dead stones is proposed. The number of neighboring Atari stones is used to determine life or death. If the number of Atari stones is equal for both players, both are always judged dead. If one player has one stone and the other player has more than two stones, the latter is judged dead. The remaining neighboring Atari are also judged dead. In Fig.2, the dead stones are marked with white and black triangles.

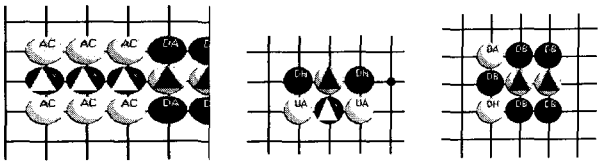


그림 2. 죽은 돌(흰색과 검은색 삼각형)들의 예
Fig. 2. Examples of dead stones, where white and black triangles represent dead stones.

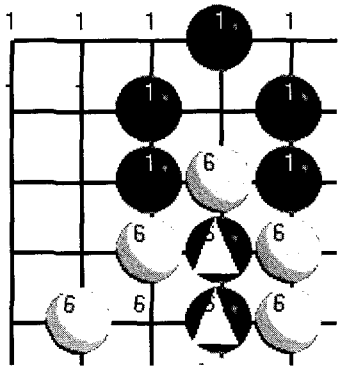


그림 3. 형세평가의 예
Fig. 3. Example of board evaluation.

The result of updating the string and group information is that the strings with a stability of more than KI are captured and calculated. Therefore, the black and white territory and captured stones are used to score the game.

In fig. 3, the stones marked by a triangle are dead, the numbers on the stones represent a group, and the numbers on the board represent the territory of the group. When the game in Figure 3 was evaluated by a professional player, the white area was 55.5: 40 on the left side, 10 in the top-right corner, and 5.5 Komi, while the black area was 41: 6 in the top-left corner, 15 on the right side, and 20 in the bottom-right corner, meaning that white won by 14. Figure 3 then shows the result of a static analysis of the game when using the proposed method, where the white area was 45.5: 31 on the left side, 9 in the top-right corner, and 5.5 Komi, while the black area was 30: 8 in the top-left corner, 16 in the bottom-right corner, and 6 on the right side, meaning that white won by

15.5. The difference between the two methods was related to the influence of the left side and top-right corner, which were regarded as territory by the professional player, yet not by the proposed method.

3. Dames and Additional moves

Counting the score in Go requires filling all dames and additional moves. Dames that are identified by searching area of group are not territory within a group. Thus, in the end game, each color places a stone alternately on an empty intersection in all the dames, and the additional move that occur when filling all the dames are not included as territory. Fig. 4 (a) shows a game where the dead strings need to be removed and the dames need to be filled, including additional moves. In Fig. 4 (b), the proposed method has identified all the dames on board, as marked by a black square.

Furthermore, some strings need additional moves to live, as they can Atari based on filling all the dames. Therefore , the additional moves are not included as territory, even if they were included as territory before the step. In fig. 4 (c), the proposed method has filled all the dames, including additional moves, as marked by a square on the stones.

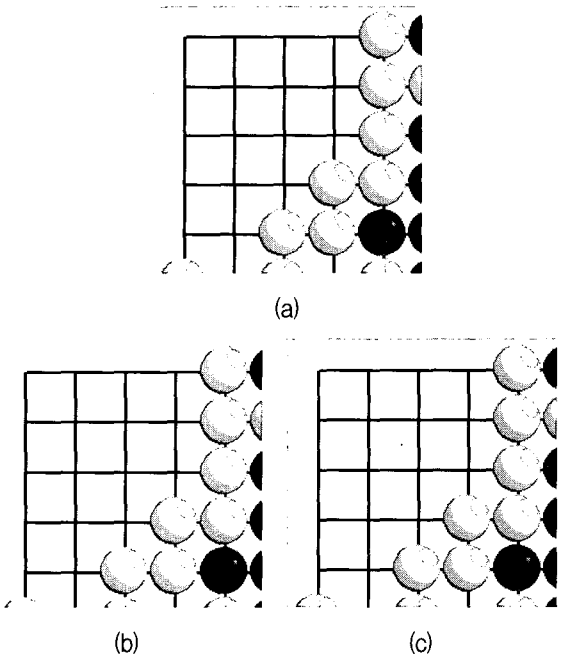


그림 4. (a) 게임의 예, (b) 모든 공배들, 검은색 사각형으로 표시, 그리고 (c) 가일수, 돌 위에 사각형으로 표시

Fig. 4. (a) Game, (b) all dames, as marked by black squares, and (c) additional moves, as marked by square on stones.

4. Score-counting algorithm

The proposed method counts the score using a score procedure, where the input is the BoardPoint and the Komi. The score procedure consists of StringProcess and RemoveDeadString. As a result, the score counting calculates the territory of black and white, the Komi, captured stones, and dead stones.

```
/* Procedure: StringProcess() */
1. Begin
2.  Evaluate the stability of S; //S is a String
3.  Call EvaluationUsingSG;
4. End
```

The StringProcess procedure evaluates the stability of the strings by classifying and evaluating the strings using a string graph[5], called the EvaluationUsingSG procedure, which classifies the string stability. The stability is not changed if the string stability is evaluated as Completely Alive (C).

Meanwhile, the RemoveDeadString procedure processes the dead strings on the board. If a string is Seki, it is alive. When the dead string is black, the removed_black value is calculated by adding the number of dead black stones, while the removed_white value is calculated by adding the number of dead white stones.

```
/* Procedure: RemoveDeadString () */
1. Begin
2.  if S is Seki then return; //S is a String.
3.  Insert S to DeadString; //DeadString has the number of Strings.
4.  Remove S on Board;
5.  if color of S is black then
6.     removed_black = removed_black + n(S);
7.  else
8.     removed_white = removed_white + n(S);
9. End
```

As such, the score procedure processes the position

and captured stone information on the board, where the StringProcess procedure evaluates the stability of the strings and completely alive strings using a string graph, then the strings are processed by the RemoveDeadString procedure if their stability is between KK and KB. Finally, all strings are re-evaluated by the StringProcess procedure.

```
/* Procedure: Score () */
1. Begin
2.  Call StringProcess;
3.  Remove strings that have extremely lower stability;
4.  Call RemoveDeadString;
5.  Call StringProcess;
6.  Find area of Groups;
7.  Call FillUpDames;
8.  Call StringProcess;
9.  Call MoveAdditionMoves;
10. Call StringProcess;
11. Compute score, white_score, and black_score;
12. End
```

The territory of a group is calculated based on the evaluation of its strings. Thereafter, the score counting processes the number of calculated territories, number of captured stones, and number of removed stones with a dead status.

III. Experiment

An experiment to evaluate the proposed score counting method was conducted using the BGA's collection at http://www.britgo.org/gopcres/gopcres1.html.

Although 883 problems by amateur Go players are included, faults were detected in some games, so the test data actually included 362 games.

Table 1 shows a comparison between the proposed method and other programs. The mean error for the proposed method was only 4.15, while the mean error for the CGoban and HandTalk methods was 8.66 and 5.96, respectively, confirming that the proposed method was very effective.

In Table 2, when the difference between the correct score and the score produced by each method was 0, CGoban produced 170, representing 47%,

표 1. 제안된 방법과 다른 프로그램간의 비교

Table 1. Comparison of proposed method and other program.

Items	Proposed Method	CGoban	HandTalk
Mean Error	4.15	8.66	5.96
Variance	110.4	422	118.7
Standard Deviation	10.5	20.5	10.9

표 2. 실험 결과. Diff는 정확한 계가결과와 다양한 방법들의 계가결과들 간의 차이, N은 hit된 개수, A는 hit 누적 개수.

Table 2. Experimental results. Diff is the difference between the correct score and the scores produced by the various methods, N is the number of hits, A is the aggregate number of hits.

Diff	Proposed method			CGoban			HandTalk		
	N	A	%	N	A	%	N	A	%
0	134	134	37	170	170	47	81	81	22.4
0.5	2	136	37.6	2	172	47.5	1	82	22.7
1	107	243	67.1	80	252	69.6	80	162	44.8
1.5	0	243	67.1	0	252	69.6	0	162	44.8
2	46	289	79.8	26	278	76.8	64	226	62.4
2.5	1	290	80.1	1	279	77.1	1	227	62.7
3	17	307	84.8	12	291	80.4	30	257	71
3.5	0	307	84.8	0	291	80.4	1	258	71.3
4	4	311	85.9	2	293	80.9	10	268	74
4.5	0	311	85.9	0	293	80.9	0	268	74
5	3	314	86.7	2	295	81.5	9	277	76.5
5.5	0	314	86.7	0	295	81.5	0	277	76.5
6	5	319	88.1	3	298	82.3	5	282	77.9
6.5	0	319	88.1	0	298	82.3	0	282	77.9
7	4	323	89.2	1	299	82.6	3	285	78.7
7.5	0	323	89.2	0	299	82.6	0	285	78.7
8	2	325	89.8	2	301	83.1	4	289	79.8
8.5	0	325	89.8	0	301	83.1	0	289	79.8
9	1	326	90.1	1	302	83.4	5	294	81.2
9.5	0	326	90.1	1	303	83.7	0	294	81.2
10	3	329	90.9	1	304	84	1	295	81.5
10.5	0	329	90.9	0	304	84	0	295	81.5
11	3	332	91.7	1	305	84.3	3	298	82.3
11.5	0	332	91.7	0	305	84.3	0	298	82.3
12 over	30	362	100	57	362	100	64	362	100
Total	362			362			362		

HandTalk produced 81, representing 22.4%, and the proposed method produced 134, representing 37%. When the difference between the correct score and the score produced by each method was 2 or less, CGoban produced 278, representing 76.8%, HandTalk produced 226, representing 62.4%, and the proposed method produced 289, representing 79.8%.

Although CGoban was better than the proposed method on correctness, the proposed method was superior when the aggregation of the difference between the correct score and the score produced by each method was 2 or less. Furthermore, when the difference between the correct score and the score produced by each method was 12 or over, CGoban produced 57, representing 15.7%, HandTalk produced 64, representing 17.7%, and the proposed method produced 30, representing 8.3%, thereby confirming the effectiveness of the proposed method.

The main problem of the proposed method is related to its reading strategies, which currently do not include strings broken by the enemy, unended gang fights, a stone supplement, and oi-otoshi. Thus, the proposed method is best used in the last stages of a game. However, it can also be used to minimize the depth of exploration when a localized judgment of life or death is made by an evaluation function.

As the proposed method is essentially a static evaluation, a search-based algorithm is still required for a few problems, such as an open-Ko, the reinforcement of one point, and a thrust.

IV. Conclusion

This paper proposed a method of score counting that considers stability, the management dead stones, filling all dames, and additional moves. In experiments using the final positions of 362 games, the mean error for the CGoban, HandTalk, and proposed methods was 8.66, 5.96, and 4.15, respectively.

In the case of neighboring Atari stones, their number is used to decide whether they are alive or dead. If the number of Atari stones is equal on both sides , both are always judged dead. If one side has one Atari stone and the other side has more than two, the latter is judged dead. The remaining neighboring Atari are also judged dead.

In addition, a method is presented for filling all dames and making additional moves, where dames are defined as empty points that cannot be included within the territory of a group, while some strings require additional moves to live, as they can Atari based on filling all the dames.

Score counting experiments were conducted using the BGA's collection. Although the problems included 883 games by amateur Go players, some faults were detected, therefore, the test data only included 362 games with completed scores.

The main limitation of the proposed method is related to its reading strategies, which currently do not include strings broken by the enemy, unended gang fights, a stone supplement, and oi-otoshi.

References

[1] http://www.andromeda.com/people/ddyer/go/scoring-games.html.

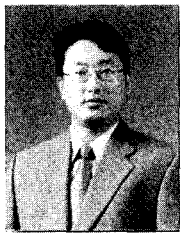
[2] J.W.H.M. Uiterwijk E.C.D. Vander Werf, H.J. van den Herik, "Learning to score final positions in game of Go," 10th Advances in Computer Games conference, pp. 143-158 2003.

[3] M. Muller, "Counting the score: Position evaluation in computer Go." *ICGA Journal*, 25(4): pp. 219-228, 2002.

[4] H. S. Park, D. H. Lee, H. J. Kim, "Static Analysis of String Stability and Group Territory in Computer Go," *Journal of the Institute of Electronics Engineers of Korea*, Vol. 40CI, no.6, pp. 77-86, Nov 2003.

[5] H. S. Park, K. W. Kang, "Evaluation of Strings in Computer Go Using Articulation Points check and Seki judgment," *Lecture Notes on Artificial Intelligence (LNAI 3809)*, Springer-Verlag, Berlin Heidelberg, pp. 197-206 2005.

저 자 소 개



박 현 수(정회원)

1992년 경성대학교 전산통계학과 이학사

1995년 경북대학교 컴퓨터공학과 공학석사

2005년 2월 경북대학교 컴퓨터공학과 박사.

1997년~현재 경동정보대학 컴퓨터정보기술과 전임강사.

<주관심분야 : 인공지능, 컴퓨터 바둑, 게임이론>