컴퓨터 바둑 프로그래밍 기법

김 영 상[†] · 이 종 철[††]

요 약

본 논문은 컴퓨터 바둑을 구현하기 위한 프로래밍 기법을 제시한다.

컴퓨터 바둑은 바둑의 여러 규칙을 가지고 스스로 국면의 형세를 판단한 후, 다음 놓을 자리를 정확히 계산할 수 있어야 한다. 주요 설계 원칙은 패턴 매칭, 알파-베타 탐색, 영향력 계산 등의 휴리스틱한 요소를 조합하는 것이다.

본 논문에서는 컴퓨터 바둑을 연구하는 사람들의 접근 방법과 그들의 연구결과를 소개하고 컴퓨터 바둑을 구현하기 위한 자료 구조와 알고리즘을 제안하였다.

Programming Methodology of the Computer Go

Young Sang Kim[†] · Jong Cheol Lee[††]

ABSTRACT

In this paper, we describe the programming methodology which can produce computer Go.

After computer Go program with the rules of Go determines a territory for itself, it must evaluate the exact next move. The common design principle of computer Go is to combine such heuristic elements as pattern match, alpha-beta pruning and influence function.

In this study, we introduce many other approaches and their results on computer Go, and then show data structures and algorithms to implement computer Go project.

1. 서 론

게임은 인공 지능 분야에서 학문적 가치가 높은 연구 대상이다. 그것은 프로그램의 성공 여부를 쉽게 예측할 수 있고, 많은 지식을 필요로 하지 않으므로 프로그래밍이 쉬우며, 무엇보다도 많은 사람의 흥미를 유발하기 때문이다.

특히, 바둑과 같은 지적 게임은 지극히 명확한 규칙으로 정의된 세계를 가진다는 점에서 컴퓨터로 해결하기에 적합하다. 바둑 게임은 두 경기자가 교대로 흑, 백의 돌을 번갈아 놓으면서 자신의 영역을 구축하는데, 최종적으로 집이 많은-혹은 자신의 영역이 넓은-경기자가 이기게 된다.

그러므로 컴퓨터 바둑 프로그램은 바둑의 여러 규칙을 가지고 스스로 국면의 형세를 판단한 후, 다음 놓을 자리를 정확히 계산할 수 있어야 한다. 컴퓨터 바둑이 추구하는 최종 목표가 프로 기사와 대적하는 수준이라고 할 때, 기존의 컴퓨터 바둑은 공통적으로 부분적인 전술에 강한 반면, 전반적인 형세를 평가하는데는 아직도 미약한 수준이다. 왜냐하면, 바둑판에서 고려해야 할 변화가 361!의 $\frac{1}{4}$ 으로 천문학적인 숫자이기 때문이다. 제1선 또는 착수 금지에 해당하는 점이나 팻감 등에 따라서 실제의 변화는 줄어들 수

† 정 회 원:한라전문대학 전산정보처리과 전임강사
†† 정 회 원:경북대학교 컴퓨터공학과 부교수
논문접수:1995년 11월 29일, 심사완료:1996년 3월 8일

있지만, 여전히 변화의 숫자는 엄청나다. 따라서, 유용한 변화만을 선택할 수 있도록 조정하여 올바른 다음 수(next move)를 생성하는 컴퓨터 바둑 프로그램을 개발하여야 한다.

바둑은 유기적 인과성이 매우 강한 현실 세계를 그대로 반영하는 게임이기 때문에, 컴퓨터 바둑은 각각의 국면마다 최선의 가치를 가지는 '다음의 한 수'라는 특정한 현실성을 고려해야 한다. 이를 테면, 각 부분 전투가 어떤 과정으로 진행되어 왔으며, 어떻게 끝날 것인가에 대한 전략적인 구도를 실현할 수 있는 새로운 접근 방법을 모색해야 한다. 기존의 컴퓨터 바둑은 문제풀이 능력이나 계산, 기계적 인과성 등을 담보로 하는 절차적 설계 방식에 의존하고 있다. 따라서, 프로 기사가 바둑을 두는 감(感)이라든가 가치 판단의 능력은 전혀 기대하지 못한다.

컴퓨터 바둑을 연구하는 그룹들은 크게 부분적인 상황의 전개를 따른 이론적 분석[1, 2]과 이를 적절히 응용한 프로그램을 개발하는[3-9] 두가지 측면으로 분류할 수 있다. 우선, 부분적 상황 전개에 대한 연구로는 제한된 범위 내에서 이루어지는 특정한 국면에 대한 알고리즘의 개발이 주축을 이루고 있다. 또한, 사용자에 대해서 적절히 응수하는 컴퓨터 바둑 프로그램으로는 Albert Zobrist가 최초의 컴퓨터 바둑을 선보인 이래로, 세계의 많은 프로그래머들이 이에 도전하여 왔고, 1986년에 국제 경진 대회가 생겨나면서부터는 더욱 많은 주목을 받으며 점진적인 향상을 보여 왔다.

이들 컴퓨터 바둑의 두드러진 특징으로는, 지엽적인 게임 트리의 탐색 알고리즘이나 정석, 패, 축, 장문 등의 패턴을 데이터베이스로 사용한다는 점이다. 패턴은 바둑에서 흘륭하다고 입증된 모범적 유형이기 때문에 부분적 가치를 판단하는 초석이 되기도 한다. 바둑은 양자간에 점령해야 할 중립적인 지역을 두고, 서로의 이해 대립이 불가피한 전면전의 성격을 가지기 때문에 특정 국면에 대한 알고리즘이라든지 지엽적인 데이터베이스만을 가진 컴퓨터 바둑은 기력이 낮을 수밖에 없다.

본 논문은 세계 각국에서 컴퓨터 바둑을 연구하는 사람들과 그들의 접근 방법론 및 지금까지의 연구 결과를 분석하고, 본 연구실에서 수행 중인 컴퓨터 바둑 프로젝트를 기반으로 컴퓨터 바둑에 적절한 프로그래밍 방법론을 제안한다.

본 논문의 구성을 보면, 제2장에서는 컴퓨터 바둑 연구가들과 컴퓨터 바둑 경진 대회를 소개하고, 실전 대국을 예시하여 컴퓨터 바둑의 특성을 분석하였다. 제3장에서는 컴퓨터 바둑 프로그램을 구현하기 위한 동적 자료 알고리즘을 제시한다. 그리고 제4장에서는 프로그램의 전략과 다음 수 선정 절차를 제안한다. 그리고 제5장에서 세계 대회 우승자와의 평가 대국을 통하여 프로그램의 성능을 평가하고 6장에서 결론 및 향후 연구 방향을 논의한다.

2. 관련 연구

이 장에서는 전세계적으로 컴퓨터 바둑을 연구하는 학자들과 프로그래머들의 설계 방법을 비교 분석하고, 컴퓨터 바둑의 최대 목표인 프로 기사에 대적하는 수준의 프로그램을 만들기까지 매년 개최하고 있는 컴퓨터 바둑 대회에 대해서 살펴본다.

2.1 컴퓨터 바둑 연구가

컴퓨터 바둑에 대한 학문적 연구가 시작된 것은 1968년, Albert Zobrist에 의해서 최초의 컴퓨터 바둑 프로그램이 소개되었을 때이다. 그 후, 1985년 첫번째 컴퓨터 바둑 세계 대회가 개최되면서 점차적으로 컴퓨터 바둑에 대한 학문적 관심이 고조되어 왔으며, 최근에는 다양한 연구결과와 더불어 상업적 목적의 프로그램도 등장하게 되었다.

연구의 기초가 다져진 70년대에는 Bruce Wilcox와 Walter Reitman이 [3]의 프로그램을 LISP언어로 작성하였는데, 이는 초반과 중반, 종반의 변화 단계를 유지하면서 부수적 목표 트리(contingency plan goal tree)를 구성하여 각각의 이동 순서에 대해서, 각 이동에 따르는 목표 스택(goal stack)을 축척하고, 두개의 목표 스택으로부터 단일 목표-비목표(non goal) 쌍으로 대립된 목표를 조합한 후, 계층적 목표 트리 구조에서 단일 노드로 동일한 목표 쌍을 조합하는 것이다. 그는 여러 논문[3-6]에서 다양한 정보 표현 기법으로 최선의 수를 선택하기 위한 알고리즘을 제안하였다. 그러나 이러한 알고리즘은 상대의 돌에 대해 직접 인접하여 착점하기를 꺼리고, 상대가 위급한 대마를 탈출하려는 상황에서 오히려 아군의 세력증 허술

한 지역이 위험에 처할 수 있게 되는 결과를 가져온다. Bruce Wilcox는 [3]을 PC상에서 1년여에 걸쳐 C 언어로 재 작성하여 Nemesis라는 이름으로 세계 대회에 참가하여 우수한 성적을 거두기도 하였다.

본격적으로 19×19의 컴퓨터 바둑 프로그램 경진이 이루어진 시기는 1987년 옹창기 세계 대회이므로, 이때부터 학문적 관심 또는 상업적 목적을 가진 여러 프로그래머들이 컴퓨터 바둑을 활성화하게 되었다. 이를 테면, 가속화된 패턴매칭 기법을 사용함으로서 정석과 초반에 강한 특성을 가진 Mark Boon의 프로그램, Goliath은 1989년부터 1991년까지 옹창기 세계 대회를 석권하였다.

Shirayanagi는 바둑에 관련된 지식을 기호적으로 다루기 위하여 LISP의 표현을 사용하여 YUGO라는 프로그램을 발표하였으며, 그는 [10]을 통하여 이러한 기본적 표기 도구로서 특별히 정의된 용어와, 표기법, 패턴 지식을 사용하는 기호적 표현법을 제안하였다.

Dargon II는 공격과 방어에 대한 많은 종류의 패턴 지식을 보유하고 이를 평가하기 위한 깊이 우선 탐색 기법을 사용한 프로그램인데, 보유한 패턴 지식에 근거하여 공격과 방어에 대한 전술적 대응을 하도록 만들어졌다[11].

또한, 60개의 패턴을 기반으로 다음의 한 수를 계산할 수 있는 COSMOS를 만들었던 David Fotland는 새로이 Many Faces of Go(MFGO)를 발표하였는데, 그는 MFGO에서 210개의 패턴을 기반으로 다음에 착점할 순서의 전체트리를 계산할 수 있게 하였다[8, 12].

Barney Pell은 컴퓨터 바둑을 확률적 평가 함수 학습(probabilistic evaluation function)과 의존기반(dependency-based) 기계 학습의 두가지 관점에서 연구하고 있다. 마스터 게임에 대해서는 교사(supervised) 학습을, 경험적 지식에 대해서는 무교사(unsupervised) 학습을 수행하고, 대마간의 의존 구조(dependency structure)를 구성하여 독립도(liberty)가 2인 상대의 대마(group)를 잡을 수 있는 나의 대마는 안정하다고 정의하였다[12]. 또한, Stoutamire는 [12]에서 실전에서 자주 나타나는 패턴을 학습시킨 후, 해싱 기법으로 다음 수를 선정하였다.

본 연구실의 Keundol[13, 14, 15]은 1991년부터 인공 지능 프로젝트로 시작하여 지금까지 연구가 진행중에 있는데, 이론적인 정립[2]과 함께 객체 지향적 개념을 사용하여 바둑판의 반면에 존재하는 요소 사이의 유기적인 관계로서 Keundol의 단면을 기술하였다[15].

이와 같이 컴퓨터 바둑에서의 인공 지능적 접근과 연구는 최근에 이르러 매우 활발히 전개되고 있는데, 다양한 알고리즘과 여러가지의 개발 도구를 가지고 최상의 프로그램을 만들기 위해 노력하고 있다. 특히 바둑에서의 형세 판단은 그 중요성에 기인하여 기존의 대부분의 컴퓨터 바둑 프로그램에서 어떠한 형태로든 구현이 되어 있으나, 다만 논문을 통한 발표나 이론적인 정립은 별로 없는 편이다.

2.2 1995년 옹창기 세계 대회

1995년 대회는 (주)한글과 컴퓨터의 주관 아래 11월 4, 5일 양일간 한국 기원 대회장에서 개최되었다. 이번 대회는 옹창기 바둑 교육 재단의 집행 비서인 양 유자가 심판관으로 참석한 가운데 옹씨룰을 적용한 스위스식 리그 방식으로 진행되었다. 당초 7개국 13개팀이 참가 신청을 냈으나 Janusz Kraszek가 교통사고로 불참하는 등 단지 10개 팀만이 참가하여 5라운드까지 대국이 벌어졌는데, 이 대회의 자세한 대국 상황은 표 1과 같다. 각 라운드에서 +는 승, −는 패를 의미한다[16].

〈표 1〉 1995년 세계 대회 최종 결과
〈Table 1〉 World computer Go congress results(95')

프로그램명	라운드					승수	순위
	1	2	3	4	5		
1 (Handtalk)	+4	+7	+3	+2	+5	5	1
2 (Go Intellect)	+6	+3	+5	−1	−4	3	3
3 (MFGO)	+9	−2	−1	+6	+7	3	4
4 (Go4++)	−1	+10	+6	+7	+2	4	2
5 (Stone)	+10	+8	−2	+9	−1	3	5
6 (Explorer)	−2	+9	−4	−3	+8	2	6
7 (Sason)	+8	−1	+10	−4	−3	2	7
8 (Jungnyeom)	−3	−6	−9	−10	−6	0	10
9 (Keundol)	−3	−6	+8	−5	+10	2	8
10 (Rex 95)	−5	−4	−7	+8	−9	1	9

Goliath과 Nemesis는 1993년 이후 연속 참가하지 않았으며, 상위 입상한 3개의 프로그램은 1993년,

1994년 대회에서 역시 5위이내에 속했던 프로그램으로 기력은 8~10급 정도로서 서로 엇비슷한 것으로 나타났다.

93년도 우승자이자 이번 대회 우승을 차지한 Handtalk의 프로그래머, Zhi Xing Chen(아마 6단, 63세)은 중국 광조우에 있는 Zhongshan 대학의 화학과 교수로서 어셈블리 언어로 프로그래밍하였다. 제1회 FOST컵 대회에서 일본 급수로 5급의 기력을 인정받기도 하였던 Handtalk은 다음 수를 계산하는 시간 및 수행 코드 크기가 짧고, 중반전에 강한 것이 특징이다. 또한, 이번 대회를 관전한 프로 기사들에 의하면 Handtalk은 우리 급수로 7, 8급 정도라고 평가했으며, 공격적이기보다는 철저한 수비형이라고 분석했다. Michael Reiss의 Go4++은 87년 대회에서 저조한 성적을 냈었으나 이 대회에서 수준이 향상되어 Hantalk과의 접전 끝에 2위를 차지하였다.

관심을 모았던 인간과의 대결에서는 Handtalk이 15점을 놓고 싶은 위치에 선점하고 대국을 벌이는 독특한 방식으로 진행되었는데, 송태곤(86년생), 박영훈(85년생), 김병희(83년생) 등 3명의 소년 기사와 대국을 벌여서 송태곤에게는 패하였으나 박영훈, 김병희를 꺾고 2승1패로 승리하였다. 연이어 벌어진 2차전에서도 송태곤, 손경진(83년생)을 꺾어 2승1패를 기록하였다.

94년 대회의 우승자이자 이 대회 3위인 Go Intellect은 패턴 추출기와 같은 20여개의 특수화된 수 생성기(move generator)를 가지고 있으며, 궁극적으로 바둑판 전체에 대해서 알파-베타 탐색을 통하여 최적의 수를 만들어 낸다. Ken Chen은 영향력과 연결성을 결정하기 위해서 휴리스틱한 평가 함수를 제안하였는데, d(p, s)를 지점 p에서 돌 s까지 최소 경로라 할 때, 돌에 미치는 영향력을 $2^{6-d(p,\ s)}$으로 계산하였다[12].

상업용으로 프로그램화하여 미국 내에서는 꽤많이 알려진 MFGO는 250여개의 착점 규정과 26,000개의 정석, 320개의 특수 패턴을 지식으로 가지고 있으며, 전술적 분석기(tactical analyzer) 및 연결성, 눈, 사활, 영역 측정기(evaluator) 등으로 가장 최적의 수를 읽어낸다. MFGO는 약간의 그래픽 인터페이스 부분을 제외하고, 프로그램 전체가 41,000라인의 C언어로 작성되었으며, 약 440 Kbytes의 메모리를 차지한다[8].

2.3 1995년 FOST컵 대회

제1회 FOST컵 대회는 동경에서 9월 29, 30 양일간 개최되었다. 이 대회의 규정은 응창기 세계 대회와는 달리 일본식 규칙을 적용하며, 5호 반의 덤을 공제한다. 각 프로그램은 1시간내에 125수 이상을 착수해야 하며, 표준 컴퓨터 바둑 모뎀 규약(standard computer Go modem protocol)을 사용하여 통신할 수 없는 프로그램에 대해서는 10분의 시간 초과 벌점을 부여하게 되어 있다.

사실 이 대회 상위 입상자들은 다른 프로그램보다는 월등히 기력이 높았다. 우승자인 Handtalk은 6급의 소녀(7세) 기사와 9점 접바둑을 두어 이겼고, MFGO는 1단의 청소년 기사에게 9점 접바둑에서 136수만에 불계승을 거뒀지만 Go 4++은 패했다. 이들 프로그램과 30수의 선점 대국을 벌인 일본 프로 기사는 Handtalk에 대해서 5급을, Go4++에 대해서 7급을, MFGO에 대해서 8급을 인정했다. 특히, Go Intellect는 Handtalk와의 마지막 라운드에서 정석 라이브러리를 사용하지 않는 바람에 자신의 게임을 잃었다. 아마 6단의 기력을 가진 Ken Chen은 프로 기사에게 사사를 받고 현재 자신의 프로그램을 계속하여 업그레이드하고 있다[7].

일본은 프로 기전에 맞먹는 상금을 걸고 FOST컵 대회를 신설하는 것과 별도로 94년부터 매년 게임 프로그래밍 워크샵을 개최하는 등 컴퓨터 바둑에 매우 각별한 관심을 쏟고 있다.

3. 자료 구조의 정의

이 장에서는 바둑을 프로그래밍하기 위한 요구사항에 대해서 살펴보고, 컴퓨터로 구현하기 위한 자료구조를 제시한다. 컴퓨터 바둑에서 동적 자료는 돌이 놓인 자리에 대해서 이웃한 공배의 수라든지 덩어리의 활로(liberty)와 같은 지역적 자료로서, 국면이 바뀔 때마다 변하게 되고, 없어지거나 새로운 형태로 바뀔 수 있다. 따라서, 한 수가 두어지면 그 국면에 대한 동적 자료는 다시 계산하여야 한다. 이러한 동적 자료는 돌, 덩어리, 연결성, 집, 대마, 영향력 등으로 구분할 수 있다.

```
typedef struct point /* 임의의 한 지점 */
```

```
{ int x, y;
} Point_type;
int color_type[]={BLACK, WHITE, EMPTY};/*
가능한 색 */
Point_type *dis2, *dis3; /* 거리가 1, 2, 3인 지점 */
```

3.1 돌

바둑판에서 돌은 가장 기본적인 객체로서 다음과 같은 자료 구조로 표현할 수 있다.

```
typedef struct stone
{ int    color, /* 돌의 색 */
         nliberties, /* 돌의 활로의 수 */
         strength; /* 돌의 세기 */
Point pos;, /* 바둑판에서의 위치 */
} Stone;

typedef struct p_list {
  int safe; /* 안정도 */
  Point_type pos; /* 돌의 리스트 */
  struct p_list *next;
} P_List_type;
```

따라서 돌에 대한 정보는, 돌이 덩어리의 일부로 존재하는 것인지 아니면 홀로 떨어져 있는 것인지 국면이 바뀔 때마다 갱신되어야 한다. 따라서, 돌의 활로가 되는 인접한 공배의 수와 위치, 인접한 돌의 색과 형태를 알아야 한다. 부가적으로, 다른 돌 혹은 다른 덩어리와 연결 가능한지 및 주위의 돌에 미치는 영향력에 대한 정보도 가져야 한다.

3.2 덩어리

덩어리는 같은 색으로 이어진 돌의 집합을 말한다. 개개의 덩어리는 독립적인 성질을 가지며, 색, 활로, 돌의 수와 같은 정보를 유지해야 한다.

```
typedf struct string
{ int    color, /* 덩어리의 색 */
         nliberties, /* 덩어리의 활로의 수 */
         nstones; /* 덩어리를 이루는 돌의 수 */
  Point pos; /* 덩어리에서 임의의 한 점의 위치 */
  Point empty_pos[MAX_EMPTY];/* 활로의 위치 */
} String;
String strings[19*19];/* 덩어리를 저장한 배열 */
int nString;/* 현재의 반면에 있는 덩어리의 수 */
```

3.3 연결성

연결되었다는 것은 상대방의 힘에 의해서 단락되지 않았음을 의미하며, 각각의 덩어리 사이에는 자기적 성질의 강도가 존재한다. 즉, 같은 색의 덩어리가 연결되면 힘은 강해지고, 다른 색의 덩어리가 인접해 있다면 적절하게 분할된 세력권을 형성한다. 따라서, 컴퓨터 바둑은 어떤 덩어리가 연결 가능한가를 예측하고, 그 덩어리의 위치를 알아낸 다음 연결 강도가 가장 강해지는 연결의 형태를 결정한다. 연결의 유형으로는 젖힘(Hane), 한간 벌림(Ikken-tobi), 반패(half knight's move), 패(knight's move), 두간 벌림(Nikken-tobi), 세간 벌림(Hiraki) 등이 있고, 연결 강도는 이미 끊김, 끊길 우려 있음, 맛좋게 이음, 꽉 이음 등으로 구분한다.

```
int conn_type[]={HANE, ONE, TWO, HIRAKI,
H_KO, KO};
int strength_type[]={CUT, CUTTABLE, TASTE,
SOLID};
```

3.4 집과 옥집

덩어리가 바둑판에서 존재하려면, 최소한 두개 이상의 완전한 집을 갖추어야 한다. 집의 유형에는 같은 색의 돌로 둘러 쌓인 공배가 하나 혹은 두개인 것, 직선 또는 직선이 교차하는 형태인 것, 옥집(1/2집) 등으로 나눌 수 있다. 예를 들어, 집의 유형을 판단하는 기준은 중앙에서 8개의 돌로 둘러 쌓인 집의 수는 완전한 한 집이고, 4개의 돌이면 옥집으로 판정한다. 귀의 경우는 각각 3개, 2개의 돌이, 변의 경우는 5개, 3개의 돌이 덩어리 안의 집수를 결정한다.

```
Point *eye_empty_points;/* 집안의 공배 수 */
int eye_type[]={HALF, ONE, TWO, MULTIPLE};
```

3.5 대 마

컴퓨터 바둑에서 대마에 대한 정의는 같은 색의 덩

어리가 4간 정도의 거리를 두고 인접한 경우를 말한다. 따라서, 대마를 이루는 덩어리의 수 또는 대마가 가진 집의 수 및 위치, 대마의 강도, 인접한 상대방 대마의 위치도 간파하고 있어야 한다.

```
typedef struct group
{ int color; /* 그룹의 색 */
  Point list_eyes[MAX]; /* 그룹안의 집의 리스트 */
  String list_strings[MAX];/* 그룹안의 덩어리 리스트 */
  int strength;/* 그룹의 강도 */
  struct group Enemy_group[MAX];/*
  적 그룹의 리스트 */
} Group_type;
Group_type groups[19*19];
int nGroups;
```

3.6 영향력

바둑 반면의 상황이 바뀔 때마다 바둑판 전체에 대한 영향력은 다시 계산되어져야 한다. [13]에서 우리는 영향력 함수를 $1/distance^2$로 하여 각 돌이 가지는 힘을 구하였다. 독립된 돌에 대한 초기치는 중앙, 변, 귀의 특수성을 감안하여 일정한 순위를 두었다. 따라서, 바둑판 상의 각 돌 또는 덩어리는 1에서부터 −1까지의 영향력을 가지게 된다. 영향력은 현 국면의 영역을 계산하거나 두터움, 급한 곳, 사활 등 바둑 반면을 평가하는데 유리하게 작용한다.

3.7 축과 장문

운영 규칙의 간단함에 비해서 컴퓨터 바둑을 프로그래밍 하기 위해서 고려해야 할 사항은 매우 복잡하다. 우선 실전에서 빈번하게 발생하는 축(Shicho)에 관한 지식이 있어야 한다. 백의 입장에서 축의 성립 여부를 판단하려면 축 머리선 상에 상대방 돌이 있는지 미리 예측하는 알고리즘이 필요하다.

【알고리즘 1】 축의 성립 여부 판단

```
bool Shicho_Test(String S)
{
 if (S->nliberties == 2) {
    L1 = S->empty_pos[0];
    L2 = S->empty_pos[1];
    Move_Possible(L1, L2);
    Put_Stone(L1);
    Update(S);
    if (Out_Board(S) return(YES);
    if (S->nliberties ) = 3) return(NO);
    else Shicho_Test(S);
 }
}
```

한편, 직접 상대의 활로를 막지 않고 포위하려면 장문을 이용한다. 축이 성립하지 않거나 축으로 잡을 경우, 자신이 불리하게 될 때 사용한다.

3.8 환격과 촉촉수

환격수는 상대의 돌을 계속 추적하여 자충으로 유도하여 돌을 잡는 방법이다. 알고리즘 2는 환격이 가능한지를 테스트한다.

【알고리즘 2】 환격 판정

```
bool Snap_back_Check (int x, int y)
/* Check Uttegaeshi *6/
 {
    if {new_lib == 0 && old_lib == 1 &&
    x == Dead_Array[Dead_Index − 1].x &&
    y == Dead_Array[Dead_Index − 1].y) {
    if (Count_Block(Other(Turn) ) 1))
      return(FALSE);
    else return(TRUE);
    }
 else return(FALSE); /* can be move */
}
```

4. 프로그램의 전략

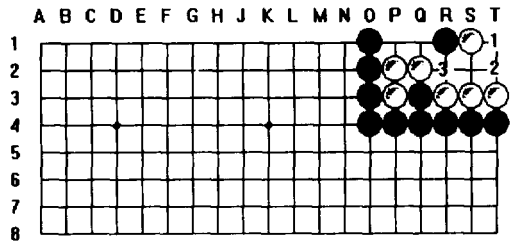
대부분의 문제 풀이 시스템은 정적인 자료를 대상으로 수식을 반복 계산하거나 결과를 산출해 낸다. 하지만, 컴퓨터 바둑은 문제 풀이 과정중에 계획이 혼합되어 목표를 변화시키므로 동적 환경에 적응할 수 있도록 프로그램을 구성하여야 한다. 이 장에서는 다음 수 생성을 위한 전략과 전술에 대해서 논의한다.

4.1 지역 접근전

바둑에서는 귀의 가치가 가장 우선되므로 당연히 흑은 귀(corner)에서부터 시작한다. 귀의 주요 착점 위치로는 화점, 소목, 외목, 고목, 삼삼 등이다. 귀로부터의 발전과 세력의 방향으로는 한칸 뜀 또는 날일자 걸침이 주효하고, 귀에 대한 점령이 끝난 후 굳히거나 걸침이 이루어진다. 굳힘은 상대의 침입을 미리 단속하여 자신의 세력을 견고하게 하고자 함이며, 걸침은 차후의 공격을 겨냥하고 상대의 굳힘 또는 허술함을 찌르려는 의도로 보여진다.

이러한 귀의 착점 및 굳힘, 걸침, 벌림은 흔히 바둑의 정석이라 일컫는다. 따라서 다섯가지의 착점 위치에 따른 많은 종류의 정석이 있다. 컴퓨터 바둑은 이러한 정석이 패턴 데이터 베이스로 저장되어 있으며 바둑판의 대칭성을 이용하여 패턴 매칭을 선택하도록 한다.

이러한 바둑판의 대칭성은 컴퓨터 바둑의 지역 접근전에서, 하나의 사분면에 대한 패턴이 그대로 적용되어질 수 있음을 뜻한다. 또한, 바둑에서 나타나는 패턴은 주로 사분면중의 하나에 표현될 수 있는데 이는 패턴을 비트맵 형식으로 저장한 다음, 바둑판의 현재 상황을 여러 패턴으로 분리하고, 해싱을 통해서 매치되는 패턴을 찾아낼 수 있다.



(그림 1) 알파-베타 탐색의 예
(Fig. 1) Alpha-beta pruning

그림 1에서 흑이 1에 둔다면 백은 2로 응수한다. 흑이 3에 두면 백은 잡히게 된다. 이러한 문제에 대해서 완전한 탐색 트리를 구성하면 공배가 모두 6이므로 22605개의 중간 노드와 51421개의 단말노드가 계산된다. 이것을 역 순서화(reverse ordering) 노드로 변환하여 알파-베타를 적용하면 각각 4941개, 11835개로 줄어들고, 알파-베타와 캐쉬를 함께 하면 141,399가 된다.

4.2 전략 및 전술

프로그램의 기본 전략은 현재의 반면에 대해서 여러 개의 수를 생성하고 그중에서 가장 호착점을 선정한다. 돌이 놓일 자리에 대해서 주위의 돌과의 영향력을 미리 계산하고, 사활을 적용한다. 적 혹은 나의 덩어리 및 대마의 연결성을 체크하여 위급한 자리를 생성한다.

프로그램에 사용된 전술은 6개의 루틴으로 구성되었다. 우선 361자리마다 가중치를 계산하여 무게(weights)에 대한 초기값을 주었다. 영향력 갱신은 임의의 돌에 대해서 8방향으로 이웃하는 돌이 있는가를 검사하여 인접한 빈자리가 자살수가 아니면 무게를 갱신한다. 대마를 방어하는 루틴은 해당 대마에 대한 안정도가 2이하일 때만 활로를 검색하고, 적을 공격하는 루틴은 알고리즘 3과 같이 적의 활로가 32이하일 때 사용하고, 적의 덩어리사이에 침입하는 루틴 및 붙임 루틴은 해당자리와 나의 돌의 연결성을 평가하여 무게가 높을 때만 사용한다.

【알고리즘 3】 적의 약한 대마를 공격하는 루틴

```
void Attack_Opponent(P_List_type **move, byte depth)
{ /* Attacks weak groups */
  integer_board terrain;
  byte count, garbage;
  Group_type *WITH;
  int x, y;
  memcpy(terrain, zero_board, sizeof(integer_board));
  count = 0;
  while (bd.groups[count] -> size != 0) {
    WITH = bd.groups[count];
    x = WITH -> to_kill.x;
    y = WITH -> to_kill.y;
    if (WITH -> owner == Other(bd.Turn) &&
    WITH -> aliveness < 32 &&
      ((1L << WITH -> aliveness) & 0x38) != 0) {
      if (O_Board(x, y) && Legal_Move(x, y) &&
      !Suicidal(x, y))
      terrain[x][y] += WITH -> size;
```

```
        else {
        garbage = Life(*bd.groups[count], depth,
        &WITH->to_kill);
        if (On_Board(x, y))
          terrain[x][y] += WITH->size;
        }
      }
      count++;
    }
  *move = (P_List_type *)Highest_Rated(terrain);
}
```

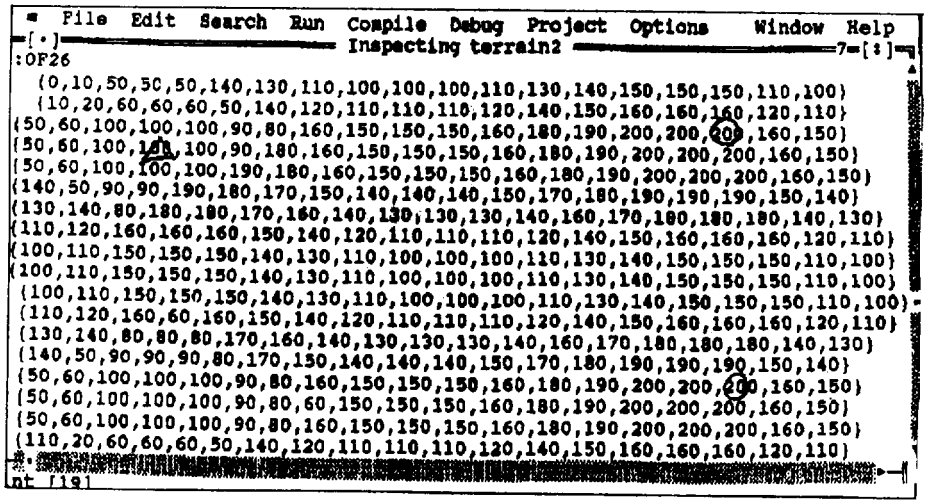
다음 수를 생성하는 절차는 다음과 같다. 상대방이 놓은 돌에 대해서 프로그램은 그 돌에 인접한 덩어리나 대마가 있는지를 체크한 후, 인접한 빈자리들에 대한 영향력을 계산하고, 사활 루틴을 수행한다. 만약, 인접한 덩어리나 대마가 없다면 붙임루틴을 수행하고, 그렇지 않으면 공격 루틴을 수행하게 된다. 공격이 여의치 않으면 방어루틴을 수행하지만 더 이상 수행할 루틴이 없으면-프로그램이 놓을 자리가 없으면-바둑 반면의 또다른 덩어리나 대마에 대해서 위의 절차를 반복하여 수행하게 된다. 반면의 형세에 대한 갱신은 알고리즘4로 수행한다.

【알고리즘 4】 현 반면의 형세를 갱신하는 루틴

```
void Update_Territory(P_List_type **move)
{
  short(*weights)[board_size];
  int x, y;

  for (x = 0; x < board_size; x++) {
    for (y = 0; y < board_size; y++) {
      if (weights[x][y] == 0 && Legal_Move(x, y)
      && !Suicidal(x, y)
        weights[x][y] = terrain[x][y] + 100;
      else
        weights[x][y] = 100/abs(weights[x][y])
        + terrain[x][y];
    }
  }
  *move = (P_List_type *)Highest_Rated(weights);
}
```

수를 생성하는데 있어서 공통적인 사항은 어떤 루틴을 수행하고 나면, 바둑 반면 전체에 대한 형세 계산이 이루어지는데 전역 변수인 terrain을 갱신한 후, 그중에서 최고의 값을 선택한다. 그림 2는 C 컴파일러에서 디버거를 통하여 terrain의 값을 출력해 본 것



(그림 2) 다음 수 생성을 위한 형세도
(Fig. 2) Terrain map for next move generation

으로, △표시는 사람이 둔 수이고, ○표시는 프로그램이 생성한 수이다.

5. 성능 평가

컴퓨터 바둑 프로그램의 성능을 평가하는 기준은 세계 대회 우승자와의 평가 대국을 통하여 기력을 검증하는 것이다. 본 논문의 프로그램은 IBM-PC를 사용하여 C언어로 구성하였다.

그림 3은 95년 세계 대회 우승 프로그램인 Handtalk (백)과의 평가 대국으로서 31수가 진행되기까지의 포석 단계에서 프로그램은 우상귀를 Handtalk에게 내주는 바람에 형세가 매우 약하게 되었다. 흑23은 〈O, 4〉보다는 〈P, 4〉로 흑 9와 연결하는 것이 올바른 수순이다. Hantalk이 흑9와 흑23사이에 백 24를 끼움으로서 프로그램은 우상귀의 근거를 빼앗기는 악수를 두었다. 백24에 대해서는 흑25로 맞단수하기 보다는 〈P, 5〉로 양단수를 하는 것이 귀 대신 중앙의 세력을 발전시킬 수 있는데 프로그램은 우상귀에 집착하다가 결국 백에게 내주고 말았다.

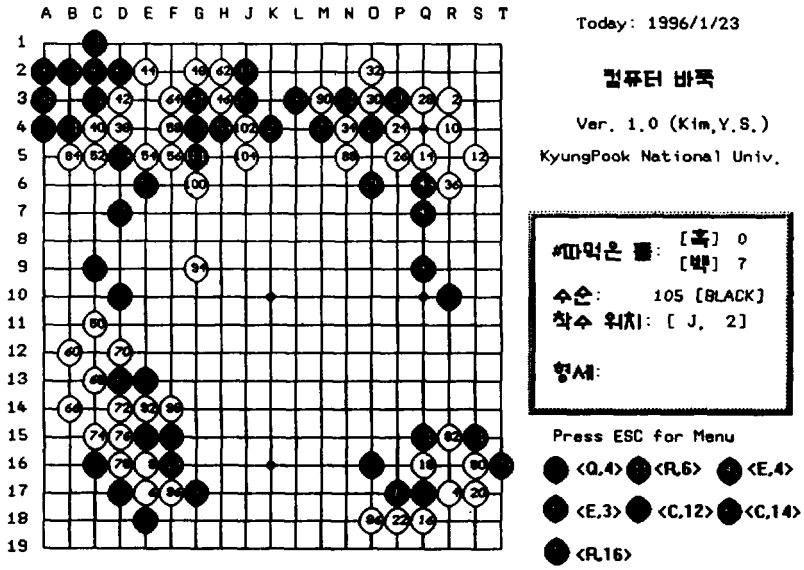
또한, 흑 63은 흑 61과 연결하려는 수단이나 백이 〈F, 3〉으로 차단함으로서 오히려 백을 도와주는 패착이었다. 특히, 프로그램은 좌변에서 흑53과 흑57로 백 50을 협공하였으나 백60에 대해서 〈D, 12〉로 응수하지 않는 바람에 백68의 끼움을 허용하고 흑 69가 무모하게 되어 좌변의 흑 일단이 곤마가 되고 말았다.

좌상귀의 흑은 백40을 차단시키지 못함으로서 두집 만들기에 급급하여 흑85, 흑87, 흑89, 흑91, 흑93 등 다섯수를 보강하였으나 사실상 흑 91과 흑 93은 지키지 않아도 될 착점이었다. 이후, 프로그램은 Handtalk과의 형세가 너무 차이가 나서 더 이상 경기를 진행하지 못하였지만 붙임이나 끼움에 대해서는 만족할 만한 성과를 얻었다.

따라서 향후 정석 라이브러리를 더 많이 구축하고, 형세 계산 함수를 개선한다면 좋은 프로그램으로 발전할 수 있을 것이다.

6. 결 론

국제 경진 대회가 생겨나면서 전세계적으로 컴퓨터 바둑에 대한 학문적 관심이 고조되고 있다. 그러나 바둑은 규칙의 간단함에 비해서 수 이동에 대한 변화가 많고 양자간에 점령해야 할 중립적인 지역을 두고 서로의 이해 대립이 불가피한 전면전의 성격을



(그림 3) 프로그램 평가 대국 : 105수
(Fig. 3) Example Game : Move 105

가지기 때문에 특정한 함수 몇개로 컴퓨터 바둑을 구현하기는 매우 어렵다.

본 논문에서는 컴퓨터 바둑의 국제 경진 대회를 소개하고, 컴퓨터 바둑을 연구하는 프로그래머들의 접근 방법 및 프로그램의 특징을 분석하고 이를 컴퓨터로 구현하기 위한 자료 구조와 알고리즘을 제시하고 실전 대국을 통하여 프로그램을 평가하였다.

여러 연구자들의 컴퓨터 바둑에서 발견할 수 있는 공통점은 반면의 형세를 평가하기 위한 영향력 산출 함수를 사용하거나, 부분적인 알파-베타 탐색을 수행한다는 사실이다. 또한, 축이나 사활과 같은 부분적인 전술은 휴리스틱한 탐색을 적용하고, 고도의 사고력을 요구하는 수 생성 및 연결성 측정은 대부분 패턴과 바둑 격언 등의 데이터 베이스를 구축하여 추출하고 있는 것으로 요약할 수 있다.

현재 본 연구실에서는 패턴과 사활에 대한 기계 학습을 통하여 최적의 수를 생성하는 실험을 수행하고 있다. 향후 연구로는, 우수한 컴퓨터 바둑을 만들기 위해서 반면에 대한 형세를 정확히 계산하고, 호착점(good move)을 예견할 수 있는 알고리즘의 개발에 주력해야할 것이다.

참 고 문 헌

[1] David B. Benson, "Life in the Game of Go," Information sciences vol.10, American Elsvier Publishing company, Inc., pp. 17-29, 1976.

[2] 김영상, 이종철, "바둑 게임의 이론적 접근," 한국정보과학회 전문대학 전산교육 논문지, Vol. 1, No 1, 1993.

[3] W. Reitman and B. Wilcox, "The Structure and Performance of the INTERIM 2 Go Program," Proceedings of the 6th International Joint Conference on Artifical Intelligence(Tokyo, Augest 20-23 1979), pp. 711-719.

[4] Bruce Wilcox, "Reflections on Building Two Go Programs," SIGART Newsletter, Bolt Beranek & Newman Inc. 1985.

[5] W. Reitman and B. Wilcox, "Pattern Recognition and Pattern-Directed Inference in a Program for Playing Go," Pattern Directed Inference Systems, Academic Press Inc.

[6] W. Reitman, J. Kerwin, R. Nado, J.Reitman, and B.Wilcox, "Goals and Plans in a programs for playing Go," Proceedings of the ACM Annual Conference, sandiago, 1975.

[7] Ander Kierulf, Ken Chen, and Jurg Nievergelt, "Smart game board and Go explorer:A Study in Software and Knowledge Engineering," Communication of the ACM, Edmonton Canada, 1990.

[8] D. Fotland, "Knowledge Representation in the Many Faces of Go," Manuscript available by Internet anonymous ftp from bsdserver.ucsf.edu, 1993.

[9] N. Sanechika(AIR), "Go Generation, A Go Playing System," ICOT Technical Report:TR-545, 1990.

[10] Shirayanagi, K., "A new approach to programming Go-Knowledge representation and refinement," Proceedings of the Workshop on New Directions in Gametree Search, Edmonton, Canada, pp. 28-31, 1989.

[11] Liu Dong-Yeh, Hsu Shun-Chin, "The Design and Construction of the Computer Go Program Dragon II," Computer Go, No. 10, 1988.

[12] D. Stoutamire, "Machine learning applied to Go," Master's thesis, Case Wastern Reverse University, Manuscript available by Internet anonymous ftp from bsdserver.ucsf.edu, 1991.

[13] 김영상, "동적 환경에 적절한 확률론적 형세 평가의 모델링에 관한 연구," 공학 석사 학위 논문, 경북대학교, 1992.

[14] 유정혜, "AND/OR 신장 트리를 이용한 바둑 사활문제 풀이에 관한 연구," 경북대학교 공학 석사 학위논문, 1991.

[15] 이두한, "The Representation of Computer Components with Object-Oriented Concept," Second Game Programming Workshop, 1995.

[16] 한국 기원, 월간 바둑, (재)한국 기원, No. 341, 1995.

[17] Dave Dyer, "Searchs, tree pruning and three

ordering in Go," Second Game Programming Workshop, 1995.

김 영 상

1990년 울산대학교 전자계산학과 졸업(학사)
1990년~1991년 현대전자(주) 산업전자연구소에서 근무
1993년 경북대학교 대학원 컴퓨터공학과 졸업(석사)
1996년 경북대학교 대학원 컴퓨터공학과 박사 수료
1993년~현재 한라전문대학 전산정보처리과 전임강사
1995년~현재 한라전문대학 전자계산소장
관심분야:소프트웨어 공학(O.O.D), 컴퓨터 바둑, Graph Theory

이 종 철

1978년 서울대학교 전자공학과 졸업(학사)
1980년 한국과학기술원 전산학과 졸업(석사)
1989년 6월~1990년 5월 캐나다 워털루 대학 교환 교수
1993년~현재 한국과학기술원 전산학과 박사과정
1980년~현재 경북대학교 컴퓨터공학과 부교수
관심분야:프로그래밍 기법, 4색 문제, Graph Embedding