한국컴퓨터정보학회 하계학술대회 논문집 제26권 제2호 (2018. 7)

게임 트리와 알파-베타 가지치기를 이용한 오목 프로그램의 설계 및 구현

이경호*, 한원근O
O*한라대학교 정보통신소프트웨어학과
e-mail: khlee@halla.ac.kr*, asdasfafsf@naver.comO

Design and Implementation of Omok Program Using Game-Tree and Alpha-Beta Pruning

Kyong-Ho Lee*, Won-keun HanO
O*Department of Information Communication Software, Halla University

● 요 약 ●

본 논문에서는 오목을 두는 지능적 프로그램을 설계하고 구현하였다. 규칙은 렌주 룰(renju rule)을 기준으로 하였으며, 15 x 15 게임 판에서 오목을 둔다. 초기에는 문제 분석을 통하여 분석된 가중치로 판단을 하여 판단을 하여 게임을 진행하도록 하였으나, 반복된 수행의 경험적 판단을 통하여 얻은 정보로 여러 차례 수정하며 고정된 가중치를 구성하고, 이 가중치를 게임에서 돌을 놓을 때 평가 기준으로 삼도록 하였으며, 최소-최대 게임 트리(min-max game tree)를 이용하여 상대가 있는 게임을 수행할 수 있도록 하였다. 또한 프로그램 자신에게 유리한 수를 찾기 위한 탐색에서 무의미한 노드들의 전개를 줄여 제한된 시간 안에 좋은 수를 찾을 수 있도록 알파 베타 가지치기(alpha-beta pruning)를 사용하도록 프로그램을 구현하였다.
이렇게 구현된 오목 프로그램은 게임을 본 프로그램과 게임 하기 원하는 주변의 일반인들에게 90% 이상의 승률을 보이고 있었다.

키워드: 오목 프로그램(Omok Program), 게임 트리(game tree), 알파 베타 프루닝(alpha-beta pruning)

I. Introduction

이세돌과의 바둑 경기에서 3:1로 완승을 거둔 알파고의 등장으로 인공지능은 더욱 더 많은 관심을 받게 되었다. 단순히 게임 인공지능을 넘어서, 음성으로 집안의 가전을 제어할 뿐만 아니라 사람과 대화를 하는 등, 먼 미래에는 인공 지능이 사람의 일을 대신 할 수 있겠다는 기대를 가지게 되었다.

본 논문에서는 오목을 두는 지능적 프로그램을 설계하고 구현하였다. 규칙은 렌주 룰(renju rule)을 기준으로 하였으며, 15 x 15 게임 판에서 오목을 둔다. 구현된 프로그램은 초기에는 문제 분석을 통하여 분석된 가중치로 판단을 하여 게임을 진행하도록 하였으나, 반복된 수행의 경험적 판단을 통하여 얻은 정보로 여러 차례 수정하며 고정된 가중치를 구성하고, 이 가중치를 게임에서 돌을 놓을 때 평가 기준으로 삼도록 하였으며, 최소-최대 게임 트리(min-max game tree)를 이용하여 상대가 있는 게임을 수행할 수 있도록 하였다. 또한 프로그램 자신에게 유리한 수를 찾기 위한 탐색에서 무의미한 노드들의 전개를 줄여 제한된 시간 안에 좋은 수를 찾을 수 있도록 알파 베타 가지치기(alpha-beta pruning)를 사용하도록 프로그램을 구현하였다.

II. Preliminaries

오목 인공지능에 대한 연구가 활발하게 이루어지는 것은 아니지만, 매 년 오목 인공지능 대회가 열리며 2012년부터 Yixin 이라는 인공지능 프로그램이 매해 우승하고 있다. 최근 2018년에 열린 대회에서 FreeStyle 룰에서만 다른 인공지능에게 1위를 내어주고 아직까지도 다른 룰에서는 1등을 유지하고 있다. 인공지능과 사람과의 대결도 있었는데 인공지능 특성상 한 수를 두는 시간이 매우 길지만, 본문에서 소개한 Yixin과 사람과의 대결에서 총 전적 1승 3무 1패를 기록하였다.

III. The Proposed Scheme

1. Applied Rules

본 논문에서 적용한 렌주 룰은 먼저 돌을 놓는 흑에게 열린 33, 열린 44, 장목은 금지되나, 43은 허용되고, 백에게는 열린 33, 열린 44, 장목, 43이 모두 허용된다. 게임에 이용되는 판의 크기는 15x15이다.

Table 1. Applied Rule

종류 / 선수	33	44	장목	43
흑(선수)	불허	불허	불허	허용
백	허용	허용	승리	허용

2. Weight for Pattern

게임에 적용한 패턴에 대한 가중치는 고정 가중치를 적용하였으며, 오목을 만들기 위하여 분석하는 과정에서 만든 초기 값을 이용하여 초기 버전의 프로그램을 제작하고 제작된 프로그램으로 게임을 수행하면서 나타난 현상을 나름대로 분석하여 경험과 판단을 바탕으로 수정하여 아래와 같이 패턴들에 대한 가중치를 만들었다.

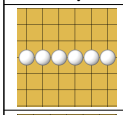
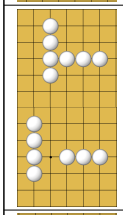
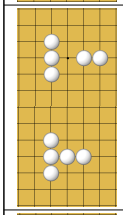
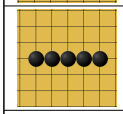
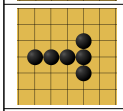
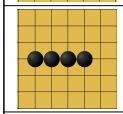
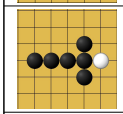
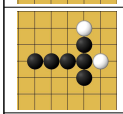
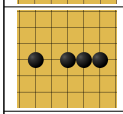
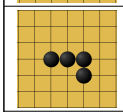
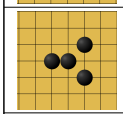
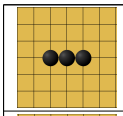
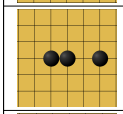
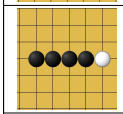
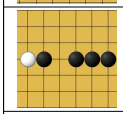
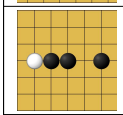
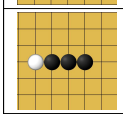
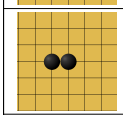
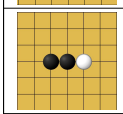
그림	상태	가중치
	장목	2000
	사사	1800
	열린 삼삼	900
	오목	2000
	사가 열린 사삼	1600
	열린 사	1500
	사가 닫힌 사삼	1300
	막을 수 있는 사삼	800
	한 칸 띄어진 열린 사	660
	열린 삼이	420
	열린 이이이	410
	열린 삼	400
		
	한 칸 띄어진 열린 삼	360
	막힌 사	200
	한 칸 띄어진 막힌 사	190
	한 칸 띄어진 막힌 삼	120
	막힌 삼	60
	열린 이	40
	막힌 열린 이	30

Fig. 1. Weight Table by Pattern

가중치는 컴퓨터가 먼저 두게 되면, 컴퓨터는 검은 색으로 자신의 가중치를 계산하고 백돌로 상태의 가중치를 계산한다. 역은 흑백을 바꾸어 생각한다.

평가 함수의 가중치 부여는 제시된 표와 논리적으로 일치할 경우에 부여하게 되며 완벽하게 일치하는 것이 아닌 패턴이 동일하며 방어자 입장에서 방어하기 위해 소비되는 돌의 개수가 같다면 같은 가중치를 부여한다.

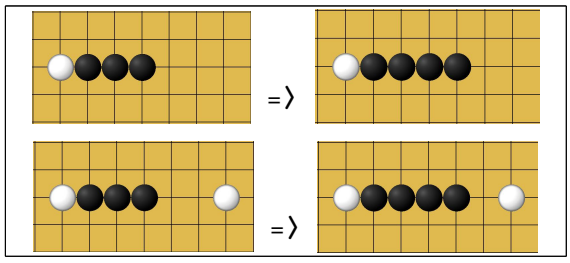


Fig. 2. Weight Calculation Example

위 그림에서 위와 아래 그림은 두 칸 뒤에 백돌의 존재 여부가 차이가 나지만 각각 오른 쪽과 같이 흑이 수를 두게 된다면 방어하기 위한 백돌의 개수는 모두 한 개이므로 물리적으로는 다르지만 논리적으로 같은 패턴이라고 판단하였다. 따라서 왼쪽에서 오른쪽의 상태를 만들 수 있는 좌표에 모두 같은 가중치를 부여한다.

3. Search the Stone Position

오목은 혼자서 하는 게임이 아닌 상대가 있는 게임으로 교대로 돌을 놓는데, 상대가 내 마음 먹은 대로 순응하며 돌을 놓아 주진 않는다. 상대도 자신에 유리한 수를 놓기 때문에 상대의 수를 계산하지 않는 깊이우선 탐색이나 너비우선탐색 방법으로 돌을 놓을 위치를 찾는 것은 의미가 없다. 따라서 내가 돌을 놓은 후 상대가 놓는 단계까지 고려해서 판단해야 한다. 이럴 때 사용하는 방법이 최소-최대 게임 트리이다. 즉 내가 놓을 수 있는 모든 상태와 그다음 상대가 놓을 수 있는 모든 상태를 이용하여 계산하되 자신이 놓을 수 있는 말의 위치에 상대가 놓을 수 있는 말의 위치별로 앞에서 언급한 패턴별 가중치를 이용하여 계산하되 상대 차례에서 상대가 얻을 수 있는 가장 큰 값들 중, 내가 놓음으로서 상대가 얻을 수 있는 값이 가장 작아지게 하는 수를 선택한다. 반대로 한다면 상대가 나에게 불리한 수를 선택하므로 min 단계의 값 중 나에게 가장 유리한 수를 그 상위 max 단계에서 내가 불리한 수중 가장 큰 수가 되게 선택하는 것이다.

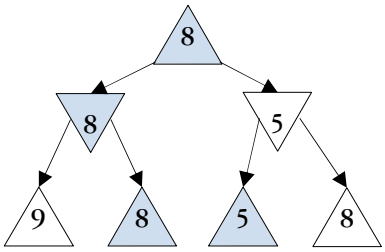


Fig. 3. Min-Max Tree

Fig. 1은 구현된 프로그램이 최소-최대 게임 트리를 이용해 어떻게 수읽기를 하는지 보여준다. 먼저 상대가 돌을 놓을 차례 레벨인 하단 노드에서 가중치를 구한다. 그 후 노드의 깊이가 홀수일 경우, 최하단 노드들의 부모 노드는 자기 자신의 자식들이 가진 가중치 중 최솟값을 선택하며, 짝수일 경우에는 최댓값을 선택한다. 이 것은 부모 노드가 자기 자식들의 가중치 중 가장 큰 값을 선택하기 위함이다. 즉, 부모 노드가 선택하게 된 수를 가장 유리한 수라고 판단하는 것이다.

렌주룰 오목의 경우 돌을 둘 수 있는 공간은 15 * 15, 225개이며 초기에 트리 깊이 3까지 앞을 내다본다고 가정하고 자신에게 유리하게 돌을 놓을 자리를 계산할 때 계산해 보아야 할 노드의 수는 225 * 224 * 223, 즉 11,239,200개가 된다. 단순히 3수 앞을 내다본다는 조건 만으로도 노드의 방문 수가 저렇게 많아진다. 한 노드를 전개하는데 1us 걸린다고 가정해도 10초 정도 걸린다. 10초 이상 걸린다. 따라서 좀 더 깊이 보려면 전개할 필요가 없는 노드들을 전개하지 않는 전략이 필요하다. 이를 위해 알파-베타 가지치기를 적용하였다. 이 방법은 깊이 우선 탐색을 수행하되 지금까지 나온 값들을 참고하여 더 전개할 필요가 없을 때 전개를 그만두는 가지치는 방법이다.

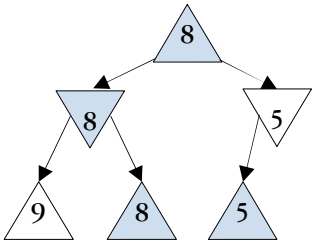


Fig. 4. Alpha-Beta pruning

Fig. 2에서 보면 미니맥스 트리를 기반으로 가지치기를 진행하며 최 하단 노드로부터 8이 선택되는 과정까지는 똑같다. 8이 선택된 이후 옆 노드를 방문하는데 옆 노드에서 5의 가중치가 발견된 순간 이를 방문할 필요가 없다고 판단하고 가지치기 한다. 5가 발견된 순간 옆 노드에서는 최솟값을 불러와야 하기 때문에 5보다 같거나 작은 값을 선택하게 된다. 하지만 루트노드에서 선택하는 것은 최댓값이다. 8보다 같거나 작은 값이 발견되는 순간 8보다 더 큰 값이 선택될 수 없기 때문에 더 이상 방문하지 않고 가지치기 한다. 예시로 제시한 그림에서는 노드 수가 작기 때문에 가지치기의 가치를 설명하기에는 부족함이 있으나 앞에서 계산한 바와 같이 오목 게임 초기에 노드의 수가 최소 11,239,200개 전개되는 것을 생각하면 알파-베타 가지치기는 대략 30% 이상의 가지치기 효율을 기대할 수 있다.

4. Evaluating Stone Positioning

오목은 돌을 연속적으로 다섯 개 놓으면 이기는 게임이다. 그렇다면 한 수를 둘 때 상대의 오목을 막을 수 있거나 혹은 자신의 오목이 만들어 질 수 있는 수를 둬야 의미가 있는 수가 되는 것이다. 따라서 각 경우마다 자신의 돌의 개수와 자신의 돌이 몇 개나 연속으로 연결되어 있는지 여부, 상대방의 돌의 개수를 센다. 여기서 상대방의 돌의 개수가 1이라도 있으면 해당 경우에서는 오목이 이루어지지 않는다고 판단하고 더 이상 돌을 세지 않는다. 해당 경우의 5의 범위에서는 오목이 이루어질 수 없기 때문이다.

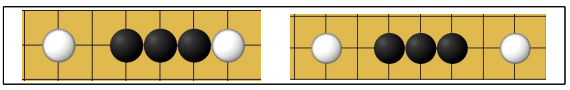


Fig. 5. Impossible and Possible Example of Omok

좌와 같은 경우는 좌우로는 오목이 만들어 질 수 없으며 좌우 방향으로는 어떻게 돌을 세도 흰 돌을 포함하고 있기 때문에 좌측 빈 공간에 부여되는 가중치는 0이 된다. 우와 같은 경우는 오목이 만들어 질 여지가 생긴다. 좌우의 빈 공간을 기준으로 백돌이 세어지지 않는 경우의 수가 만들어지기 때문이다. 이 경우에는 오목이 완성될 수도 있기 때문에 좌우로 빈 공간에 가중치를 부여하게 된다. 앞에서 제시한 가중치를 살펴본다면 해당 그림은 막힌 사(4)에 해당하므로 200이 된다. 평가 함수는 자신의 차례에 그림에 해당하는 상태를 만들 수 있다면 해당 좌표에 주어지는 가중치를 부여하는 함수이다.

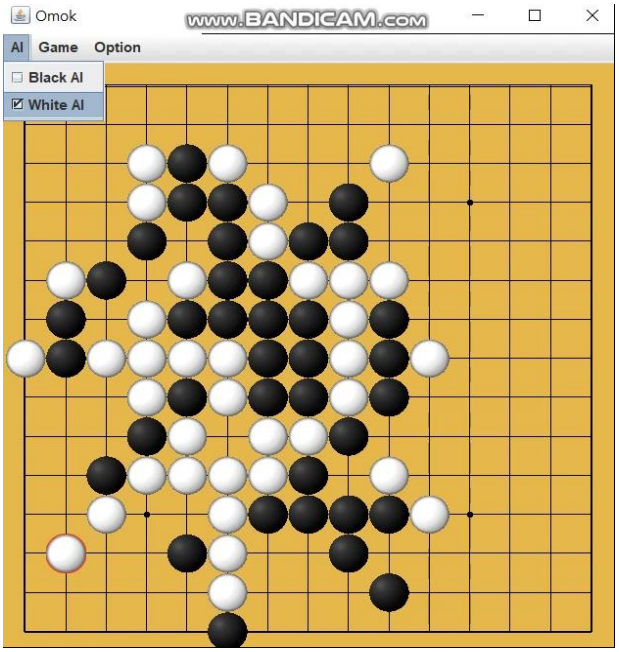


Fig. 6. Example of game execution of completed program

IV. Conclusions

본 연구에서는 오목 프로그램을 구현하였다. 평가를 위한 트리의 깊이는 3으로 하였으며, 트리의 깊이를 더 추가할 경우 메모리에 대한 압박이 있으며 연산 량 증가로 인해 한 수를 두는 시간이 더 길어졌다. 시간적인 제약으로 인해 트리의 깊이를 3으로 두고 실험을 진행하였는데 기본적인 수는 모두 막으나 평가 함수에 의한 행동 패턴이 존재하기 때문에 일정한 패턴으로 수를 두었으며 자신이 공격할 수 있음에도 불구하고 지나치게 수비적인 모습을 보여 공격 기회를 놓치는 모습도 보인다. 첫 번째로 노드 수 증가에 따른 메모리 효율 개선이 필요하며 두 번째로는 수비 가중치와 공격 가중치의 분리만 이루어지면 꽤나 그럴듯한 프로그램이 만들어 질 것이다.

REFERENCES

[1] http://cafe.naver.com/omoknara/28650
[2] https://ko.wikipedia.org/wiki/%EC%B5%9C%EC%86%8C%EA%B7%B9%EB%8C%80%ED%99%94
[3] https://ko.wikipedia.org/wiki/%EC%95%8C%ED%8C%8C-%EB%B2%A0%ED%83%80_%EA%B0%80%EC%A7%80%EC%B9%98%EA%B8%B0