인지과학, 제 14 권 제 2 호, 2003년

DNA 컴퓨팅을 이용한 원숭이와 바나나 문제 해결*

Solving the Monkey and Banana Problem Using DNA Computing

박 의 준**,*** 이 인 희*** 장 병 탁**,***
(Eui-Jun Park) (In-Hee Lee) (Byoung-Tak Zhang)

요 약 원숭이와 바나나 문제는 인공지능과 관련된 여러 문헌에서 문제 해결(problem solving) 과정을 설명하는 예제로 자주 등장한다. 그러나 이 문제에 대한 전통적인 접근 방식은 추론을 수행함에 있어 절차적(procedural) 관점의 도입을 필요로 하며, 이는 복잡한 문제 해결에 제약 조건으로 작용한다. 그러나 대규모 병렬 처리가 가능한 DNA 컴퓨팅 기법을 이용하면, AI 본연의 의미를 퇴색시키지 않고서도 이 문제를 효과적으로 해결할 수 있다. 본 논문에서는 DNA 분자를 사용해서 원숭이와 바나나 문제를 표현하는 방법을 설계한 후, 컴퓨터 시뮬레이션을 통해 다양한 해들(solutions)이 생성됨을 확인하였다. 전통적인 방법으로 구현된 Prolog 프로그램이 단 하나의 최적해밖에 제공해 주지 못한다는 사실과 비교해 볼 때, 이것은 확실히 흥미로운 결과이다.

주제어 원숭이와 바나나 문제, 논리적 추론, DNA 컴퓨팅

Abstract The Monkey and Banana Problem is an example commonly used for illustrating simple problem solving. It can be solved by conventional approaches, but this requires a procedural aspect when inferences are processed, and this fact works as a limitation condition in solving complex problems. However, if we use DNA computing methods which are naturally able to realize massive parallel processing, the Monkey and Banana Problem can be solved effectively without weakening the fundamental aims above. In this paper, we design a method of representing the problem using DNA molecules, and show that various solutions are generated through computer-simulations based on the design. The simulation results are obviously interesting in that these are contrary to the fact that the Prolog program for the Monkey and Banana Problem, which was implemented from the conventional point of view, gives us only one optimal solution. That is, DNA computing overcomes the limitations of conventional approaches.

Keywords Monkey and Banana Problem, Logical Inference, DNA Computing

1. 서론

DNA 컴퓨팅은 DNA 분자들을 사용해서 정보 처리를 수행하려는 시도이다[1]. 이것의 역사는 1994년 에이들만(L. M. Adleman)이 DNA를 이용하여 해밀토니안 경로 문제(Hamiltonian Path Problem)를 해결한 것[2]에서부터 사실상 시작되며, 이후 지금까지 계산학적으로 난제인 NP-완전(NP-complete) 문제들의 풀이에 관해 특히 많은 연구들이 있어 왔다[3]. 왜냐하면 DNA 컴퓨팅은 그것의 가장 큰 특징인 대규모 병렬 처리(massive parallel processing)로써 주어진 NP-완전 문제의 상태 공간(state space) 내 모든 상태들을 탐색하는 것을 가능하게 해 주는 장점을 갖고 있기 때문이다. 관련된 연구로서, DNA 컴퓨팅의 이러한 특성을 버전 공간(version space) 학습[4]이나 명제 논리 및 삼단 논증 정리

* 본 연구는 산자부 차세대 신기술 사업 수퍼 지능칩 과제의 Molecular Evolutionary Computing(MEC)에 의해 지원되었음.
** 서울대학교 대학원 인지과학 협동과정
*** 서울대학교 컴퓨터공학부 바이오지능 연구실
연구 세부분야: DNA 컴퓨팅 / 바이오지능
주소: 151-742 서울특별시 관악구 신림동 산 56-1
서울대학교 공과대학 컴퓨터공학부 바이오지능 연구실
전화: 02)880-1847 FAX: 02)875-2240
전자우편: {ejpark, ihlee, btzhang}@bi.snu.ac.kr

(theorem) 증명[5, 6, 7]에 이용한 연구 결과들이 발표된 바 있다.

한편, 원숭이와 바나나 문제(Monkey and Banana Problem)는 상식적인 추리를 수행하는 자동 문제 풀이기(automatic problem solver)의 동작을 설명하기 위해 인공지능(AI)과 관련된 여러 문헌에서 자주 언급되는 문제이다. 그런데 고전적인 AI의 입장에서는 근본적으로 모든 문제에 대해 순차적(sequential) 혹은 절차적(procedural)으로 접근할 수밖에 없다.

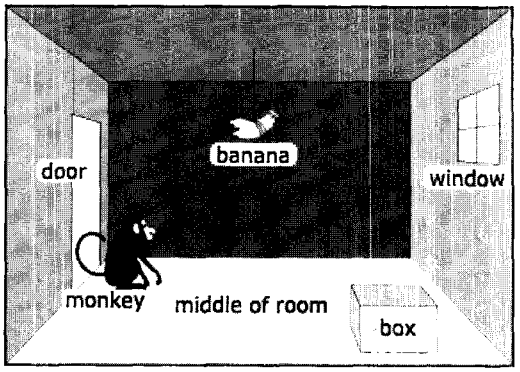
이에 본 논문에서는 먼저 원숭이와 바나나 문제에 대한 고전적 AI의 해결 방식을 살펴보고, 그것이 갖는 근본적인 어려움, 즉 절차적 관점의 도입에 대해 토론한다. 그리고 이 문제에 대해 DNA 컴퓨팅을 이용한 새로운 접근 방식을 제시함으로써, 절차적 관점에서 벗어난 추론 과정의 구현이라는 측면에서 우리의 방식이 인공지능을 추구하는 원래의 목적에 좀 더 충실한 시도라는 점을 보이고자 한다. 또한 이 새로운 방식은 사람의 행동과 유사하게, 최적해 외에도 다양한 해법들을 생성해 낼 수 있다는 사실을 이끌어 냄으로써, 일반적인 문제 풀이 과정에 존재하는 어떤 인지적인 특성을 포착하고자 한다. 이것은 물론 원숭이와 바나나 문제의 목표 자체를 기존의 시각과는 다른 관점에서 설정하고, 여기에 병렬 처리 기반의 접근 방식을 적용한 데에 결정적으로 의존한다.

단, 제반 여건 상 DNA 분자들을 사용한 실제 실험은 추후로 미루고, 대신 컴퓨터 시뮬레이션을 통해 결과를 확인하였음을 밝힌다. 물론 현재의 DNA 컴퓨팅 기술 수준을 고려할 때, 시뮬레이션만으로 실제 결과를 정확하게 예측하는 것은 사실 무리이다. 그러나 다행히 몇몇 선행 연구들에서 본 논문의 시뮬레이션과 절차상으로 유사한 실험을 실제로 설계하여 성공적으로 결과를 얻은 바가 있고[2, 4, 5], 이러한 경험은 본 논문의 결과에 대해서도 어느 정도의 신뢰성을 제공해 준다고 기대한다.

2. 원숭이와 바나나 문제

원숭이와 바나나 문제는 다음과 같다:

어떤 방에 원숭이, 상자, 그리고 바나나가 있다. 바나나를 얻기 위해 원숭이는 어떤 동작들을 수행하여야 하는가? 즉 원숭이는 상자 있는 곳으로 가서(walk), 상자를 바나나 있는 곳까지 밀고(push), 상자 위로 올라가(climb), 바나나를 잡아야(grasp) 하는 것이다.



(그림 1) 원숭이와 바나나 문제

2.1. 고전적 AI(Symbolic AI)에서의 문제 표현 및 해결

이 문제가 가정하고 있는 세계의 각각의 가능한 상태는 다음과 같이 벡터로 표현된다:

(*w, x, y, z*)

(단, 여기서 w는 원숭이의 위치(2차원), x는 원숭이가 상자 위에 있는가 여부(0 또는 1), y는 상자의 위치(2차원), z는 원숭이가 바나나를 가졌는가 여부(0 또는 1)이다.)

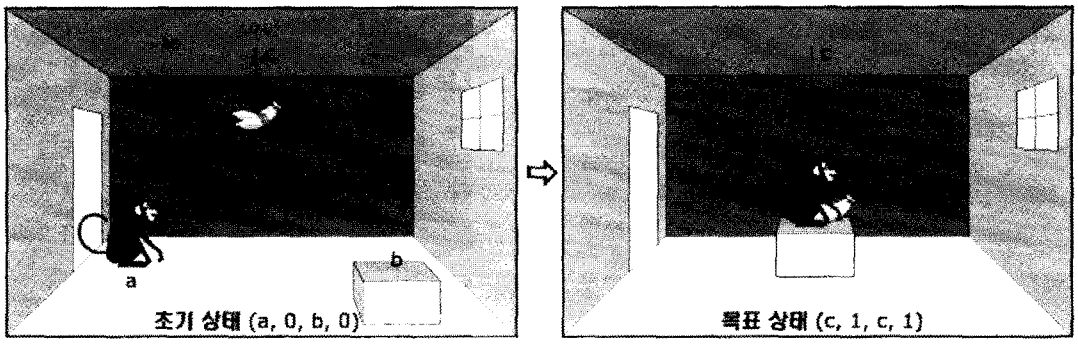
이제 이 벡터에 대해 다음과 같은 4개의 연산을 정의함으로써 원숭이의 행동을 나타낼 수 있게 된다:

① *walk*(u): $(w, 0, y, z) \rightarrow (u, 0, y, z)$
② *push*(v): $(w, 0, w, z) \rightarrow (v, 0, v, z)$
③ *climb*: $(w, 0, w, z) \rightarrow (w, 1, w, z)$
④ *grasp*: $(c, 1, c, 0) \rightarrow (c, 1, c, 1)$

따라서 초기 상태와 목표 상태는 (그림 2)와 같다. 여기서 원숭이, 상자, 바나나의 초기 위치 조건은 각각 a와 b와 c이다.

원숭이와 바나나 문제 풀이에 있어서 가장 중요하게 고려해야 할 사항은 원숭이나 상자의 위치를 나타내는 무한한 값(2차원 벡터)들을 어떤 식으로 해결할 것인가 하는 점이다. 이에 대해서는 두 가지 방법을 생각해 볼 수 있다:

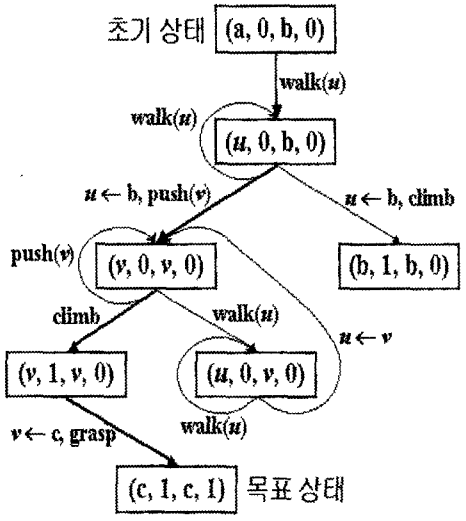
방법 (1) 방을 일정한 간격으로 등분함으로써 위치들의 집합을 유한하게 만든다. 그 결과 우리는 유한한 상태 공간을 얻게 된다. 그러나 적당히 실용적인 간격을 취한다 해도 상태 공간은 지극히 커질 수 있다.



(그림 2) 초기 상태와 목표 상태

방법 (2) 상태를 표현하기 위해 틀 변수(schema variable)를 사용한다. 다시 말해, 위치를 나타내는 상수(constant)는 변수(variable)로 대치될 수 있고, 변수는 또한 다른 변수로 대치되거나 혹은 상수로 예화(instantiation)될 수 있다.

고전적 AI에서는 어떤 문제에 대해서도 항상 순차적으로 접근하므로, 방법 (1)의 적용은 현실적으로 힘들다. 왜냐하면 상태 공간의 크기가 커짐에 따라 목표 상태에 이르는 풀이 경로(solution path)의 길이는 급격하게 증가하게 되기 때문이다. 따라서 틀 변수의 사용만 지원이 된다면, 방법 (2)를 적용시키는 편이 보다 바람직하다. 방법 (2)의 적용 하에서 상태 공간의 탐색을 통한 풀이 경로는 다음 (그림 3)에서의 굵은 화살표로 표시된다:



(그림 3) 원숭이와 바나나 문제의 그래프

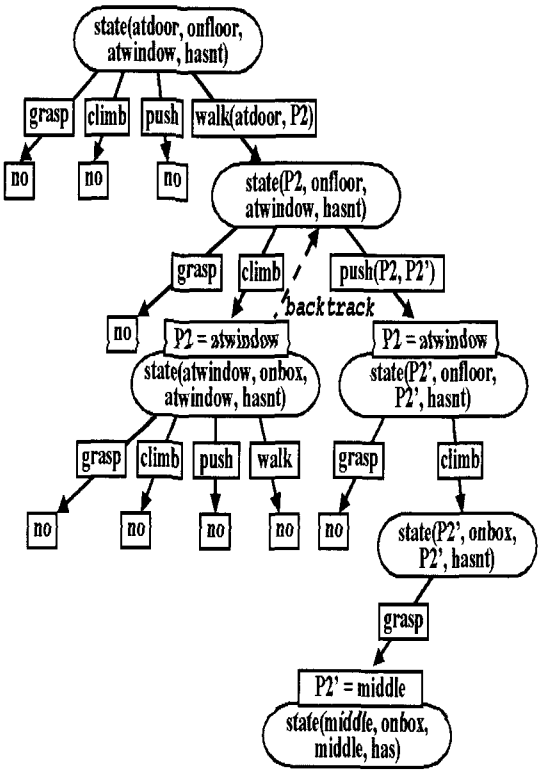
2.2. Prolog를 사용한 문제 표현 및 해결

Prolog는 LISP와 더불어 고전적 AI 연구자들이 즐겨 이용하는 대표적인 언어이다. 이것은 주어진 문제에 대해 선언적(declarative) 접근, 즉 목표 지향적(goal-oriented) 프로그래밍을 가능하게 하므로, 원숭이와 바나나 문제를 해결하는 데 매우 적합한 도구이다. 나아가 Prolog의 추론 기제는 기본적으로 패턴 매칭(pattern matching)과 자동 백트래킹(backtracking)에 기반하고 있으므로, 구현된 Prolog 프로그램은 제 2.1절 방법 (2)에서의 틀 변수의 사용을 만족스럽게 지원한다. Prolog 프로그램의 소스 코드는 다음과 같다([8], p.55):

```
% move( State1, Move, State2): making Move in State1 results in State2:
%     a state is represented by a term:
%     state( MonkeyHorizontal, MonkeyVertical, BoxPosition, HasBanana)

move(state( middle, onbox, middle, hasnot), % Before move
     grasp,                                 % Grasp banana
     state( middle, onbox, middle, has) ). % After move

move( state( P, onfloor, P, H),
     climb,                                 % Climb box
     state( P, onbox, P, H) ).

move( state( P1, onfloor, P1, H),
     push( P1, P2),                         % Push box from P1 to P2
     state( P2, onfloor, P2, H) ).

move( state( P1, onfloor, B, H),
     walk( P1, P2),                         % Walk from P1 to P2
     state( P2, onfloor, B, H) ).

% canget( State): monkey can get banana in State

canget( state( _, _, _, has) ).             % can 1: Monkey already has it

canget( State1) :-                          % can 2: Do some work to get it
   move( State1, Move, State2),             % Do something
   canget( State2).                         % Get it now
```

(그림 4) 원숭이와 바나나 문제 프로그램

여기서는 (그림 2)에서와 같이 목표 상태를 state(middle, onbox, middle, has)로 가정하고 있다. 이 프로그램은 "?-canget(state(atdoor, onfloor, atwindow, hasnot))."라는 질의에 대해 "Yes"라고 대답한다. 즉, 이것은 초기 상태 state(atdoor, onfloor, atwindow, hasnot)로부터 목표 상태 state(middle, onbox, middle, has)에 이르는 경로가 존재함을 추론해 낸 것이다. 이 때, 추론해 낸 경로는 최단 경로이며, 사실상 이 프로그램은 최단 경로밖에 찾을 수 없다. 이 과정을 그림으로 표현하면 (그림 5)와 같다:



(그림 5) Prolog로 구현된 경우의 추론 과정([9]에서 발췌)

2.3. 고전적 AI의 해결 방식이 갖는 근본적인 어려움

앞서 제 2.2절에서는 고전적 AI의 관점에 충실하게 상태 공간의 순차적인 탐색을 통해 목표 상태에 도달할 수 있었다. 그런데 실제로 Prolog 프로그램이 성공적으로 수행되는 데에는 그것이 갖는 선언적 측면 이외에 프로그램을 구성하고 있는 절(clause)들의 순서가 결정적인 역할을 한다. 예컨대 'walk'가 포함된 절이 가장 처음에 나오도록 앞의 프로그램을 수정하면, 이 수정된 프로그램은 무한 루프에 빠지게 됨을 쉽게 알 수 있다. 즉, 이 경우 불쌍한 원숭이는 방 안을 무한정 이리저리 헤매게 되는 것이다.

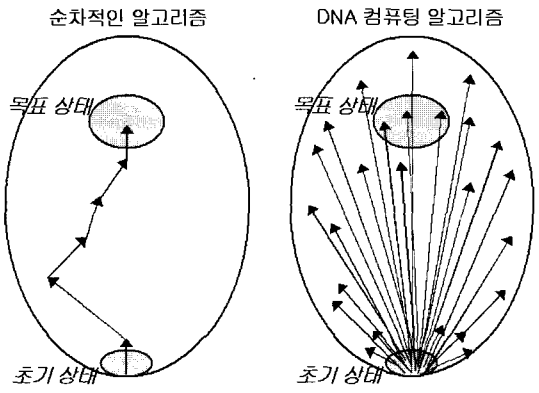
선언적 관점 외에 절차적 관점을 필요로 한다는 것은 우리가 인공지능을 추구하는 본연의 목적을 충족시키지 못함을 의미한다. 왜냐하면 선언적 언어인 Prolog를 사용하여 문제를 해결하려는 시도가 갖는 근본적인 의의는 사람은 문제를 해결하기 위해 필요한 지식들만을 제공하고, 실제 추론은 인공적인 추론 기관이 담당하게끔 하는 데 있기 때문이다. 다시 말해, 절차적 관점의 도입은 *추론 과정에 인간의 사고 작용을 필연적*으로 개입시키게 되는데, 이것은 바로 고전적 AI의 의미를 퇴색시키는 일인 것이다. 그리고 적어도 원숭이와 바나나 문제의 경우, 이것은 순차적 접근이라는 틀을 유지하면서 제 2.1절의 방법 (2)를 적용시키려는 고전적 AI 일반의 한계이다. 결코 제 2.2절에서 소개된 Prolog 프로그램만의 문제가 아닌 것이다. 물론 고전적 AI에도 절들의 순서에 구애받지 않고 무한 루프를 배제시키는, 일반적이고 신뢰할만한 여러 방법들이 있다([8], p.58). 그러나 각종 테크닉에 의존하는 이런 방법들이 아닌, 보다 근본적인 해결책은 과연 존재하지 않는 것일까?

본 논문의 문제 의식은 바로 이 점에 있다. 에이들만(L. M. Adleman)이 [2]에서 DNA 컴퓨팅을 이용한 해결책을 제시했던 것은 근본적으로 헤밀토니안 경로 문제의 경우, 방문해야 할 점(point)들의 개수가 증가함에 따라 가능한 경로의 수가 폭발적으로 증가한다는 사실에 기인한다. 반면 원숭이와 바나나 문제는 이런 계산학적인 관점과는 다소 다른 시각에서 DNA 컴퓨팅의 새로운 의미를 드러나게 해 준다. 그것은 인공지능을 구현하기 위한 바람직한 방법론에 관한 문제 제기이다.

2.4. DNA 컴퓨팅을 이용한 접근 방식의 필요성 및 기술적 난점

고전적 AI가 제 2.1절의 방법 (1)을 고려하지 않은 것은 계산학적인 이유에서이다. 실용적인 관점에서는 무한 집합을 다룰 수 없다는 것은 심각한 문제가 아니다. 중요한 것은 입도(granularity)이므로, 필요한 만큼 얼마든지 큰 유한 집합으로 대신하면 되기 때문이다. 이 맥락에서 정말로 중요한 문제는, 유한하지만 매우 큰 상태 공간을 탐색하는 데 걸리는 비용—시간과 메모리이다.

DNA 컴퓨팅 기법은 이 문제를 해결할 수 있는 가능성을 제공한다. DNA 컴퓨팅은 기본적으로 대규모 병렬 처리이므로, 원숭이와 바나나 문제의 상태 공간의 크기와 무관하게 상태 공간 내 상태들을 동시에 탐색할 수 있다. 이러한 점은 다음의 (그림 6)으로 쉽게 설명될 수 있다[10].



(그림 6) 순차적인 알고리즘과 DNA 컴퓨팅 알고리즘의 비교

그러나 DNA 컴퓨팅 기법으로 제 2.1절의 방법 (2)를 사용하는 것, 즉 상수나 변수에 대한 대치와 예화를 구현하는 것은 현재의 기술 수준으로는 곤란하다. 방법 (2)를 사용하기 위해서는 무엇보다도 DNA 시퀀스에 대한 임의의 돌연변이(mutation) 연산을 필수적으로 요구하는데, 이는 지금으로서는 부분적으로 가능하기 때문이다. 따라서 본 논문에서는 DNA 컴퓨팅 기법에 기반한 방법 (1)을 사용하여 원숭이와 바나나 문제를 해결하고자 한다. 이 방법은 Prolog를 사용한 방식과는 달리, 가능한 유한 개의 모든 상태들과 그에 따른 연산들을 DNA로 미리 코딩(coding)해 놓는 대신에 추론 과정에는 전혀 관여하지 아니한다. 물론 실제적인 비용의 측면에서 볼 때, 현재로서는 Prolog를 사용한 방식보다 바람직하지 않다는 점은 명백하다. 그러나 오히려 인공지능 본연의 의미를 추구하는 데에는 더 적합한 접근 방식이라고 생각한다.

3. DNA를 이용한 문제 표현— 추상적인 관점

본 논문에서는 유한한 상태 공간을 유지하기 위해, 원숭이의 위치와 상자의 위치가 취할 수 있는 값을 '창 옆(atwindow)', '방문 옆(atdoor)', '방 가운데(middle)'의 3 가지로 제한하기로 한다. 앞 절에서 말했듯이, 상태 공간의 크기는 만약 필요하다면 얼마든지 늘일 수 있으므로, 이러한 식의 제한이 문제 풀이의 일반성을 잃게 만드는 것은 아니다. 또 상태들의 개수를 줄이기 위해, 원숭이가 바나나를 가졌는지 여부는 무시하기로 하자. 즉, 초기 상태에서 원숭이는 당연히 바나나를 갖고 있지 않고, 이것은 목표 상태에 이르기 직전까지 마찬가지이므로, 우리는 목표 상태 직전의 상태 (middle, onbox, middle, hasnot)를 새로운 목표 상태로 간주하여, 초기 상태로부터 이 새로운 목표 상태에 이르는 것으로 원래의 문제를 재구성할 수 있다. 이렇게 되면, 상태 벡터의 4 번째 요소는 불필요하게 되므로 모든 상태들의 개수는 12 개가 된다(원숭이가 방바닥 위에 있는 경우: 9 (= 3 × 3)개, 상자 위에 있는 경우: 3개). 또 walk 연산은 방바닥 위(onfloor)에서만 가능하므로 모두 18 (= 9 × 2)개이고, push 연산은 원숭이의 위치와 상자의 위치가 동일할 때 방바닥 위에서만 가능하므로 6 (= 3 × 2) 개이며, climb 연산은 원숭이의 위치와 상자의 위치가 동일할 때 상자 위(onbox)에서만 가능하므로 3 개이다. 이제 grasp 연산은 당연히 불필요하다.

하나의 상태에 대한 DNA 시퀀스는 기본적으로 3 개의 단위를 필요로 한다. 예컨대 상태 (atdoor, onfloor, atwindow)는 (그림 7)과 같이 나타낼 수 있다:

atdoor	onfloor	atwindow

(그림 7) 상태에 대한 DNA 시퀀스

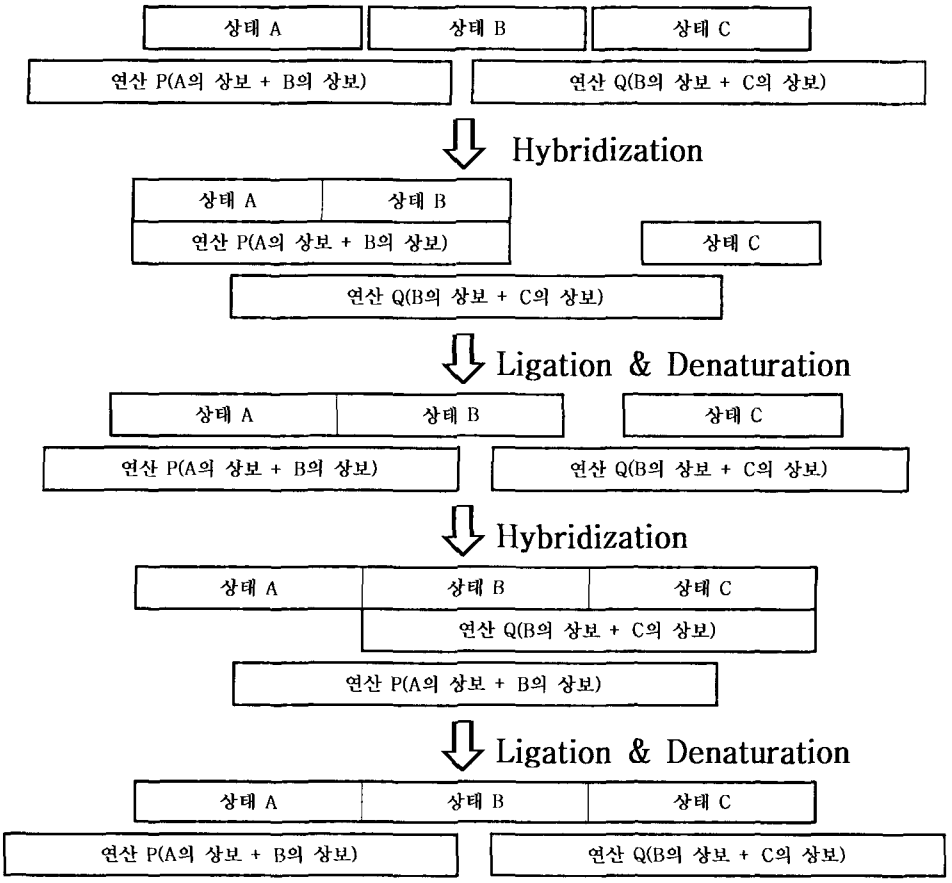
연산은 두 개의 상태를 매핑(mapping)시켜 주는 역할을 하므로 6 개의 단위를 필요로 한다. 예컨대 상자가 창 옆에 있을 때 원숭이가 방문 옆에서 방 가운데로 걸어가는(walk) 연산은 다음 (그림 8)과 같이 나타낼 수 있다(단, '^ atdoor', '^ onfloor', '^ atwindow', '^ middle'는 각 'atdoor', 'onfloor', 'atwindow', 'middle'의 상보적 염기 서열을 뜻한다). 즉, 하나의 연산에 대한 시퀀스는 하나의 상태를 나타내는 그것보다 2 배로 길고, 따라서 그 연산의 가능한 논항 쌍에 대응하는 두 시퀀스와 완벽한 Watson-Crick 상보 결합을 이룰 수 있다.

이와 같은 방법으로 원숭이와 바나나 문제를 해결하기 위해 요구되는 지식들을 추상적인 수준에서 DNA로 표현해 내었다. 이제 남은 것은 추론 과정을 거쳐 실제로 문제를 해결하는 일이다. 이 과정을 도식적으로 나타내면 (그림 9)와 같다:

상태 A를 상태 B로 전이시키는 연산 P가 존재하는

^ atdoor	^ onfloor	^ atwindow	^ middle	^ onfloor	^ atwindow

(그림 8) 연산에 대한 DNA 시퀀스



(그림 9) 문제 해결 과정-- 추상적인 관점

경우 Hybridization을 통해 P에 해당하는 연산-시퀀스를 매개로 하여 두 상태-시퀀스는 서로 인접하게 되며, Ligation 및 Denaturation을 거쳐 두 상태-시퀀스는 연결된다. 유사하게, 상태 B를 나타내는 시퀀스는 상태 C를 나타내는 그것과 다시 연결될 수 있다. 이러한 과정을 여러 번 반복함으로써 상태들의 연속체를 표현하는 DNA 시퀀스들이 생성된다. 이제 이렇게 생성된 시퀀스들 중 한쪽 끝은 초기 상태를, 그리고 다른 쪽 끝은 목표 상태를 나타내는 것들만이 우리가 원하는 해가 된다. 따라서 해를 도식적으로 나타내면 (그림 10)과 같다:

상태 1	상태 2	상태 3	·····	상태 N-1	상태 N

(그림 10) 최종 해에 대한 DNA 시퀀스

단, 여기서 초기 상태인 상태 1과 목표 상태인 상태 N은 각각 (그림 11)의 위 그림, 아래 그림과 같다:

atdoor	onfloor	atwindow	상태 1
middle	onbox	middle	상태 N

(그림 11) 초기 상태, 목표 상태에 대한 DNA 시퀀스

4. DNA 컴퓨팅을 이용한 문제 해결

4.1. DNA 시퀀스 코딩 방법

어떤 상태에 대한 DNA 시퀀스를 실제로 코딩하기 위해 다음 두 가지 방법을 생각해 볼 수 있다: (1) 상태 벡터를 구성하는 각각의 성분들을 DNA 시퀀스로 나타내고, 이러한 성분-시퀀스들의 연속체로 하나의 상태를

표현해 낸다(제 3절의 표현 방법). 곧, 원숭이 및 상자의 위치를 나타내는 3 종류의 시퀀스들과 원숭이가 상자 위에 있는가 여부를 나타내는 2 종류의 시퀀스들을 차례로(원숭이 위치-원숭이가 상자 위에 있는가 여부-상자 위치) 연결하여 하나의 상태를 표현해 내는 방법이다. (2) 구성 성분에 관계없이, 12개의 상태들을 각각 서로 다른 DNA 시퀀스들로 나타낸다.

첫 번째 방법은 생성된 성분-시퀀스들의 연속체들, 곧 상태를 나타내는 시퀀스들이 가능한 한 서로 유사하지 않도록 애초에 각각의 성분에 해당하는 시퀀스를 디자인해야 하기 때문에, 실제 코딩이 어렵다는 단점을 갖는다. 그러나 서로 다른 상태 i, j에 공통적인 성분이 존재하는 경우, 두 상태 i, j를 나타내는 DNA 시퀀스가 유사하므로, 실제 반응 시 두 시퀀스가 비슷한 행동 양상을 보이게 되기 쉽다는 장점이 있다. 반면에 12개의 상태들을 각각 서로 다른 DNA 시퀀스들로 표현하는 방법은 디자인이 쉽다는 장점이 있으나, 동일한 성분을 포함하는 상태들 간의 유사성을 표현하기 어렵다는 단점이 있다.

상태들 간의, 곧 시퀀스들 간의 유사성은 DNA를 이용한 실제 실험에서는 반응의 효율성이란 측면에서 중요한 특징이나, 컴퓨터 시뮬레이션에서는 그다지 고려할 필요가 없다. 본 논문에서는 오직 컴퓨터 시뮬레이션을 통해 결과를 확인하고자 하므로, 두 번째 방법을 써서 DNA 시퀀스를 디자인하도록 한다. 실제로 이 작업은 서울대학교 바이오지능 연구실에서 개발한 DNA 컴퓨팅 시뮬레이터인 NACST(Nucleic Acid Computing Simulation Toolkit)[10]를 써서 수행되었다. NACST는 시뮬레이션에 사용되는 DNA 시퀀스들을 진화 알고리즘(Genetic Algorithm)을 이용해 최적화시킨다. <표 1>에는 각각의 상태를 나타내는 12 개의 시퀀스들이 정리되어 있다.

상태(상태번호)	시퀀스(5' - 3')
a 0 a (0)	GACCCTGCTG GAGAGACTAC CCCTTTAGAC
a 0 b (1)	GATCTCAGAC ACAGAATTGT GTCGACTAGG
a 0 c (2)	CTTCGTCGTC TCCGACAGGA ATGAATTGCA
a 1 a (3)	GTAGTTCATG CCACAAGCAC ACTATTTGGA
b 0 a (4)	CAACTTGAGA TTGTCACGGC TGGGCGAGGC
b 0 b (5)	GTTCGTTGCG GTTCCTCGGC CCTAAAAGTC
b 0 c (6)	GGCCTTGAAG CCGGCAAGTT AAATAAAGTG
b 1 b (7)	ATGCACTATG ATTGCGCTCT AGTCACACAC
c 0 a (8)	TCTGCGGCAG CGAGATATGT ATCATGGTAA
c 0 b (9)	CAGCAGGGGT TAGTCAGTAG CGGGGTTTTA
c 0 c (10)	TTCCTAGGCG TATGACTAGA GGAATGAGGC
c 1 c (11)	ATTTCGAGCG GCAATTCGTT CAAGATGTAT

<표 1> NACST를 이용하여 생성된 12 개의 상태-시퀀스들

4.2. 실험 설계 및 절차

DNA 분자들을 이용한 실제 실험 과정을 설계하면 대략 다음과 같다:

(1) Hybridization & Ligation : 원숭이가 이동할 수 있는 다양한 경로 생성
(2) Affinity separation : 초기 상태와 목표 상태를 모두 포함하는 경로 분리
(3) PCR : 위의 단계 (2)에 부합하는 경로 증폭

(1) 먼저 하나의 튜브 안에 가능한 모든 상태-시퀀스들(12 개)과 모든 연산-시퀀스들(27 개)을 넣고 Hybridization과 Ligation, 그리고 Denaturation을 반복적으로 수행한다. 그 결과, Hybridization과 Ligation을 통해 상태들 간의 연결이 이루어지고, Denaturation을 통해 상태-시퀀스들과 경로-시퀀스들이 다시 분리되어 다음 사이클(cycle)에서의 반응을 준비한다. 이어 다시 Hybridization 및 Ligation을 거쳐 이전 사이클에서 생성된, 연결된 상태-시퀀스들은 이제 튜브 내의 다른 상태-시퀀스들 및 경로-시퀀스들과 확률적으로 반응하게 된다. 이러한 사이클을 반복하면서 원숭이의 이동 가능한 경로들 중 상당수가 생성되게 된다.

(2) 다음 단계에서는 Affinity Separation이라는 실험 기법을 이용하여, 원숭이의 출발 상태와 목표하는 상태를 모두 포함하는 경로들만을 분리해 낸다. Affinity Separation은 Magnetic Bead에 특정한 시퀀스를 붙여서 테스트 튜브 내의 시퀀스들과 상보 결합반응을 일으킨 다음, Bead 상의 시퀀스와 결합한 시퀀스들을 그렇지 않은 시퀀스들로부터 분리해 내는 기법이다. 실제 실험에서 우리가 원하는 경로들만을 분리해 내려면, 출발 상태에 해당하는 시퀀스와 상보 결합하는 시퀀스가 부착된 Bead를 이용하여 출발 상태를 포함한 경로들을 걸러낸 후, 다시 목표 상태에 해당하는 시퀀스와 상보 결합하는 시퀀스가 부착된 Bead를 이용하여 목표 상태를 포함하는 경로들을 걸러내면 된다.

(3) 마지막으로 출발 상태를 표현하는 시퀀스와 목표 상태를 표현하는 시퀀스를 각각 Primer로 사용하여 PCR을 수행한다. 이 단계를 통해 우리가 원하는 경로들만을 (만약 존재할 경우) 증폭하게 된다.

4.3. 시뮬레이션 설계 및 결과

여기서는 제 4.2절의 실험 과정을 컴퓨터 시뮬레이션으로 구현해 보았다. 시뮬레이션 절차 및 결과는 아래와 같다.

4.3.1. Hybridization & Ligation

초기에는 12 개의 상태-시퀀스들과 27 개의 연산-시퀀스들만이 각각 균일한 농도로 존재하고, 한 차례의 Hybridization 및 Ligation을 통해, 두 개의 상태(혹은 경로)-시퀀스와 하나의 연산-시퀀스가 반응하여 하나의 긴 경로-시퀀스와 연산-시퀀스를 생성하는 것으로 가정하였다. 따라서 반응이 반복적으로 진행됨에 따라 짧은 경로-시퀀스들은 점차 줄어들고 상대적으로 긴 경로-시퀀스들이 새로이 생성될 것이다. 그리고 연산-시퀀스들의 농도에는 변화가 없을 것이다. 한 사이클 당 이러한 Hybridization 및 Ligation이 일정한 회수 ($REACTION^{h}_{MAX}$)만큼 일어나는 것으로 모델링하였다. 그리고 이 과정은 총 $CYCLE^{h}_{MAX}$ 사이클 동안 반복된다.

Hybridization 및 Ligation 과정을 자세히 살펴보면 다음과 같다: 우선 반응에 참여할 두 개의 경로-시퀀스는 현재까지 생성된 경로-시퀀스들 중에서 10 개를 무작위로 고른 다음 (중복 허용), 이 중에서 현재 가장 높은 농도를 가진 경로-시퀀스를 고르는 일을 두 번 반복하여 선택하였다. 이 방식을 사용하면 현재 가상의 튜브 안에 다수가 존재하는 경로-시퀀스들이 Hybridization 및 Ligation에 참여할 확률이 높아지므로, 실제 반응 결과에 가까운 시뮬레이션 결과를 얻게 되리라 기대한다.

위의 과정을 통해 선택된 두 경로-시퀀스는 이들과 완벽하게 상보 결합할 수 있는 연산 시퀀스가 존재하는 경우에만 Ligation을 통해 보다 긴 경로-시퀀스를 만들 수 있다고 가정하였다. 연산-시퀀스와 상보 결합할 경우, 반응 전에 선택된 두 경로 시퀀스의 수는 각각 1 씩 줄이고, Ligation을 통해 생성된, 보다 긴 경로-시퀀스의 수는 1 만큼 늘리도록 하였다. 만약 새로 생성된 경로-시퀀스가 현재까지 생성되지 않은 것이라면, 이것을 현재까지 생성된 경로-시퀀스 목록에 추가하도록 하였다.

4.3.2. Affinity Separation

여기서는 Bead에 부착된 시퀀스가 한 종류뿐이라고 가정하였다. 이 때, 가상의 튜브 내에 존재하는 모든 경로-시퀀스들은 (1) Bead에 붙은 시퀀스와 상보 결합할 수 있는 것들과 (2) 상보 결합할 수 없는 것들의 두 종류로 나뉠 수 있는데, 각각에 대하여 다음의 반응을 수행하였다:

(1)의 경우, 현재 튜브 내에 존재하는 수에 수율(yield)을 곱한 만큼의 개수를 가지도록 하였다. 그리고 (2)의 경우, 각각의 경로-시퀀스는 상태-시퀀스들의 연결로 이루어져 있으므로, 경로-시퀀스를 이루는 상태-시퀀스들 중 Bead에 붙은 시퀀스와 상보 결합할 확률이 가장 높은 부분을 찾은 다음, 그 지점에서 상보 결합할 확률 p_{hyb}을 계산하였다. 다음에, 0~1사이의 수를 임의로 생성하여 그것이 p_{hyb}보다 작으면 (1)과 같은 반응을 수행하였고, 그렇지 않으면 현재 가상 튜브 내에 존재하는 수를 0으로 만들었다.

본 시뮬레이션에서는 이러한 Affinity Separation을 초기 상태와 목표 상태에 대하여 각각 한 번씩 거치도록 하였다.

4.3.3. PCR

Primer로 시퀀스 $primer_1$(5' 방향), $primer_2$(3' 방향)가 주어진다고 가정하였다. 이 때, 가상 튜브 내의 경로-시퀀스들은 Primer와의 관계에 따라 다음과 같이 나누어진다:

(1) 경로-시퀀스의 가장 끝 부분과 Primer가 상보적인 경우
(2) 경로-시퀀스를 이루는 상태-시퀀스 중 하나와 Primer가 상보적인 경우
(3) 어느 상태-시퀀스와도 상보적이지 않은 경우

PCR 시뮬레이션 과정에서는 각각의 Primer에 대하여 위의 세 가지의 경우로 나누어 다음의 반응을 $REACTION^{PCR}_{MAX}$만큼 수행하였다. 그리고 이것을 일정한 사이클 수($CYCLE^{PCR}_{MAX}$)만큼 반복하였다.

(1)의 경우, 첫 부분과 $primer_1$이 상보적인 경우나 마지막 부분과 $primer_2$가 상보적인 경우에는 해당 경로-시퀀스의 개수를 1 증가시켰고, 나머지 경우에는 개수의 변화가 없게 하였다. 이것은 PCR 중 Extension에는 방향성이 존재하기 때문에 일정한 방향으로만 Extension이 일어나는 것을 모델링한 것이다.

(2)의 경우, 첫 부분에서 $primer_2$와 상보적인 부분까지, 또는 $primer_1$에서 상보적인 부분에서 끝부분까지에 해당하는 부분 경로-시퀀스가 생성되게 되는데, 이러한 경로-시퀀스가 이미 존재하는 경우에는 해당 시퀀스의 개수를 1 증가시켰고, 그렇지 않은 경우에는 기존의 경로-시퀀스 목록에 추가한 다음에 이것의 개수를 1 증가시켰다.

(3)의 경우, 제 4.3.2절 Affinity 과정의 Step 4와 비슷하게, 각 Primer와 가장 상보 결합할 가능성이 큰 부분 시퀀스를 찾아 (2)에서와 같이 부분 경로-시퀀스를 생성하여 처리한다. 이것은 PCR 과정 중에 발생할 수 있는 Primer와 경로-시퀀스들 간의 잘못된 반응도 모델링하기 위한 것이다.

4.3.4. 시뮬레이션 결과

본 논문에서는

$REACTION^{h}_{MAX}=50000$,

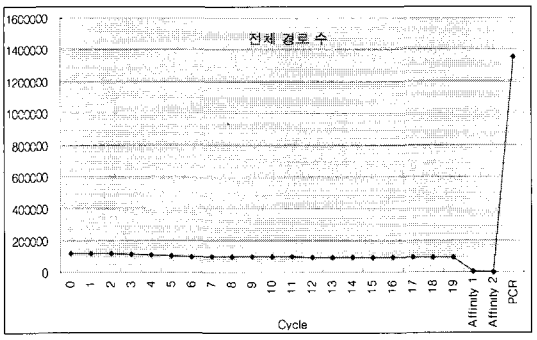
$CYCLE^{h}_{MAX}=20$,

$REACTION^{PCR}_{MAX}=50000$,

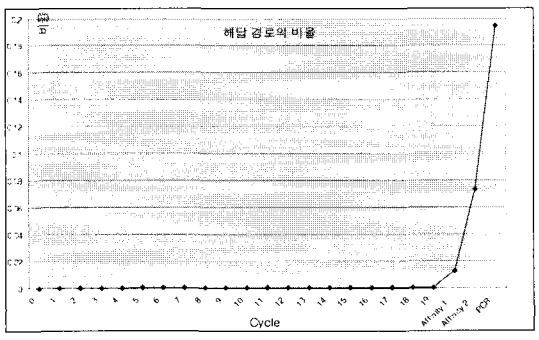
$CYCLE^{PCR}_{MAX}=20$, $YIELD=0.1$

로 초기 조건을 설정하여 시뮬레이션을 수행하였다.

우선 시뮬레이션이 진행됨에 따른 전체 경로의 개수와 해답 경로의 비율은 (그림 12) 및 (그림 13)과 같다. (그림 12)에서 전체 경로의 개수는 처음에 주어진 상태-시퀀스들의 개수를 줄곧 유지하다가 Affinity Separation을 두 번 거친 후 급격히 줄어든 다음, PCR 후 다시 증가하는 것을 볼 수 있다. 또한 (그림 13)에서 해답 경로의 비율은 Affinity 전까지는 매우 낮은 비율이었지만, Affinity Separation을 두 번 거친 후 0.08까지 증가하였고, PCR 이후에는 0.2 정도까지 증가하였다. 따라서 우리의 시뮬레이션 결과는, 계획한 실험 단계를 거치면 해답 경로의 비율이 점차 증가할 것이라는 예측을 가능하게 해 준다.

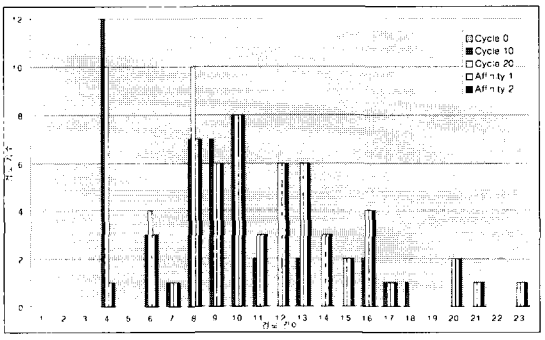


(그림 12) 전체 경로의 개수

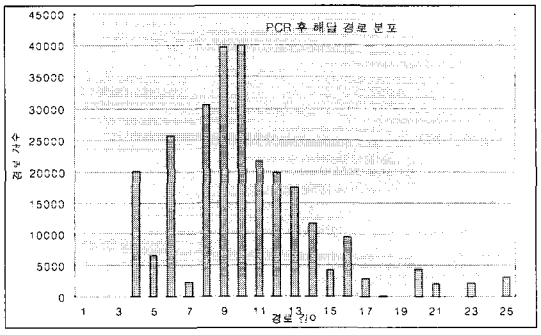


(그림 13) 전체 경로에 대한 해답 경로의 비율

(그림 14)와 (그림 15)는 각각 Affinity Separation 이후 생성된 경로들과 PCR 이후 생성된 경로들의 다양성을 나타내고 있다. 이러한 다양성은 DNA 컴퓨팅의 특징인 대규모 병렬 탐색의 결과이다. (그림 14)에서는 반응 초기(사이클 10)에 짧은 길이의 경로들이 많이 생기고, 사이클이 거듭될수록 보다 긴 길이의 경로들이 점차 발생하는 것을 볼 수 있다. 또한 (그림 15)에서는 Affinity Separation을 거치면서 상당히 적은 양으로 줄어든 짧은 길이의 해답 경로들의 개수도 PCR을 통해 많이 증가한 것을 볼 수 있으며, 해답이 아닌 경로에 한 부분으로 포함되어 있던 해답 경로(길이 25)가 새로이 생성된 것도 또한 관찰할 수 있다. 이것은 PCR 이전에는 아예 존재하지 않았던 경로이다.



(그림 14) 다양한 길이의 해답 경로들



(그림 15) PCR 후 해답 경로들의 분포

5. 결론

지금까지 DNA 컴퓨팅에 대한 시뮬레이션을 통해, 원숭이가 바나나를 얻게 되는 다양한 경로들을 발견할 수 있었다. 이것은 단순히 초기 상태로부터 목표 상태로 전

이할 수 있는가에 대한 답(Yes-No)만을 주는 것이 아니라, 목표 상태에 이르는 다양한 해법들을 제공해 준다는 점에서 제 2절에서 살펴 본 Prolog를 사용한 방식과 다르다. 그리고 이 결과는 L. M. Adleman[2]과는 달리 본 논문에서는 해집합의 조건에 대해 별다른 제약을 가하지 않은 데에서 기인한다. 왜냐하면 원숭이와 바나나 문제의 경우, 최단 경로를 찾는 일은 매우 사소하기 때문이다. 우리에게 보다 의미 있는 작업은 사람이 고안할 수 있는 다양한 해법들을 인공적인 추론 기관이 또한 생성해 낼 수 있다는 것을 보이는데 있다고 생각한다.

이와 유사하게, AI의 구현을 위한 바람직한 방법론이란 관점에서 원숭이와 바나나 문제에 대한 시뮬레이션 결과는 DNA 컴퓨팅에 또 다른 의미를 부여한다. 이 점에 대해, 앞서 제 2.3절에서 설명한 바를 한 번 더 도식적으로 정리하면 다음과 같다: 고전적 AI의 해결 방식에는 근본적으로 어려움이 있는데, 그것은 제 2.1절의 방법 (1)을 적용할 경우 계산학적인 난점에 봉착하게 되고, 방법 (2)를 적용할 경우 AI를 추구하는 본래의 목적을 충족시키지 못하게 되기 때문이다. 반면, DNA 컴퓨팅을 이용한 해결에 관해서는 현 단계에서 현실적으로 어려움이 있다. 본 논문에서처럼 방법 (1)을 사용하는 것은 개별 문제마다의 기호 구체화(Symbol Grounding)라는 점에서 실험자에게 시퀀스 디자인의 부담과 이에 따른 경제적 비용을 지우며, 방법 (2)를 사용하는 것은 현재의 기술 수준으로는 사실상 불가능하기 때문이다. 이렇듯 두 가지 해결 방식이 모두 문제점을 갖고 있으나, 논리적이나 물리적인 가능성이 아닌, 현실적인 가능성은 얼마든지 변화될 수 있기 때문에, 향후 DNA 컴퓨팅 기술이 발전하게 되면, 계산하는 데 소요되는 비용은 점점 떨어지게 될 것이고, 또 DNA 시퀀스들을 지금보다 훨씬 더 자유롭게 조작할 수 있게 되면서 기호 구체화 문제도 또한 해결되게 될 것이다.

고전적 AI, 연결주의적 AI, 그리고 DNA 컴퓨팅 등을 이용한 생물학 기반의 AI 등은 분명히 각각 특정한 영역에서 장단점을 갖고 있으며, 서로 상대방의 부족한 부분을 보완해 주는 역할을 현재까지 하고 있다. 요컨대, 지금까지 살펴본 원숭이와 바나나 문제와 같이 그래프 탐색의 그것으로 환원될 수 있는 문제의 경우, DNA 컴퓨팅을 이용한 접근 방식은 더 이상 절차적 관점을 요구하지 않는다는 점과 이로 인해 백트래킹(backtracking) 기법의 도입이나 무한 루프에 빠지는 현상에서 벗어날 수 있다는 점을 밝히고자 하는 것이 바로 우리의 논지라고 할 수 있다. 반면, 문제 해결을 목표 상태에 도달하기 위한 하위 목표(sub-goal)를 생성해 내고 이를 다시 새로운 목표 상태로 설정함으로써 진행되는 과정이라고 간주하는 입장에서는 우리의 시도가 불만족스러운 것으로 보일 수도 있다. 왜냐하면, 실제적인 성능에 있어 설사 유용성이 크다 하더라도, 이 경우 문제 자체가 갖는 고유한 인지적, 심리적인 성격의 많은 부분을 잃게 된다고 생각할 수 있기 때문이다.

이번 시뮬레이션에서는 Hybridization과 Ligation 과정 중 일어날 수 있는 오반응(Mis-hybridization)은 생각하지 않았다. 이 점을 고려한 시뮬레이션을 수행하면, DNA를 이용한 실제 실험의 경우처럼 훨씬 더 복잡하고 다양한 결과를 산출할 것으로 기대한다.

참고문헌

[1] 장병탁 (2002), 분자정보처리기술, 전자공학회지, 29-3, 286-298.

[2] Adleman L. M. (1994), Molecular Computation of Solutions to Combinatorial Problems, *Science*, 266, 1021-1024.

[3] Setubal, J. C., Meidanis. J. (1997), *Introduction to Computational Molecular Biology*, Ch. 9, PWS Pub. Co., Boston.

[4] Lim, H.-W., Yun, J.-E., Jang, H.-M., Chai, Y.-G., Yoo, S.-I., and Zhang, B.-T. (2003), Version Space Learning with DNA Molecules, *Lecture Notes in Computer Science*, 2568, 143-155.

[5] Lee, I.-H., Park, J.-Y., Jang, H.-M., Chai, Y.-G., and Zhang, B.-T. (2003), DNA Implementation of Theorem Proving with Resolution Refutation in Propositional Logic, *Lecture Notes in Computer Science*, 2568, 156-167.

[6] 박의준, 이인희, 장병탁 (2002), DNA 컴퓨팅을 이용한 삼단논증의 결론 증명, 한국정보과학회 가을 학술발표 논문집 (II), 29-2, 382-384.

[7] Wasiewicz, P., Janczak, T., Mulawka, J. J., Plucienniczak, A. (2000), The Inference Based on Molecular Computing, *International Journal of Cybernetics and Systems*, 31-3, 283-315.

[8] Bratko, I. (1990), *Prolog Programming for Artificial Intelligence*, 2nd Ed., Ch. 2, Addison-Wesley Pub. Co., Wokingham.

[9] Williams, R., *The Monkey and the Banana*, Tutorial Materials, School of Computing, University of Tasmania, Australia, http :// www.comp.utas.edu.au

/ units / kxa252 / tutorials / MonkeyBan.pdf (2002. 12. 1 현재).

[10] Zhang, B.-T., Shin, S.-Y. (1998), Molecular Algorithms for Efficient and Reliable DNA Computing, *Proceedings of the Third Annual Genetic Programming Conference* (GP-98), 735-742, Morgan Kaufmann.

접 수	2003년 1월 28일
게재승인	2003년 3월 31일