전향 상태 공간 계획을 위한 계획 그래프 휴우리스틱

신행철° 김만수 김인철
경기대학교 전자계산학과
{zest133°, i3rew,ickim}@kyonggi.ac.kr

Planning Graph Heuristics for Forward State-Space Planning

Haeng-Chul Shin° Man-Soo Kim In-Cheol Kim
Department of Computer Science, Kyonggi University

요 약

Graphplan 계획기에서 유래된 계획 그래프는 탐색에 매우 유용한 휴우리스틱을 손쉽게 얻을 수 있는 수단이다. 본 논문에서는 전향 상태 공간 계획방식을 정의한 뒤, 이 계획방식에 적용 가능한 계획 그래프 기반의 휴우리스틱 계산법들을 소개한다. 또 본 논문에서는 각 휴우리스틱 계산법이 갖는 특징을 정리하고 이들이 계획생성에 미치는 효과를 실험을 통해 비교, 분석해본다. 또한 이러한 전향 상태 공간 탐색 알고리즘과 휴우리스틱 계산법을 토대로 본 연구에서는 전향 상태 공간 계획기인 JPLAN을 구현하였고 그 구조에 대해 설명한다.

1. 서론

최근 몇 년 동안에 이루어진 자동화된 계획생성 기술의 발전은 과거에 비해 놀라울 정도로 높은 계획기(planner)의 성능 개선을 가져왔다. 이러한 계획기의 높은 성능 개선은 상당 부분 효과가 높은 도달성 휴우리스틱(reachability heuristic) 계산법을 발견하고 이것을 탐색과정에 적용한 결과 때문이다. 최근에 이용되는 대부분의 도달성 휴우리스틱은 계획 그래프(planning graph)라 불리는 자료구조를 이용하여 얻는다. Graphplan 계획기[1]에서 유래된 계획 그래프는 탐색에 매우 유용한 휴우리스틱을 손쉽게 얻을 수 있는 수단이다[8]. 본 논문에서는 전향 상태 공간 계획방식을 정의한 뒤, 이 계획방식에 적용 가능한 계획 그래프 기반의 휴우리스틱 계산법들을 소개한다. 본 논문에서는 각 휴우리스틱 계산법이 갖는 특징을 정리하고 이들이 계획생성에 미치는 효과를 실험을 통해 비교, 분석해본다. 또한 이러한 전향 상태 공간 탐색 알고리즘과 휴우리스틱 계산법을 토대로 본 연구에서는 전향 상태 공간 계획기인 JPLAN을 구현하였고 그 구조에 대해 설명한다.

2. 상태 공간 계획

일반적으로 하나의 계획문제(planning problem)는 $P=(A,I,G)$ 로 주어진다[4]. 여기서 A 는 동작(action)의 집합을 나타내며, I 는 초기 상태, G 는 목표 조건을 각각 나타낸다. 상태변경을 의미하는 각 동작 $o \in A$ 는 전통적으로 다음과 같이 정의한다: $o=(pre(o),add(o),del(o))$. 여기서 $pre(o)$ 는 동작의 적용 전조건(precondition)을, $add(o)$ 와 $del(o)$ 는 동작의 효과(effect)를 각각 나타낸다. 그림 1은 하나의 예로서, 행성 탐사용 Rover의 작업계획을 위한 동작모델과 계획문제를 나타내고 있다. 그림의 좌측은 동작들의 정의를 나타내고, 우측은 초기 상태와 목표 조건으로 구성된 하나의 계획문제를 나타낸다. 전조건 $pre(o) \subset s$ 가 만족되는 동작 o 는 상태 s에서 적용 가능한 것으로 판단한다. 그리고 동작 o 의 실행 결과로 얻어지는 다음 상태 s' 은 $(s-del(o)) \cup add(o)$ 로 정의된다.

```
(define (domain rovers_classical)
 (:requirements :strips :typing)
 (:types waypoint data)
 (:predicates
        (at ?x - waypoint)
        (avail ?d - data ?x - waypoint)
        (comm ?d - data)
        (have ?d - data))
 (:action drive
  :parameters (?x ?y - waypoint)
  :precondition (at ?x)
  :effect (and (at ?y) (not (at ?x))))
 (:action commun
  :parameters (?d - data)
  :precondition (have ?d)
  :effect (comm ?d))
 (:action sample
  :parameters (?d - data ?x - waypoint)
  :precondition (and (at ?x) (avail ?d ?x))
  :effect (have ?d))
)
```

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

그림1. PDDL로 정의된 계획 문제의 예

전향 상태 공간 계획(forward state space planning)은 초기상태에서 시작하여 이와 같은 상태변경을 통해 목표상태를 찾아가는 탐색과정으로 볼 수 있다. 초기 상태에서 시작하여 목표 상태에 도달하는 하나의 경로를 구성하는 일련의 동작들은 주어진 계획문제에 대한 하나의 해 계획(solution plan)이 된다. 이와는 반대로 목표 상태에서 시작하여 초기 상태를 찾아가는 역방향 탐색을 펼치는 계획방식을 후향 상태 공간 계획이라고 한다.

이와 같은 상태 공간 계획 방식을 복잡도가 높은 실세계 응용문제들에 성공적으로 적용하기 위해서는 효율성이 높은 탐색 알고리즘의 개발과 효과적인 휴우리스틱의 계산이 반드시 필요하다. 특히 전향 상태 공간 계획방식은 후향 계획 방식에 비해 일반적으로 탐색트리의 가지 수(branching factor)가 더 높아 탐색공간 축소를 위한 휴우리스틱 계산법이 계획기의 성능에 큰 영향을 미친다[2].

상태 공간 계획을 위해서 다양한 탐색 알고리즘들이 이용될

수 있다. 대표적인 최적-우선 탐색(Best-First Search) 알고리즘인 A*는 허용성이 있는 휴우리스틱(admissible heuristic)을 이용하는 경우 최적의 해를 찾을 수 있으나, 전역 탐색방법의 하나이기 때문에 탐색시간이 길고 요구되는 메모리 양이 매우 크다는 제한점이 있다. 따라서 많은 전향 상태 공간 계획기들은 휴우리스틱에 따라 목표 도달 가능성이 낮은 탐색가지들은 가지치기(pruning)를 하여 탐색공간을 축소하는 지역 탐색 알고리즘들을 더 선호한다. 표 1은 대표적인 지역 탐색 알고리즘인 Enforced Hill-Climbing (EHC)[5]을 나타내고 있다. EHC는 기존의 언덕오르기(Hill-Climbing) 탐색의 문제점을 개선한 것으로서, 현재 상태 노드에서 시작하여 넓이-우선 탐색(Breadth-First Search)을 전개하여 최초로 현재 상태 노드보다 낮은 휴우리스틱 값을 갖는 후손노드를 찾는다. 그리고 이와 같은 탐색과정을 반복함으로써 목표상태를 찾아간다.

표1. 상태 공간 탐색 알고리즘

```
function EnforcedHillClimbing() computes
validPlan: Plan or fails
input: InitialState : State
input: GoalState : State
local: S : State /* the current computed state */
local: S' : State /* possible successor of S */
local: currPlan : Plan /* current plan */
local: h_s: int
 /* the distance of S to a goal computed by use of of Relaxed Planning Graph
*/
local: h_s' : int
local: N_s [] : List of Actions /* List of helpful action based on state S*/
local: N_s' [] : List of Actions
begin
    /* The initial plan is empty */
    currPlan = <>;
    S = InitialState;
     /* Compute the distance from starting state to goal */
    h_s = BuildRelaxedPlangraph(S,GoalState);
    N_s = GetHelpfulActions(S);
    while h_s ≠ 0 do
      S' = BFS Expand(S, N_s);
      if S' == NULL then
          return FAILURE;
      else
         /* If a state S' is found, the action sequence is attached to the end
         of the current plan, that enables to get from S to S'. */
         currP lan = currP lan + ActionsPath(S, S');
         UpdateGlobalFluents(S,S');
         /* The search goes on beginning with S'. N_s' is computed before
         by BFS Expand and can still be use. */
         S = S';
         N_s = N_s' ;
      end
    end
    return currPlan;
End
```

3. 계획 그래프 기반의 휴우리스틱

계획 그래프(planning graph)는 탐색공간 상의 한 상태를 시작노드로 상아 실제에 근사한 탐색트리를 펼쳐봄으로써 매우 효과적인 도달성 휴우리스틱(reachability heuristic)을 손쉽게 계산할 수 있는 자료구조이다[3, 8]. 계획 그래프는 Blum과 Furst에 개발된 Graphplan 계획기[1]를 통해 처음 소개되었다. 이후 계획 그래프는 많은 상태 공간 계획 알고리즘과 계획기에서 휴우리스틱 계산에 널리 이용되고 있다. 그림 2에 주어진 예에서 보듯이, 계획 그래프는 일반적으로 여러 레벨(level)로 구성되며 각 레벨i는 다시 하나의 동작층(action layer) A_i과 사실층(fact layer) P_i으로 구성된다. 동작층에는 이전 레벨의 사실들에 의해 적용 전조건(precondition)이 만족되는 동작들이 놓여지고, 사실층에는 이러한 동작들이 실행됨으로써 얻어지는 효과들 중 추가 조건(add list)에 속하는 사실들만 놓여진다. 사실과 동작간의 링크는 이들간의 지원관계(supporting relationship)와 역지원관계(supported relationship)를 의미한다. 따라서 하나의 계획 그래프는 원래 계획문제보다 더 간략화된 계획문제(relaxed planning problem)를 풀어가는 탐색과정을 표현한다.

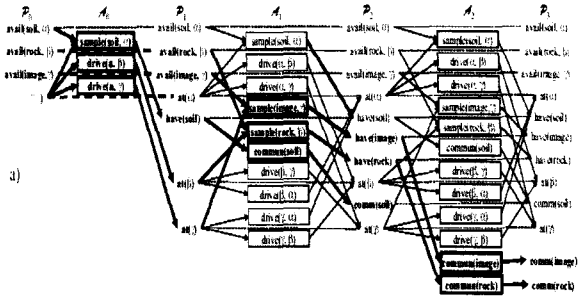

그림2. 계획 그래프의 예

표2. 계획 그래프 생성 알고리즘

```
function BuildRelaxedPlanningGraph() computes
relaxedPlanningGraph or fails
input: InitialFacts[] : List of Facts
input: GoalFacts[] : List of Facts
local: CurrentlayerFacts[], NextLayerFacts[] : List of Facts
local: CurrActivActions[] : List of Actions
local: CurrentLayer : int
begin
    CurrentlayerFacts = InitialFacts; CurrentLayer = 0;
    while ! AllGoalsActive(GoalFacts) do
       foreach Fact f in CurrentLayerFacts do
           Increment precondition counter of actions
           which f is a precondition of;
       end
       foreach Fact f in CurrentLayerFacts do
       /* Select all executable actions, that are part of the current layer */
          CurrActivActions +=
          GetActiveActions(CurrentLayer, f);
       end
       foreach Action a in CurrActivActions do
          if all preconditions of a are satisfied AND
            Layer of a == CurrentLayer then
               NextLayerFacts +=
               GetAddedFactsFromAction(a);
               RemoveFromList(a, CurrentActiveActions);
          end
       end
       CurrentLayerFacts = NextLayerFacts;
       NextLayerFacts = <>;
       /* Increasing layer counter and continue with next layer */
       Currentlayer = CurrentLayer + 1;
       if CurrentlayerFacts == <> then
       /* If a fix point is reached regarding facts and actions and
          the goal isn't fulfilled, the problem isn't solvable */
       if ! AllGoalsActive(GoalFacts) then
          return FAILURE;
       end
    end
    return CurrentLayer;
end
```

계획 그래프는 단계적으로 레벨을 늘여가는 방식으로 확장되며, 모든 목표 조건들이 만족되는 사실층을 만날 때까지 확장을 계속한다. 표 2는 이와 같은 방식으로 하나의 계획

그래프를 생성하는 알고리즘을 나타내고 있다. 그림 2의 예에서 맨 좌측에 위치한 첫 번째 레벨의 사실층 P_1에는 초기 상태를 나타내는 (at alpha), (avail soil alpha), (avail rock beta) 등이 포함되고, 맨 우측에 위치한 마지막 레벨의 사실층에는 목표조건들인 (comm soil), (comm. image), (comm. rock) 등이 모두 포함되어 있다.

계획 그래프를 이용하여 그림 3, 그림 4와 같이 다양한 유형의 휴우리스틱을 계산해낼 수 있다. 그림 3은 계획 그래프의 레벨 정보만을 이용하는 3 가지 유형의 휴우리스틱 계산 방법(set-level, max-level, sum-level)을 나타내고 있다. 이들 중에 set-level는 현재 상태에서 시작해서 목표 상태에 도달할 때까지 계획 그래프를 확장한 다음, 그때 최대 레벨 수(maximal level number)를 현재 상태에서 목표 상태에 이르는 예측거리(estimated distance)로 삼는 방법이다. max-level은 확장된 그래프에서 각 단위 목표별 레벨을 따로 구한 다음 최대치를 선택하는 방법이다. 각 동작의 소요비용(cost)이 모두 일정한 경우, set-level 계산법과 max-level 계산법의 실제결과는 모두 일치한다. 반면에 sum-level은 각 단위 목표가 처음 달성되는 레벨들을 모두 합산하여 휴우리스틱을 계산하는 방법이다. 그림 4는 이들과는 달리 역방향 탐색을 통해 각 단위 목표를 달성하기 위해 필요한 실제 동작들을 그래프 상에서 찾아내고 이들의 수를 합산함으로써 휴우리스틱을 계산하는 sum-action방법을 보여주고 있다. 그림 2의 계획 그래프 경우, 단위 목표 (comm soil), (comm. image), (comm. rock)들이 처음 나타나는 레벨들은 각각 2, 3, 3이다. 따라서 그림 3과 같이 set-level과 max-level 계산법에 따르면 휴우리스틱 값은 모두 3이고, sum-level 계산법에 따르면 8이다. 반면에, 그림 4와 같이 목표 달성에 필요한 동작들을 찾아내어 sum-action을 계산하면 휴우리스틱 값은 8이 된다. 표 3은 동작 기반의 휴우리스틱 계산법인 sum-level에 대한 알고리즘을 나타낸다.

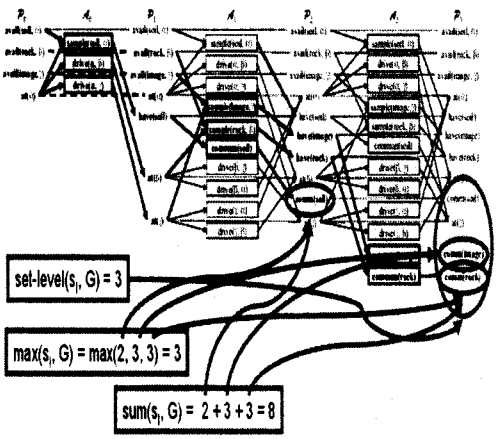

그림 3. 레벨 기반의 휴우리스틱들

위에서 소개한 계획 그래프 기반의 휴우리스틱 계산방법들은 모두 계획을 구성하는 동작들 사이에 존재하는 부정적 상호작용(negative interaction)을 고려하지 않는다. 따라서 4 가지 방법 모두 목표까지의 거리를 실제보다 과소평가하여 휴우리스틱의 허용성(admissibility)을 만족한다. 한편, 간략화된 계획 휴우리스틱의 경우는 레벨 기반의 휴우리스틱들과는 달리 동일 레벨에 존재하는 동작들이 모두 병행 수행 가능하다는 가정을 하지 않는다. 그뿐만 아니라 계획을 구성하는 동작들 사이에 존재하는 긍정적 상호작용(positive interaction)을 찾아내어 휴우리스틱 계산에 반영함으로써 다른 휴우리스틱보다 더 풍부한 정보를 가진다.

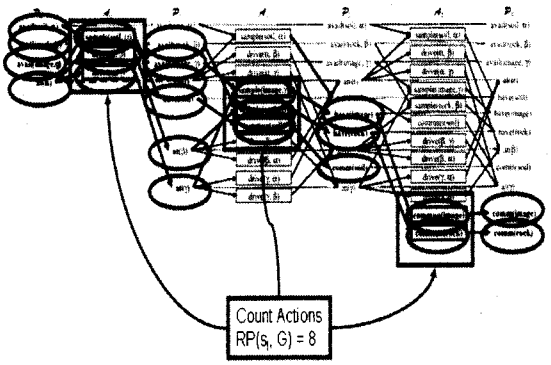

그림4. 동작 기반의 휴우리스틱

표3. 동작 기반 휴우리스틱 계산 알고리즘

```
function ActionBasedHeuristic() computes HeuristicValue: int
local: GoalFacts[] : List of GoalFacts
local: CurrentLayerFacts[] : List of Facts
local: AllLayers[]:List of Layer
local: CurrActivActions[] : List of Actions
local: CurrentLayer : int
local: CurrentActionCount : int
local: H_Value: int

begin
CurrentLayer = 0;
foreach Fact f in AllLayers do
    if Fact of f == GoalFacts then
        SetCurrentLayerGoalFact(f);
    end
    CurrentLayer ++;
end

while CurrentLayer > 1 do
    foreach Fact f in CurrentLayerFacts do
        if Fact f in GetCurrentLayerGoalFact == CurrentLayerFacts then
            if all effects of a are satisfied AND
                Layer of a == CurrentLayer then
                CurrentActionCount += GetSatisfiedActions(a);
            end
        end
    end

    foreach Action a in CurrActivActions do
        if Action a in GetSatisfiedActions == CurrActivActions then
            if all preconditions of a are satisfied AND
                Layer of f == CurrentLayer then
                GetSatisfiedfacts(f);
            end
        end
    end
    CurrentLayer--;
end
H_Value = CurrentActionCount;
return H_Value;
end
```

4. 전향 상태 공간 계획기

본 연구에서는 앞서 설명한 상태 공간 탐색 알고리즘과 계획

그래프 기반의 휴우리스틱 계산법을 토대로 전향 상태 공간 계획기(forward state space planner) JPLAN을 구현하였다. JPLAN은 그림 5와 같이 Parser, Grounding, State Manager, Heuristic, Search, Plan Synthesizer 등의 주요 구성요소들과 State Queue, Planning Graph 등의 기타 자료 구조들로 구성된다. Parser는 PDDL로 기술된 동작모델과 계획문제에 대한 구문분석을 통해 Domain, problem, Predicate, Action 등의 내부 자료구조들을 만드는 역할을 하며, Grounding은 술어논리(predicate logic) 형태의 상태정보와 동작모델을 명제논리(propositional logic)로 변환하는 역할을 한다.

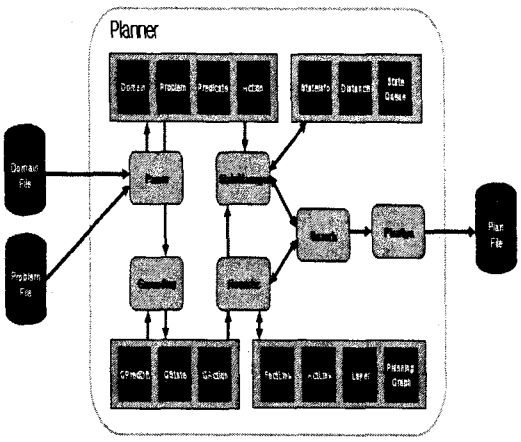

그림5. 계획시스템의 구조

Search부는 상태 공간 탐색을 통해 해 계획을 찾는 역할을 수행하며, StateManager부와 Heuristic부는 각각 이를 위해 탐색과정에 발생하는 상태 정보를 관리하고 이들에 대한 휴우리스틱을 계산하는 역할을 한다. 앞서 설명한대로 각 상태의 휴우리스틱은 계획그래프를 이용하여 계산된다. 마지막으로 PlanSyn부는 탐색과정을 통해 구해진 해를 하나의 작업 계획으로 합성하고 변환하는 역할을 수행한다.

5. 실험

본 논문에서는 앞서 제시한 서로 다른 3 가지 유형의 계획 그래프 휴우리스틱들(max-level, sum-level, sum-action) 사이의 탐색효율과 결과계획을 비교하기 위한 실험을 전개하였다. 실험을 위해서 전향 탐색 알고리즘으로 A* 알고리즘을 사용하였고, JPLAN 계획기 내부에 서로 다른 휴우리스틱 계산법을 구현하였다.

실험에는 총 8 가지 서로 다른 영역의 계획문제들을 사용하였고, 각 영역당 임의로 생성된 3 개의 계획문제를 실험에 적용하였다. 실험에 사용된 8 가지 서로 다른 영역의 계획문제들은 다음과 같다. 학생이 어떤 수업을 듣고 어느 강의실로 이동해야 하는지를 계획하는 Class 문제, 매복하고 있던 병사가 적과 교우 시에 어떻게 대처를 해야 하는지를 계획하는 Ambush 문제, 화물을 어떻게 옮길 것 인가를 계획하는 Cargo 문제, 홈 서비스 로봇이 화재 발생 시에 취해야 할 행동을 계획하는 Robot_Emergency 문제, 홈 서비스 로봇이 사용자에게 음료수 심부름을 제공하기 위한 Robot_Service 문제, 여행 경로를 계획하는 Travel 문제, 세일즈맨이 어떻게 효율적으로 판매활동을 할지를 계획하는 Tsp 문제, 블록을 어떻게 쌓을 지를 계획하는 Block 문제 등이다. 실험에 사용된 각 계획문제의 특성을 표 4에 요약하였다. 표 4는 각 계획문제를 구성하는 연산자(operator)의 수, 평균 가지수(branching factor), 최적 해(optimal solution)의 길이 등을 나타내고 있다.

표4. 실험에 사용된 계획문제들

	연산자 수	평균 가지수	최적 해의 길이
Class	2	4.3	4
		4.0	2
		4.3	6
Ambush	13	9.7	4
		9.3	4
		14.3	5
Cargo	3	6.2	6
		6.2	6
		5.6	4
Robot_Emergency	7	2.4	8
		3.4	6
		2.3	6
Robot_Service	9	4.4	5
		4.4	5
		3.6	4
Travel	4	3.3	2
		3.0	2
		3.5	9
Tsp	4	1.2	6
		2.6	10
		1.7	10
Block	3	2.9	3
		3.5	4
		7.8	6

본 실험에서는 해 계획을 도출할 때까지 각 휴우리스틱이 유도한 탐색의 효율성을 분석하기 위해, 생성된 총 상태의 수, 확장된 총 상태의 수를 구해보았다. 그리고 마지막으로 각 휴우리스틱이 유도한 탐색결과로 얻어진 해 계획의 질을 판단하기 위해 해 계획의 길이를 비교해 보았다. 표 5는 실험결과를 나타내고 있다.

max-level 휴우리스틱은 최종 목표가 만족되는 마지막 레벨을 휴우리스틱 값으로 취하게 된다. 따라서 간단한 계획문제의 경우 선택 가능한 대부분의 상태들이 모두 비슷한 휴리스틱 값을 가지게 된다. 이러한 이유로 너비 우선 탐색과 유사한 상태 공간 탐색을 전개하였다. 생성상태 수와 확장상태 수면에서 max-level 휴우리스틱이 다른 휴우리스틱 계산법에 비해 많다는 것을 표 5에서 알 수 있다. 그러나 A* 탐색 알고리즘의 특성으로 인해 결국 최적의 해 계획을 구하였다는 사실을 표 4과 표5의 비교를 통해 확인할 수 있다.

sum-level 휴우리스틱의 경우, max-level 휴우리스틱과 마찬가지로 레벨기반의 알고리즘이다. 이 휴우리스틱은 기존의 max-level 휴우리스틱과 다르게 각 단위 목표가 처음 달성되는 레벨들을 모두 합산하여 휴우리스틱을 계산하는 방법이다.

표5. 실험결과

	Max-Level			Sum-Level			Sum-Action		
	생성 노드	확장 노드	계획 길이	생성 노드	확장 노드	계획 길이	생성 노드	확장 노드	계획 길이
Class	239	52	4	26	6	4	26	6	4
	9	2	2	9	2	2	9	2	2
	2504	564	6	43	10	6	43	10	6
Ambush	2602	238	4	63	7	4	63	7	4
	1006	109	4	39	4	4	78	8	5
	5243	462	5	347	21	5	128	8	6
Cargo	15098	2541	6	255	40	6	255	40	6
	14258	2253	6	178	31	6	178	31	6
	780	134	4	122	24	4	122	22	4
Robot_Emergency	844	333	8	212	86	8	212	86	8
	230	74	6	93	25	6	93	25	6
	471	194	6	43	19	6	43	19	6
Robot_Service	322	73	5	322	73	5	322	73	5
	322	73	5	95	21	5	95	21	5
	35	10	4	43	19	4	43	19	4
Travel	1	4	2	1	4	2	1	4	2
	13	4	2	13	4	2	70	18	6
	32544	9270	9	270	86	9	103	31	9
Tsp	440	239	6	34	20	6	34	22	6
	907	740	10	27	14	10	31	20	10
	965	907	10	51	37	10	43	33	10
Block	12	3	3	12	4	3	125	44	3
	47	11	4	58	4	4	58	4	4
	4592	562	6	193	26	6	193	26	6

그러므로 이 휴우리스틱은 max-level 휴우리스틱에 비해 언제나 큰 휴우리스틱 값을 갖게 되며, 또한 좀더 많은 정보를 포함하게 된다. 따라서 실제 실험결과도 표 5에서 보듯이 대부분의 경우 탐색효율이 더 높은 결과를 나타내었다. 즉, max-level 휴우리스틱과 비교했을 때, 생성상태 수와 확장상태 수가 모두 더 적다는 것을 확인할 수 있다. 또한 탐색을 통해 최적의 해 계획을 도출하였다는 것을 알 수 있다.

한편, 동작 기반의 sum-action 휴우리스틱은 각 단위 목표를 달성하기 위해 필요한 그래프 상의 동작들을 모두 합산하는 방법으로 구한다. 또한 이 휴우리스틱은 계획을 구성하는 각 동작들 사이의 긍정적 상호작용도 고려한다. 따라서 계획 그래프의 레벨만을 고려하는 휴우리스틱들에 비해 더 풍부한 정보를 포함한다. 표5에서 보듯이 실제 실험결과에서도 대부분의 경우 max-level에 비해 상대적으로 더 높은 탐색효율을 보여 주었고, sum-level과도 비슷한 탐색효율을 보여 주었다.

6. 결론

본 논문에서는 계획 그래프의 개념과 효율적인 상태 공간 계획을 위한 계획 그래프 휴우리스틱 계산법들을 소개하였다. 본 논문에서 소개한 계획 그래프 기반의 휴우리스틱 계산법들은 모두 원래 계획문제에 대한 간략화를 통해 현재 상태에서 목표까지의 거리를 추정한다. 본 논문에서는 다양한 영역의 계획문제를 대상으로 비교 실험을 수행하였고 이를 통해 계획 그래프 휴우리스틱들의 특징과 효과를 확인할 수 있었다.

참고 문헌

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

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

[3] D. Bryce and S. Kambhampati, " Planning Graph Heuristics for Belief Space Search" , Journal of Artificial Intelligence Research, 2006.

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

[5] J. Hoffmann and B. Nebel," The FF Planning System: Fast Plan Generation through Heuristic Search" , Journal of Artificial Intelligence Research, 2001.

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

[7] D. McDermott, " Using Regression-Match Graphs to Control Search in Planning" , Artificial Intelligence, Vol.109, No.1-2, pp.111-159, 1999.

[8] X. Nguyen and S. Kambhampai, " Extracting Effective and Admissible Heuristic from the Planning Graph" , Proc. of AAAI-00, 2000.