휴우리스틱 자원 시간 계획을 위한 탐색 알고리즘

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

A Search Algorithm for Heuristic Resource Temporal Planning

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

요 약

본 논문에서는 휴우리스틱 자원 시간 계획을 위한 새로운 탐색 알고리즘인 Strictly Enforced Hill-Climbing (SEHC)을 제안한다. 이 탐색 알고리즘은 FF 등의 계획기에 적용되어 매우 높은 효율성을 보인 Enforced Hill-Climbing (EHC)을 확장한 것이다. EHC는 목표를 찾아가는 과정 동안 매번 현재 상태에서 그 상태보다 더 낮은 휴우리스틱 값을 갖는 첫 번째 후손 상태를 찾아 넓이 우선 탐색을 펼치는 데 반해, 본 논문에서 제안하는 SEHC는 찾아진 첫 번째 후손 상태와 같은 깊이의 나머지 형제 상태들까지 탐색을 연장하여 최소의 휴우리스틱 값을 갖는 후손 상태를 찾아낸다. 이와 같은 SEHC 탐색방법은 매 주기마다 소량의 추가 탐색을 통해 탐색의 전체과정 동안 EHC 보다 우수한 탐색경로를 유지할 수 있도록 해준다. 본 논문에서는 다양한 영역의 계획문제를 대상으로 A* 알고리즘, EHC 알고리즘 등과의 비교실험을 통해 SEHC알고리즘의 우수성을 알아본다.

1. 서론

자원 시간 계획 문제(Resource Temporal Planning Problem)는 전통적인 계획문제를 확장하여 각 동작의 소요시간(duration time)과 소요자원(resource)까지 고려하여 최소의 실행시간과 실행비용을 갖는 계획을 찾아내는 문제이다. 이러한 자원 시간 계획 문제를 풀기 위한 다양한 계획방법들이 연구되고 있으나, 이중에서 대표적인 방법이 전통적인 상태공간상의 휴우리스틱 탐색을 이용하는 휴우리스틱 자원 시간 계획방식이다. 그 동안 상태 공간 탐색에 널리 적용되어온 휴우리스틱 탐색 알고리즘들로는 A* 알고리즘, Hill-Climbing, Enforced Hill-Climbing 등이 있다. 특히 FF 계획기에서 소개된 Enforced Hill-Climbing (EHC)은 Helpful Action들을 이용한 가지치기(pruning)와 함께 매우 높은 탐색 효율성을 보이는 것으로 알려져 있다. 본 논문에서는 휴우리스틱 자원 시간 계획을 위한 새로운 탐색 알고리즘인 Strictly Enforced Hill-Climbing (SEHC)을 제안한다. 이 탐색 알고리즘은 기존의 Enforced Hill-Climbing (EHC)을 확장한 것이다. EHC는 목표를 찾아가는 과정 동안 매번 현재 상태에서 그 상태보다 더 낮은 휴우리스틱 값을 갖는 첫 번째 후손 상태를 찾아 넓이 우선 탐색(Breadth-First Search, BFS)을 펼치는 데 반해, 본 논문에서 제안하는 SEHC는 더 낮은 휴우리스틱 값을 갖는 첫 번째 후손 상태와 같은 깊이의 나머지 형제 상태들까지 탐색을 연장하여 최소의 휴우리스틱 값을 갖는 후손 상태를 찾아낸다. 이와 같은 SEHC 탐색방법은 매 주기마다 소량의 추가 탐색을 통해 탐색의 전체과정 동안 EHC 보다 우수한 탐색경로를 유지할 수 있도록 해준다. 본 논문에서는 다양한 영역의 계획문제를 대상으로 A* 알고리즘, EHC 알고리즘 등과의 비교실험을 통해 SEHC알고리즘의 우수성을 알아본다.

2. 계획과 상태 공간 탐색

일반적으로 계획(plan)은 목표를 달성하기 위한 동작들의 시퀀스를 말하며, 탐색을 통해 이러한 계획을 찾아가는 과정을 계획수립(planning)이라 부른다. 고전적인 계획문제(planning problem)는 $P=(A,I,G)$ 로 표현된다. 여기서 A 는 동작(action)의 집합을 나타내며, I 는 시작 상태, G 는 목표 상태를 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. 상태 s 에서 $pre(o) \subset s$ 가 만족되면 동작 o 는 실행 가능하며, 실행 결과상태는 $(s-del(o)) \cup add(o)$ 가 된다[1]. 계획수립과정은 이와 같은 상태변경을 통해 초기상태에서 시작하여 목표상태를 찾아가는 탐색과정으로 볼 수 있다. 그림 1은 이와 같은 계획을 위한 상태 공간상의 탐색을 나타내고 있다.

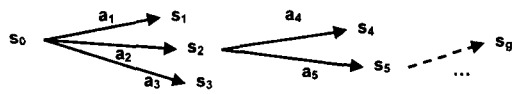

그림1. 계획을 위한 상태 공간 탐색

3. 휴우리스틱 자원 시간 계획

휴우리스틱 자원 시간 계획방식은 상태 공간상의 휴우리스틱 탐색을 통해 최소 실행시간과 실행비용을 가지는 계획을 찾는 방식을 말한다. 이러한 휴우리스틱 자원 시간 계획을 위해서는 동작들의 소요시간과 소요자원을 고려한 각 상태의 휴우리스틱 계산법과 이러한 휴우리스틱에 기초한 상태 공간 탐색 알고리즘이 중요하다[2]. Sapa를 비롯한 대부분의 휴우리스틱 자원 시간 계획기들은 각 상태의 휴우리스틱 계산을 위해 확장된 계획 그래프(planning graph)를 이용한다. 전통적으로 계획 그래프를 이용한 휴우리스틱 계산방법은 각 동작을 간략화한 다음, 모든

동작의 소요시간과 소요비용이 단위시간과 단위비용으로 모두 동일하다고 가정하여 목표상태까지의 추정거리를 계산한다. 하지만 자원 시간 계획을 위해서는 목표상태에 이르는 각 동작의 서로 다른 실행시간과 소요비용을 추적하여 계획 그래프를 작성하고 이것을 바탕으로 각 상태의 휴우리스틱을 계산한다.

4. 탐색 알고리즘

상태 공간 탐색에 널리 적용되어온 휴우리스틱 탐색 알고리즘들로는 A* 알고리즘, Hill-Climbing, Enforced Hill-Climbing 등이 있다. A* 탐색 알고리즘은 각 상태 n에 대해 평가치 f(n)=g(n)+h(n)를 계산하여 이 평가치가 최소인 노드를 다음 확장 노드로 선택하는 과정을 반복한다. 즉, A* 알고리즘은 상태 평가에 목표까지의 거리를 의미하는 휴우리스틱 h(n)외에 초기상태에서 그 상태에 이르는 거리 g(n)도 함께 평가한다. 또, A* 탐색 알고리즘은 탐색과정 동안 다음에 확장할 대상을 현재 상태의 자식 상태들로만 선택범위를 제한하지 않고, 아직 확장되지 않은 채 저장되어 있는 모든 상태 대해 평가치에 따라 선택할 수 있도록 한다. 이런 의미에서 A* 알고리즘은 전역탐색(global search)을 통해 탐색의 완전성(completeness)를 보장한다. 반면에 목표상태를 찾을 때까지는 생성된 모든 상태들을 저장하여야 하므로 메모리가 매우 많이 요구되고 탐색과정이 매우 느리다[3].

```
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 Relaxed Graphplan */
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
```

표1. Enforecd Hill-Climbing

A* 알고리즘과는 대조적으로 Hill-Climbing과 Enforced Hill-Climbing 탐색방법은 목표상태까지의 추정거리인 휴우리스틱만을 상태 평가에 이용하며, 매 순간 자식 상태들이나 후손 상태들 중에서만 다음 확장할 상태를 선택한다. 즉, 이들은 현재 상태를 중심으로 한 지역탐색(local search)을 진행하면서 선택되지 않은 상태들에 대해서는 가지치기를 하게 된다. 따라서 상태 저장을 위한 메모리 요구량이 적고, 평가함수가 우수한 경우 매우 빠른 속도로 목표상태에 접근할 수 있으나 완전성을 보장할 수 없어 해 계획을 찾지 못하는 경우도 있다. 일반적으로 Hill-Climbing에서는 자식 상태들 중에서 최소의 휴우리스틱 값을 갖는 상태를 다음 확장 상태로 선택한다. 하지만 선택된 자식 상태가 반드시 부모 상태보다 더 개선된 휴우리스틱 값을 가진다는 보장은 없다. 이에 반해 Enforced Hill-Climbing에서는 넓이 우선 탐색을 통해 현재 상태보다 낮은 휴우리스틱 값을 갖는 후손 상태를 찾아내고 그 상태까지 계획을 확장하는 과정을 반복한다. 이 탐색 알고리즘의 문제는 귀로(backtrack)를 할 수 없고, 그림 2에서 보듯 현재 상태보다 좋은 휴우리스틱 값을 갖는 최초의 후손 상태를 만나면 그것을 선택하고 그 이후에 등장할 더 좋은 휴우리스틱 값을 갖는 형제 상태가 존재하더라도 탐색하지 않는다는 것이다. 따라서 이러한 이유로 목표에 도달할 수 있는 더 나은 탐색경로를 놓치게 될 가능성이 있다[4].

본 논문에서 제안하는 Strictly Enforced Hill-Climbing (SEHC) 탐색방법은 그림 3에 보듯이 현재 상태 노드에서 가장 좋은 휴우리스틱 값을 갖는 후손 상태 노드를 찾아 이동한다. 이를 위해 SEHC는 현재 상태 보다 더 낮은 휴우리스틱 값을 갖는 첫 번째 후손 상태뿐만 아니라 같은 깊이의 나머지 형제 상태들까지 넓이 우선 탐색을 연장하여 최소의 휴우리스틱 값을 갖는 후손 상태를 찾아낸다.

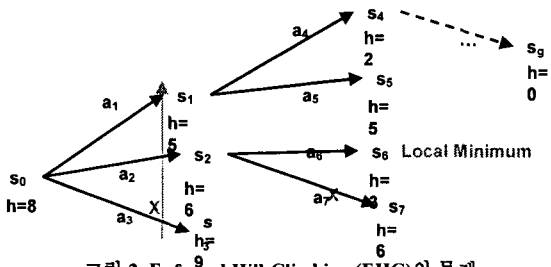

그림 2. Enforced Hill-Climbing (EHC)의 문제

이 탐색 알고리즘 역시 가지치기와 지역탐색의 한계로 인해, 좋지 않은 휴우리스틱 평가함수를 이용하는 경우 최적의 해를 찾지 못할 수도 있다. 그러나, SEHC 탐색 알고리즘은 EHC에 비해 탐색의 전체과정 동안 보다 우수한 탐색경로를 유지할 수 있도록 해준다.

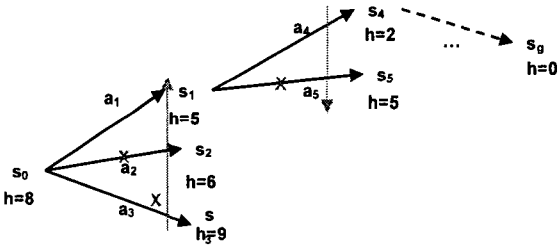

그림3. Strictly Enforced Hill-Climbing (SEHC)

5. 실험

본 논문에서는 A*, EHC, SEHC 등의 3개 알고리즘간의 성능 비교 실험을 하였고, 각 탐색 알고리즘에 의한 확장 상태의 수, 생성 상태의 수, 총 탐색시간, 그리고 결과계획의 최적성 여부를 비교하여 보았다. 또한 Helpful-Action을 이용한 가지치기 기능을 추가하였을 때의 각 알고리즘의 성능도 비교하여 보았다.

	A*		EHC		SEHC	
	Time	Op	Time	Op	Time	Op

Ambush	93	O	62	O	78	O
Blocks	563	O	F	F	F	F
BlocksWorld	125	O	F	F	93	O
Block3	156	O	F	F	172	O
Cargo-Dom	47	O	31	O	31	O
Gripper-Dom	78	O	47	O	47	O
Travel	125	O	F	F	78	F
TSP	93	O	156	F	94	O
Class	16	O	16	O	16	O
RobotDomain	63	O	62	O	47	F

(T= milisecs)

표2. 탐색시간과 계획의 최적성

	A*		EHC		SEHC	
	Exp	Gen	Exp	Gen	Exp	Gen
Ambush	8	84	8	10	8	9
Blocks	972	1541	F	F	F	F
BlocksWorld	12	45	F	F	25	19
Block3	14	78	F	F	27	21
Cargo-Dom	9	30	10	12	10	12
Gripper-Dom	18	56	22	23	23	23
Travel	74	183	F	F	89	53
TSP	35	41	43	85	19	35
Class	3	10	4	4	4	4
RobotDomain	16	42	18	15	21	17

표3. 확장상태의 수와 생성상태의 수

표 2와 표 4에서 Time은 총 탐색시간을, Op는 결과 계획의 최적성을 나타낸다. 또 O표시는 최적으로 목표 상태에 도달하였음을 나타내고, F표시는 실패를 나타낸다. 표 2에서 보듯이, Block 도메인의 경우 EHC와 SEHC는 지역탐색으로 인해 최적의 해 계획을 찾지 못한 경우가 있다. 또, EHC는 Blockworld-3ops, Block3, Travel의 경우 지역최소(Local Minimum)에 빠져서 해를 찾지 못한 경우를 나타내며, SEHC은 Travel, RobotDomain에서 보듯, 계획이 최적이진 않지만, 목표에 도달한다는 것을 알 수 있다. 탐색 시간을 비교해보면, A*탐색 알고리즘에 비해 EHC와 SEHC가 대부분 탐색시간이 적게 소요되었다는 것을 알 수 있다.

표3을 살펴보면, A* 알고리즘은 생성 상태 수가 EHC와 SEHC 알고리즘에 비해 2~3배 가까이 많은 것을 알 수 있고, EHC와 SEHC를 비교해보면 SEHC가 EHC보다 확장상태의 수와 생성상태의 수가 약간 적음을 알 수 있다. 이로써 본 실험에서는 SEHC가 A*와 기존의 EHC에 비해 탐색 효율이 더 우수하다는 것을 확인할 수 있다.

표 4와 표 5는 각각 Helpful-Action(ha)을 이용한 가지치기 기능을 추가한 실험 결과들이다. 표 2와 비교 했을 때 탐색 시간은 매우 단축 되었다는 것을 알 수 있다. 표 3과 표 5에서 보듯, 생성상태의 수 역시 ha를 적용 하였을 때 생성 상태의 수가 더 감소하였다는 것을 알 수 있다. 하지만 EHC와 SEHC를 비교해보면, Domain Ambush, Block3, RobotDomain 경우 ha를 적용하지 않았을 때와 마찬가지로 생성상태의 수가 비슷하다는 것을 알 수 있다. 또 Blocks 도메인의 경우 가지치기를 하여 A*탐색 알고리즘일 경우 목표 상태에 도달하지 못하는 경우가 생기고, 이와 반대로 Block3 도메인의 경우 EHC는 최적의 해 계획을 나타낸다. 마찬가지로 RobotDomain의 경우 ha를 적용하지 않았을 때는 최적의 해를 찾지 못했지만, ha를 적용 함으로써 최적의 해 계획을 나타내는 것을 알 수 있다.

6. 결론

본 논문에서는 휴우리스틱 자원 시간 계획을 위한 새로운 탐색 알고리즘인 Strictly Enforced Hill-Climbing (SEHC)을 제안하였다. SEHC 탐색 알고리즘은 소량의 추가 탐색을 통해 탐색의 전체과정 동안 기존의 Enforced Hill-Climbing(EHC)보다 우수한 탐색경로를 유지할 수 있도록 해준다. 본 논문에서는 다양한 영역의 계획문제를 대상으로 A* 알고리즘, EHC 알고리즘 등과의 비교실험을 통해 SEHC 탐색 알고리즘의 효율성을 확인할 수 있었다.

	A*		EHC		SEHC	
	Time	Op	Time	Op	Time	Op
Ambush	47	O	32	O	32	O
Blocks	F	F	F	F	F	F
BlocksWorld	79	O	F	F	78	F
Block3	110	O	78	O	78	O
Cargo-Dom	32	O	31	O	16	O
Gripper-Dom	47	O	32	O	31	O
Travel	265	O	F	F	F	F
TSP	78	O	141	F	94	F
Class	16	O	16	O	16	O
RobotDomain	47	O	32	O	32	O

(T= milisecs)

표4. Helpful-Action을 적용을 했을 때, 탐색시간과 계획의 최적성

	A*		EHC		SEHC	
	Exp	Gen	Exp	Gen	Exp	Gen
Ambush	7	13	8	10	8	9
Blocks	F	F	F	F	F	F
BlocksWorld	12	30	F	F	25	19
Block3	11	37	12	13	12	13
Cargo-Dom	9	17	10	12	10	12
Gripper-Dom	16	25	22	23	22	22
Travel	74	183	F	F	F	F
TSP	35	41	85	43	35	19
Class	3	3	4	4	4	4
RobotDomain	14	22	17	15	17	15

표5. Helpful-Action을 적용을 했을 때, 확장상태의 수와 생성상태의 수

참고 문헌

[1] F.Bacchus and M.Ady, "Planning with Resources and Concurrency: A Forward Chaining Approach.", Proc of IJCAI-2001, pp 417-424, 2001.

[2] M. B. Do and S. Kambhampati, "Sapa: A Scalable Multi-objective Heuristic Metric Temporal Planner", Journal of AI Research, Vol.20, pp.155-194, 2003.

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

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

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