논문 2004-41CI-4-7

컴퓨터 바둑에서 String Graph를 사용한 정적분석

(Static Analysis In Computer Go By Using String Graph)

박 현 수*, 김 항 준**

(Hyun Soo Park and Hang Joon Kim)

요 약

본 논문은 정적 분석을 하기위해서 SG(String Graph)를 정의하고 ASG(Alive String Graph)를 정의한다. String의 사활의 판단을 위해 돌이 포함되지 않은 상태와 돌이 포함된 상태로 나누어 Rule을 적용한다. 돌이 포함되지 않은 상태에서 SR(String Reduction), ER(Empty Reduction), ET(Edge Transform), 그리고 CG(Circular Graph) Rule을 정의한다. 돌이 포함되어진 상태에서 DESR(Dead Enemy Strings Reduction)과 SCSR(Same Color String Reduction) Rule을 정의한다. 이러한 Rule을 사용하여 SG(String Grpah)가 ASG(Alive String Graph)인지를 평가한다. 그리고 관절점의 개수에 따라 사활을 판단하기 위해 APC(Articulation Point Check)를 사용하였다. 우리의 방법에 대한 성능은 Computer Go Test Collection의 IGS_31_counted 문제 집합에 대해 실험했다. 이 Test set은 11,191 Points와 1,123 Strings을 가진다. 우리는 실험 결과에서 Points에 대해 92.5% 정확성과 Strings에 대해 95.7%의 정확성을 얻었다.

Abstract

We define a SG(String Graph) and an ASG(Alive String Graph) to the purpose to do static analysis. For a life and death judgment, we apply the rule to the situation which the stone is included and not included. We define the rules that are SR(String Reduction), ER(Empty Reduction), ET(Edge Transform), and CG(Circular Graph), when the stone is not included. We define the rules that are DESR(Dead Enemy Strings Reduction) and SCSR(Same Color String Reduction, when the stone is included. We evaluate a SG that it is an ASG or not by using rules. And we use APC(Articulation Point Check) rule according to number of articulation points for a life and death judgment. The performance of our method has been tested on the problem set IGS_31_counted form the Computer Go Test Collection. The test set contains 11,191 Points and 1,123 Strings. We obtain 92.5% accuracy of Points and 95.7% accuracy of Strings.

Keywords : Static Analysis, String Graph, Alive String Graph, String Reduction, Circular Graph

I. 서 론

바둑은 흑과 백이 번갈아 두는 경기이다. 흑과 백은 19 X 19의 바둑판의 교차 되어지는 빈 공배에 번갈아 두어진다. 바둑의 최종적인 목표는 적보다 더 큰 영역을 차지하는 것이다. 바둑의 탐색 범위는 서로 다른 자리에 둘 수 있는 경우로 계산하면 $3^{19\times19} \fallingdotseq 10^{170}$으로 체스나 다른 게임보다 매우 크다. 그러므로 바둑 전체의 탐색 보다는 지역적 분할을 통한 탐색이 효율적이다. 사활은 지역적 분할의 대표적인 문제이다. 사활에서 스트링이 살았는지 혹은 죽었는지 결정하는 것은 사활탐색(Life and Death Search)에 의해 가능하다. 그리고 이러한 탐색에 덧붙여 매우 좋은 정적 분석(Static Analysis)이 가능하다면 탐색의 속도를 크게 향상 시키고 효율성을 높일 수 있다.[1] 또한 다음 수 선택을 위해서 형세판단이 필수적이다. 형세판단의 기본은 Strings와 Points의 영역의 정확한 정적분석에 있다.

David B. Benson[2]은 Strings에 대해 연속적으로 두는 적의 수에 대해서 잡히지 않는 무조건적 삶(Unconditionally Alive)을 정의하였다. Popma와

* 정회원, 경동정보대학 컴퓨터정보기술과
(Kyundong college of techno-information)
** 정회원, 경북대학교 컴퓨터공학과
(Dept. of Computer Eng. at Kyungpook National Univ.)
접수일자: 2004년3월19일, 수정완료일: 2004년6월30일

Allis[3]는 방어자가 X번 수 넘김을 해도 여전히 Group이 살아 있음을 'X life'라고 표기 하였다. Wolf[4]는 사활 문제를 트리 탐색을 사용하여 문제를 해결하고자 하였다. 그는 무제한의 깊이 우선 탐색을 사용하고 탐색 종료에 특정한 평가 함수를 사용하지 않는다. Wolf의 프로그램은 GoTools이며 35개의 다른 휴리스틱과 서브루틴을 사용하여 조금 더 일찍 삶과 죽음을 판단한다. Thore Graepel[5]은 machine learning의 관점에서 바둑을 고려하여 Common Fate Graph(CFG)를 사용하여 학습을 위한 위치 정보를 표현하였다. Muller[1]는 정적 룰과 로컬 탐색을 사용하는데 서로 번갈아 두는 조건하의 안정성에 대해 연구하였다. Muller는 아래와 같이 Benson의 Unconditionally Alive를 정의 하였으며 그의 논문에는 보다 확장한 개념들을 정의 하였다.

$$\forall\, b \in B \sum_{r \in R} SLC_{\cdots\cdots}(b, r, B) \geq 2 \qquad (1)$$

K .Chen과 Z. Chen은 패와 무관하게 그룹의 사활에 대해 정적 분석을 연구하였다.

이러한 연구에도 불구하고 아직까지 사활에 대해 정적 분석과 게임 트리를 이용한 방법들이 꾸준히 연구되고 있다[6].

본 논문은 SG(String Graph)를 사용한 String의 사활과 Points를 정적 분석하였다. 우리는 SG(String Graph)를 정의하고 ASG(Alive String Graph)를 정의한다. 기존의 정적 분석과의 차이는 String과 Group을 Graph 표현을 사용하여 삶의 상태를 정적 평가하였다. 우리는 실험에서 Points에 대해 92.5% 정확성과 Strings에 대해 95.7%의 정확성을 얻었다. 이 결과는 같은 실험 데이터를 사용한 Muller[1]의 Static Rule과 Search를 이용한 방법 보다 매우 좋은 효율을 보였다.

SG(String Graph)는 String과 Empty 그리고 그들 간의 관계를 표현한다. 우리는 SG(String Graph)를 사용하여 String의 직관적 삶을 표현하기위해 정적 평가를 통한 String의 사활 상태를 연구한다. 본 논문은 Section Ⅱ에서 SG(String Graph)에 대한 정의 및 표현에 대해 설명하고 또한 돌을 포함하지 않은 상태와 돌을 포함한 상태 그리고 관절점에 관한 Rule을 정의한다. Section Ⅲ에서는 실험 및 분석을 통해 제안한 방법의 효율성을 보인다. 그리고 Section Ⅳ에서는 결론적으로 제안한 방법의 요약과 문제점과 앞으로의 과제에 대해 논의한다.

Ⅱ. SG(String Graph)와 Rules

1. SG(String Graph)

String은 바둑 판에서 죽었을 때 제거 되어지는 기본적인 단위이다. 바둑 판에 놓인 돌들은 19 X 19 이차원 그리드의 구조를 가진 board graph로 표현되어진다. board graph는 undirected graph, $G_B = (V,E)$ $G_B \in G_U$ 로 정의된다. 대칭적 이진 에지 관계, $E = \{ e_1,\ldots,e_n\}$ with $e_i \in \{(v,v'): v, v' \in V\}$는 종적 횡적 교점의 관계를 표현한다. 우리는 아래에 정의 1에 G_B를 정의하였다.

정의 1. 그래프 GB는 아래와 같이 정의한다.

GB = (V,E)

V = { (i, j) | 1 ≤ i, j ≤ 19 },

E = { ((i, j), (i', j')) | (i = i' ∧ |j j'| = 1) ∨ (|i i'| = 1 ∧ j = j') },

f : V -> {black, white, Empty}

우리는 흑돌의 String을 BS vertex로 백돌의 String을 WS vertex로 표현한다. Empty의 덩어리를 ES vertex로 표현한다. String Graph, $G_{SG} = (V,E)$는 vertex 유한 집합 V과 edge 유한 집한 E로 구성된다. 각 edge는 vertex의 순서쌍으로 되어진다. 우리는 SG(String Graph)에서 vertices를 $v_0,v_1,v_2,\ldots$로 표현하고 edge를 $e_0,e_1,e_2,\ldots$로 표현한다. SG(String Grpah)는 directed와 undirected graph이다. undirected edge e_u는 두 vertex 사이에 이웃한 관계를 표현하고 directed edge e_d는 포함관계를 표현한다. 즉, String이 Empty points나 다른 String을 완전히 포함하고 있으면 directed edge로 표현가능하다.

정의 2. 임의의 vertex v_i와 v_j에 대해서 v_i가 완전히 v_j.를 포함하면 v_i는 v_j로 directed edge, $e_d = (v_i, v_j)$를 가진다.

우리는 black String을 vertex BS로 표현하고 white String을 vertex WS로 표현하고 Empty String을 vertex ES로 표현한다. 정의 2는 String이 완전히 다른 String을 포함하면 directed edge로 표현되어지는 것을 정의하고 있다. 그림 1의 왼쪽은 BS1에 의해 ES1이 완전히 포함되어진 것을 보여주며, 그림1의 오른쪽은 이것을 SG(String Graph)로 표현한 것으로 directed edge

그림 1. Directed Edge의 예
Fig. 1. Example of Directed Edge.

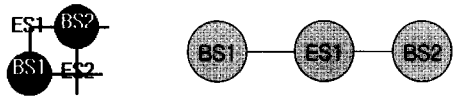
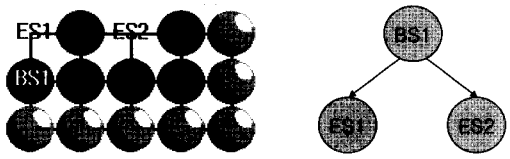


그림 2. Undirected Edge의 예
Fig. 2. Example of Undirected Edge.



(a) (b)

그림 3. ASG(Alive String Graph)의 기본
Fig. 3. Portotype of ASG(Alive String Graph).

로 표현되어진다.

정의 3. 임의의 vertex v_i와 v_j에 대해서 v_i가 v_j에 이웃하면 v_i와 v_j는 undirected edge, $e_u = (v_i, v_j)$를 가진다.

String과 Empty가 다른 String이나 Empty와 이웃하다면 undirected edge에 의해 표현된다. 그림2의 왼쪽 그림은 ES1이 BS1과 이웃한 상태이고 그림 오른쪽은 왼쪽 그림을 SG(String Graph)로 표현한 것이다. ES2도 실질적으로는 BS1과 BS2와 이웃한 관계에 있다.

우리는 크게 두 상태로 나누어 구분한다. 이 상태는 돌을 포함하지 않는 상태와 돌을 포함한 상태로 나누어지며 각각의 상황에 Rule을 정의하여 정적 분석을 하고자 한다.

가. 돌을 포함하지 않은 상태

돌을 포함하지 않는 상태는 기본적인 형태이다. 그림 3 (a)는 ES1과 ES2가 BS1에 완전히 포함되어진 것으로 삶이 보장된 String의 예이다. 이것은 무조건적 삶(Unconditionally Alive)이고, 또한 그림 3 (b)는 (a)의 SG(String Graph)이다. 그림 3은 삶의 프로토타입이다. 우리는 이러한 형태를 ASG(Alive String Graph)라고 정의한다. ASG는 적어도 두개의 directed edges를 가진다.

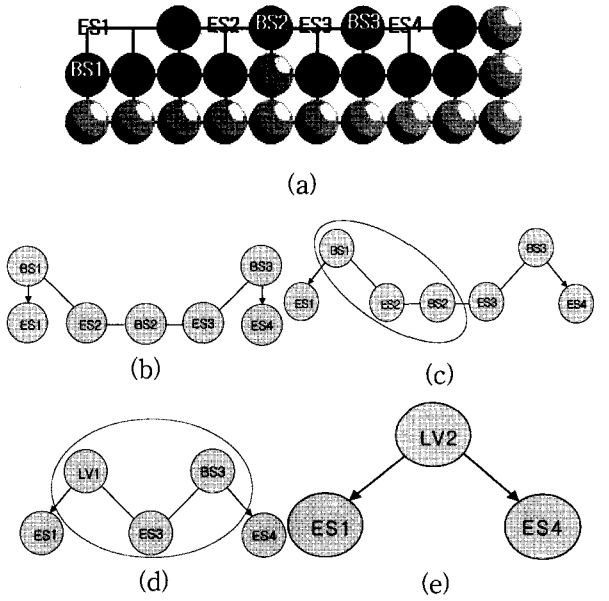


그림 4. String Reduction 의 예: (a) board graph (b) String Graph (c) (d) String Reduction의 과정 (e) Alive String Graph
Fig. 4. Example of String Reduction : (a) board graph (b) String Graph (c) (d) the process of String Reduction (e) Alive String Graph.

그림 3의 ASG(Alive String Graph)는 initial vertex는 BS1이고 terminal vertex는 ES1 그리고 ES2인 그래프이다. 이러한 ASG 그래프의 특성은 self-loop와 parallel edge가 없고 initial vertex는 항상 BS와 WS이고 terminal vertex는 ES와 BS 그리고 WS가 될 수 있다. initial vertex의 out degree는 적어도 2이고 내부의 Empty points의 수나 모양은 중요하지 않다.

Rule (SR: String Reduction). ES가 같은 색의 vertex 사이에 있다면 같은 색의 두 vertex와 ES는 새로운 vertex로 축약(Reduction)이 이루어진다. 이 새로운 vertex를 LV(Link Vertex)라 한다.

Group은 Empty points에 의해서도 완전한 삶을 보장받을 수 있다. 그림 4는 ES1은 BS1에 포함되어졌고, ES4는 BS3에 포함되어있다. 그러나 BS1은 ES1만으로 살지 못하고 BS3도 ES4만으로만 살지 못한다. ES2와 ES3가 그룹 삶에 중요한 역할을 한다. 그림 4 (c)에서 BS1과 ES2 그리고 BS2를 SR(String Reduction)하여 새로운 vertex인 LV1을 생성시킨다. 그리고 다시 LV1과 ES3 그리고 BS3가 LV2로 SR(String Reduction)이 되어진다. 마지막으로 그림 4 (e)에서 보여주는 것과 같이 ASG(Alive String Graph)가 되어 지므로 결론적으로 흑 Group은 '살아있다'라고 판명되어진다.

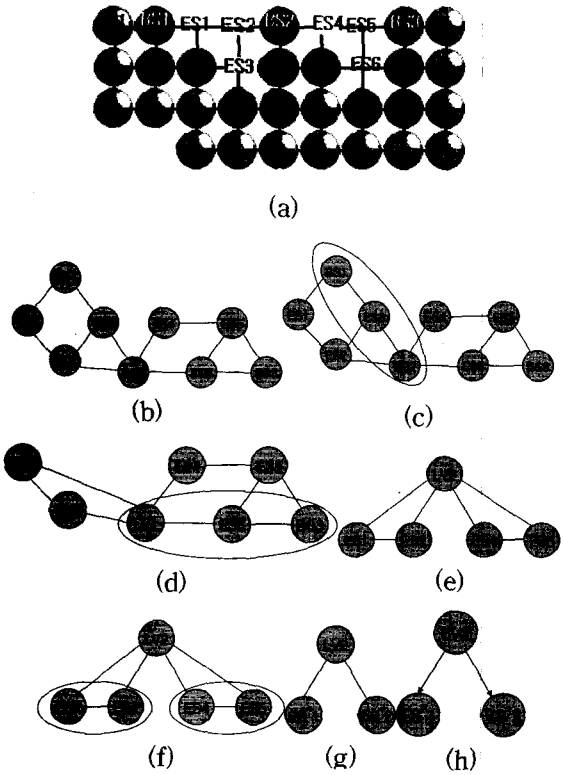


그림 5. Empty Reduction과 Edge Transform의 예 : (a) Board Graph (b) String Graph (c) (d) String Reduction의 과정 (e) String Reduction의 결과 (f) Empty Reduction의 과정 (g) Edge Transform 의 과정 (h) Alive String Graph

Fig. 5. Example of Empty Reduction and Edge Transform : Board Graph (b) String Graph (c) (d) the process of String Reduction (e) the result of String Reduction (f) the process of Empty Reduction (g) the precess of Edge Transform (h) Alive String Graph.

Rule (ER: Empty Reduction). SR(String Reduction)이 일어난 결과에서 이웃한 Empty vertex들은 새로운 Empty vertex로 축약되어진다.

Rule (ET: Edge Transform). ER(Empty Reduction)이 일어난 결과에서 만약 ES가 단지 단일 색 vertex에 undirected edge를 가지고 있다면, 이 undirected edge는 directed edge로 변환시킨다.

그림 5 (c)와 (d)에서, SR(String Reduction)이 일어나고, (f)에서는 ER(Empty Reduction)이 일어난다. 그리고 (g)에서 ET(Edge Transform)이 일어나고. 마지막으로 (h)에서 graph는 ASG(Alive String Graph)가 되어진다. Empty points(vertices)가 새로운 Empty vertex로 축약이 되면서 undirected edge가 directed edge로 변환되어 지는 것을 보여준다.

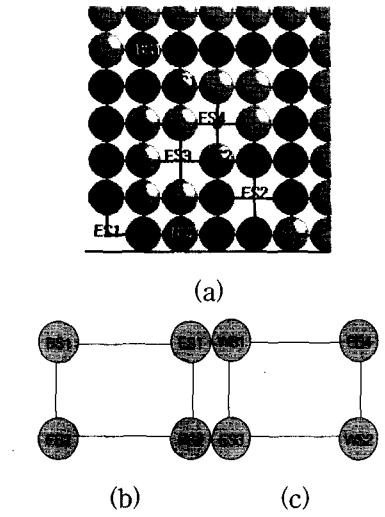


그림 6. Circular Graph의 예 : (a) board graph (b) (c) Circular Graph

Fig. 6. Example of Circular Graph : (a) board graph (b) (c) Circular Graph.

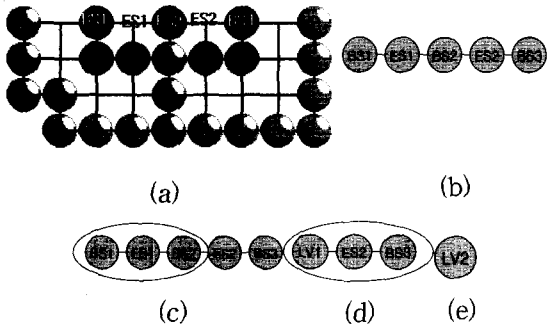


그림 7. 비 circular와 죽음 : (a) Board Graph (b) String Graph (c) (d) String Reduction의 과정 (e) String Reduction의 결과

Fig. 7. No circular and dead (a) Board Graph (b) String Graph (c) (d) String Reduction의 과정 (e) String Reduction의 결과.

Rule (CG: Circular Graph). Graph G_{SG}에서 vertex들과 undirected edges가 번갈아 있는 유한 시컨스, $v_0,e_1,v_1,e_2,..v_{k-1},e_k,v_k$ 가 walk이고 tail 그리고 closed이면, 이것을 Circular graph라 하며, 이것은 살아있는 상태이다.

False eye는 실제로 영역에 포함되지 않고 연결점으로 사용되어진다. 그림 6(a)에서 BS1과 BS2는 ES1과 ES2를 가지는 Group이다. WS1과 WS2는 ES3와 ES4를 가진 또 다른 Group이다. ES1과 ES2는 false eye 형태이므로 BS Group은 살아 있지 못하는 것 같지만 실제로는 완전히 살아있다. BS1과 BS2는 ES1과 ES2에 의해 순환적으로 연결되어있으며 ES1과 ES2는 적이 놓을 수 없는 자리이다.

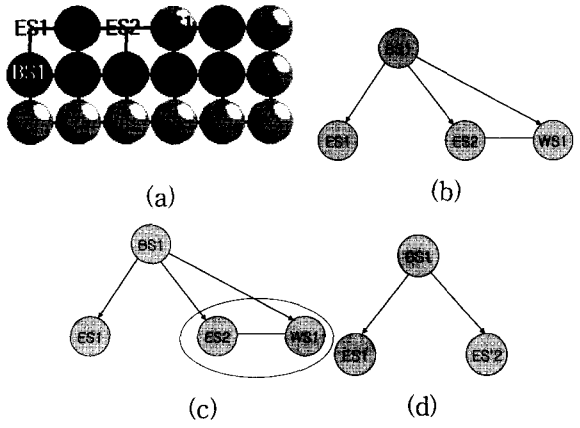


그림 8. 죽은 적 String 축약 : (a) Board String (b) String Graph (c) Dead Enemy Strings Reduction 의 과정 (d) Alive String Graph

Fig. 8. Dead Enemy Strings Reduction : (a) Board String (b) String Graph (c) the process of Dead Enemy Strings Reduction (d) Alive String Graph.

ES의 연속은 하나의 ES로 간주하여 처리한다. 그리고 ES의 연속되는 시컨스가 closed로 오인 되는 것을 피하였다. 그리고 self-loop가 없으므로 자신에 의한 순환과 parallel edge가 없으므로 reflex에 의한 순환도 제거한다는 의미이다.

그림 7에서, false eye인 ES1과 ES2는 circular가 아니므로 (c)와 (d)에서 SR(String Reduction)이 일어나며 결국 (e)로 되어진다. 결과적으로 이 graph는 ASG가 아니므로 alive가 아니다 라는 결과이다. BS1과 BS3는 백돌들에 의해 막혀있는 상황이며 흑 String은 단수로 몰릴 수밖에 없으며 백돌에 의해 죽임을 당하게 되어진다. 앞의 예 중에 그림 4에서의 다른 false eye는 결국 LV vertex가 되어지고 이 LV는 Group을 삶의 상태로 만들었다. LV와 circular false eye는 안전한 Group을 만들기 위해 매우 중요하다.

나. 돌들을 포함한 상태

Rule (DESR: Dead Enemy Strings Reduction). 만약 죽은 적 String이 포함되어있고 또한 포함되어진 Empty와 이웃하다면 이웃한 Empty와 죽은 적 String은 새로운 Empty string으로 축약되어진다.

그림 8 (d)에서, 흑 String인 BS1은 ASG이다. ES2와 WS1은 BS1에 포함되어져있고 각각 서로 이웃한 관계에 있다. ES1은 BS1에 따로 떨어져 포함되어있다. 그림 (b)에서 SG를 보여 주고 있으며 (c)에서 DESR(Dead Enemy Strings Reduction)이 일어나고 ES2와 WS1은 새로운 하나의 vertex, ES'2로 축약이 일어난다. 그리고

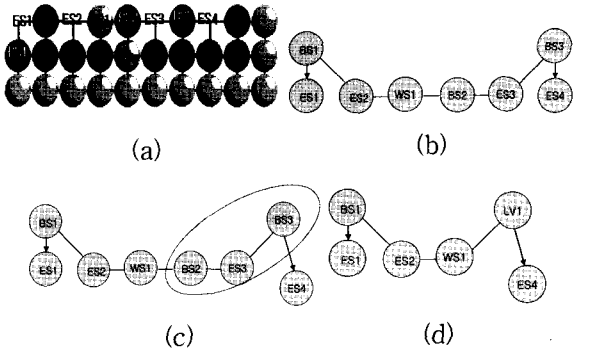


그림 9. 적 String을 가진 Group의 예 : (a) Board Graph (b) String Graph (c) String Reduction의 과정 (d) String Reduction의 결과

Fig. 9. Example of a Group with enemy String : (a) Board Graph (b) String Graph (c) the process of String Reduction (d) the result of String Reduction.

그림 (d)에서 결과 graph에서 ASG이다.

죽은 적 돌이 있는 다른 예를 그림 9에서 보여준다. BS1과 BS2는 WS1에 의해 끊어져있으며, WS1은 ES2와 이웃하다. ES1은 BS1에 포함되어있으므로 directed edge로 표현 되어진다. 또한 BS3는 ES4로 향한 directed edge를 가지고 있다. 그림 9 (c)에서 SR(String Reduction)이 일어나고 새로운 vertex LV1이 만들어진다. 그러나 WS1은 BS1에 포함이 되지 않았고 어떤 다른 축약이 일어나지 않는다. 결과적으로 BS1과 BS2 그리고 BS3 인 흑 Group은 ASG가 아니다. 그러므로 alive가 아니다. 이 예에서 아시다시피 현재 우리가 판단하는 것은 순번에 관계없이 현 상태에 대한 정적 판단이다. 그러므로 현재 상태에서 흑이 ES2에 돌을 놓고 따 낸다면 이것은 ASG로 변환 되어 질 것이다. 만약 백이 ES3에 놓고 따 낸다면 역시 ASG가 되지 못할 것이다.

Rule (SCSR: Same Color String Reduction). 임의의 String에 그 String과 같은 색의 String이 둘이상의 Empty String과 각각 이웃하다면 임의의 String에 아군 String은 축약이 되어진다. 그러므로 이 String은 ASG이다.

다른 경우에 대해서 알아보면, 임의의 String이 아군 String과 적 String을 가지고 있을 수 있다. 만약 아군 String과 둘 이상의 Empty String이 서로 각각 이웃하고 또한 모두 임의의 String에 포함되어진 상태이면 SCSR(Same Color String Reduction)이 일어난다. 그림 10 (b)는 SG(String Graph)이고 (c)에서 SR(String

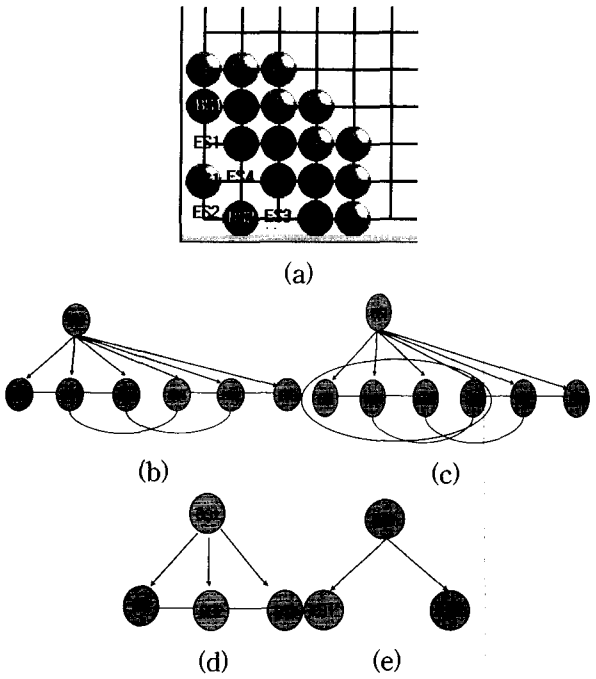


그림 10. SCSR(Same Color String Reduction)의 예: (a) Board Graph (b) String Graph (c) Same Color String Reduction의 과정 (d) Same Color String Reduction의 과정 (e) Alive String Graph

Fig. 10. Example of SCSR(Same Color String Reduction): (a) Board Graph (b) String Graph (c) the process of Same Color String Reduction (d) the process of Same Color String Reduction (e) Alive String Graph.

Reduction)이 일어나고 (d)에서 ES'과 BS2그리고 ES3가 각각 이웃하며 이것은 BS1에 포함되어있다. 그러므로 SCSR(Same Color String Reduction)이 발생이 되어 BS2는 BS1에 흡수 축약이 되어진다. 결국 그림(e)의 graph는 ASG(Alive String Graph)가 되어진다.

다. 관절점(Articulation Point)

Rule (APC: Articulation Point Check). 임의의 String에 포함된 내부 Empty String이 두개이상의 관절점(Articulation Point)을 가지면 이 String은 살아있다.

임의의 String에서 내부에 포함되어진 Empty의 관절점(Articulation Point)을 조사하여 2개 이상이면 무조건적으로 삶이 보장이 되어진다. 이러한 관절점은 눈을 만드는 자리로 1개가 있으면 그 자리에 적 돌이 놓이게 되면 죽게 되어지고 아군이 놓게 되면 삶이 보장이 되는 급소의 자리이다. 그러므로 내부에 포함되어진 Empty에 관절점이 몇 개가 존재하는 지는 매우 중요하다. 그림 11에서의 예에서처럼 ES2와 ES3가 관절점에

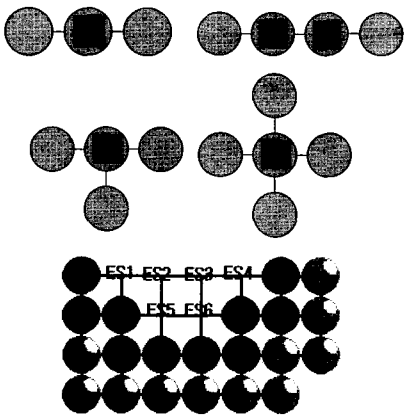


그림 11. 관절점(ES2와 ES3) 예

Fig. 11. Example of Articulation Points(ES2 and ES3).

표 1. 실험 결과 비교

Table 1. The comparison of experiment results.

Method	Points		Strings	
	Accuracy	%	Accuracy	%
Benson	1886	16	103	9
Muller Static	2481	22	168	14
Muller Search	2954	26	198	17
String Graph	10348	92.5	1075	95.7

해당이 되어진다. 여기에서는 두개가 있으므로 흑 String은 당연히 살아있다.

III. 실험 및 분석

우리는 앞에서의 정의와 Rule들을 이용하여 제안한 방법의 성능을 Computer Go Test Collection 중 IGS_31_counted problem (www.cs.ualberta.ca/~mmueller)을 이용하여 실험하였다. 이들 문제들은 아마추어 단급의 선수들에 의해 두어진 31개의 게임을 포함하고 있다. 이들 게임들은 완전히 두어졌으며 계가가 이루어진 게임들이다. 이 실험은 31개의 게임 마지막 형세에 대해 실험한다. 실험 데이터는 Points가 31 × 361 = 11,191개이고 Strings이 1,123개이다. 이 실험의 목표는 Strings의 안정도와 영역의 분류의 정확성을 보이기 위함이다. 공배에 대한 정확성은 Points가 포함함으로 제외시켰다.

표 1의 실험 결과 비교는 Muller의 결과[1]와 제안한 방법의 결과를 나타낸다.

제안한 방법은 Points는 92.5%의 분류 정확성 그리고 Strings은 95.7%의 분류 정확성을 나타내었다. 이러한 결과는 제안한 방법이 상당히 효율성이 있음 보여 주는 것이다.

아래의 표 2는 실험 기보 31개를 Points와 Strings에

표 2. 실험 결과
Table 2. The results of experiment of games.

게임 번호	Points		Strings	
	개수	Accuracy	개수	Accuracy
1	361	335	35	33
2	361	320	28	26
3	361	343	39	39
4	361	361	22	22
5	361	335	36	38
6	361	312	36	31
7	361	325	42	41
8	361	310	31	25
9	361	296	35	35
10	361	333	23	22
11	361	346	43	41
12	361	319	37	36
13	361	358	40	40
14	361	352	29	29
15	361	345	41	38
16	361	332	44	43
17	361	321	53	50
18	361	353	24	24
19	361	332	52	48
20	361	302	42	42
21	361	339	42	41
22	361	345	29	29
23	361	361	34	34
24	361	324	33	31
25	361	361	42	42
26	361	348	30	29
27	361	347	35	35
28	361	239	51	37
29	361	332	33	32
30	361	361	28	28
31	361	361	34	34
합계	11191	10348	1123	1075

대해 실험한 결과를 보여준다.

기보 4의 경우에. Muller의 방법에서는 Points 분류가 최고 좋은 결과로 92%를 내는데 반해 제안한 방법을 적용한 결과는 100%의 Points 분류가 되었다. 아래 그림 12는 기보 4에 제안한 방법을 적용한 예이다. 'CA'는 Strings의 안정도가 '살아있음'을 'KJ'는 Strin -gs의 안정도 '죽었음'을 표시한 것으로 Strings 위에 표시하였다. 그리고 바둑판위에 적힌 숫자는 속한 Group의 번호이다.

기보 9번은 Muller의 방법에서 최악의 경우로 0%의 Points 분류가 되어졌지만 제안한 방법의 경우는 82%

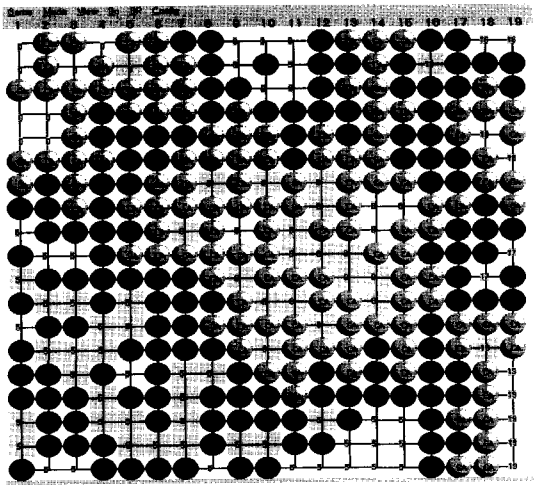


그림 12. 기보 4의 실험결과
Fig. 12. The result of the 4th problem.

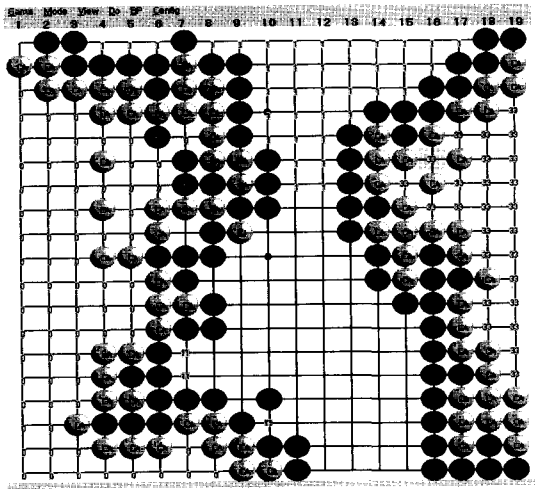


그림 13. 기보 9의 실험결과
Fig. 13. The result of the 9th problem.

의 양호한 결과가 나왔다.

그러나 제안한 방법의 문제점은 같은 색의 String과 String이 적 돌에 의해 끊어진 상황에서 같은 Group으로 포함이 되지 못하는 문제점이 발생한다. 이는 Points들의 영역 분류에 오류를 발생한다. 또한 영역이 정확하게 분류되지 못하므로 Strings의 포함 영역과 이웃 영역을 정확히 찾지 못해 Strings의 사활 오류가 발생한다. 이는 휴리스틱 안정도 정의에 의한 정적 분석과 통합하여 문제를 해결 할 수 있으므로 다음 연구 과제로 남겨둔다.

Ⅳ. 결 론

우리는 정적 분석을 하기위해서 SG(String Graph)를 정의하고 ASG(Alive String Graph)를 정의하였으며 그

성능을 실험하였다. String의 안정도 평가를 하여 삶과 죽음의 판단이 잘못되어진 개수를 구하였으며, Points의 분류로 어느 Group에 속한 영역인지를 판단하여 정확히 판단된 개수를 구하였다. 실험에 사용된 경기는 Computer Go Test Collection 중 IGS_31_counted 문제로 Points는 11,191개이고 Strings는 1,123개이다. Poinits 분류는 92.5%와 Strings 분류는 95.7%의 정확성을 결과로 얻었다. 이는 이전 논문들에 비해 매우 높은 정확성을 갖는다.

우리는 컴퓨터 바둑에서 String의 사활의 판단을 위해 먼저 돌이 포함되어있지 않은 상태에서 SG(String Graph)를 사용하여 SR(String Reduction)과 ER(Empty Reduction) 그리고 ET(Edge Transform)을 수행하였다. 또한 CG(Circular Graph)로 사활을 판단하였다. 그리고 돌이 포함되어진 상태는 적 돌이 죽은 상태일 경우 DESR(Dead Enemy Strings Reduction)으로 죽은 적돌과 Empty와 축약을 하고 아군이 포함 되어있는 상태이면 SCSR(Same Color String Reduction)으로 축약을 해서 ASG인지 아닌지를 판단하였다. 그리고 관절점의 개수에 따라 사활을 판단하기 위해 APC(Articulation Point Check)를 사용하였다.

제안한 방법의 문제점으로는 적 돌에 의해 끊어진 상태, 미결 패싸움, 한점 보강, 그리고 회도리 수 등이 아직 문제로 남아있다. 이러한 것은 모두 수 읽기문제와 연관이 되는 것이다.

제안한 방법은 휴리스틱 안정도 평가와 탐색트리와 통합하여 정확한 계가에 사용되어질 수 있으며 또한 종반의 끝내기에 이용되어질 수 있다. 그리고 부분적으로 사활 문제 풀이에 평가함수로 이용되어 탐색 깊이 최소화에 이용되어질 것으로 판단되어진다. 이러한 연구들은 앞으로의 과제이다.

참 고 문 헌

[1] Muller, M. Playing it safe: Recognizing secure territories in computer Go by using static rules and search. *Game Programming Workshop in Japan '97*(Ed. H. Matsubara), pp.80-86, Computer Shogi Association, Tokyo, Japan, 1997.

[2] Benson, D.B. *Life in the game of Go,* Information Sciences, Vol. 10, pp 17-29, ISSN 0020-0255. Reprinted in *Computer Games,*(Ed. D.N.L.Levy), Vol. II, pp. 203-213, Springer-Verlag, New York, N.Y. ISBN, 1976.

[3] Popma, R. and Allis, L.V. Life and death refined. *Heuristic Porgramming in Artificial Intelligence 3*(Eds. H.J. van den Herik and L.V.Allis), pp. 157-164. Ellis Horwood Ltd., Chichester, Englad. ISBN

[4] T.Wolf, Investigating Tsumego problems with RisiKo, inL D. Levy, D. Beal(Eds,), *Heuristic Programming in Artificial Intelligence 2*, pp. 153-160, Ellis Horwood, Chichester, 1991. *In International Conference on Artificial Neural Networks* (ICANN-01), Vienna, Austria, 2001.

[5] Thore Graepel, Mike Goutrie, Marco Krüger, and Ralf Herbrich. Learning on graphs in the game of Go.

[6] Muller, M. Computer Go. *Artificial Intelligence,* Vol. 134, Nos. 1-2,pp. 145-179. ISSN 0004-3702. 2002.

저 자 소 개

박 현 수(정회원)

1992년 경성대학교 전산통계학과 이학사.

1995년 경북대학교 컴퓨터공학과 공학석사.

1997년 경북대학교 컴퓨턱공학과 박사수료.

1997년~현재 경동정보대학 컴퓨터정보기술과 전임강사.

<주관심분야: 인공지능, 컴퓨터 바둑>

김 항 준(정회원)

1977년 서울대학교 전자공학과 공학사.

1979년 한국과학기술원 전산학과 공학석사.

1997년 Shizuoka University 공학박사.

1979년~현재 경북대학교 공과대학 컴퓨터공학과 교수.

<주관심분야 : Pattern Recognition, Image Understanding.>