변형된 점증 깊이 우선 탐색 방법을 사용한 로봇 계획 시스템

임 재 걸 †

요 약

본 논문은 목적 상태를 구성하는 부분목적들 사이의 선취 관계를 이용하는 새로운 탐색 방법을 제안한다. 제안된 방법은 기존의 인공지능 탐색 방법에 부분 목적들간의 선취관계를 이용하여 분지인수(branching factor)와 탐색 공간의 깊이를 줄이고, 직접 성취 가능한 극대 부분 목적과 필연적인 작업들을 즉시 실행하는 전략을 사용하여 효율성을 더욱 제고시킨다. 이러한 전략을 사용하는 제안된 알고리즘의 효율성을 이론적으로 보일 뿐 아니라, 점증깊이우선 탐색(DFID : Depth-First Iterative Deepening Search) 방법과 IDA*(Iterative Deepening A*) 알고리즘에 제안된 방법을 적용하여 얻은 변형된 탐색 알고리즘을 이용하는 로봇 계획 시스템을 구현하여 제안된 전략의 효율성을 실험적으로도 보인다.

A Robot Planning System Based on a Modified DFID Search Method

Jaegeol Yim†

ABSTRACT

We propose a new search method which is based on the precedence relationship between subgoals. The proposed method reduces both the branching factor of and the depth of the search space by making use of the precedence relationship between subgoals, and further improves the efficiency of A.I. search by immediately achieving directly achievable maximal subgoals and immediately performing the directly applicable actions which must be eventually done. The efficiency of our method has been analysed theoretically. We have also implemented a robot planning system equipped with versions of DFID and IDA* which are modified by applying our proposed strategies, and experimentally showed the efficiency of our strategy.

1. 서 론

로봇 계획 문제에 접근하는 방법은 목적 상태를 구성하는 부분 목적들을 성취되어져야 하는 순서에 맞추어 알맞게 나열하는 방법과[1, 2], 각각의 부분 목적들을 성취하는 일련의 동작(a sequence of actions)들(부분 계획들)을 순서에 맞추어 알맞게 나열하는 방법의 두가지가 있다[3, 4, 5]. 접근 방법이 무엇이든지 모든 로봇 계획 시스템의 공통점은 지수복잡도의 비효율성이다. 즉 전자의 접근 방법을 사용하는 시스템은 부분 목적의 수를 지수로 하는 지수복잡도이고, 후자의 접근 방법을 사용하는 시스템은 부분 계획의 수를 지수로하는 지수복잡도이다.

그런데, 부분 목적의 수나 부분 계획의 수는 모두 로봇 계획 문제의 영역의 크기(domain size)에 비례하므로 결국 모든 범용 로봇 계획 시스템 알고리즘은 문제 영역의 크기를 지수로 하는 지수복잡도이다.

Chapman은 [6]에서 특정 문제 영역에만 적용되는 로봇계획자 Pengi를 소개하고, Pengi를 구현한 방법으로 45개 상자까지 쌓는 문제를 해결할 수 있다고 하였다. 본 논문에는 목적 상태를 구성하는 부분 목적들 간에 직관적인 선취관계가 존재하는 문제 유형에 적용되는 효율성이 크게 개선된 새로운 인공지능 탐색 방법이 제안되며, 이 방법은 또한 상자 쌓기 문제의 효율적인 해로서 100개가 넘는 수의 상자들도 얼마든지 다룰 수 있다. 이 탐색 방법을 사용하는 로봇 계획 시스템으로 상자 세계 문제를 풀어, 제안된 방법의 효율성을 실험적으로 분석한다.

인공지능 탐색의 정의는 상태 집합과 연산 집합으로 구성된다. 연산 집합은 연산이라 불리우는 상태 집합을 영역으로 하는 함수로 구성된다.

† 정 회 원 : 동국대학교 전자계산학과 조교수
논문접수 : 1995년 1월 27일, 심사완료 : 1995년 4월 27일

상태 집합의 원소 중에는 초기상태와 목적상태(목적상태에 사용되는 구성 요소를 부분 목적이라함.)라고 불리우는 특별한 상태가 있는데, 인공지능 탐색의 목적은 초기상태를 목적상태로 변형시키는 일련의 연산, 특히 최적의 일련의 연산(최적해)을 찾는 것이다. 인공지능 탐색 방법은 사용되는 전략에 따라 맹목적 탐색과 휴리스틱 탐색으로 대별된다.

맹목적 탐색은 초기상태로부터 임의의 연산을 적용하여 새로운 상태를 생성하는 작업을 목적상태를 만날 때까지 반복한다. 맹목적 탐색 중 최소의 메모리를 사용하면서 항상 최적해를 찾는 탐색 방법으로 DFID(Depth-First Iterative Deepening : 점증깊이우선)[7] 탐색 방법이 있다. DFID는 탐색 공간의 깊이를 제한하면서 반복적으로 깊이우선탐색을 수행한다. 깊이의 제한은 처음에는 1로 시작하여 점차 1씩 증가한다. 깊이의 제한이 i일 때 깊이우선탐색이 성공적으로 끝이나면 전체 탐색이 성공적으로 끝이 나지만 그렇지 않을 경우에는 깊이의 제한을 1증가하여(i+1로 하여) 다시 깊이우선탐색을 수행한다.

이러한 과정을 탐색이 성공할 때 까지(탐색 성공) 혹은 깊이의 제한이 너무 큰 수가 될 때까지(탐색 실패) 반복한다.

휴리스틱 탐색에서는 새로운 상태를 생성할 때 임의의 연산을 적용하는 대신 목적상태와 가장 가까우리라고 예상되는 상태를 생성하는 연산을 먼저 적용하는 점이 맹목적 탐색과 다르다. 휴리스틱 탐색 방법 가운데 A*와 비슷한 시간을 소비하면서 구현이 간단하고 최소의 메모리를 사용하는 탐색 방법으로 IDA*(Iterative Deepening A*)[7] 알고리즘이 있다.

기존의 인공지능 탐색 방법의 단점으로 탐색 깊이가 깊어짐에 따라 분지수가 증가하는 경향이 있음을 들 수 있다. 본 논문에서는 탐색 깊이에 따라 분지 수가 증가하는 기존의 탐색 방법의 단점을 보완하기 위하여 분지 인수와 탐색 깊이를 다 같이 줄이는 방법으로 부분 목적에 대한 선취관계를 이용하는 방안을 제시한다. 목적상태를 이루는 부분 목적들 사이에는 어떤 부분 목적이 다른 것보다 반드시 먼저 성취되어져야 한다는 선취관계가 있는 것이 대부분임을 알 수 있다. 상자세계 문제에서는 밑에 있는 상자를 위에 있는 상자보다 먼저 정위치 시켜야 하며, 자동차 조립에서는 몸체를 먼저 조립한 다음 바퀴를 조립하여야 하고, 건설 공정이라면 먼저 땅을 파고 지반을 다진 후 지하층부터 차례로 지어 올라가야 한다.

이러한 선취관계를 이용하여 탐색의 효율성을 크게 제고시킬 수 있다. 문제에 따라서는 반드시 최적해가 아니더라도 최적에 가까운 근사해에 만족하는 경우가 많이 있다. 본 논문에는 대부분의 경우에 최적해를 찾고 최적해를 찾는 데 실패하더라도 최적해에 매우 가까운 근사해를 찾는 매우 빠른 탐색 방법 또한 제시된다. 제시된 방법들을 구현하여 여러가지 상자세계문제들을 풀어봄으로써 이들의 성능을 실험적으로 비교하여 본다.

2. 선취관계

주어진 탐색 문제의 목적상태를 분석함으로써 목적상태를 구성하는 부분 목적간의 선취관계를 규명할 수 있다. 예를들어 상자세계 문제에서는 목적상태에서 밑에 위치하는 상자를 위에 위치하는 상자보다 먼저 성취하여야 하고, 로봇의 공정을 찾는 문제에서도 어떤 일은 반드시 다른 일보다 선취되어야 한다는 것을 직관적으로 알 수 있는 경우가 흔히 있다. 본 절에서는 상자세계 문제[8]를 예를들어 부분 목적간의 선취관계에 대하여 알아보자. 상자세계 문제에는 다음과 같은 성질이 있다.

〈사실 1〉 ON(A, x)가 부분 목적이 아닌 어떤 상자 세계에서 어떤 최적계획 p에 사용된 stack(A, x)를 putdown(A)로 대체하여 얻은 계획도 최적계획이다.

〈증명〉 p=(초기상태 I에서) a11, a12, a13, ... stack(A, x)(상태 s1에 도달), a21, a22, a23, ... (sg(sg ⊇ G)에 도달)이 어떤 최적계획이라 하자. 그러면 p'=(초기상태 I에서) a11, a12, a13, ... putdown(A)(상태 s1'에 도달), a21', a22', a23', ... (sg'(sg' ⊇ G)에 도달)도 최적계획임을 증명하자. p의 s1에서는 A가 어떤 상자 x의 위에 있

고 p'의 s1'에서는 A가 테이블 위에 있다는 것만을 제외하고는 s1과 s1'은 서로 똑 같다. p에서 a21, a22, a23, ...는 어떤 상자를 A의 위에 올려 놓든지, 혹은 A와 상관 없는 동작들이든지, 혹은 A를 unstack하고 그 밑에 있는 상자들을 사용하는 동작들로 구분할 수 있다. 상태 s1'에서 A가 CLEAR함으로 A의 위에 올려 놓는 동작들은 s1' 이후에도 그대로 적용 가능하고 이렇게 얻는 상태는 A의 위치(ONTABLE(A))만 다르고 나머지는 똑 같다. 상태 s1과 s1'은 A의 위치만 다르므로 s1에 적용되는 A와 상관없는 동작들은 s1'에도 역시 적용 가능하다.

s1 이후에 A를 unstack하는 작업이 수행된다면 이는 x를 CLEAR하여 x와 그 밑에 있는 상자들을 사용하기 위함인데, s1'에서는 이미 x가 CLEAR하므로 unstack을 pickup(A)로 대체하여 A를 사용하고 x와 그 밑의 상자들에는 똑 같은 동작을 적용하여 사용할 수 있다. 이러한 동작이 a2j에 포함되었을 경우에는 sg = sg'이 되고, 그렇지 않을 경우에는 sg ∋ ON(A, x) ∧ sg !∋ ONTABLE(A)인데 반하여 sg' !∋ ON(A, x) ∧ sg ∋ ONTABLE(A)인 차이점이 있을 뿐 나머지는 똑 같다. 그런데 사실에서 ON(A, x)는 부분 목적이 아니라고 하였으므로 sg ⊇ G이면 sg' ⊇ G이다. 그러므로 p'도 최적계획이다.

〈사실 2〉 임의의 상자 A, B, C에 대하여, ON(A, B)와 ON(B, C)가 B를 공유하는 두 개의 부분 목적이라면 최적계획에서 ON(B, C)는 ON(A, B)에 반드시 선취되어져야 한다. 즉, 선취(ON(B, C), ON(A, B))이다.

〈증명〉 어떤 최적계획 p가 ON(B, C)가 아닌 상태에서 ON(A, B)를 먼저 성취한다고 하자. 즉, 초기상태에서 a11, a12, a13, 등을 수행하여 도착한 현재상태, si,에서 ON(B, C)를 달성하기 위한 작업을 선행할 수 있음에도 불구하고 ON(A, B)를 달성하기 위한 작업을 선행한다고 하자(이 작업에는 pickup(A)나 unstack(A, x)를 하고 stack(A, B)를 수행하는 것이 포함됨.) 이 최적해는 추후에, 즉 일련의 동작 a21, a22, a23, 등을 수행한 후에, ON(B, C)를 달성하기 위하여 stack(B, C)를 수행하여야 하는데, 이의 선행 조건 중에는 HOLDING(B)가 포함되며, HOLDING(B)를 성취하는 동작은 모두 선행 조건으로 CLEAR(B)를 포함한다. ON(A, B)일때 CLEAR(B)를 성취하려면 unstack(A, B)를 수행하여야 하고, 다음에 HANDEMPTY를 성취하기 위하여 putdown(A) 혹은 stack(A, y)를 수행하여야 한다. 그런데 <사실 1>에서 본 바와 같이 stack(A, y)는 putdown(A)로 교체할 수 있으므로 stack(A, y)를 수행하는 경우는 고려하지 않아도 된다. 그리고는 stack(B, C)를 수행하기 위한 작업을 하는데 이러한 계획을 p라 하면, p는 다음과 같다.

p = (초기상태에서 시작하여)a11, a12, a13, ... (si에 도착) pickup(A) or unstack(A, x), stack(A, B)(s1에 도착), a21, a22, a23, ... (s2에 도착) unstack(A, B), putdown(A)(s3에 도착) or stack(A, y)(s4에 도착, <사실 1>에 의하면 이 경우는 고려할 필요 없음.), pickup(B) or unstack(B, z), stack(B, C), ..., (목적상태 sg(sg ⊇ G)에 도착). p가 로봇계획이라면 다음과 같은 P'도 로봇계획인 것을 증명하자.

p' = (초기상태에서 시작하여)a11, a12, a13, ...(si에 도착) pickup(A) or unstack(A, x), putdown(A)(s1'에 도착), a21', a22', a23', ... (s3'에 도착) pickup(B) or unstack(B, z), stack(B, C), ..., (목적상태 sg'(sg' ⊇ G)에 도착).

s1과 s1'의 차이는 s1 ∋ ON(A, B) ∧ s1 !∋ ONTABLE(A)이고 s1' !∋ ON(A, B) ∧ s1' ∋ ONTABLE(A) ∧ s1' ∋ CLEAR(B)인 것 외에는 s1과 s1'은 꼭 같다. 그러므로 a21, a22, a23, ...은 s1'에도 그대로 적용 가능하다. 즉, a21=a21', a22=a22', a23=a23', ...일때 도달되는 s3'과 s2의 차이는 s1'과 s1의 차이와 같다. 즉, s3'에서는 이미 s3' ∋ ONTABLE(A) ∧ s3' ∋ CLEAR(B)이다. 따라서 s3와 s3'은 서로 똑 같다. 그런데 |p'| < |p| 이므로 p는 최적계획이 아니다. 이는 p가 최적계획이라는 가정에 모순이므로 ON(A, B)를 ON(B, C)보다 선취하는 최적계획은 없다.

선취관계는 〈사실 3〉에서 보이는 바와 같이 추이관계이다.

〈사실 3〉 : ON(A, B), ON(B, C) 그리고 ON

(C, D)가 부분 목적이면 선취(ON(C, D), ON(A, B))이다.

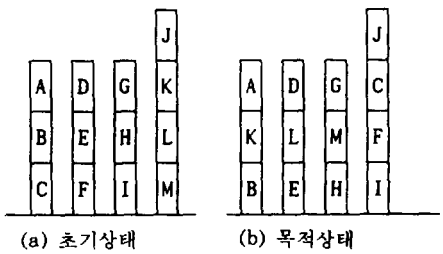
〈증명〉 ON(A, B), ON(B, C) 그리고 ON(C, D)가 부분 목적이면 〈사실 3〉에 의하여 (1)과 (2)가 만족된다.

선취(ON(B, C), ON(A, B)) ... (1)

선취(ON(C, D), ON(B, C)) ... (2)

만일 어떤 최적해에서 ON(A, B)를 ON(C, D)보다 먼저 성취한다고 가정하면 이 최적해가 성취하는 부분 목적들의 순서는 ON(B, C)를 ON(A, B) 보다 먼저((1)에 의하여), ON(A, B)를 ON(C, D)보다 먼저(가정에 의하여) 그리고 ON(C, D)를 ON(B, C) 보다 먼저((2)에 의하여) 성취한다. 즉 이 최적해에 의하여 성취되는 부분 목적의 순서에서 ON(B, C)가 ON(A, B)와 ON(C, D)보다 먼저이고 ON(C, D)가 ON(B, C) 보다 먼저인 모순에 도달한다. 그러므로 반드시 선취(ON(C, D), ON(A, B))이다.

〈사실 2〉와 〈사실 3〉에서 술어기호 ON을 사용하는 부분 목적들이 어떤 상자를 인수로 공유하면 이들 부분 목적 사이에는 선취관계가 있으며, 선취관계는 추이관계임을 알 수 있다. 따라서 목적상태를 이루는 각 상자의 파일(pile)은 밑에서부터 차례로 성취되어져야 한다라는 것을 알 수 있다.



(그림 1) 상자문제의 예제
(Fig. 1) An example block world problem

(그림 1)[9]에 보이는 문제의 경우, B가 먼저 자리를 잡기 전에 K를 B의 위에 올려 놓으면 B를 정위치 시키기 위하여 K를 다시 내려놓아야 함으로 ONTABLE(B)가 ON(K, B)보다 먼저 성취되어져야 한다. 마찬가지로 ONTABLE(E)는 ON(L, E)보다 선취되어져야 한다. (그림 1)의 부분 목적들을 선취 순서에 따라 나열하면 다음과 같은 네 개의 선형순서집합이된다.

ONTABLE(B), ON(K, B), ON(A, K).

ONTABLE(E), ON(L, E), ON(D, L).

ONTABLE(H), ON(M, H), ON(G, M).

ONTABLE(I), ON(F, I), ON(C, F), ON(J, C).

이 상황에서 ONTABLE(B), ONTABLE(E), ONTABLE(H) 그리고 ON(F, I)는 모두 극대 부분 목적이며, 즉, 선취(x, ONTABLE(B)), 선취(x, ONTABLE(E)), 선취(x, ONTABLE(H)), 혹은 선취(x, ON(F, I))를 만족하는 부분 목적 x가 존재하지 않으며, 이들 사이에는 무엇을 먼저 성취해야 하는지 선취관계가 분명하지 않다. 여기에서 ONTABLE(I)가 아닌 ON(F, I)가 극대 부분 목적이 되는 이유는 ONTABLE(I)가 초기상태에서 이미 성취되어 있기 때문이다.

따라서 선취관계에 의하여 정렬된 부분 목적을 사용하는 탐색은 이 곳에서 네 개의 가지를 만든다. 여기에서 한가지 밝혀 둘 사실은 극대 부분 목적의 수는 최초의 극대 부분 목적의 수((그림 1)에서는 4)를 넘는 일이 결코 없으며 따라서 탐색공간의 어느 노드(node)에서도 분지의 수는 최초의 극대 부분 목적의 수를 결코 넘지 않는다는 것이다. 맹목적 탐색 방법을 사용하면 일반적으로 탐색 깊이가 깊어감에 따라 분지 수가 증가하는 경향이 있는 반면에 이와 같이 부분 목적간의 선취관계를 사용하면 분지의 수를 제한할 수 있다. 뿐만 아니라 목적상태에 가까와 짐에 따라 노드의 분지 수는 서서히 감소한다. 상자세계문제에서 목적 상태가 한 개의 상자 파일로 구성되어 있을 때에는 부분 목적들이 선취관계에 대하여 선형 순서 관계집합을 이루므로(극대 부분 목적이 많아야 한 개임) 로봇계획을 찾는 알고리즘이 분기(branch)나 되돌아옴(bounce)이 없이 손쉽게 로봇계획을 찾을 수 있다.

본 절에서는 상자세계 문제에서, 목적 상태를 구성하는 부분 목적들 사이에 존재하는 선취 관계에 대하여 알아 보았다. 부분 목적들 간의 선취 관계를 알아내는 효율적인 알고리즘은 아직 발견된 바가 없다. 그러나 발견된 선취 관계를 일목요연하게 표현하는 방법으로는 하세 그래프가 널리 알려져 있다.

3. 변형된 점증깊이우선 탐색 방법

본 논문의 목적은 목적상태를 이루는 부분 목적들간의 선취관계를 이용한 매우 빠른 탐색 전략을 제안하는 것이다. 본 절에서는 부분 목적간의 선취관계를 맹목적 탐색 방법 중 DFID에 적용하는 방법에 관하여 살펴본다. 휴리스틱 탐색을 포함한 다른 기존의 탐색 방법에 선취관계를 적용하는 방법은 이와 매우 흡사하므로 설명을 생략한다.

부분 목적들에 대한 선취관계는 부분순서관계(Partially ordered set)이므로 이를 이용하여 변형시킨 DFID를 DFID_pos라고 명명한다. DFID_pos는 DFID와 매우 유사하다. 차이점은 DFID가 가능한 모든 자식 노드를 생성하여 나가는 데 반하여, DFID_pos는 극대 부분 목적을 성취하는 자식 노드만 생성하여 나가는 것이다.

DFID_pos의 주요 부분은 〈표 1〉에 보이는 바와 같다.

알고리즘 DFID_pos는 극대 부분 목적(max : maximal subgoal) 각각에 대하여 이것을 성취하는 데 필요한 첫번째 동작을 수행하여 자식 노드를 만들고 탐색 공간의 깊이를 증가시키면서

〈표 1〉 선취관계를 이용하여 변형된 점증깊이우선탐색의 주요 부분

〈Table 1〉 An important part of a modified DFID which is based on the precedence relation.

```
.....
max_block[]에 극대부분 목적 기재;
성취해야 할 극대 부분 목적이 없으면 return
(1);/*탐색 성공*/
각각의 극대 부분 목적, gi에 대하여
begin
  gi를 성취하여 자식 노드를 만든다; ...(1)
  자식 노드를 인수로하여 DFID_pos를 재귀
       적으로 부른다;
  IF DFID_pos가 성공이면
  begin
    (1)에서 수행한 동작을 프린트;
    return(1);
  end
end
```

DFID_pos를 재귀적으로 부른다. 그러므로, DFID_pos의 탐색 공간은 성취하여야 할 부분 목적을 차례로 성취하면서 깊이가 한번 증가할 때 마다 로봇계획의 길이도 1 씩 증가하면서 성장하여 간다. 그러다가 현재상태에서 목적상태가 성취되어 있을 때에 성공적으로 끝난다. 즉, DFID_pos는 길이가 i 이하인 모든 가능한 일련의 동작들을 탐색한 후 길이가 i+1인 일련의 동작들을 시도함으로 DFID_pos가 찾는 로봇계획은 최적이다.

〈정리 1〉 알고리즘 DFID_pos는 최적해를 발견한다.

〈증명〉 선취관계의 정의에 의하여 최적해는 부분 목적들을 선취관계에 의하여 정렬된 순서대로 성취하여 나아간다. DFID_pos는 선취관계를 만족하는 모든 순서의 상태들을 탐색한다. 더구나 탐색 공간의 깊이 i+1을 탐색하기 전에 깊이 i를 모두 탐색한다. 그러므로 DFID_pos가 찾는 로봇계획은 최적계획이다.

알고리즘 DFID_pos는 부분 목적들에 대한 선취관계와 부분 목적들이 부분순서집합을 형성한다라는 가정(많은 실제 문제에 적용되는 타당한 가정임)하에 현재 상태에서의 모든 극대 부분 목적 각각에 대하여 자식 노드를 생성한다. 여기에서 극대 부분 목적은 다른 부분 목적보다 반드시 먼저 성취되어져야 하는 부분 목적일 뿐만 아니라 다른 부분 목적을 성취하는 도중에 파기(undone)되는 일이 결코 없는 성질을 갖고 있다. 그러므로 노드의 분지 수는 선취관계를 구성하는 선형 순서 관계의 수 보다 클 수 없다.

〈정리 2〉 성취된 어떤 상태의 극대 부분 목적을 파기(undo)하는 로봇계획은 최적계획이 될 수 없다.

〈증명〉 : 어떤 최적계획, P가 임의의 상태에서 극대 부분 목적 g1을 성취한 후 상태 si에 이르렀다가, 상태 si에서 그 g1을 파기하는 동작을 수행하여 술어 g2를 성취하였다고 하자. 이들 사이의 관계는 다음 중 한가지이다.

1. 선취(g2, g1), or
2. 선취(g1, g2), or
3. 서로 선취관계가 없는 경우.

1의 경우에는 선취관계의 정의에 의하여 g2가 g1 보다 먼저 성취되어져야 하는 데 P가 g1을 성취한 다음 g2를 성취하였으므로 모순이고, 2의 경우에는 g1이 부분 목적이므로 이후 언젠가 다시 성취하여야 하는데, 이렇게 되면 g1을 g2보다 먼저 성취하여야 한다는 선취(g1, g2)의 정의에 모순이다. 3의 경우에는 g1을 달성하기 위하여 수행되었던 동작을 P에서 삭제하여도 로봇계획이 된다. 왜냐하면 g1과 g2가 아무런 선취관계가 없기 때문이다. 만일 g2를 성취하는 데 g1이 꼭 필요하였다면 선취(g1, g2)이었어야 한다. 즉 P가 최적이라는 가정에 모순이다. 세 가지 경우 모두 모순이므로 최적계획은 성취된 극대 부분 목적을 결코 파기하지 않는다.

부분 목적 gi에 대하여, 이것이 성취되어 있지 않은 상태에서 이를 성취하기 위하여 수행하여야 할 최소한의 동작의 수를 gi를 성취하기 위한 최소 비용이라고 하자. 예를들어 [8]에 소개된 상자세계 문제에서는 부분 목적 ON(A, B)의 최소 비용은 2이다. 또한 최소 비용으로 성취 가능한 부분 목적을 직접성취가능이라고 하자. 어떤 극대 부분 목적(부분 목적 g1)이 직접성취가능이라면 이를 당장 성취하는 데 드는 비용은 g1을 성취하기 위하여 반드시 드는 최소 비용이고, 〈정리 2〉에서 한번 성취된 극대 부분 목적은 결코 파기되지 않는다라고 하였으므로 어떤 상태에 도달 할 때마다 직접성취가능인 극대 부분 목적을 당장 성취하도록 변형된 DFID_pos도 역시 최적해를 찾는다. 이와같이 변형된 방법을 DFID_pos1이라 하자.

〈정리 3〉 어떤 상태에 도달 할 때 마다 직접성취가능한 극대 부분 목적을 성취하는 작업을 첨가하도록 변형된 DFID_pos(DFID_pos1)도 역시 최적해를 찾는다.

알고리즘 DFID_pos는 깊이를 더하여 갈 때 마다 부분계획에 한 동작 씩을 첨가함으로써 반드시 최적해를 찾는다. 그런데 상자세계 문제에서는 초기상태와 목적상태에서 모두 HANDEMPTY가 참이라면 로봇계획의 길이는 반드시 짝수이고 따라서 매 반복마다 두개의 동작을 계획에 첨가시켜서 얻는 로봇계획도 최적이다. 본 논문에서 상자세계에 대한 실험치를 얻기 위하여 사용한 DFID_pos는 두개 동작씩 첨가한다. 만일 탐색 깊이가 1 증가할 때 부분계획에 넷 혹은 여섯 동작까지 증가시키는 것을 허용한다면 반드시 최적해를 찾는 성질은 상실할 수 있으나 그 대신 탐색 공간의 깊이는 크게 감소시킬 수 있다. 이 때 찾아지는 로봇계획은 최적계획과 매우 근사하다. 이러한 아이디어를 확장하여 극대 부분 목적 각각에 대하여 이것을 성취하여 자식 노드를 만들고 탐색 공간의 깊이를 증가 시키면서 자신을 재귀적으로 부르는 탐색 방법을 DFID_pos2라 부른다.

알고리즘 DFID_pos2는 한 반복에서 취하여지는 동작의 수를 증가시킴으로써 탐색 공간의 깊이를 감소시킬 수 있으나, 최적계획을 찾는 성질은 상실하였다. 한 반복에서 취하여지는 동작의 수를 더욱 증가시키고, 최적 계획과 더욱 근접한 해를 구하기 위하여 자식 노드에 이를 때 마다, 필연적으로 수행하여야 할 동작들과 직접 성취할 수 있는 극대 부분 목적들을 성취함으로써 탐색 공간을 더욱 축소시킬 수 있다. 이러한 작업이 보강된 방법을 DFID_pos3라 한다.

상자세계의 경우에는 현재상태에서 목적상태에 쓰이는 상자들의 위에 위치한 목적상황 표현에 사용되지 않는 제삼의 상자들을 치우는 작업들이 필연적으로 수행되어야 하는 작업들에 속한다. 이러한 작업을 수행하는 함수를 move_third_boxes라고 하자. 제삼의 상자를 치운 다음에는 move_ready_boxes를 사용하여 직접성취가능인 극대 부분 목적을 성취한다. 하나의 극대 부분 목적을 성취하면 max_block의 내용과 현재상태가 바뀌게되므로 move_ready_boxes는 직접성취가능인 극대 부분 목적이 없는 상태에 도달할 때

〈표 2〉 알고리즘 DFID_pos3에 필요한 문장 (상자세계의 경우)
〈Table 2〉 Sentences needed for DFID_pos3 (Blocks world case)

```
move_third_boxes(C, G, max_block, actions);
for(;move_ready_boxes(C, G, max_block, actions););
```

까지 반복적으로 수행한다. 상자세계문제의 경우, DFID_pos3는 〈표 1〉의 문장 (1) 다음에 〈표 2〉의 두 문장을 첨가하여 얻을 수 있다.

휴리스틱 함수를 이용하여 비용을 예측하고 예측된 비용이 threshold보다 큰 노드는 버리는 문장을 첨가함으로써 DFID_pos1은 손쉽게 선취관계를 고려한 IDA*로 변형될 수 있다. 이때 초기상태의 휴리스틱 함수값을 threshold의 초기값으로 하고 i번째 반복에서의 threshold는 i−1번째 반복에서 버려진 노드의 예측 비용 중 가장 작은 것으로 한다. 이렇게 변형된 알고리즘을 IDA*_pos1이라 하자. 여기에 사용될 수 있는 휴리스틱 함수는 많이 있지만 현재상태와 위치가 다른 목적상태의 상자 수(manhatan distance)를 함수값으로 하는 휴리스틱 함수를 사용한다. Manhatan distance는 적당하고 단조하므로 IDA*_pos1이 발견하는 해도 역시 최적해이다. 또한 IDA*_pos1에 의하여 생성되는 노드의 수는 DFID_pos1에 의하여 생성되는 노드의 수보다 많을 수 없다.

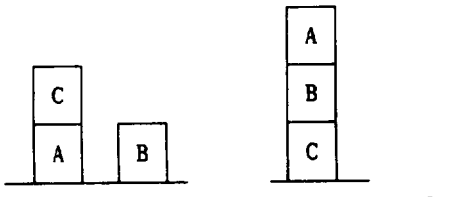
4. 실 험

점증깊이우선 탐색(DFID)은 최적해를 구하는 알고리즘으로 잘 알려져 있다. 앞에서는 이 방법의 변형으로 DFID_pos와 DFID_pos 1, 2, 3을 제안하였다. 알고리즘 DFID_pos와 DFID_pos1이 발견하는 로봇계획은 최적해이며, 탐색 공간은 DFID 알고리즘의 경우보다 훨씬 작다라고 주장하였다. 또한 DFID_pos2와 3는 최적해를 구하지는 못하지만 최적해와 근사한 로봇계획을 훨씬 빠른 속도로 구한다라고 하였으며, 특히 DFID_pos3는 DFID_pos2 보다 더욱 빠르고 구하는 로봇계획도 최적에 더욱 근사하다라고 주장하였다. 끝으로 DFID_pos1을 최적탐색으로 변형시킨 IDA*_pos1이 생성하는 노드 수는 DFID_pos1에 의하여 생성되는 노드의 수보다 작다라고 하였다.

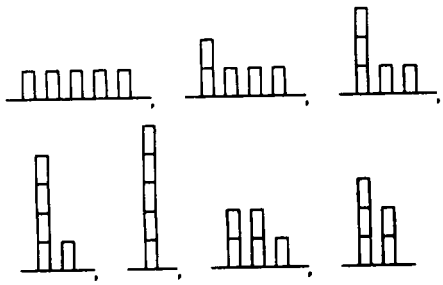
본 절에서는 이러한 주장들을 실험적 수치로 뒷받침하기 위하여 위의 알고리즘들을 SPARC 10급 workstation에 프로그래밍 언어 C로 구현하여 다음과 같은 실험을 하였다. 첫째, DFID_pos와 DFID_pos1이 구하는 로봇 계획도 최적해라는 주장(이미 이론적으로 충분히 증명하였지만)을 실험적으로 뒷받침하기 위하여 수십개의 5상자 문제를 DFID, DFID_pos 그리고 DFID_pos1, 2, 3으로 풀어보았다.

이 실험에서 5상자 문제를 채택한 이유는 DFID의 처리 속도가 너무 느려서 상자의 수가 늘어날 경우 해를 구하는데 걸리는 시간이 너무 길기 때문이었다. 또한 (그림 1)에 보이는 [8]에 소개된 문제와 (그림 2)에 보이는 Sussman's anomaly 문제도 DFID_pos와 DFID_pos1, 2, 3로 풀어 보았다. 실험 결과 모두 최적해를 구함을 알 수 있었다. DFID_pos2와 3는 언제나 최적해를 구하지는 못하지만 다섯상자 문제들과 (그림 1), (그림 2)에 대하여 최적해를 구한다. 이로부터 DFID_pos2와 3의 해가 최적해와 매우 근사함을 알 수 있다.

다음에는 여러 종류의 문제를 각 알고리즘으로 풀어 보고 처리 속도를 비교하였다. 실험의 객관성을 보장하기 위하여 실험에 사용되는 문제를 무작위로 만든다. 다른 상자 문제의 경우를 예로 들어 실험에 사용되는 문제의 초기 상태와 목적



(그림 2) Sussman's anomaly 문제
(Fig. 2) Sussman's anomaly problem



(그림 3) 다섯 상자의 전형적인 배열 유형
(Fig. 3) The typical configurations of 5 blocks

상태를 구축하는 방법을 간단히 설명한다. 다섯 상자가 책상위에 배열되는 경우는 (그림 3)에 보이는 바와 같이 일곱가지가 있다. 함수 random을 사용하여 일곱가지 배열의 유형 중 한 가지를 무작위로 선택한다. 다음에는 비슷한 방법으로 각 상자의 이름을 무작위로 결정한다. 이러한 방법으로 무작위 상태를 구축하여 문제의 초기 상태와 목적 상태를 결정한다. 이후에 수행되는 10 상자 문제, 20 상자 문제 등의 실험에서도 유사한 방법으로 문제를 구축하여 사용한다.

DFID와 부분순서 관계를 고려하여 변형된 DFID_pos 그리고 DFID_pos1의 처리 속도를 비교하기 위하여 이들이 5상자 문제를 푸는데 걸리는 CPU time들을 〈표 3〉에 보인다. DFID에 의하여 소모된 CPU time은 제안된 방법에 의하여 변형된 알고리즘들에 의하여 소모된 CPU time보다 훨씬 길다.

〈표 3〉 DFID, DFID_pos 그리고 DFID_pos1의 CPU time 비교
〈Table 3〉 CPU times spent by DFID, DFID_pos and DFID_pos1

STEP	DFID	DFID_pos1	DFID_pos
2	0.28	0.00	0.00
4	2.18	0.00	0.06
6	50.53	0.00	0.11
8	210.91	0.00	0.30
10	962.38	0.00	0.53
12	2111.858	0.01	0.66
14	4057.09	0.06	0.67

DFID는 10 상자 문제를 해결하는 데 너무 긴 시간을 사용하므로, 제안된 방법으로 변형된 방법들(DFID_pos, DFID_pos1, DFID_pos2)만으로 10상자 문제를 풀어 보아, CPU time을 〈표 4〉에 보인다. 로봇계획의 길이가 길어짐에 따라 DFID_pos는 다른 알고리즘보다 비교할 수 없을 만큼 느린 것을 알 수 있다. 더구나 DFID_pos1은 최적해를 구하면서 근사해를 구하는 DFID_pos2보다 훨씬 더 빠름을 보인다. 이로부터 직접 성취 가능한 부분 목적을 즉각 성취하는 전략의 공헌도가 매우 높음을 알 수 있다.

〈표 4〉 DFID_pos, DFID_pos1, DFID_pos2의 CPU time 비교
〈Table 4〉 CPU times spent by DFID_pos, DFID_pos1 and DFID_pos2

STEP	DFID_pos2	DFID_pos	DFID_pos1
4	0.00	0.00	0.00
8	0.10	0.44	0.00
12	1.16	2.86	0.01
16	2.50	27.48	0.03
20	5.38	96.97	0.05
24	9.35	194.12	0.07
28	13.78	373.66	0.26
30	19.75	767.20	1.50

〈표 5〉 (a) 20상자 문제를 DFID_pos1으로 푸는데 걸리는 시간과
(b) 40상자 문제를 DFID_pos3으로 푸는 데 걸리는 시간
〈Table 5〉 (a) CPU times spent by DFID_pos1 in solving 20 blocks problems
(b) CPU times spent by DFID_pos3 to find approximate solutions of 40 blocks problems

STEP	DFID_pos1
24	0.04
30	0.12
36	0.51
42	2.60
48	7.73
54	21.92

〈(a) 20상자 문제〉

STEP	DFID_pos3
38	0.85
58	2.94
78	46.50
100	500.48
114	1387.95

〈(b) 40상자 문제

〈표 5(a)〉에는 20상자 문제를 DFID_pos1으로 푸는데 걸리는 CPU 시간을 보인다. 이것은 지금까지 알려진 상자세계 문제를 푸는 알고리즘 중 가장 빠른 결과라고 생각된다. 〈표 5(b)〉에는 임의의 40상자 문제에 대하여 DFID_pos3가 로봇계획을 찾는데 걸리는 CPU 시간을 보인다. DFID_pos3는 최적해를 못 구하는 경우는 있지만 표에서 보는 바와 같이 매우 빠른 시간내에 근사해를 구한다.

지금까지 IDA*_pos에 대한 실험치는 언급을 보류하여 왔다. 그 이유는 Manhatan distance를 사용하는 IDA*_pos의 성능이 DFID_pos1의 성능과 대동소이하기 때문이다. IDA*_pos가 생성하는

〈표 6〉 DFID_pos1과 IDA*_pos의 비교
(a) 생성되는 노드 수, (b) CPU 시간
〈Table 6〉 Comparing DFID_pos1 to IDA*_pos: (a) Number of nodes explored, (b) CPU time

STEP	IDA*_pos	DFID_pos1
24	3.7	3.7
30	8.1	7.6
36	57.2	56.6
42	116.3	111.2
48	303.9	297.8
54	1093.5	1070.1

〈(a) 생성되는 노드 수〉

STEP	IDA*_pos	DFID_pos1
24	0.04	0.04
30	0.12	0.13
36	1.04	1.08
42	2.18	2.23
48	5.76	5.86
54	20.72	21.08

〈(b) CPU 시간〉

노드의 수는 DFID_pos1이 생성하는 노드의 수보다 약간 적지만 〈표 6(a)〉에 보이는 바와 같이 거의 비슷하다. 그러나 IDA*_pos가 소비하는 CPU 시간은 〈표 6(b)〉에 보이는 바와 같이 DFID_pos1의 CPU 시간과 거의 같지만 오히려 약간 더 길다. 이로부터 직접 성취 가능한 극대 부분 목적을 당장 성취하는 전략의 공헌도에 비해 Manhatan distance를 사용하는 휴리스틱 방법의 공헌도가 매우 미미하다는 것을 알 수 있다.

더구나 Manhatan distance를 구하는 데 소모되는 여벌 시간이 탐색 공간을 감소시켜 주는 공헌을 상쇄하고도 남는 것을 알 수 있다. 여기에서 한번 더 강조하고 싶은 것은 극대 부분 목적은 부분 목적을 선취관계에 의하여 나열함으로써 얻을 수 있다는 것이다.

5. 결 론

탐색은 인공지능에서 가장 중요한 연구 분야 중 하나이다. 본 논문에는 목적상태를 이루는 부분 목적들 간의 선취관계를 이용하여 탐색 공간을 크게 줄이는 방안이 제시되었다. 알고리즘 DFID_pos의 탐색 공간의 크기는 부분 목적에 대한 선취 관계로 이루어지는 선형 순서 관계의 수를 밑으로하고 탐색공간의 깊이를 지수로하는 수로 나타낼 수 있다.

필연적으로 수행되어야 하는 동작들과 직접 성취할 수 있는 극대 부분 목적들을 당장 성취하면 최적해를 구하면서 탐색 공간의 깊이를 더욱 얕게하여 줌으로써 탐색 공간을 더욱 크게 축소 시킨다. DFID_pos에 이러한 전략을 가미하여 변형시킨 알고리즘을 DFID_pos1이라 부른다.

알고리즘 DFID_pos와 DFID_pos1은 모두 최적해를 구하며 DFID 보다 훨씬 효율적이다. DFID나 DFID_pos의 경우에는 깊이가 하나 증가할 때마다 한 개의 동작이 계획에 첨가되고, DFID_pos1의 경우에는 한 개의 동작을 첨가하는 외에 직접성취 가능한 극대 부분 목적을 성취하는 동작을 첨가함으로써 최적해를 구하는 것을 보장하였다. 그런데 깊이가 하나 증가할 때 여러 개의 동작을 계획에 첨가하면 최적해를 구하는 것을 보장 할 수 없게 된다. 그러나 탐색 공간의 깊이는 줄어들어 빠른 시간내에 해를 구할 수 있다. 알고리즘

DFID_pos2의 경우에는 깊이가 증가 할 때마다 각각의 극대 부분 목적을 성취한다. 따라서 하나의 깊이가 증가할 때 여러개의 동작이 계획에 첨가되고 결국 탐색공간의 깊이를 얕게하여 줌으로써 탐색의 효율성을 높인다.

DFID_pos3의 경우에는 매 반복마다 어차피 수행해야할 동작과 직접 성취할 수 있는 극대 부분 목적들까지 성취함으로써 탐색 공간의 깊이를 더욱 얕게하고 따라서 탐색공간 자체도 DFID의 경우보다 훨씬 작다. 상자세계를 푸는 제안된 알고리즘들을 구현하여 다수의 상자세계 문제들을 풀어 보았다. DFID_pos와 DFID_pos1은 DFID보다 훨씬 빠르면서 항상 최적해를 구하였다. IDA*_pos의 성능은 DFID_pos1의 성능과 대동소이하다. 알고리즘 DFID_pos1은 해의 길이가 50인 20상자 문제를 20초 내에 푸는 매우 빠른 알고리즘으로 이것은 지금까지 발표된 가장 빠른 알고리즘이라고 생각된다.

DFID_pos3는 최적해를 못 찾는 경우가 있지만 임의로 만든 40상자 문제의 근사해를 구할 만큼 빠르다. 본 논문에 소개된 알고리즘은 부분 목적들 간의 선취 관계를 응용하는 탐색 방법이다. 그러나 부분 목적간의 선취 관계를 규명하는 일이 항상 용이한 것은 아니다.

실제 로봇 계획 시스템에서는 범용성인 기존의 계획자와 본 논문의 방법을 함께 설치하여, 풀고자 하는 문제에 따라서 선취 관계가 분명한 문제는 본 논문의 방법으로 해를 구하고 그렇지 않은 경우에는 기존의 방법으로 해를 구하도록 한다.

참 고 문 헌

[1] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, Vol. 2, pp. 189-208, 1971.

[2] J. Cheng and B. Irani, "Ordering Problem Subgoals," The Proc. of the 1989 International Joint Conference on Artificial Intelligence(IJCAI), pp. 931-936, 1989.

[3] E. D. Sacerdoti, "The Nonlinear Nature of Plans," The Proc. of the 1975 IJCAI, pp. 206-214, 1975.

[4] D. Chapman, "Planning for Conjunctive Goals," Artificial Intelligence, Vol. 32, pp. 333-377, 1987.

[5] J. Hertzberg and A. Horz, "Towards a Theory of Conflict Detection and Resolution in Nonlinear Plans," The Proc. of the 1989 IJCAI, pp. 937-942, 1989.

[6] D. Chapman, "Penguins Can Make Cake," AI Magazine, Winter 1989, pp. 45-50, 1989.

[7] R.E. Korf, "Depth-first Iterative-Deepening: An Optimal Admissible Tree Search," Artificial Intelligence 27, pp. 97-109, 1985.

[8] N.J. Nilsson, Principles of Artificial Intelligence, Tioga, NY, 1980.

[9] N. Gupta and D.S. Nau, "On the Complexity of the Blocks-World Planning," Artificial Intelligence, 56, pp. 223-254, 1992.

임 재 걸

1974년 인천교육대학 졸업
1981년 동국대학교 전자계산학과 졸업(학사)
1987년 석사, 1990년 박사:University of Illinois at Chicago, Dept. of Electrical Engineering and Computer Science
1974년 ~ 79년 초등교사
1981년~81년 경제기획원 조사통계국 전산처리사
1991년~92년 현대전자
1992년~현재 동국대학교 전자계산학과(경주 캠퍼스) 조교수
관심분야 : 페트리 네트 응용과 인공지능.