제약을 갖는 POMDP를 위한 점-기반 가치 반복 알고리즘

김동호 ○ 이재송 김기응
한국과학기술원 전산학과
{dkim, jaesong, kekim}@cs.kaist.ac.kr

Pascal Poupart
School of CS, Univ. of Waterloo
ppoupart@cs.uwaterloo.ca

Point-Based Value Iteration for Constrained POMDPs

Dongho Kim○ Jaesong Lee Kee-Eung Kim
Department of Computer Science
KAIST

Pascal Poupart
School of Computer Science
University of Waterloo

요 약

제약을 갖는 부분 관찰 의사결정 과정(Constrained Partially Observable Markov Decision Process; CPOMDP)는 정책이 제약(constraint)를 만족하면서 가치 함수를 최적화하도록 일반적인 부분 관찰 의사결정 과정(POMDP)을 확장한 모델이다. CPOMDP는 제한된 자원을 가지거나 여러 개의 목적 함수를 가지는 문제를 자연스럽게 모델링할 수 있기 때문에 일반적인 POMDP에 비해 더 실용적인 장점을 가진다. 본 논문에서는 CPOMDP의 확률적 최적 정책 및 근사 최적 정책을 계산할 수 있는 최적 및 근사 동적 프로그래밍 알고리즘을 제안한다. 최적 알고리즘은 동적 프로그래밍의 각 단계마다 미니맥스 이차 제약 계획 문제를 계산해야 하는 반면에 근사 알고리즘은 선형 계획 문제만을 필요로 하는 점-기반(point-based) 가치 업데이트를 이용한다. 실험 결과, 확률적 정책이 결정적(deterministic) 정책보다 더 나은 성능을 보이며, 근사 알고리즘을 통해 계산 시간을 줄일 수 있음을 보였다.

1. 서 론

부분 관찰 마코프 의사결정 과정(Partially Observable Markov Decision Process; POMDP)은 마코프 의사결정 과정(Markov Decision Process; MDP)을 확장하여, 부분적 또는 부정확한 관찰값을 가지는 확률적 의사결정 문제를 모델링하는 데에 널리 쓰이고 있다. 일반적인 POMDP에서는 특정한 상태에서 어떤 행동을 수행했을 때에 즉각적으로 얻을 수 있는 효용(utility)을 보상 함수(reward function)으로 표현하고, 장기적인 보상을 최대화하는 최적 정책을 계산한다. 하지만, 실제 문제에서는 효용 가치가 여러 개의 목표로 표현되는 경우가 많다.

제약을 갖는 POMDP(Constrained POMDP; CPOMDP)는 이러한 문제를 해결하기 위해서, 하나의 기준(보상)은 최대화하면서, 다른 기준(비용)은 특정 값보다 낮아야 하는 제약을 가진다. CPOMDP는 제한된 자원을 가진 에이전트, 예를 들면 한정된 에너지로 최대한 많은 일을 달성하여야 하는 로봇 등에 유용하게 쓰일 수 있다. 또한 POMDP의 여러 응용에 있어서 많은 문제들이 제약조건을 이용하여 자연스럽게 표현된다.

이러한 장점에도 불구하고, 제약을 갖는 MDP(Constrained MDP; CMDP)[1]와는 달리 CPOMDP는 결정적(deterministic) 정책을 찾는 동적 프로그래밍(Dynamic Programming; DP) 알고리즘[2]외에 많은 연구가 이루어지지 않았다. CPOMDP에서는 결정적 정책 만으로는 최적 정책을 계산할 수 없으므로, 본 논문에서는 확률적(randomized) 최적 정책을 찾는 최적 및 근사 알고리즘을 제안한다. 최적 알고리즘은 필요없는 정책을 제거하기 위해서 계산량이 많은 미니맥스(minimax) 이차 제약 계획 문제(Quadratic Constrained Program; QCP)를 필요로 하기 때문에 실제로 사용하기는 힘들다. 반면, 근사 알고리즘은 일반적인 POMDP에서의 점-기반 가치 반복(Point-Based Value Iteration; PBVI) 알고리즘을 확장하여, 확률 상태와 admissible cost[3]를 함께 수집한다. 근사 알고리즘은 minimax QCP 대신에 선형 계획 문제(Linear Program; LP)만을 계산하면 되는 장점을 가진다. 실험 결과, 근사 알고리즘을 이용하여 수천개의 상태를 가지는 CPOMDP 문제를 풀 수 있음을 확인하였다.

2. Constrained POMDP

일반적인 POMDP는 $\langle S, A, Z, T, O, R, \gamma, b_0 \rangle$ 으로 정의된다. 여기서 S는 상태 s 들의 집합, A는 행동 a들의 집합, Z는 관찰값 z들의 집합이다. T는 전이 함수로, $T(s,a,s')$는 상태 s에서 행동 a를 수행했을 때 상태 s'으로 이동할 확률 $\Pr(s'|s,a)$을 의미한다. O는 관찰 함수로 $O(s,a,z)$는 행동 a를 수행하여 상태 s로 이동하였을 때 관찰값 z를 관측할 확률 $\Pr(z|s,a)$를 의미한다. R은 보상 함수로, $R(s,a)$는 상태 s에서 행동 a를 수행하였을때 에이전트가 받게 되는 즉각적인 보상값을 의미한다. $\gamma \in [0,1)$ 은 discount factor이고, b_0 는 초기 확률 상태로서 $b_0(s)$는 에이전트가 초기에 상태 s에 있을 확률값을 의미한다.

† 본 논문은 2011년 International Joint Conference on Artificial Intelligence (IJCAI-2011)에 게재될 예정인 "Point-Based Value Iteration for Constrained POMDPs" 논문을 국내 연구자들의 편의를 위해 한글로 요약한 논문임.

COMDP는 $\langle S, A, Z, T, O, R, \{C_k\}_{k=1}^{K}, \{\hat{c}\}_{k=1}^{K}, \gamma, b_0\rangle$으로 정의된다. C_k는 k번째 비용을 뜻하는 함수로, $C_k(s,a) \geq 0$는 상태 s에서 행동 a를 수행하는데에 소모되는 비용을 의미한다. $\hat{c}_k$는 k번째 비용에 대한 누적 비용의 상한선을 의미한다. 즉, 에이전트는 $\hat{c}_k$ 이상의 비용을 소모할 수 없다. 따라서 CPOMDP의 최적 정책 π를 구하는 것은 다음의 최적화 문제를 푸는 것을 의미한다:

$$\begin{aligned} &\text{maximize } E_\pi[\textstyle\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)] \\ &\text{subject to } E_\pi[\textstyle\sum_{t=0}^{\infty}\gamma^t C_k(s_t,a_t)] \leq \hat{c}_k \quad \forall k. \end{aligned} \tag{1}$$

POMDP나 CPOMDP에서는 에이전트가 현재 상태를 정확히 할 수 없으므로, 현재 상태에 대한 확률 분포인 확률 상태 b를 사용하게 된다. $b' = \tau(b,a,z)$는 확률 상태 b에서 행동 a를 수행하고 관찰값 z를 관측하였을 때 업데이트 된 확률 상태를 의미한다. 이는 베이즈 정리(Bayes theorem)을 이용하여 다음과 같이 계산할 수 있다:

$$b'(s') = O(s',a,z)\textstyle\sum_s T(s,a,s')b(s)/P(z|b,a) \tag{2}$$

한편, CMDP에서는 최적 정책이 확률적일 수 있음이 알려져 있다[1]. 또한, 모든 CMDP는 정확한 관찰값을 제공하는 CPOMDP로 생각할 수 있으므로, CPOMDP에서도 최적 정책의 계산을 위해서는 확률적 정책을 고려해야 한다.

3. CPOMDP를 위한 최적 정책 알고리즘

CPOMDP의 *결정적* 정책을 찾기 위한 DP 알고리즘이 제안된 바 있다[2]. 편의를 위해 하나의 제약식만 존재한다고 가정하면($K=1$), CPOMDP의 가치 함수는 누적 보상 함수와 하나의 누적 비용 함수로 표현될 수 있다. 즉, 가치 함수는 누적 보상 백터와 누적 비용 백터의 쌍들의 집합으로 표현한다. 현재의 가치 함수를 V'라고 했을 때, DP 업데이트는 다음과 같이 수행된다:

$$\begin{aligned} \alpha_{i,r}^{a,z}(s) &= R(s,a)/|Z| + \gamma\textstyle\sum_{s'\in S} T(s,a,s')O(s',a,z)\alpha'_{i,r}(s') \\ \alpha_{i,c}^{a,z}(s) &= C(s,a)/|Z| + \gamma\textstyle\sum_{s'\in S} T(s,a,s')O(s',a,z)\alpha'_{i,c}(s') \\ V &= \textstyle\bigcup_{a\in A}\bigoplus_{z\in Z}\{\langle\alpha_{i,r}^{a,z}, \alpha_{i,c}^{a,z}\rangle | \forall i\}, \end{aligned}$$

여기서 한번의 DP 업데이트는 최악의 경우 $|A||V'|^{|Z|}$ 개의 벡터 쌍을 생성하게 된다. 즉 벡터 쌍의 수가 기하급수적으로 늘어나기 때문에 각 행동과 관찰값에 대해서 벡터를 생성할 때 필요없는 벡터들을 제거하는 pruning 과정이 필요하다[4]. Pruning 과정에서는 새로 생성된 벡터 쌍인 $\langle\alpha_r, \alpha_c\rangle$이 가치 함수를 표현하는 벡터 쌍의 집합에 포함할지를 결정하게 된다. 이 때, 어떠한 확률 상태 b에서 현재 벡터 쌍이 비용에 대한 제약을 만족하면서, V에서 제약을 만족시키는 모든 벡터 쌍들보다 더 높은 누적 보상값을 가진다면, 이러한 벡터 쌍은 가치 함수 집합에 포함되어야 한다. 이 문제는 혼합 정수 선형 계획 문제(Mixed Integer Linear Program; MILP)로 표현될 수 있다:

$$\max_{h,b,d^i} h \quad \left|\begin{array}{l} \alpha_c \cdot b \leq \hat{c}, \\ (\alpha_r - \alpha_{i,r})\cdot b \geq h - d_i M \quad \forall i, \\ \alpha_{i,c}\cdot b \geq d_i \hat{c} \quad \forall i, \\ d_i \in \{0,1\} \quad \forall i, \\ \textstyle\sum_{s\in S} b(s) = 1, \\ b(s) \geq 0 \quad \forall s \in S, \\ h \geq 0, \end{array}\right. \tag{3}$$

여기서 M은 충분히 큰 양의 상수이다. 이 문제의 해가 존재하지 않는다면 $\langle\alpha_r, \alpha_c\rangle$은 제거된다.

하지만 이 방법은 몇가지 문제점을 가지고 있다. 첫째, 앞서 설명한 것처럼 결정적 정책은 CPOMDP에서 준최적일 수 있으므로 벡터의 제거 여부를 판단하는 데에 있어서 확률적 정책, 즉 여러 벡터들의 convex 조합으로 표현되는 정책을 고려하여야 한다. 둘째, 이 방법은 매 DP 업데이트 마다 누적 비용 제약을 만족하지 않는 벡터들은 제거한다. 즉 모든 중간 t-시간에서의 정책들이 누적 비용 제약을 만족한다. 하지만 (1)의 누적 비용 제약을 위해 매 단계에서 누적 비용 제약을 만족할 필요는 없다.

따라서 본 논문에서는 (3)의 MILP 대신에 다음의 minimax QCP를 이용하는 것을 제안한다:

$$\min_{b}\max_{w_i,h} h \quad \left|\begin{array}{l} \alpha_c \cdot b \geq b\cdot\textstyle\sum_i w_i\alpha_{i,c} + h, \\ \left(\textstyle\sum_i w_i\alpha_{i,r} - \alpha_r\right)\cdot b \geq h, \\ \textstyle\sum_{s\in S} b(s) = 1, \\ b(s) \geq 0 \quad \forall s \in S, \\ \textstyle\sum_i w_i = 1, \\ w_i \geq 0 \quad \forall i. \end{array}\right. \tag{4}$$

첫번째와 두번째 제약식은 $h \geq 0$이면, b에서 α_c보다 적은 누적 비용을 가지면서도 α_r보다 큰 누적 보상값을 가지는 벡터들의 convex 조합이 존재함을 의미한다. 위 문제는 모든 확률상태 공간에 대해서 최대 h를 최소화하기 때문에, 최종해 h가 0 이상일 때 $\langle\alpha_r, \alpha_c\rangle$는 제거된다. 하지만 위 minimax QCP는 많은 계산량을 필요로 한다.

4. CPOMDP를 위한 근사 정책 알고리즘

4.1 점-기반 가치 반복 (PBVI)

일반적인 POMDP에서의 PBVI 알고리즘은 확률 상태 공간 전체 대신에 미리 수집된 유한 개의 확률 상태들, 즉 $B = \{b_0, b_1, \ldots, b_q\}$에서만 가치 함수를 업데이트한다[5]. 이러한 점-기반 DP 업데이트를 CPOMDP에 적용하기 위해서 마찬가지로 알고리즘 1을 이용해 각 확률상태 $b \in B$에 대해 α-벡터의 regression을 구한 다음, $b \in B$에 대해서 필요없는 벡터들을 제거할 수 있다. 이러한 점-기반 DP 업데이트는 알고리즘 2에 나타나 있다. Pruning을 B에 대해서만 수행하게 되므로, (4)의 minimax QCP는 다음의 LP로 줄어든다:

$$\max_{w_i,h} h \quad \left|\begin{array}{l} \alpha_c \cdot b \geq b\cdot\textstyle\sum_i w_i\alpha_{i,c} + h, \\ \left(\textstyle\sum_i w_i\alpha_{i,r} - \alpha_r\right)\cdot b \geq h, \\ \textstyle\sum_i w_i = 1, \\ w_i \geq 0 \quad \forall i. \end{array}\right. \tag{5}$$

이는 (4)의 minimax QCP에서 확률상태 b가 더 이상 변수가 아닌 것을 제외하면 동일하다. 이 LP를 이용한 pruning 방법은 알고리즘 3에 나타나 있다.

하지만 각각의 확률상태에 단 하나의 α-벡터만 유지하게 되는 일반적인 PBVI와 비교하였을 때, 위의 단순한 점-기반 알고리즘에서는 여전히 α-벡터의 수가 크게 증가하게 된다. 이는 알고리즘에서 도달 가능한 확률상태의 집합만 이용하고, 그 확률상태에 도달하기까지 얼마나 많은 비용이 소모되었는지에 대한 정보를 전혀 사용하고 있지 않기 때문이다.

알고리즘 1: regress V'

input : V'; output : $\{\langle\alpha_r^{a,*},\alpha_c^{a,*}\rangle\}$, $\{\Gamma^{a,z}\}$
foreach $a \in A$ and $z \in Z$ do
  $\langle\alpha_r^{a,*},\alpha_c^{a,*}\rangle \leftarrow \langle R(\cdot,a), C(\cdot,a)\rangle$, $\Gamma^a \leftarrow \emptyset$
  foreach $\langle\alpha'_{i,r},\alpha'_{i,c}\rangle \in V'$ do
    $\alpha_{i,r}^{a,z}(s) = \sum_{s'\in S} T(s,a,s')O(s',a,z)\alpha'_{i,r}(s')$
    $\alpha_{i,c}^{a,z}(s) = \sum_{s'\in S} T(s,a,s')O(s',a,z)\alpha'_{i,c}(s')$
    $\Gamma^{a,z} \leftarrow \Gamma^{a,z} \cup \langle\alpha_{i,r}^{a,z},\alpha_{i,c}^{a,z}\rangle$
  end
end

알고리즘 2: update $V = \tilde{H}V'$

input : B, V'; output : V
$V \leftarrow \emptyset$
$\{\langle\alpha_r^{a,*},\alpha_c^{a,*}\rangle\}$, $\{\Gamma^{a,z}\} \leftarrow$ regress(V')
foreach $a \in A$ do $\Gamma^a \leftarrow \langle\alpha_r^{a,*},\alpha_c^{a,*}\rangle \oplus \oplus_{z\in Z}\gamma\Gamma^{a,z}$ end
$\Gamma \leftarrow \cup_{a\in A}\Gamma^a$
foreach $b \in B$ do $V \leftarrow V \cup$ prune(b,Γ) end

알고리즘 3: prune

input : b, Γ; output : $\tilde{\Gamma}$
$\tilde{\Gamma} \leftarrow \emptyset$
foreach $\langle\alpha_r,\alpha_c\rangle \in \Gamma$ do
  Solve the LP (5) and get the solution h
  if $h < 0$ then $\tilde{\Gamma} \leftarrow \tilde{\Gamma} \cup \{\langle\alpha_r,\alpha_c\rangle\}$ end
end

4.2 Admissible cost를 이용한 점-기반 가치 반복 (PBVI)

본 논문에서 제안하는 근사 알고리즘의 주요한 아이디어는 확률상태를 수집할 때 누적 비용에 대한 정보도 함께 수집하는 것이다. 먼저 시간 t 에서의 *admissible cost*, 즉 남은 시간 $\{t, t+1, \ldots\}$동안 더 소모할 수 있는 기대 누적 비용을 의미하는 변수 d_t를 도입한다[3]. 또한 W_t를 시간 t까지 소모된 누적 비용, 즉 $W_t = \sum_{\tau=0}^{t}\gamma^\tau C(b_\tau,a_\tau)$ 라고 정의하면, 시간 $t+1$ 에서의 admissible cost는 $d_{t+1} = \frac{1}{\gamma^{t+1}}(\hat{c} - W_t)$ 으로 정의될 수 있다. 즉, d_{t+1} 은 $\hat{c}$ (허용된 기대 누적 비용의 최대값)과 W_t (지금까지 소모된 누적 비용)의 차이에 $1/\gamma^{t+1}$을 곱한 값이다. Admissible cost는 다음과 같이 재귀적으로 계산될 수 있다:

$$\begin{aligned} d_{t+1} &= \frac{1}{\gamma^{t+1}}(\hat{c} - W_t) = \frac{1}{\gamma^{t+1}}\left(\hat{c} - W_{t-1} - \gamma^t C(b_t,a_t)\right) \\ &= \frac{1}{\gamma}\left(d_t - C(b_t,a_t)\right). \end{aligned} \quad (6)$$

여기서 초기 admissible cost는 $d_0 = \hat{c}$로 정의된다.

한편 현재의 확률상태-비용의 쌍 (b,d) 이 주어지면, 최적의 행동은 다음의 LP을 이용하여 계산한다:

$$\max_{w_i} b\cdot\sum_i w_i\alpha_{i,r} \;\Big|\; \begin{array}{l} b\cdot\sum_i w_i\alpha_{i,c} \le d, \\ \sum_i w_i = 1, \\ w_i \ge 0 \quad \forall i, \end{array} \quad (7)$$

알고리즘 4: update $V = \tilde{H}V'$ (PBVI with admissible cost)

input : B, V'; output : V
$V \leftarrow \emptyset$; $\{\langle\alpha_r^{a,*},\alpha_c^{a,*}\rangle\}$, $\{\Gamma^{a,z}\} \leftarrow$ regress(V')
foreach $(b,d) \in B$ do
  foreach $a \in A$ do
    foreach $z \in Z$ do
      $d_z \leftarrow \frac{1}{\gamma}\left(d - C(b,a)\right)\Pr(z|b,a)$
      Solve the LP (7) with $\forall\langle\alpha_{i,r},\alpha_{i,c}\rangle \in \Gamma^{a,z}$ and (b,d_z), and get the solution $\tilde{w}_i$
      $\tilde{\alpha}_r^{a,z} \leftarrow \sum_i \tilde{w}_i\alpha_{i,r}$; $\tilde{\alpha}_c^{a,z} \leftarrow \sum_i \tilde{w}_i\alpha_{i,c}$
    end
    $\alpha_r^{(b,d),a} = \alpha_r^{a,*} + \gamma\sum_{z\in Z}\tilde{\alpha}_r^{a,z}$; $\alpha_c^{(b,d),a} = \alpha_c^{a,*} + \gamma\sum_{z\in Z}\tilde{\alpha}_c^{a,z}$
  end
  $\Gamma^{(b,d)} \leftarrow \cup_{a\in A}\left\{\langle\alpha_r^{(b,d),a},\alpha_c^{(b,d),a}\rangle\right\}$
  Solve the LP (7) with $\Gamma^{(b,d)}$ and (b,d), and get w_i.
  $V \leftarrow V \cup \left\{\langle\alpha_{i,r},\alpha_{i,c}\rangle \in \Gamma^{(b,d)} | w_i > 0\right\}$
end

알고리즘 5: Execution

input : $b = b_0$, $d = \hat{c}$
while *true* do
  Solve the LP (7) with (b,d)
  Randomly choose the index i with probability w_i
  Perform a_i corresponding to $\langle\alpha_{i,r},\alpha_{i,c}\rangle$; Receive z
  $d \leftarrow \alpha_{i,c}\cdot b$; $d \leftarrow \frac{1}{\gamma}\left(d - C(b,a_i)\right)$; $b \leftarrow \tau(b,a,z)$
end

여기서 w_i 는 $\langle\alpha_{i,r},\alpha_{i,c}\rangle$ 에 해당하는 행동을 수행할 확률을 의미한다. 위의 LP는 최대 $|V|+2$개의 제약식을 가지고 최소한 $|V|$개의 제약식은 극점(extreme point)에서 active하게 되므로 위 LP의 해 w_i 중에서 0보다 큰 값을 가지는 경우는 최대 2개가 된다. K 개의 제약을 가지는 CPOMDP의 경우에는 최대 $K+1$개가 0보다 큰 값을 가지게 된다.

Admissible cost를 이용하도록 수정된 점-기반 DP 업데이트는 알고리즘 4에 나타나 있다. 수집된 각각의 확률상태-비용의 쌍 $(b,d) \in B$에 대해서, 다음의 벡터를 계산한다:

$$\alpha_r^{(b,d),a} = \alpha_r^{a,*} + \gamma\sum_{z\in Z}\tilde{\alpha}_r^{a,z}$$

여기서 $\tilde{\alpha}_r^{a,z}$는 행동 a를 수행하고 관찰값 z를 얻은 다음 상황에 대한 최적 확률적 백터를 의미한다. $\alpha_c^{(b,d),a}$ 역시 유사하게 계산할 수 있다. 마지막으로, (b,d) 현재 확률상태와 admissible cost에 대해서 (7)의 LP를 계산하여, 가치 벡터의 최적 convex 합을 계산한다.

여기서 각각의 $\tilde{\alpha}_r^{a,z}$ 는 admissible cost를 가능한 관찰값들에 대해서 $\frac{1}{\gamma}\left(d - C(b,a)\right)\Pr(z|b,a)$ 로 분배하여 얻은 결과이다.

b에서 가장 좋은 convex 합을 얻기 위해서는 이러한 제약이 없어야 하지만, 그러한 경우에는 α-벡터의 수가 아주 커지게 된다. 실험에서는 위의 방식을 이용해여 충분히 좋은 α-벡터를 계산할 수 있음을 확인하였다. 결론적으로, 위의 알고리즘은 각각의 확률상태에 대해 최대 $K+1$개의 벡터 쌍을 유지하게 된다. 따라서 총 벡터 쌍의 수는 $(K+1)|B|$가 된다.

4.3 정책 수행 방법

Admissible cost를 이용한 PBVI를 이용하여 얻은 근사 정책을 에이전트가 수행할 때에는, 현재 확률상태와 admissible cost에 따라서 최적의 행동을 결정하게 된다. 이 방법은 알고리즘 5에 나타나 있다. 현재 확률상태와 admissible cost (b_t, d_t)가 주어지면, (7)의 LP를 계산하여 최적의 확률적 정책을 구한다. 그리고, w_i의 확률에 따라서 하나의 벡터 쌍 $\langle \alpha_{i,r}, \alpha_{i,c} \rangle$을 선택한다. 이 때, 현재 admissible cost d_t는 $d'_t = \alpha_{i,c} \cdot b$로 리셋된다. 이는 에이전트가 남은 시간동안 $\alpha_{i,c} \cdot b$만큼의 비용을 발생시키는 $\langle \alpha_{i,r}, \alpha_{i,c} \rangle$에 해당하는 정책을 따르기로 결정하였기 때문이며, 이는 w_i의 확률에 따라서 결정되기 때문에 에이전트는 평균적으로 d_t의 비용을 남은 시간동안 소모하게 된다. $\langle \alpha_{i,r}, \alpha_{i,c} \rangle$에 해당하는 행동을 수행한 다음에는 (6)과 (2)에 따라 admissible cost와 확률상태를 업데이트한다.

5. 실험 결과

CPOMDP를 위한 최적 정책 알고리즘과 근사 알고리즘의 성능을 비교하기 위해서 Quickest Change Detection(QCD) 문제[2]를 이용하였다. 그림 1의 결과에서 볼 수 있듯, MILP나 minimax QCP를 이용한 알고리즘은 계산량의 증가로 인해 DP 업데이트를 각각 7회, 6회 이상 수행할 수 없었다. 근사 알고리즘은 500개의 확률상태-비용 샘플을 이용하였으며, 10회 이상 어려움 없이 DP 업데이트를 수행할 수 있었다. 또한 확률적 근사 정책의 성능은 결정적 정책의 성능보다 높았으며, 최적 정책에 아주 가까움을 알 수 있었다.

좀 더 큰 문제에 대한 근사 알고리즘의 성능 평가를 위해서는 n-city ticketing 문제[6]을 이용하였다. 이는 비행기 예약을 위해 사용자가 원하는 출발지와 목적지를 n개의 도시 중에서 알아내는 음성 대화 시스템을 위한 문제이며, 음성 인식의 오류로 인해 사용자의 의도를 정확히 파악하는 데 어려움이 따른다. 실험에서 에이전트는 매 시간마다 -1의 보상값을 받으며, 잘못된 예약을 수행할 때마다 1씩의 비용을 소모하게 된다. 따라서 보상값은 시스템의 효율(대화 길이), 비용은 시스템의 정확도(대화의 성공률)을 의미한다. 실험에는 0.2의 인식 에러율을 가진 3-City Ticketing 문제를 이용해 $(|S| = 1945, |A| = 16, |Z| = 18, \gamma = 0.95)$ 1000번 시뮬레이션에 따른 평균 누적 보상/비용값과 95% 신뢰구간을 계산하였다. 그림 2에서, 더 작은 $\hat{c}$에 대해서는 정보를 더 정확히 알아내기 위해 더 많은 시간을 소모함을 알 수 있다.

6. 결론

본 논문은 CPOMDP에서 확률적 정책을 계산하는 최적 및 근사 알고리즘을 제안하였다. 실험 결과, 확률적 정책이 결정적

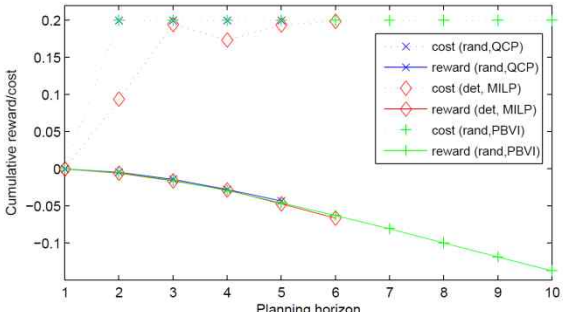

그림 1 QCD 문제의 결과 $(\gamma = 0.95,\ \hat{c} = 0.2)$

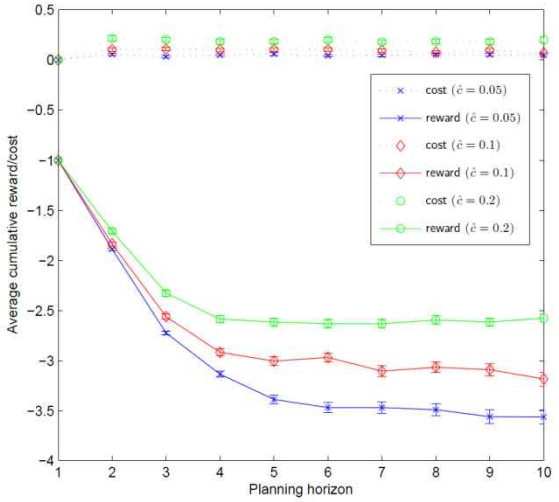

그림 2 3-City Ticketing 문제($P_e = 0.2$)의 결과.

정책 보다 더 높은 성능을 보였으며, 점-기반 알고리즘은 근사 정책을 효율적으로 계산하였다. 실험에서는 하나의 제약을 가진 CPOMDP를 이용하였지만, 본 알고리즘은 여러 개의 제약 및 서로 다른 discount factor가 사용되는 경우에도 확장 가능하다. 향후 연구로는 확률 상태 수집을 위한 휴리스틱의 적용 및 factored CPOMDP를 위한 알고리즘 개발 등이 있다.

참고문헌

[1] E. Altman, *Constrained Markov Decision Processes*, Chapman & Hall/CRC, 1999.

[2] J. D. Isom, S. P. Meyn, and R. D. Braatz, Piecewise linear dynamic programming for constrained POMDPs, In *Proc. of AAAI*, 2008.

[3] A. B. Piunovskiy and X. Mao, Constrained Markovian decision processes: the dynamic programming approach, Operations Research Letters, 27(3):119-126, 2000.

[4] A. Cassandra, M. Littman, and N. Zhang, Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes, In *Proc. of UAI*, 1997.

[5] J. Pineau, G. Gordon, and S. Thrun, Anytime point-based approximations for large POMDPs, Journal of Artificial Intelligence Research, 27:335-380, 2006.

[6] J. D. Williams and S. Young, Partially observable Markov decision processes for spoken dialog systems, *Computer Speech and Language*, 21(2):393-422, 2007.