DOI: 10.3745/KIPSTB.2008.15-B.2.159

컴포넌트 서비스 기반의 휴리스틱 탐색 계획기

김 인 철[†] · 신 행 철[††]

요 약

최근 들어 로봇 작업 계획기에 요구되는 중요한 기능 중의 하나가 이미 존재하는 컴포넌트 서비스들을 결합하여 새로운 서비스로 조합해낼 수 있는 계획 기능이다. 본 논문에서는 이러한 컴포넌트 서비스 조합을 위한 커널모듈로 개발된 휴리스틱 탐색 계획기인 JPLAN의 설계와 구현에 대해 설명한다. JPLAN은 효율적인 상태 공간 탐색을 위해 지역 탐색 알고리즘과 계획 그래프 휴리스틱을 이용한다. 본 논문에서 제안하는 지역 탐색 알고리즘인 EHC+는 FF 등의 상태 공간 계획기에 적용되어 높은 효율성을 보인 Enforced Hill-Climbing (EHC)을 확장한 것이다. EHC+는 EHC에 비해 소량의 추가적인 지역 탐색을 필요로 하지만 목표 상태까지 전체 탐색 양을 줄일 수 있고 더 짧은 계획을 얻을 수 있다. 또한 본 본문에서는 대규모 상태 공간 탐색에 필수적인 효과적인 휴리스틱 추출 방법을 제안한다. 본 논문에서 제안하는 휴리스틱 추출 방법은 Graphplan에서 계획 생성을 위해 처음 제안된 계획 그래프를 이용한다. 본 논문에서는 이러한 계획 그래프 기반의 다양한 휴리스틱들을 소개하고, 이들이 계획 생성에 미치는 효과를 실험을 통해 분석해본다.

키워드 : 컴포넌트 서비스, 상태 공간 계획, 지역 탐색, 계획 그래프 휴리스틱

A Heuristic Search Planner Based on Component Services

Incheol Kim[†] · Hangcheol Shin[††]

ABSTRACT

Nowadays, one of the important functionalities required from robot task planners is to generate plans to compose existing component services into a new service. In this paper, we introduce the design and implementation of a heuristic search planner, JPLAN, as a kernel module for component service composition. JPLAN uses a local search algorithm and planning graph heuristics. The local search algorithm, EHC+, is an extended version of the Enforced Hill-Climbing(EHC) which have shown high efficiency applied in state-space planners including FF. It requires some amount of additional local search, but it is expected to reduce overall amount of search to arrive at a goal state and get shorter plans. We also present some effective heuristic extraction methods which are necessarily needed for search on a large state-space. The heuristic extraction methods utilize planning graphs that have been first used for plan generation in Graphplan. We introduce some planning graph heuristics and then analyze their effects on plan generation through experiments.

Key Words : Component Service, State-Space Planning, Local Search, Planning Graph Heuristics

1. 서 론

최근 들어 지능 로봇 작업 계획 분야에서는 로봇에 이미 내장되어 있거나 적재 가능한 모듈들의 서비스를 서로 결합하여 사용자의 요구에 맞는 새로운 작업 계획을 생성해내는 컴포넌트 서비스 조합 기술에 대한 관심이 높아지고 있다. 이러한 작업 계획 패러다임은 이미 여타 분야에서 활발히 진행되고 있는 서비스 지향 구조(SOA)와 서비스 지향 컴퓨팅(SOC) 연구의 연장선상에 있다고 볼 수 있다. 특히 계획(planning) 기술을 이용한 자동화된 서비스 조합에 관한 연구는 웹 서비스 분야에서 이미 몇몇 선행 연구들이 수행된 바가 있으며, SHOP2[16], Xplan[11], P4J-CP[12] 등은 웹 서비스 조합을 위해 적용된 대표적인 계획기(planner)들이다. 이 계획기들의 공통점은 탐색에 의한 자동화된 계획 생성의 복잡도를 줄여 보려는 노력의 하나로, 사용자가 제시하는 별도의 영역-의존적 제어지식을 이용한다는 점이다. SHOP2와 Xplan의 경우는 사용자가 이미 알고 있는 동작들의 부분 순서관계와 계층관계를 별도로 제공받아 탐색에 이용하였으며, P4J-CP의 경우는 사용자로부터 관심조건, 제한조건, 계획 유형 등을 별도로 입력받아 탐색에 이용하였다.

본 논문에서는 컴포넌트 서비스 조합을 위한 핵심모듈로 개발된 휴리스틱 탐색 계획기인 JPLAN의 설계와 구현에 대해 설명한다. 효율적인 컴포넌트 서비스 조합을 위해서는 효과적인 서비스 표현법(representation), 다양한 서비스 실

※ 본 연구는 2006학년도 경기대학교 학술연구비(연구그룹연구과제) 지원에 의하여 수행되었음.
† 종신회원 : 경기대학교 컴퓨터과학과 교수
†† 준 회 원 : 경기대학교 컴퓨터과학과 석사과정
논문접수 : 2008년 3월 4일, 심사완료 : 2008년 3월 26일

행 제어구조(control structure)의 지원, 초기 환경지식의 불완전성(incompleteness), 서비스 실행 결과의 불확실성(uncertainty), 보안(security) 등 해결해야 할 여러 가지 문제들이 있으나 본 논문에서는 주로 서비스 조합을 위한 계획 생성의 효율성 문제를 다룬다. JPLAN은 별도의 영역-의존적 제어지식을 이용하는 기존의 계획기들과는 달리 계획 문제의 명세로부터 자동으로 추출하는 영역-독립적인 휴리스틱을 이용하여 보다 효율적인 탐색을 전개한다는 점이 큰 특징이다. JPLAN은 효율적인 탐색을 위해 지역 탐색(local search) 알고리즘과 계획 그래프 휴리스틱(planning graph heuristics)을 이용한다. 지역 탐색 알고리즘인 EHC+는 FF[8, 9] 등의 상태 공간 계획기에 적용되어 높은 효율성을 보인 Enforced Hill-Climbing (EHC)을 확장한 것이다. EHC+는 EHC에 비해 소량의 추가적인 지역 탐색을 필요로 하지만 목표 상태까지 전체 탐색 양을 줄일 수 있고 더 짧은 계획을 얻을 수 있다. 본 논문의 휴리스틱 추출법은 Graphplan[1]에서 처음 시도된 계획 그래프(planning graph) 자료구조를 이용한다. 본 논문에서는 계획 그래프 기반의 휴리스틱들을 소개하고, 이들이 계획 생성에 미치는 효과를 실험을 통해 분석해본다.

2. 컴포넌트 서비스와 계획 문제 표현

전통적인 인공지능 계획기들은 전-조건(preconditions)과 효과(effect)로 정의되는 동작(action)들을 이용한다[7]. 하지만 현재 웹 서비스 표준 언어인 WSDL에서는 각 웹 서비스를 입력(input)과 출력(output)만을 가지는 것으로 표현한다. 반면에, 시맨틱 웹 서비스 언어들인 OWL-S[13]와 WSMO [15] 등에서는 서비스의 입력(input), 출력(output), 전-조건(precondition), 효과(effect) 모두를 표현할 수 있다. 하지만 기존 연구들을 통해 서비스의 입력과 출력을 적절한 형태의 전-조건과 효과로 변환될 수 있음이 이미 알려져 있다[12]. 특히, 저자는 선행 연구를 통해 OWL-S로 기술된 서비스 명세를 표준 계획 문제 명세 언어인 PDDL로 변환하는 방법과 변환기 구현에 대해 이미 논문으로 발표한 바 있다[10]. 따라서 본 논문에서는 PDDL[4]로 표현된 하나의 계획 문제가 계획기 JPLAN의 입력으로 주어진다고 가정한다.

계획기에 주어지는 계획 도메인과 계획 문제를 다음과 같이 정의할 수 있다.

2.1 계획 도메인(planning domain)

하나의 계획 도메인 D는 (C, A)로 표현되며, 이때 C는 술어 논리 조건(predicate logic condition)들의 집합이고, A는 동작들의 집합을 의미한다.

2.2 동작(action)

동작 a는 $(Prec_a, Effect_a)$의 쌍으로 표현되며, 이때 $Prec_a$와 $Effect_a$는 각각 동작 a의 전-조건(preconditions)과 효과(effect)를 나타내는 술어 논리 조건들의 집합이다.

자동 계획 수립을 위한 도메인 명세는 위에서 정의한 바와 같이 술어 논리 조건들의 집합과 이들을 이용해 정의되는 동작들의 집합으로 주어진다. 그리고 각 동작은 전-조건과 효과로 표현된다.

2.3 계획 문제(planning problem)

하나의 계획 문제 P는 (D, I, G)로 표현된다. 이때 D는 계획 도메인을 의미하며, I와 G는 각각 초기 상태와 목표 조건들을 나타내는 술어 논리 조건들의 집합을 의미한다.

<표 1>은 탐사 로봇 Rover의 작업 계획을 위한 도메인과 계획 문제를 PDDL(Planning Domain Description Language) 로 간략히 기술한 예이다. 표의 상단에 위치한 도메인 정보에는 도메인 이름 외에 사물의 유형(type), (at ?x - location)과 같은 술어 논리 조건식(predicate)들의 종류, 그리고 전-조건(precondition)과 효과(effect)로 구성된 동작 명세들을 포함한다. 표의 하단에 위치한 계획 문제 정의는 작업에 연관된 사물(object)들과 초기 조건들의 집합(init), 목표 조건들의 집합(goal)들을 포함한다.

〈표 1〉 PDDL로 기술한 Rover 작업 계획 문제

```
(define (domain rovers_classical)
  (:requirements :strips :typing)
  (:types location data)
  (:predicates (at ?x - location) (avail ?d -data ?x - location)
               (comm ?d - data ?x - location) (have ?d - data))

  (:action drive
   :parameters (?x ?y - location)
   :precondition (at ?x )
   :effect (and (at ?y) (not (at ?x ?y))))

  (:action commun
   :parameters (?d - data)
   :precondition (have ?d)
   :effect (comm ?d))

 (:action sample
  :parameters (?d - data ?x - location)
  :precondition (and (at ?x ) (avail ?d ?x))
  :effect (have ?d))
)

(define (problem rovers_classical1)
  (:domain rovers_classical)
  (:obejcts soil image rock -data
            alpha beta gamma - location)
  (:init (at alpha)
       (avail soil alpha)
       (avail rock beta)
       (avail image gamma))
  (:goal (and (comm soil)
              (comm image)
              (comm rock)))
)
```

3. 지역 탐색 알고리즘

3.1 상태 공간 모델과 탐색

앞서 정의한 형태로 하나의 계획 문제 $P=(D, I, G)$가 주어지면, 이 문제를 아래와 같이 하나의 상태 공간 모델(state space model) $S_p = < S, s_0, S_G, A(s), f(s,a), c(s,a) >$로 정

형화할 수 있다. 이때 각 구성요소의 의미는 다음과 같다.

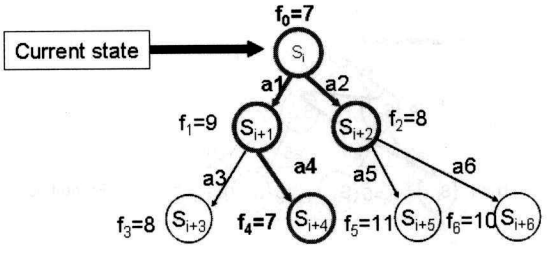
- S는 가능한 상태들의 집합
- s_0는 집합 I의 조건들로 표현되는 초기 상태
- S_G는 집합 G의 조건들을 만족하는 목표 상태들
- $A(s)$는 상태 s에 적용 가능한 동작들, 즉 $A(s)=\{ a \mid a \in A, Prec_a \subseteq s \}$
- $f(s,a)$는 동작 a에 의한 상태 s에서의 상태전이를 나타내는 상태전이함수
- $C(s,a)$는 상태 s에서 동작 a를 적용하는데 드는 비용(cost), 모든 경우 $C(s,a)$=1로 가정

상태 공간 모델의 해(solution)는 일련의 상태전이 s_0, $s_1=f(s_0,a_0)$, ..., $s_{n+1}=f(s_n,a_n)$를 통해 하나의 목표 상태 $s_{n+1}=s_G$에 도달할 수 있는 동작들의 시퀀스 a_0, a_1, ..., a_n를 말한다. 그리고 특히 총 비용 $\sum_{i=0}^{n} c(s_i,a_i)$이 최소인 해를 최적의 해(optimal solution)라고 한다. 이러한 상태 공간 모델에 기초하여 계획 문제에 대한 해를 찾는 과정은 상태 공간상의 한 탐색과정으로 볼 수 있다[3, 5]. 전형적인 탐색 방법은 탐색 방향에 따라 초기 상태에서 출발하여 하나의 목표 상태를 찾아가는 전향 탐색(forward search)과 거꾸로 목표 상태에서 초기 상태로 찾아가는 후향 탐색(backward search) 등이 있다. 일반적인 상태 공간상의 전향 탐색과정은 <표 2>에 기술한 알고리즘과 같다.

실세계 계획 문제들은 대부분 매우 큰 상태 공간을 가지므로 계획기들은 탐색의 효율성을 위해 휴리스틱 탐색 알고리즘들을 많이 채용한다. 대표적인 휴리스틱 탐색 알고리즘으로는 A* 알고리즘이 있다. A* 알고리즘은 생성된 임의의 상태들 중에서 평가함수 $f(s) = g(s) + h(s)$ 값이 최소인 상태를 매번 선택하여 확장하는 최적-우선 탐색(best-first search) 알고리즘이다. 이때 평가함수 $f(s)$는 초기 상태 s_0에서 현재 상태 s까지의 추정거리인 함수 $g(s)$와 현재 상태 s에서 한 목표 상태 s_g까지의 추정거리를 나타내는 휴리스틱

〈표 2〉 상태 공간상의 전향 탐색

```
Forward-Search(s0, SG, A, f)
  s ← s0
  π ← the empty plan
  loop
    if s ∈SG, then return π
    A(s) ← {a | a∈A and Preca ⊆ s}
    if A(s) is empty, then return failure
    Nondeterministically choose an action a from A(s)
    s ← f(s,a)
    π ← π.a
  endloop
```

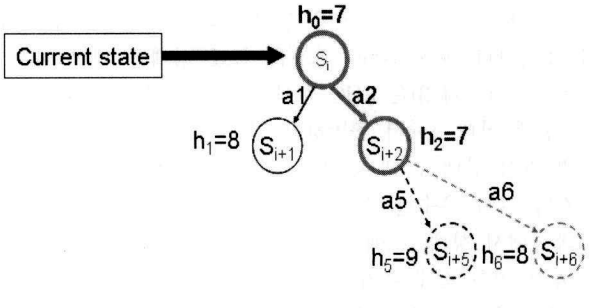


(그림 1) A* 탐색 알고리즘의 동작방식

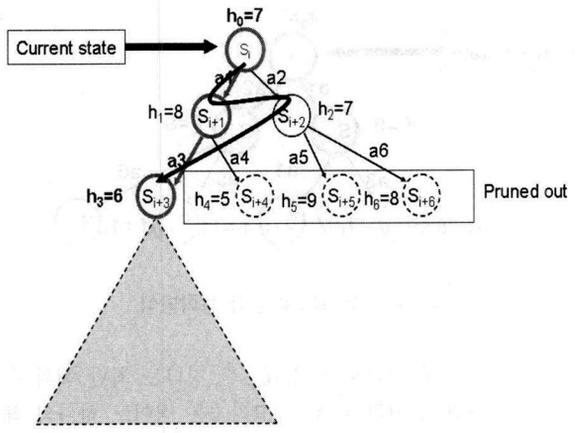
함수 $h(s)$를 합한 것으로서, 상태 s를 지나는 최단 해의 추정 길이를 나타낸다. 따라서 A* 알고리즘은 적어도 하나의 해를 찾을 수 있는 완전성(completeness)과 최적의 해를 보장하는 허용성(admissibility)을 함께 갖는 전역 탐색(global search) 알고리즘이다. (그림 1)은 A* 알고리즘에 의한 상태 공간 탐색 과정의 일부를 나타내고 있다. A* 알고리즘에 따르면 현재 상태 s_i를 확장한 후, 자식 상태들 중 평가함수 f 값이 더 작은 s_{i+2}를 선택하여 확장한다. 하지만 s_{i+2}를 확장한 다음에는, 새로운 자식 상태들 s_{i+5}와 s_{i+6}의 평가함수 f 값이 각각 11과 10인데 비해 아직 선택되지 않은 채 저장중인 형제 상태 s_{i+1}의 평가함수 f 값이 9로서 더 우수하므로 탐색 트리의 다른 가지 위에 존재하는 상태 s_{i+1}을 선택하여 확장한다. 이와 같이 전역 탐색을 전개하는 A* 알고리즘은 탐색 도중에 생성된 모든 상태들을 메모리에 저장해야만 한다. 따라서 복잡도가 큰 계획 문제들을 풀 때 메모리 부족 현상을 유발할 수 있으며, 일반적으로 탐색이 매우 느리게 진행된다.

3.2 지역 탐색 알고리즘들

일반적으로 탐색 중에 가지치기(pruning)를 통해 아직 방문하지 않은 기존의 다른 탐색 가지들을 보관하지 않고 잘라버리는 탐색 방법들을 지역 탐색(local search)이라고 부른다. 언덕 오르기(Hill-Climbing, HC)는 휴리스틱 정보를 이용하는 대표적인 지역 탐색(local search) 알고리즘이다. (그림 2)에서 보듯이, 언덕 오르기에서는 현재 상태 s_i의 자식 상태들 s_{i+1}과 s_{i+2} 중에서 휴리스틱 값이 가장 우수한 상태 s_{i+2}를 선택하여 확장하는 반면, 선택되지 않은 상태 s_{i+1}은 가지치기를 통해 메모리에서 삭제한다. 따라서 후속 탐색은 상태 s_{i+2}를 뿌리로 하는 서브트리 안에서만 진행된다.



(그림 2) 언덕 오르기(HC)의 동작방식



(그림 3) EHC 탐색 알고리즘의 동작방식

지역 탐색 알고리즘인 언덕 오르기의 장점은 가지치기를 통해 탐색 공간을 축소함으로써 복잡도가 높은 계획 문제들에 대해서도 탐색이 빠르게 진행될 수 있고, 보관해야 하는 상태들의 수가 상대적으로 많지 않아 메모리 부족 문제를 잘 겪지 않는다는 점이다. 따라서 언덕 오르기는 부모 상태에 비해 자식 상태들의 휴리스틱 값이 더 우수한 단조-감소형 휴리스틱을 이용하는 경우, 높은 탐색 효율을 얻을 수 있다. 하지만 그렇지 못한 경우, 현재의 탐색 트리 안에서는 목표 상태에 도달 할 수 있는 해를 찾지 못하거나 최적의 해를 얻지 못할 수 있다. (그림 2)의 예는 선택된 자식 상태인 s_{i+2}와 그것의 후손 상태들인 s_{i+5}와 s_{i+6}의 휴리스틱 값이 부모 상태인 s_i의 그것에 비해 같거나 더 큰 경우를 보여준다. 하지만 이때 선택되지 않은 상태 s_{i+1}는 가지치기를 통해 이미 탐색 트리에서 삭제되었으므로 그쪽으로 탐색을 되돌릴 수는 없다. 이러한 언덕 오르기의 문제점을 보완하고자 FF 계획기에서는 새로운 지역 탐색 알고리즘인 EHC (Enforced Hill-Climbing)을 제시하였다[8, 9].

EHC 탐색 알고리즘은 매번 다음에 확장할 후손 상태를 선택하기 위해, 현재 상태에서 시작하여 그 아래에 존재하는 후손 상태들에 대해 넓이-우선 탐색(breadth-first search)을 전개하여 현재 상태보다 더 우수한 휴리스틱 값을 갖는 첫 번째 후손 상태를 찾는다. 일단 현재 상태보다 더 우수한 휴리스틱 값을 갖는 후손 상태가 찾아지면, 현재 상태에서 이 후손 상태에 이르는 경로 상에 존재하는 모든 동작들을 계획에 추가 한다. 그리고 이 후손 상태에서 출발하여 다시 넓이-우선 탐색을 통해 다음에 확장할 후손 상태를 찾아간다. 이와 같은 과정은 목표 상태를 만날 때까지 반복한다. (그림 3)은 EHC 탐색 과정의 일부를 나타내고 있다. 현재 상태 s_i에서 출발하여 넓이-우선 탐색을 전개함으로써 자식 상태 s_{i+1}과 s_{i+2}를 거쳐 최초로 현재 상태보다 휴리스틱 값이 우수한 후손 상태 s_{i+3}를 찾는다. 그리고 현재 상태 s_i에서 상태 s_{i+3}에 이르는 경로 상에 존재하는 동작 a_1과 a_3를 계획 속에 추가 한 다음, 상태 s_{i+3}에서 출발하여 다시 넓이-우선 탐색을 전개한다.

언덕 오르기에서는 부모 상태보다 더 나은 휴리스틱 값을 갖지 못한 자식 상태들이라도 그 중에서 가장 우수한 상태를 선택하는 반면, EHC 탐색 알고리즘에서는 이러한 휴리스틱 지역 최소점(local minimum)을 탈출하여 반드시 더 나은 후손 상태를 찾아 확장 한다는 점이 큰 차이점이다. 따라서 EHC 탐색 알고리즘은 언덕 오르기에 비해 경우에 따라서는 확장할 후속 상태 선택을 위한 추가 탐색이 요구되지만 전체적으로 탐색의 효율성면이나 해의 질적인 면에서 더 나은 탐색 결과를 얻을 수 있다.

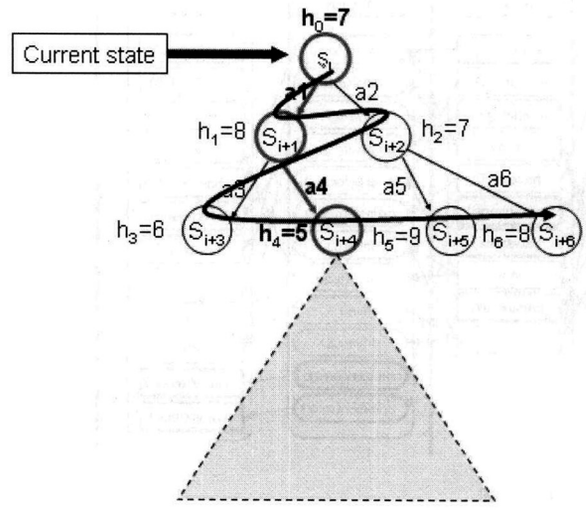
3.3 확장된 지역 탐색 알고리즘

EHC 탐색 알고리즘에 따르면, 다음에 확장할 후손 상태를 찾기 위한 넓이-우선 탐색은 현재 상태보다 더 좋은 휴리스틱 값을 가지는 첫 번째 후손 상태를 만날 때까지만 진행된다. 일단 이러한 후손 상태를 찾으면 동일한 레벨에 놓인 다른 형제 상태들은 더 이상 검사하지 않고 바로 이 상태를 선택하여 확장한다. 그리고 후속 탐색은 첫 번째로 찾은 상태 아래의 서브 트리 안에서만 진행된다. 따라서 첫 번째로 찾은 후손 상태보다 더 우수한 동일 레벨의 형제 상태가 나중에 등장하더라도 EHC 탐색 알고리즘은 이들을 찾지 못하게 된다. 예컨대, (그림 3)의 예에서 휴리스틱 값이 5로 더 우수한 형제 상태 s_{i+4}가 바로 다음에 나타날 수 있음에도 불구하고, EHC 탐색 알고리즘은 첫 번째로 찾은 후손 상태 s_{i+3}을 선택하고 나머지 상태들 s_{i+4}, s_{i+5}, s_{i+6} 등은 탐색하지 않고 모두 버린다. 따라서 EHC 탐색 알고리즘은 상태 s_{i+5}와 s_{i+6} 뿐만 아니라 더 우수한 휴리스틱 값을 갖는 상태 s_{i+4} 를 뿌리로 하는 서브 트리들은 모두 탐색에서 제외함으로써 더 나은 탐색 결과를 기대하기 어렵다.

이러한 사실을 기초로 본 논문에서는 기존의 EHC 알고리즘을 확장한 EHC+ 알고리즘을 제안한다. EHC+ 탐색 알고리즘에서는 매번 넓이-우선 탐색을 현재 상태 보다 더 나은 휴리스틱 값을 갖는 첫 번째 후속 상태를 찾을 때까지만 진행하는 EHC 알고리즘과는 달리 넓이-우선 탐색을 동일한 레벨에 존재하는 나머지 형제 상태들까지 연장한다. 이와 같은 소량의 추가 탐색을 통해 더 나은 후속 상태를 찾을 기회를 확보할 수 있고, 아무런 고려 없이 탐색에서 배제되는 상태공간을 줄일 수 있다. (그림 4)는 EHC+ 탐색 과정의 일부를 나타내고 있다. 그림에서 EHC+ 탐색은 상태 s_{i+3}뿐만 아니라 동일한 레벨의 다른 형제들까지 탐색함으로써 더 나은 후속 상태 s_{i+4}를 찾을 수 있음을 확인할 수 있다. <표 3>는 간략히 요약된 EHC+ 탐색 알고리즘을 기술하고 있다.

일반적으로 초기 상태에서 출발하여 그 상태까지는 도달 가능하지만, 그 상태에서 목표 상태까지는 도달 할 수 없는 상태를 교착 상태(deadlock state)라고 부른다. 그리고 어떤 계획 문제가 이러한 교착 상태를 전혀 포함하고 있지 않은 경우, 이것을 교착 상태가 없는 계획 문제라고 한다 (deadlock-free planning problem). 본 논문에서 제안하는 EHC+ 탐색 알고리즘은 교착 상태가 없는 어떠한 계획 문제에 대해서도 완전하다(complete). 즉, 교착 상태가 없는 계

획 문제에 대해 언제나 적어도 하나의 해를 찾을 수 있다.



(그림 4) EHC+ 탐색 알고리즘의 동작방식

〈표 3〉 EHC+ 탐색 알고리즘

```
EHC+-Search(s0, SG, A, f)
  s ← s0
  π ← the empty plan

Compute the heuristic function of s, h(s)
while h(s) > 0 do
    Begin the breadth-first search for a state s'
          s.t. h(s') < h(s)
    if such a state can not be found then
        return failure
    endif
    Further search for a sate s'' on the same level as s
          s.t. h(s'') < h(s')
    if a state s'' can be found then
        s' ← s''
    endif
    Add the actions on the path to s' at the end
        of the current plan π
    s ← s'
endwhile
    if s ∈SG, then return π
```

4. 계획 그래프 휴리스틱

4.1 간략화한 계획 그래프

상태 공간 계획기(state space planner)를 복잡도가 높은 실세계 계획 문제에 성공적으로 적용하기 위해서는 탐색에 이용할 양질의 휴리스틱 정보가 필수적이다. 일반적으로 이러한 탐색 제어 지식은 적용 영역과 문제의 특성에 따라 고유한 영역-의존적 지식이 더욱 효과적이며, 이들은 사용자나 영역 전문가로부터 제공될 수 있다. 하지만 영역에 따라서는 이러한 영역-의존적 제어 지식을 확보하기 용이하지 않거나 불가능한 경우도 많다. 따라서 본 논문에서 Graphplan[1]에서 계획 생성을 위해 처음 이용되었던 자료구조인 계획 그래프(planning graph)를 이용하여 효과적인 영역-독립적인 휴리스틱을 추출하는 방법을 소개한다. 본 논문에서 이용할 자료구조는 동작들 간의 상호배제(mutex) 정보를 포함하는 본래의 계획 그래프에 비해 보다 간략화한 계획 그래프(relaxed planning graph)이다.

• 간략화한 계획 그래프(relaxed planning graph)

동작들의 집합 A를 이용하여 생성된, 상태 s를 위한 하나의 간략화한 계획 그래프 $PG(s, A)$는 $(P_0(s), A_1(s), P_1(s), ..., A_k(s), P_k(s))$와 같은 연속 계층들로 구성된 하나의 레벨 그래프이다. 이때 각 레벨 i는 하나의 동작 계층$A_i(s)$와 하나의 명제 계층$P_i(s)$의 쌍으로 구성된다. 본 절 이하에서는 간략화한 계획 그래프를 당분간 짧게 계획 그래프로 줄여 부른다.

• 첫 번째 명제 계층(the first proposition layer)

계획 그래프 $PG(s, A)$의 첫 번째 명제 계층$P_0(s)$는 상태 s를 묘사하는 명제들, 즉 논리 조건들로 구성된다. 즉, $P_0(s) \supseteq s$

• i 번째 동작 계층(the i st action layer)

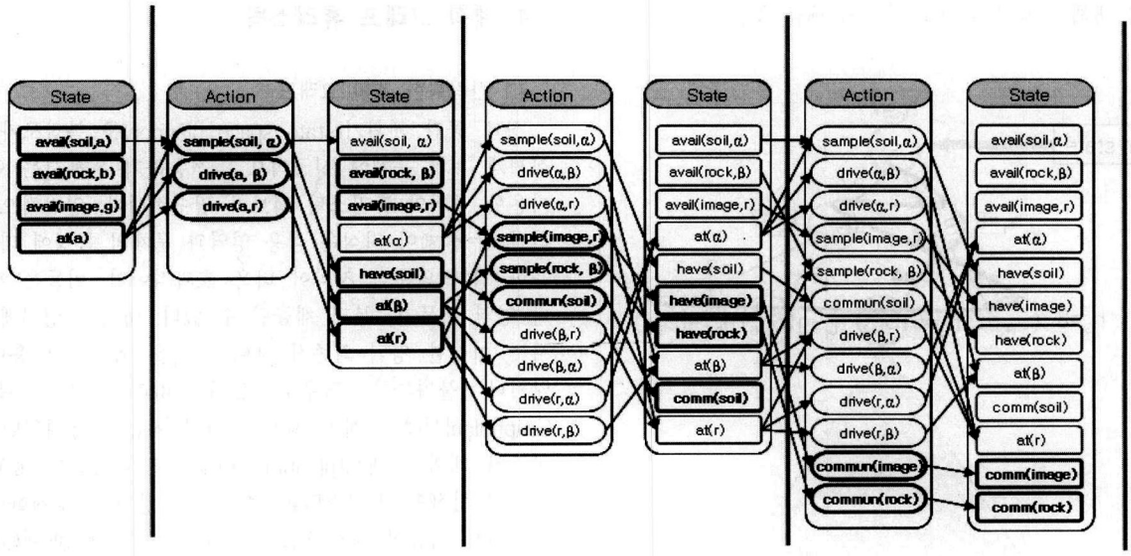
계획 그래프 $PG(s, A)$의 i 번째 동작 계층$A_i(s)$는 i-1 번째 명제 계층$P_{i-1}(s)$에 자신의 모든 전-조건(precondition)들이 포함되어 있는 동작들로 구성된다. 즉, $A_i(s) \supseteq \{ a \mid a \in A, Prec_a \subseteq P_{i-1}(s) \}$

• i 번째 명제 계층(the i st proposition layer)

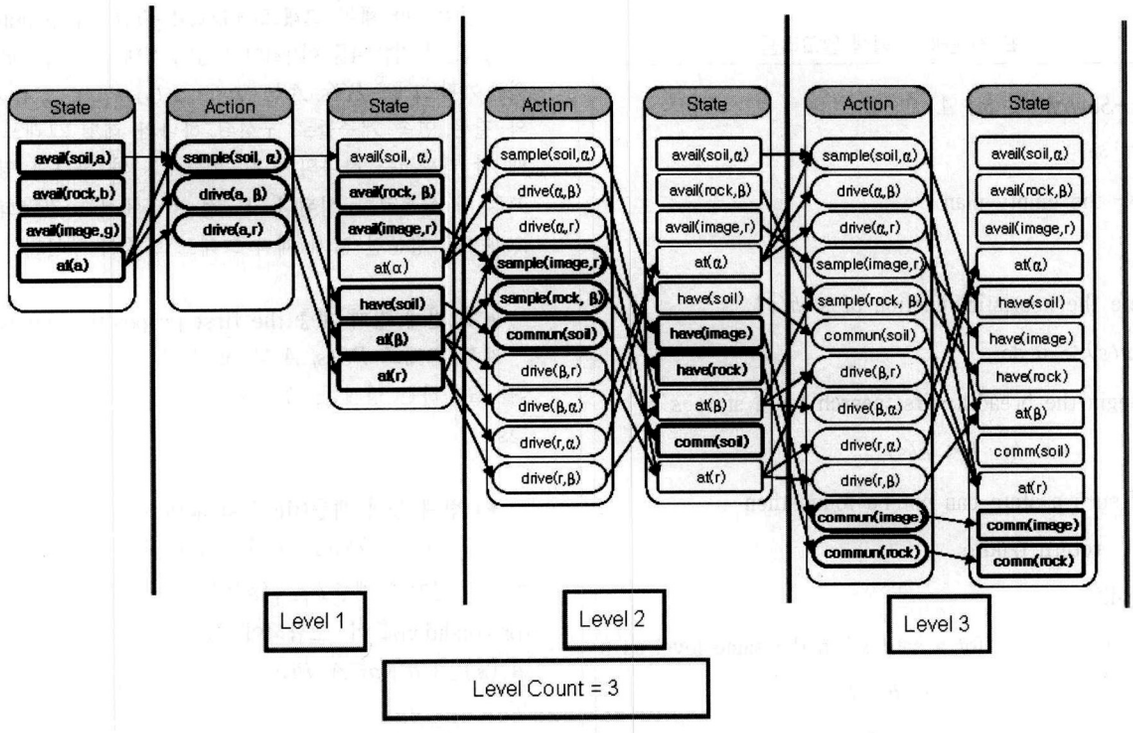
계획 그래프 $PG(s, A)$의 i 번째 명제 계층$P_i(s)$는 i 번째 동작 계층$A_i(s)$에 존재하는 각 동작들이 생성하는 긍정 효과(positive effect)들로 구성된다. 즉, $P_i(s) \supseteq \{ Effect^+_a \mid a \in A_i(s) \}$

• 유지 동작(noop action)

유지 동작 a_p는 명제 계층$P_{i-1}(s)$에 존재하는 한 명제 p를 다음 레벨의 명제 계층$P_i(s)$에도 그대로 유지시키는 특수 동작이다. 따라서 유지 동작 a_p의 유일한 전-조건과 효과



(그림 5) 간략화한 계획 그래프의 예



(그림 6) Set-Level과 Max-Level 휴리스틱

는 모두 p로 일치한다. 즉, $Prec_{ap} = Effect^{+}_{ap} = \{p\}$

• 목표 도달(goal achievement)

계획 문제의 모든 목표 조건들이 계획 그래프 *PG(s, A)*의 한 명제 계층$P_i(s)$에 모두 등장하면, 이때 이 계획 그래프는 i 번째 레벨에서 목표에 도달하였다고 한다. 즉, $P_i(s) \supseteq G$

• 레벨 종료(leveled off)

연속된 두 명제 계층의 명제들이 모두 아무런 차이가 없이 서로 일치할 때, 계획 그래프 *PG(s, A)*는 레벨 종료되었다고 한다. 즉, 명제 계층$P_{i-1}(s)$와 $P_i(s)$가 서로 똑같이 일치할 때, 계획 그래프 *PG(s, A)*는 레벨 i에서 레벨이 종료되었다고 한다.

• 계획 그래프 확장(planning graph expansion)

첫 번째 명제 계층$P_0(s)$에서 시작하여 다음의 두 조건 중 하나를 만족할 때까지 계획 그래프 *PG(s, A)*의 확장을 계

속한다: (i) 계획 그래프가 목표에 도달하거나, (ii) 레벨이 종료된다.

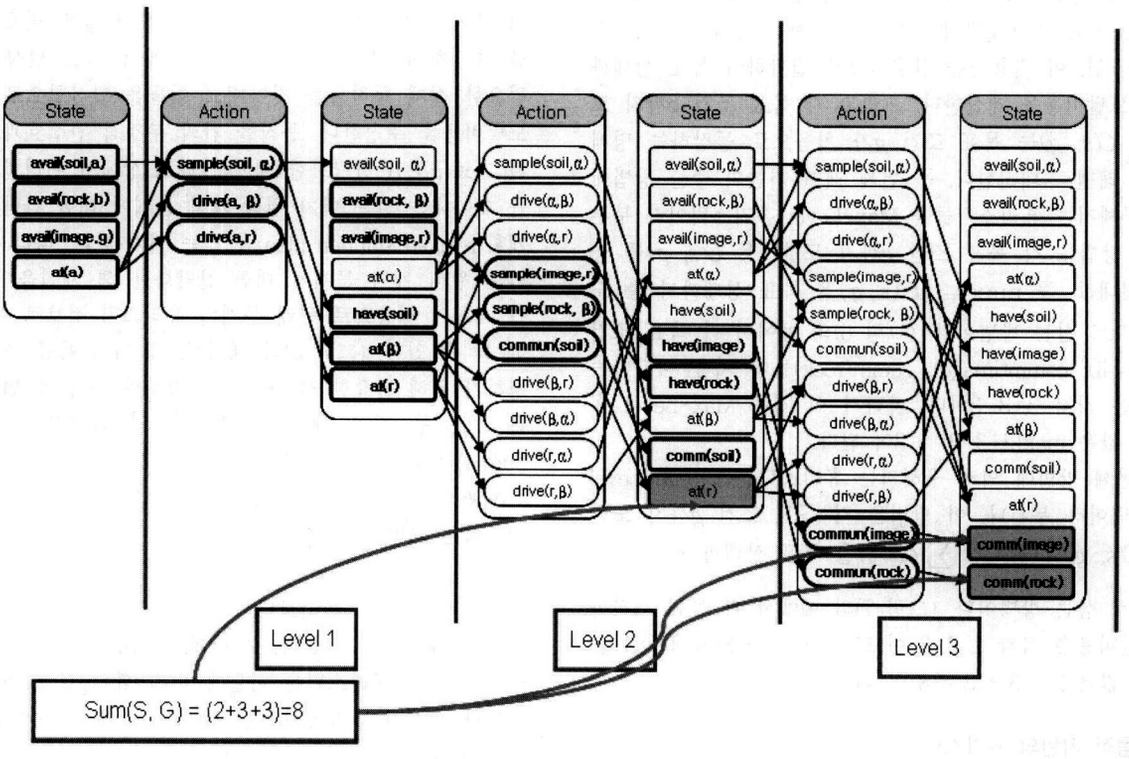
(그림 5)는 <표 1>의 Rover 작업 계획 문제를 위해 생성된 간략화한 계획 그래프의 한 예를 보여주고 있다. 그림에서 동작 계층은 Action으로, 명제 계층은 State로 표시하였다. 그림의 계획 그래프는 맨 좌측에 위치하는 초기 상태를 나타내는 첫 번째 명제 계층$P_0(s)$에서 시작하여 맨 우측에 위치한 명제 계층P_3까지 확장한 상태이다. 이 계획 문제는 {(comm(soil), comm(rock), comm(image)}와 같은 서로 다른 세 가지 목표 조건을 가지며, 이들은 모두 명제 계층 P_3에서 함께 등장한다. 또한 더 이상 새로운 명제가 추가 되지 않음으로써 명제 계층 P_3와 향후 P_4는 서로 일치하게 된다. 따라서 이 계획 그래프는 명제 계층 P_3에서 목표에 도달하였고, P_4에서 레벨 종료됨을 알 수 있다.

위에서 정의한 간략화한 계획 그래프는 다음과 같은 몇 가지 성질들을 가지고 있다. (i) 서로 다른 레벨의 동작 계층$A_i(s)$와 $A_j(s)$ $(i < j)$에 속한 동작들 간에는 레벨의 순서에 따라 순서적으로 실행이 가능하다. (ii) 동일한 레벨의 동작 계층$A_i(s)$에 속한 동작들끼리는 병행 실행이 가능하다. (iii) 레벨이 증가함에 따라 각 레벨의 명제 계층의 크기는 단조 증가한다. 즉, $P_i(s) \subseteq P_{i+1}(s)$이다. 그 이유는 명제 계층$P_i(s)$에 포함된 각 명제에 대해 각각의 유지 동작이 동작 계층$A_{i+1}(s)$에서 적용될 뿐 아니라, 동작 계층 $A_{i+1}(s)$에서 적용되는 모든 동작들의 부정 효과(negative effect)는 명제 계층$P_{i+1}(s)$에 반영되지 않기 때문이다. (iv) 계획 그래프의 각 명제 계층$P_i(s)$는 앞선 각 동작 계층 $A_1(s)$, $A_2(s)$, ..., $A_i(s)$에 속한 모든 동작들을 상태 s에 차례대로 적용한 실제 결과 상태와는 다르다. 명제 계층 $P_i(s)$는 실제 상태에 비해 더 많은 동작들의 전-조건들을 만족시킬 수 있다.

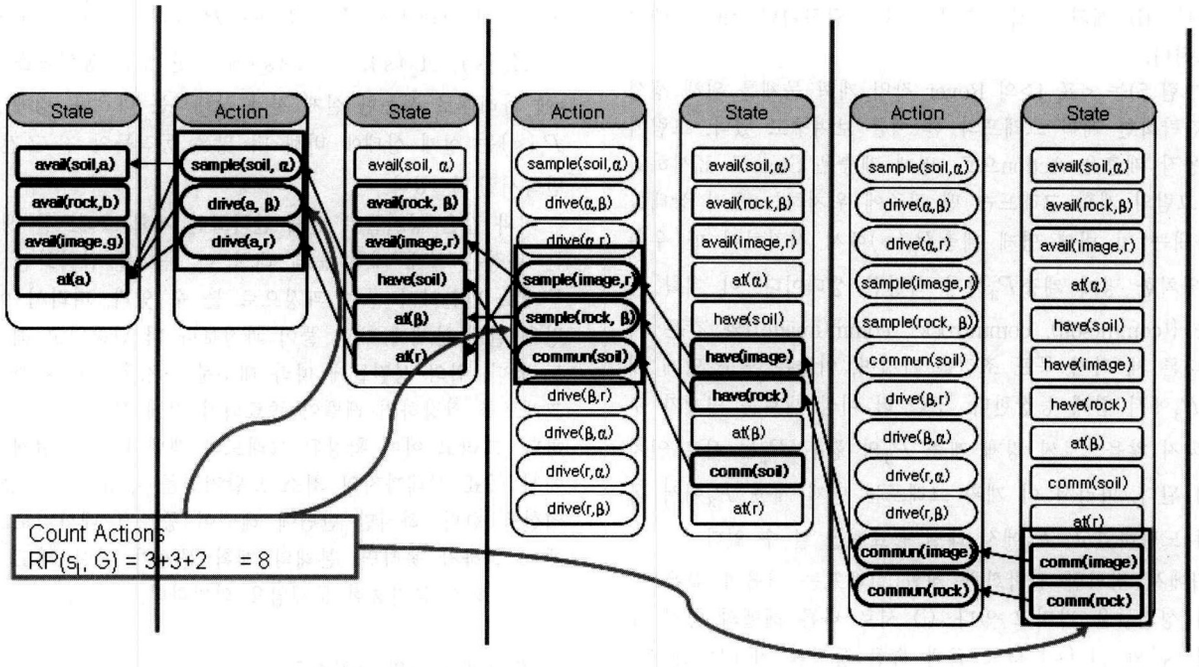
이와 같은 성질들로 인해 간략화한 계획 그래프를 생성하는 과정은 실제 계획 문제 대신 이 문제를 간략화한 계획 문제로 변환하여 푸는 과정으로 볼 수 있다. 따라서 이 과정은 실제 문제에 대한 풀이 과정보다 더 단순하고 해 길이도 짧다. 위의 성질들에 따라 대부분의 경우, 간략화한 계획 그래프를 확장하면 레벨이 종료되기 전에 목표에 도달 가능하다. 그리고 이때 확장된 그래프의 레벨의 수는 현재 상태에서 목표 상태까지의 최소 도달거리를 추정하는데 중요한 역할을 한다. 하지만 반면에 레벨이 종료될 때까지도 목표에 도달하지 못하면, 본래의 계획 문제가 어떤 해도 갖지 못하는 해결 불가능한 문제임을 의미한다.

4.2 레벨 기반의 휴리스틱

간략화한 계획 그래프 *PG(s, A)*는 상태 s에서 목표 상태에 도달하기 위한 실제 계획의 최소 길이, 즉 도달성 휴리스틱(reachability heuristic)을 도출할 수 있는 토대를 제공한다. 하나의 계획 그래프로부터 목표까지 도달성 휴리스틱을 자동으로 추출하는 방법은 크게 레벨 기반의 방법과 동작 기반의 방법으로 나눌 수 있다. 레벨 기반의 휴리스틱 추출 방법들은 모두 목표 상태까지 총 도달 비용은 각 단위 목표 조건들이 몇 번째 레벨의 명제 계층에 처음으로 등장하는가에 관련 있다고 가정한다. 첫 번째 레벨 기반의 추출



(그림 7) Sum-Level 휴리스틱



(그림 8) Sum-Action 휴리스틱

방법에 의해 구해지는 휴리스틱들을 Set-Level 휴리스틱으로 부른다. 이 방법은 (그림 6)에서 나타내듯이, 모든 목표 조건들이 함께 등장하는 계획 그래프 상의 첫 번째 레벨을 구하고 이것으로 목표 상태까지 도달하는데 필요한 실제 계획의 비용을 대신하는 방법이다. (그림 6)의 예에서는 세 가지 목표 조건들이 모두 명제 계층 P_3에서 처음으로 함께 등장하므로, 주어진 상태의 Set-Level 휴리스틱 값은 3이 된다.

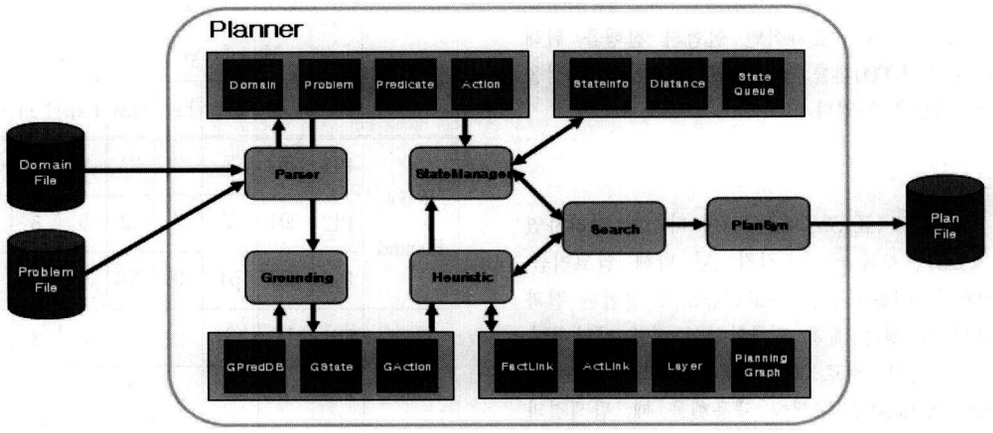
나머지 레벨 기반의 휴리스틱 추출 방법들은 모두 상태 s에서 각 목표 조건 g까지 도달하기 위한 비용 $C(s, g)$를 각각 구한 뒤, 이 비용들을 산술적으로 결합하여 목표 상태까지 총 도달비용을 계산한다. 그리고 각 목표 조건 g까지 도달비용 $C(s, g)$는 목표 조건 g가 처음으로 등장하는 명제 계층의 레벨로 대신한다. 두 번째 레벨 기반의 추출 방법에 의해 구해지는 휴리스틱들을 Max-Level 휴리스틱이라 부른다. 이 방법은 각 목표 조건 g까지 도달비용 $C(s, g)$들 중에서 최대값, 즉 $\max_{g \in G} C(s, g)$을 목표 상태까지 총 도달비용으로 삼는 방법이다. (그림 6)의 예에서 각 목표 조건 comm(soil), comm(image), comm(rock)들이 처음 나타나는 레벨들은 각각 2, 3, 3이다. 따라서 이 경우 Max-Level 휴리스틱 값은 max{2, 3, 3} = 3이 된다.

세 번째 방법에 의해 구해지는 휴리스틱들은 Sum-Level 휴리스틱이라 부른다. 이 방법은 각 목표 조건 g까지 도달비용 $C(s, g)$의 합, 즉 $\sum_{g \in G} C(s, g)$을 목표 상태까지 총 도달비용으로 삼는 방법이다. (그림 7)의 예에서 각 목표 조건까지 도달비용은 각각 2, 3, 3 이므로, 이 경우 Sum-Level 휴리스틱 값은 2 + 3 + 3 = 8 이 된다.

4.3 동작 기반의 휴리스틱

레벨 기반의 휴리스틱 추출 방법들은 모두 하나의 목표 조건을 달성하는데 드는 비용은 그 목표 조건이 처음으로 등장하는 명제 계층의 레벨과 일치한다는 가정을 가지고 있다. 하지만 계획 그래프 자료구조의 특성상 동일한 동작 계층에 속한 다수의 동작들이 모두 병행 수행 가능하다고 낙관적으로 가정하고 있으므로, 목표까지 도달비용을 계산하기 위해 레벨을 대신 이용하는 것은 문제가 있다. 이러한 한계점은 동작 기반의 휴리스틱 추출 방법을 사용함으로써 극복할 수 있다. Sum-Action이라고 부르는 이 방법은 단순히 계획 그래프의 레벨 정보로만 목표 달성에 필요한 비용을 추정하지 않고, 계획 그래프 안에서 목표 달성을 위해 필요한 실제 동작들을 찾아내고 이들을 합산함으로써 목표 도달비용을 추정한다. 동작들 간의 부정적 간섭효과가 배제된 자료구조인 간략화한 계획 그래프 상에서 이러한 동작들을 찾아내는 일은 마치 원래 계획 문제에 대한 간략화한 해 계획(relaxed solution plan)을 찾는 일과 같다. 따라서 이 방법은 원래 계획 문제에 대한 간략화한 해 계획을 구한 뒤 그 계획의 비용으로 원래 문제의 휴리스틱 정보로 이용한다고 볼 수 있다. (그림 8)의 예에서, 세 가지 목표 조건을 달성하기 위해 필요한 각 레벨별 동작들의 수가 첫 번째 레벨은 3개, 두 번째 레벨은 3개, 마지막 레벨은 2개이므로, Sum-Action 휴리스틱 값은 3 + 3 + 2 = 8 이 된다.

한편, 앞서 소개한 Set-Level 휴리스틱과 Max Level 휴리스틱은 개념적으로는 서로 다르지만 계산 결과 값은 언제나 동일하다. 또한, 이 휴리스틱들은 모두 목표까지 도달 비용을 언제나 실제보다 적게 추정하므로 허용성(admissibility)을 갖는다. 반면에, Sum-Level 휴리스틱과 Sum-Action 휴리스틱은 이들에 비해 훨씬 실제 비용에 더 근사한 추정치를 제공하지만, 때로는 실제 비용보다 더 크게 추정하기도 하기 때문에 허용성을 갖지는 못한다.



(그림 9) JPLAN 계획기의 구조

5. 계획기의 구조

본 논문에서는 앞서 설명한 지역 탐색 알고리즘과 계획 그래프 휴리스틱을 토대로 휴리스틱 탐색 계획기(heuristic search planner)인 JPLAN을 구현하였다. (그림 9)는 JPLAN 계획기의 내부 구성도이다. JPLAN은 Parser, Grounding, StateManager, Heuristic, Search, PlanSyn 등의 주요 처리 모듈들과 Grounded Predicate DB(GPreDB), Grounded Action(GAction), State Information(StateInfo), State Queue (StateQueue), Planning Graph(PlanningGraph) 등의 주요 자료구조들로 구성된다. Parser는 PDDL로 기술된 도메인 모델과 계획 문제에 대한 구문 분석을 수행하는 역할을 한다. Parser는 이를 통해 Domain, Problem, Predicate, Action 과 같은 내부 자료구조들을 만든다. Grounding은 술어논리(predicate logic) 형태의 조건식과 동작 모델을 명제논리(propositional logic)로 변환하는 역할을 한다. 즉, Domain, Predicate, Action 정보로부터 Grounded Predicate DB (GPreDB), Grounded Action(GAction)들을 생성해낸다. Search 모듈은 상태 공간상의 탐색을 통해 해 계획을 찾는 역할을 수행하며, StateManager 모듈은 탐색과정에 발생하는 상태 정보들을 관리하는 역할을 수행한다. Search 모듈에는 본 논문에서 제안한 EHC+ 탐색 알고리즘 외에도 전역 탐색 알고리즘인 A*와 지역 탐색 알고리즘인 EHC도 함께 구현하고 있다. Heuristic 모듈은 계획 그래프를 이용하여 탐색에 필요한 특정 상태의 휴리스틱 값을 도출하는 역할을 한다. 따라서 이 모듈은 각 명제 계층을 나타내는 FactLink와 행위 계층을 나타내는 ActLink, 그리고 이들로부터 얻어지는 각 Layer와, Planning Graph 자료구조를 이용한다. Heuristic 모듈은 현재 Set-Level, Max-Level, Sum-Level, Sum-Action 등 총 4 가지 계획 그래프 휴리스틱을 구현하고 있다. PlanSyn 모듈은 탐색과정을 통해 찾아진 동작들을 하나의 해 계획으로 합성하고, 이것을 계획기 내부의 표현법에서 미리 정의되어 있는 외부의 작업 계획 표현법으로 변환하는 역할을 수행한다. 이와 같은 구성요소들로 이루어진 JPLAN은 순수 Java 언어로만 구현되어 실행환경의 별다른 제한을 받지 않는다.

6. 실 험

6.1 실험 목적과 방법

본 절에서는 비교 실험을 통해 계획기 JPLAN의 성능을 분석해보고자 한다. 특히 실험을 통해 JPLAN의 두 핵심요소인 지역 탐색 알고리즘 EHC+와 계획 그래프 기반의 휴리스틱들이 탐색의 효율에 미치는 영향과 탐색 결과물로 얻어지는 해 계획(solution plan)의 질을 평가하고자 한다. 본 실험에서 탐색의 효율은 해 계획이 얻어질 때까지 탐색을 전개하는 동안 생성된 상태의 수와 확장된 상태의 수를 기준으로 판단한다. 반면에, 해 계획의 질은 계획의 길이 즉, 계획에 포함된 동작의 수로만 판단한다. 실험을 통해 비교할 탐색 알고리즘들은 3 가지 서로 다른 탐색 알고리즘 즉, 전역 탐색 알고리즘인 A*와 지역 탐색 알고리즘들인 EHC와 EHC+ 등이다. 또 비교할 서로 다른 휴리스틱들은 Max-Level 휴리스틱, Sum-Level 휴리스틱 등 2개의 레벨 기반 휴리스틱들과 Sum-Action과 같은 동작 기반의 휴리스틱 등 총 3 가지 휴리스틱들이다. Set-Level 휴리스틱의 계산결과는 언제나 Max-Level 휴리스틱과 동일하므로, 비교 실험에서 배제하였다.

계획 도메인과 문제의 특성을 고려하여, 서로 다른 5개의 로봇 도메인과 각 도메인별로 난이도가 다른 3개의 계획 문제를 임의로 생성하여 실험에 이용하였다. 도메인별로 기술되는 로봇 동작들은 각각 하나의 독립적인 기능 컴포넌트이면서, 단위 서비스를 나타낸다고 가정하였다. 실험에 이용하는 5개의 계획 도메인들은 가정용 서비스 로봇이 사용자가 수행해야 할 활동을 미리 상기시켜주는 Robot Remind 도메인, 음료수나 신문 배달 심부름을 수행하는 Robot Home-Service 도메인, 손전등과 같은 간단한 물건들을 조립하는 Robot Assembly 도메인, 탐사 작업을 계획하는 Rover 도메인, 마지막으로 화재와 같은 재난 상황에 대처하는 Robot Rescue 도메인 등이다. 실험은 3.0 GHz Intel

Pentium 4 프로세서와 1G 바이트 메모리를 가진 윈도우즈 XP 컴퓨터에서 수행하였다. 효과적인 실험의 진행을 위해 생성된 상태의 수가 10000개를 넘어가면 계획기의 탐색을 종료하는 것으로 제한을 두었다.

6.2 실험 결과

실험결과는 <표 4>, <표 5>, <표 6>에 나누어 정리하였다. <표 4>, <표5>, <표 6>은 각각 A* 탐색 알고리즘, EHC 탐색 알고리즘, EHC+ 탐색 알고리즘으로 실험한 결과를 나타낸다. 표의 각 행은 계획 도메인과 문제를 나타내며, 표의 각 열은 3 가지 서로 다른 휴리스틱(Max-Level, Sum-Level, Sum-Action)을 탐색에 채용했을 때, 탐색결과로 생성된 상태의 수(Gen), 확장된 상태의 수(Exp), 해 계획의 길이(Len)를 나타낸다. 또한, 각 표에서 탐색 제한 범위 내에 해 계획을 구하지 못한 경우는 "-" 기호로 표시하였다.

탐색 알고리즘들의 실험결과를 비교해보면, 전체적으로 두 지역 탐색 알고리즘들인 EHC와 EHC+가 전역 탐색 알고리즘인 A*에 비해 생성된 상태의 수와 확장된 상태의 수가 훨씬 적다는 것을 알 수 있다. 특히 <표 4>에서 문제의 복잡도가 높은 P33과 P53의 경우, A* 알고리즘은 제한 범위 안에 해 계획을 구하지 못한 경우도 있었다. <표 5>와 <표 6>을 비교해보면, 지역 탐색 알고리즘들 중에서는 본 논문에서 제안한 EHC+가 대부분의 경우 기존의 EHC 알고리즘보다 더 효율적인 탐색을 하였음을 확인할 수 있다. 다만 문제 P11, P41과 같이 매우 쉬운 일부의 계획 문제들에 대해서는 EHC+ 탐색 알고리즘이 EHC 알고리즘에 비해 좀 더 많은 수의 상태를 생성하고 확장한 것으로 나타났다. 그 이유는 이 문제들에서는 EHC를 통해 찾은 첫 번째 우수 후손상태보다 더 나은 후손상태를 EHC+가 발견하지 못하여, 결과적으로 매번 EHC와 동일한 후손상태를 선택하면서도 소량이지만 추가 탐색을 유발하였기 때문인 것으로 추정한다. 한편 <표 5>를 살펴보면, 문제 P23과 같이 문제의 복잡도가 크고 휴리스틱 정보가 정확하지 못한 경우 EHC 탐색 알고리즘은 해 계획을 찾지 못한 결과도 보이고 있다.

탐색 알고리즘들이 구한 해 계획의 길이를 비교해보면, <표 4>에서 전역 탐색 알고리즘인 A*는 본 실험의 모든 문제들에 대해 길이가 가장 짧은 최적의 해를 구하였다. 하지만 <표 5>와 <표 6>을 살펴보면, 지역 탐색 알고리즘들인 EHC와 EHC+는 Robot Home_Service 도메인과 Rover 도메인의 계획 문제들(P22, P23, P43)과 같이 최적의 해보다 더

〈표 4〉 A* 알고리즘의 실험 결과

도메인	문제	Max-Level			Sum-Level			Sum-Action		
		Gen	Exp	Len	Gen	Exp	Len	Gen	Exp	Len
Robot Remind	P11	42	11	2	31	7	2	31	7	2
	P12	416	43	5	79	13	5	49	11	5
	P13	2431	71	9	421	43	9	141	28	9
Robot Home_ Service	P21	34	9	2	19	5	2	19	5	2
	P22	79	24	4	43	12	4	31	9	4
	P23	1322	73	9	426	51	9	177	34	9
Robot Assembly	P31	1169	73	4	316	41	4	144	22	4
	P32	6082	152	6	751	67	6	327	45	6
	P33	-	-	-	3607	121	9	861	68	9
Rover	P41	103	29	3	37	11	3	22	7	3
	P42	579	33	7	92	17	7	41	11	7
	P43	1028	198	8	154	42	8	54	13	8
Robot Rescue	P51	1502	118	5	152	27	5	71	9	5
	P52	7069	456	9	1194	217	9	116	28	9
	P53	-	-	-	-	-	-	4292	107	11

(Gen: 생성된 상태들의 수, Exp: 확장된 상태들의 수, Len: 해 계획의 길이)

〈표 5〉 EHC 알고리즘의 실험 결과

도메인	문제	Max-Level			Sum-Level			Sum-Action		
		Gen	Exp	Len	Gen	Exp	Len	Gen	Exp	Len
Robot Remind	P11	31	7	2	23	5	2	23	5	2
	P12	204	25	5	42	9	5	42	9	5
	P13	872	54	9	276	31	9	115	17	9
Robot Home_ Service	P21	11	3	2	9	2	2	9	2	2
	P22	67	17	5	67	17	5	25	6	4
	P23	-	-	-	622	113	11	124	29	9
Robot Assembly	P31	869	59	4	255	32	4	93	15	4
	P32	2731	97	6	362	52	6	187	27	6
	P33	7633	178	9	1412	87	9	567	31	9
Rover	P41	15	7	3	8	3	3	8	3	3
	P42	281	29	7	51	16	7	23	9	7
	P43	821	48	10	59	18	8	45	15	8
Robot Rescue	P51	327	39	5	83	18	5	69	8	5
	P52	4069	131	9	927	46	9	96	21	9
	P53	9599	201	11	2312	52	11	935	47	11

〈표 6〉 EHC+ 알고리즘의 실험 결과

도메인	문제	Max-Level			Sum-Level			Sum-Action		
		Gen	Exp	Len	Gen	Exp	Len	Gen	Exp	Len
Robot Remind	P11	25	6	2	25	6	2	25	6	2
	P12	114	17	5	36	7	5	32	7	5
	P13	637	47	9	196	25	9	51	13	9
Robot Home_ Service	P21	11	3	2	9	2	2	9	2	2
	P22	52	14	4	52	14	4	35	8	4
	P23	411	33	10	134	19	10	67	16	9
Robot Assembly	P31	712	53	4	123	22	4	81	13	4
	P32	1982	84	6	277	41	6	118	21	6
	P33	5313	131	9	973	61	9	182	28	9
Rover	P41	15	7	3	11	4	3	11	4	3
	P42	81	21	7	38	14	7	19	8	7
	P43	521	31	10	34	11	8	25	10	8
Robot Rescue	P51	214	23	5	64	7	5	69	8	5
	P52	2069	51	9	496	31	9	75	16	9
	P53	6599	97	11	1312	49	11	292	27	11

긴 계획을 구할 수 있다는 것을 알 수 있다.

한편, <표 4>, <표 5>, <표 6>을 통해 서로 다른 휴리스틱들 간의 탐색 결과를 비교해보면, 어떤 탐색 알고리즘의 경우든지 휴리스틱의 정확도에 따른 순서대로 서로 다른 탐색 효율을 보여 주고 있다. 즉 목표까지 실제 비용에 가장 근사한 Sum-Action 휴리스틱, 그리고 Sum-Level 휴리스틱, 마지막으로 Max-Level 휴리스틱 순서대로 생성된 상태의 수와 확장된 상태의 수가 증가한 것을 알 수 있다. 특히 Sum-Level 휴리스틱을 이용한 EHC+ 탐색의 경우가 다른 어떤 휴리스틱과 탐색 알고리즘의 조합보다 가장 우수한 탐색 효율을 보여 주고 있다. <표 5>와 <표 6>에서 휴리스틱에 따른 지역 탐색 알고리즘의 계획 길이를 비교해보면, 이 부분 역시 Sum-Action 휴리스틱과 Sum-Level 휴리스틱의 경우가 Max-Level 휴리스틱의 경우에 비해 더 짧은 해 계획을 구하였다는 것을 확인 할 수 있다. 또 한 가지 주목할 점은 목표까지 실제 비용보다 언제나 적게 예측하는 Max-Level 휴리스틱의 경우, <표 4>에서 보듯 전역 탐색 알고리즘인 A*와 결합되었을 때는 항상 최적의 해를 찾을 수 있었으나 <표 5>와 <표 6>에서 보듯 지역 탐색 알고리즘들과 결합되면 최적의 해를 보장하지는 못하였다. 따라서 허용성을 갖지는 못하지만 실제 비용에 더 근사한 Sum-Action 휴리스틱과 Sum-Level 휴리스틱이 허용성을 갖는 Max-Level 휴리스틱에 비해 탐색 효율성뿐만 아니라 해 계획의 질적인 면에서도 우수한 실험결과를 보여주었다.

7. 결 론

본 논문에서는 컴포넌트 서비스 조합을 위해 개발된 휴리스틱 탐색 계획기인 JPLAN의 설계와 구현에 대해 설명하였다. JPLAN은 복잡도가 높은 계획 문제를 효과적으로 해결하기 위해 지역 탐색 알고리즘인 EHC+와 계획 그래프 기반의 휴리스틱들을 이용한다. 기존의 EHC 알고리즘을 확장한 EHC+는 매번 현재 상태보다 더 나은 휴리스틱 값을 갖는 후손 상태를 찾기 위해 EHC에 비해 소량의 추가적인 지역 탐색을 더 필요로 한다. 하지만, 목표 상태까지 전체 탐색 양을 줄일 수 있고 더 짧은 해 계획을 얻을 수 있는 잇점이 있다. 또한 JPLAN은 사용자가 제시하는 영역-의존적인 제어지식을 이용하는 다른 계획기들과는 달리 계획 문제 자체로부터 효과적인 휴리스틱 정보를 자동으로 추출하는 방법들을 적용하고 있다. 본 논문에서는 탐색 알고리즘 EHC+와 계획 그래프 휴리스틱들의 성능을 분석하기 위해 다양한 계획 문제들을 이용한 실험을 전개하였다. 이 실험을 통해 EHC+ 알고리즘이 일반적으로 전역 탐색 알고리즘인 A*와 지역 탐색 알고리즘인 EHC에 비해 더 적은 탐색 공간을 필요로 한다는 것과 EHC에 비해 더 짧은 계획을 얻을 수 있음을 보였다. 또, 계획 그래프 휴리스틱들 중에서 동작 기반의 휴리스틱인 Sum-Action 휴리스틱이 특히 레벨 기반의 휴리스틱들인 Max-Level 휴리스틱이나 Sum-Level 휴리스틱에 비해 더 효율적인 탐색을 유도한다는 사실도 실험을 통해 확인하였다. 현재 순수 Java 언어로 구현된 JPLAN은 특별한 제약 없이 다양한 컴퓨팅 환경에서 실행될 수 있다. 본 논문에서는 표준 계획 명세언어인 PDDL로 변환만 가능하다면 계획 생성에 이용하는 컴포넌트 서비스들의 표현법에 대해 서는 어떤 특별한 제한도 두고 있지 않다. 하지만 향후에는 보다 정형적인 컴포넌트 서비스 표현법에 관한 연구도 필요할 것으로 생각된다. 또, 현재 JPLAN의 간략화한 계획 그래프에는 각 동작의 부정적 효과(negative effect)는 포함되어 있지 않다. 따라서 휴리스틱 계산에 계획을 구성하는 동작들 간의 상호 간섭효과는 고려하지 않고 있다. 하지만 향후에는 계획 그래프에 이러한 부분들도 포함시킴으로써 좀 더 실제 비용에 근사한 계획 그래프 휴리스틱을 구하는 연구도 필요할 것으로 판단한다.

참 고 문 헌

[1] A. Blum and M. Furst, "Fast Planning through Planning Graph Analysis", Proc. of IJCAI-95, 1995.

[2] B. Bonet and H. Geffner, "HSP: Heuristic Search Planner", AI Magazine, Vol.21, No.2, 2000.

[3] B. Bonet and H. Geffner, "Planning as Heuristic Search", Journal of Artificial Intelligence, Vol.129, pp.5-33, 2000.

[4] D. McDermott, "PDDL-the Planning Domain Definition Language", Technical Report, www.cs.yale.edu/homes/dvm, 1998.

[5] H. Geffner, "Classical, Probabilistic and Contingent Planning: Three Models, One Algorithm", Proceedings of AIPS'98 Workshop on Planning as Combinatorial Search, 1998.

[6] A. Gerevini and I. Serina, "Fast Plan Adaptation through Planning Graphs: Local and Systematic Search Techniques", Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00), AAAI Press, Breckenridge, Colorado, USA, 2000.

[7] M. Ghallab, D. Nau, and P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufmann, 2004.

[8] J. Hoffmann and B. Nebel, "The FF Planning System: Fast Plan Generation through Heuristic Search", Journal of Artificial Intelligence Research, Vol.14, pp.253-302, 2001.

[9] J. Hoffmann and B. Nebel, "What Makes The Difference Between HSP and FF?", Proceedings of IJCAI-2001, 2001.

[10] H.S. Kim, I.C. Kim, "Mapping Semantic Web Service Descriptions to Planning Domain Knowledge", Proceedings of WC-06, Aug. 2006.

[11] M. Klusch, A. Gerber, M. Schmidt, "Semantic Web Service Composition Planning with OWLS-XPlan." Proceedings of the 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web, Arlington VA, USA, AAAI Press, 2005.

[12] A. Mediratta and B. Srivastava, "Applying Planning in Composition of Web Services with a User-Driven Contingent Planner", IBM Research Report RI 06002, Feb. 2006.

[13] OWL Services Coalition, "OWL-S: Semantic Markup for Web Services", 2003.

[14] J. Peer, "Web Service Composition as AI Planning", University of St. Gallen, Switzerland, 2005.

[15] D. Roman, H. Lausen, U. Keller, "WSMO Working Draft V1.0", Sep. 2004.

[16] E. Sirin, B. Parsia, D. Wu, J.A. Hendler, D.S. Nau, "HTN Planning for Web Service Composition Using SHOP2", J. Web Sem. Vol.1, No.4, pp.377-396, 2004.

김 인 철

e-mail : kic@kyonggi.ac.kr

1985년 서울대학교 수학과(학사)

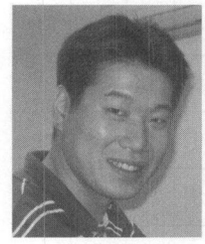
1987년 서울대학교 대학원 계산통계학과 (이학석사)

1995년 서울대학교 대학원 계산통계학과 (이학박사)

1996년~현재 경기대학교 컴퓨터과학과 교수

2003년~2004년 미시간 주립대학교 방문교수

관심분야 : 인공지능, 지능로봇, 지능형 에이전트, 자동계획 및 기계학습 등



신 행 철

e-mail : zest133@kyonggi.ac.kr

2005년 경기대학교 컴퓨터과학과 (학사)

2007년 경기대학교 대학원 컴퓨터과학과 (이학석사)

2007년~현재 경기대학교 컴퓨터과학과 AI 연구실 연구원

관심분야 : 인공지능, 지능로봇, 자동계획 및 기계학습 등