Dynamic CBDT : Q-learning의 강화기법을 응용한 CBDT 확장 기법

진영균°, 장형수
서강대학교 컴퓨터학과
{jiny1u°, hschang}@sogang.ac.kr

Dynamic CBDT : Extension of CBDT via Reinforcement Method of Q-learning

Y. K. Jin°, H. S. Chang
Department of Computer Science, Sogang University

요 약

본 논문에서는 불확실한 환경 상에서의 의사결정 알고리즘인 "Case-based Decision Theory" (CBDT) 알고리즘을 dynamic하게 연동되는 연속된 의사결정 문제에 대하여 강화학습의 대표적인 Q-learning의 강화기법을 응용하여 확장한 새로운 의사결정 알고리즘 "Dynamic CBDT"를 제안하고, CBDT알고리즘에 대한 Dynamic CBDT의 효율성을 테트리스 실험을 통하여 확인한다.

1. 서 론

본 논문은 불확실성 하에서의 의사 결정 이론중 하나인 "Case-based Decision Theory" (CBDT)[1][2]를 dynamic하게 연동되는 연속적인(sequential) 의사결정문제에 대하여 강화학습(reinforcement learning)의 대표적인 Q-learning[3]의 강화기법(reinforcement method)을 응용하여 확장하는 기법을 제안한다.

불확실한 환경 하에서의 의사결정문제에 직면하였을 때의 일반적인 의사결정은 "Expect Utility Theory" (EUT)[4]의 패러다임에 의존하였다. EUT는 불명확한 환경 하에서의 의사결정에 대한 가장 대표적인 이론으로서 불확실성을 가지는 많은 문제들이 EUT에 의하여 해결되었다. 하지만 EUT가 가지는 몇몇 제약으로 인하여 모든 불확실성을 가지는 문제들을 EUT로 표현하는 데는 한계가 있고, 정형화하는 것도 간단하지 않다[1]. 최근에는 "Qualitative Decision Theory"[5], 불안정한 정보에 대한 제약을 고려하는 "Mixed Constraint Satisfaction"[6], formal logic 등을 이용하여 의사결정을 하는 방법[7]들이 연구되고 있다.

최근에 Gilboa와 Schmeidler가 제안한 CBDT알고리즘[1][2]은 거의 모든 문제에 적용이 가능한 "Case-Based Reasoning" (CBR)[8]에 기반을 둔 알고리즘으로 CBR과 같이 거의 모든 문제에 적용이 가능하다. CBR은 "유사한 문제는 유사한 해결법을 가진다."라는 것을 전제로 가지고 있고 CBDT 또한 이러한 전제를 의사결정에 적용하여 문제를 해결한다. 하지만 단지 현재 처한 문제에 대한 해결만을 목표로 하기 때문에 연동되어 주어지는 연속된 의사결정 문제에 대한 대처는 전혀 고려하지 않는다. 지금까지 CBDT를 개선한 알고리즘은 나오지 않았고 CBDT를 응용한 타 알고리즘은 "Experience-Based Decision Making" (EBDM)[9]이 유일하다. 하지만 EBDM은 사전지식을 가지고 classification 하여 그에 따라 의사결정을 수행하기 때문에 사전 지식 없이도 학습이 가능한 CBDT와의 비교는 어렵다. 특히 dynamic한 환경에서의 CBDT를 확장하는 방법은 고려되지 않았다.

본 논문에서는 dynamic하게 연동되는 연속된 의사결정 문제에 대한 대처를 고려하지 않는 CBDT의 약점을 보완하기 위하여, 강화학습의 대표적인 Q-learning의 최적(optimal) Q값을 근사하는 강화기법을 응용하여, dynamic하게 연동되는 연속된 의사결정문제에 대하여 CBDT를 보완한 새로운 학습기법, "Dynamic CBDT"을 제안하고 이것을 테트리스에 적용하여 기존의 CBDT알고리즘보다 더 좋은 성능을 보임을 확인한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 CBDT에 대하여 간략히 소개하며, 3장에서는 기존의 CBDT알고리즘확장기법을 설명하고, 4장에서는 성능 차이의 결과를 테트리스 실험을 통하여 보인다.

2. CBDT

CBDT는 CBR과 유사하지만 하나의 사례만을 이용하여 문제를 해결하는 것이 아니라 경험한 모든 사례를 "종합하여" 현재 문제와 유사한 과거 사례들에 대하여 가장 좋은 결과를 이끌어 주었던 해를 찾는다.

CBDT는 CBR과 마찬가지로 경험을 사례화하여 저장한다. 이러한 사례들의 집합은 $C = P \times A \times R$ 로 주어진다. 여기서 P는 problem의 집합, A는 action의 집합, 그리고 R은 result의 집합을 의미한다. 특히 $r \in R$은 $p \in P$에 $a \in A$를 적용하였을 때의 결과를 의미하고 $r = r(p,a)$로 쓴다. 유사도 함수(similarity function) $\sigma : P \times P \rightarrow [0,1]$은 두 문제에 대한 유사도를 배정하고, 효용함수(utility function) $u : R \rightarrow \Re^+$는 과에 대한 효용 값을 의미한다. 의사 결정자는 사례를 저장하기 위한 제한된 크기의 메모리

$$M = \{(p_0, a_0, r_0), (p_1, a_1, r_1), \ldots\},$$
$$p_i \in P, a_i \in A, r_i \in R$$

을 가지고 있다. 주어진 문제 $p \in P$에 대하여, CBDT는 아래식 (1)을 통하여 M에 저장된, p와 유사한, 과거 사례들에 대

한 가장 좋은 결과를 이끌어주었던 action d를 선택한다.

$$d \in \arg\max_{a \in A} V(a)$$

$$V_p(a) = \sum_{(p',a,r) \in M} [\sigma(p,p') \cdot u(r)], a \in A \quad (1)$$

하지만 식(1)을 사용하면, action d에 대한 사례의 개수가 최적 action a^*에 대한 사례의 개수보다 많을 경우에 문제 p에 대한 a^*보다 좋지 않은 d가 선택될 수 있다. 즉, 식 (1)은 가장 좋은 결과를 이끌어낸 a^*를 선택함을 보장하지 못한다. 그래서 사례의 개수에 영향을 받지 않도록 유사도 함수를 normalized하여 식 (1)을 다음과 같이 재정의 한다:

$$V_p(a) = \sum_{(p',a,r') \in M} \left[\frac{\sigma(p,p') \cdot u(r')}{\sum_{(p'',a,r'') \in M} \sigma(p,p'')} \right], a \in A \quad (2)$$

재정의된 CBDT역시 사례기반 추론(CBR)과 마찬가지로 기존의 유사한 학습 데이터(사례)가 없을 경우 주어진 의사결정 문제 해결할 수 없는 약점이 있다. Gilboa와 Schmeidler[1]는 이러한 약점을 극복하고자 Simon의 idea인 "Satisficing"[1] 개념[10]을 적용하였다. 즉 임의의 정해진 *threshold*보다 큰 V값을 가지는 action을 찾을 때까지 *exploration*을 하는 것이다. CBDT는 이러한 exploration strategy를 통하여 관련 학습 데이터가 없는 문제에 대해서도 시행착오를 통하여 일정 수준 이상의 결과를 이끌어주는 action을 찾아낸다. 이때의 threshold값의 범위는 0과 효용 함수 u의 최대 값 사이의 값을 가진다. 여기서 threshold값이 최대 값에 가까울수록 exploration이 빈번하게 일어나고 0에 가까울수록 exploitation이 빈번하게 일어난다. 하지만 threshold에 근거한 exploration-exploitation은 단지 현재 문제에 대한 최선의 선택만을 위한 것이고, 각 문제의 최선의 선택이 전체 의사 결정 과정의 순차적으로 연동된 문제에 대한 최선의 선택이라고는 할 수 없다. 예를 들어 Figure 1과 같이 목적지와 출발점 사이에 장애물이 있다면, Figure 1.(a)와 같이 진행하면 만나게 될 장애물을 미리 예측하고 우회하는 것과, (b)와 같이 단지 목적지와의 거리만을 고려하여 움직이다 장애물과 조우한 후 대처하는 것의 차이라고 할 수 있다.

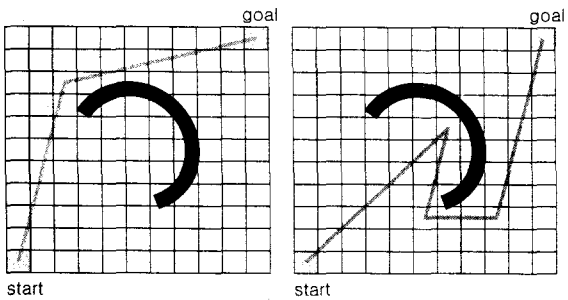

Figure 1. 장애물 예제

현재 문제만을 고려한 CBDT에 이렇듯 다음 문제까지 고려할 수 있다면 연속된 의사결정이 필요한 환경에서 더 좋은 성능을 낼 수 있을 것이다.

다음 장에서는 현재 CBDT 알고리즘의 동작을 간단하게 설명하고 알고리즘을 확장하는 방법에 대하여 서술한다.

3. Dynamic CBDT 알고리즘

3.1 CBDT 알고리즘의 분석

CBDT알고리즘은, 연속되는 의사 결정 과정에서도 다음 문제는 고려하지 않은 채, 현재 문제에 대한 최적의 action만을 결정한다. 하지만 의사 결정자가 어떠한 problem에서 어떠한 action을 수행하였을 때의 효용 값을 가지고 의사를 결정 한다는 점에서는 강화 학습의 보상 값에 의존하여 의사를 결정하는 것과 유사하다. 이러한 CBDT 알고리즘은 Figure 2와 같다.

INIT:
  SELECT $threshold \in [0, \max_{r \in R} u(r)]$
END INIT

GET p_{cur}
CALCULATE $\forall a' \in A, V_{p_{cur}}(a')$ by Eqn. (2)
IF $\max_{a' \in A} V_{p_{cur}}(a') < threshold$ THEN
  SELECT $a \in A$ by the uniform distribution over A
ELSE
  SELECT $a \in \arg\max_{a' \in A} V_{p_{cur}}(a')$
ENDIF
EXECUTE a
OBTAIN $r = r(p_{cur}, a)$
STORE (p_{cur}, a, r) in M

Figure 2. CBDT 알고리즘

불확실한 환경에서의 연속되는 의사결정 과정에 CBDT가 적용될 경우 위의 동작은 반복되어 수행된다. 하지만 CBDT는 다음 문제를 고려하지 않고 현재 문제만을 고려한 의사 결정만을 하므로 연속되는 의사결정 과정에 적용하는데 어려움이 있다. result r이 problem을 표현 할 수 있는 속성을 포함하고 있다고 가정하면, 주어진 현재 문제를 p_{cur}라고 할 때, 위의 CBDT 동작에서 result $r(p_{cur}, a)$가 주어졌을 때 다음 문제 p_{nxt}역시 알 수 있을 것이다. 이 사실을 이용하여, 기존의 효용 함수 u를 다음 문제 p_{nxt}의 효용 값을 포함하도록 재정의 하는 것이 가능하다.

3.2 Dynamic CBDT

효용 함수를 어떻게 재정의 하는가에 대하여 설명하기에 앞서 강화 학습을 대표하는 Q-learning의 강화 기법에 대하여 간단히 살펴보면 다음과 같다.

Q-learning은 RL의 대표적인 알고리즘중 하나로 주어진 상황(state x_{cur})에서 행위(action k)를 취하였을 때 받은 보상 값(reward ρ)을 바탕으로 최적 Q값을 근사하기 위하여 함수

1) satisfice = satisfy + sacrifice

$\widetilde{Q}$를 다음과 같이 update한다[11].

$$\widetilde{Q}(x_{cur},k)\leftarrow(1-\alpha)\widetilde{Q}(x_{cur},k) + \alpha\left[\rho+\gamma\max_{k'\in A}\widetilde{Q}(x_{nxt},k')\right] \quad (3)$$

식(3)에서 x_{cur}는 현재 state를 의미하고 k는 x_{cur}에서 취해진 결정을 의미한다. x_{nxt}은 x_{cur}에서 k를 취한 이후, 의사결정자가 인지하는 state이다. ρ은 x_{cur}에 대한 k의 보상 값이다. 범위 (0,1)에서 값을 갖는 α와 γ는 각각 learning rate와 discount factor를 의미한다.

여기서, state는 CBDT의 problem과 대응되고, action은 CBDT의 action과 같은 의미를 가진다. 보상 값은 CBDT의 효용 함수를 통하여 알 수 있는 효용 값에 대응된다.

$\widetilde{Q}$값을 통한 action의 선택은 현재의 결과 뿐 만 아니라, 이후의 결과까지 반영한 것이다. 이러한 Q-learning의 식 (3)을 응용하면 CBDT역시 연속된 의사 결정 과정에서 다음 problem을 고려한 action을 선택하게 하는 것이 가능하다. Q-learning에서는 exploitation시 $\widetilde{Q}$값을 최대로 가지는 action을 취하는 반면, CBDT에서는 V값을 최대로 가지는 action을 취한다. 여기서 V값은 과거 경험의 효용 값의 weighted sum이기 때문에 과거 경험의 효용 값에 현재 이후의 문제들에 대한 예측 효용 값을 유지할 수 있다면, 어떤 문제에서의 최대 V값을 가지는 action은 현재 문제뿐만 아니라 다음 문제까지 고려하게 된다.

이러한 idea를 구현하기 위하여 result r이 problem을 표현할 수 있는 속성을 포함하고 있다고 가정하고, result r이 주어져 있을 때, 다음 문제 p_{nxt}를 r로부터 구하는 함수 $N: R\rightarrow P$을 정의한다. 그리고 기존의 CBDT의 효용 함수 u를 $u': R\rightarrow\Re^{+}$로 다음과 같이 재정의 한다:

$$u'(r)=u(r)+\gamma\max_{a\in A}V_{N(r)}(a) \quad (4)$$

여기서 r은 $r(p_{cur},a)$를 의미하고, 함수 u는 앞서 정의된 CBDT의 효용 함수이다. γ은 discount factor로서 u'의 값이 무한히 증가하지 않고 일정 값으로 수렴하도록 하는 역할을 한다. 이렇게 재정의된 u'으로 CBDT를 적용하기 위하여 사례를 $C=P\times A\times R\times\Re^{+}$로 확장하여 사용한다. 즉 하나의 사례는 (p,a,r,v), $v=\max_{a'\in A}V_{p_{nxt}}(a')\in\Re^{+}$로 나타낼 수 있다.

이렇게 재정의한 식 (4)와 새롭게 정의된 사례를 이용, 주어진 p에 대하여 식 (2)를 다음과 같이 쓸 수 있다.

$$V_p(a)=\sum_{(p',a,r',v')\in M}\left[\frac{\sigma(p,p')\bullet u'(r')}{\sum_{(p'',a,r'',v'')\in M}\sigma(p,p'')}\right],a\in A \quad (5)$$

더욱이, 지금까지 알고 있는 가장 좋은 action에 대한 V값이 threshold값을 만족하여 exploitation만을 수행함으로서 performance가 더 이상 증가하지 않는 현상을 방지하기 위하여, exploitation을 할 경우 "ϵ-greedy strategy"[11]를 사용하여 ϵ값을 0에 가깝게 놓고, exploration-exploitation을 하도록 한다.

연속되는 의사결정 문제에 동작하는 확장된 CBDT 알고리즘, Dynamic CBDT는 Figure 3과 같다.

INIT:
  SELECT $\gamma\in(0,1)$
  SELECT $\epsilon\in(0,1)$
  SELECT $threshold\in[0,\max_{r\in R}u'(r)]$
  SET *stopping condition*
END INIT

GET p_{cur}
Select $a_{cur}\in A$ by the uniform distribution over A
WHILE a *stopping condition* is satisfied
  EXECUTE a_{cur}
  GET $r(p_{cur},a_{cur})$, $p_{nxt}\leftarrow N(r(p_{cur},a_{cur}))$
  $d\in\arg\max_{a\in A}V_{p_{nxt}}(a)$
  $v=\max_{a\in A}V_{p_{nxt}}(a)$
  STORE (p_{cur},a_{cur},r,v) in M
  IF v < *threshold* THEN
    SELECT $a_{nxt}\in A$ by uniform distribution over A
  ELSE
    SELECT $a_{nxt}=d$ with $(1-\epsilon)$ probability and
    $a_{nxt}\in A$ by the uniform distribution over A with
    ϵ probability
  ENDIF
  $p_{cur}\leftarrow p_{nxt}$
  $a_{cur}\leftarrow a_{nxt}$
END WHILE

Figure 3. Dynamic CBDT 알고리즘

Figure 3에서 stopping condition은 사용자가 결정한 최대 iteration이 될 수도 있고, 의사 결정자가 어떤 goal에 도달하였을 경우도 될 수 있다.

기존의 CBDT를 확장한 알고리즘인 Dynamic CBDT를 적용하여 연속된 의사 결정과정에서 다음 문제를 고려한 의사 결정을 함으로서 기존의 CBDT보다 더 나은 결과를 얻을 수 있음을 실험을 통하여 확인하고자 한다. 다음 장에서는 이러한 Dynamic CBDT알고리즘을 CBDT알고리즘과 비교 분석하였다.

4. Dynamic CBDT알고리즘의 실험 및 분석

4.1 실험환경

CBDT알고리즘과 Dynamic CBDT간의 성능 비교를 하기 위하여 잘 알려진 게임 중 하나인 테트리스를 사용하였다. 일반적인 테트리스[12]는 20×10크기의 필드와 7가지 모양의 블록을 가진다. 이러한 특성을 가지는 테트리스의 problem을

{필드 의 모양, 현재 블록의 모양}

과 같이 표현할 경우 problem space는 $2^{200}\times7$의 크기를 가

진다. Problem space를 줄이기 위하여, 기존의 테트리스를 간단하게 재구성한, melax[13]가 제안한 테트리스를 사용하였다.

Melax 테트리스는 기존의 테트리스의 state size를 줄이기 위하여 제안된 테트리스로 Figure 4와 같이 블록의 종류가 기존의 테트리스의 블록보다 적고 모양이 단순하다. 블록의 집합 B는 Figure 4와 같다.

$B=\{$  $\}$

Figure 4. melax 테트리스의 블록

각각의 블록은 uniform distribution하게 선택되어 필드에 떨어진다.

실험을 위하여 20×4크기의 field를 사용하였다. 이때의 problem size는 $2^{80}\times5$를 가진다. 여기서 problem size를 더 줄이기 위하여 problem p를 다음과 같이 재정의 하였다. field의 넓이를 n이라고 하고 m번째 column의 최대 높이를 h_m, $m=1,2,...,n$라고 할 경우 다음과 같이 벡터로 표현하였다.

$$p=<h_1-h_2,h_2-h_3,...,h_{n-1}-h_n,b>,\ b\in B$$

여기서 마지막 속성인 현재 블록 모양을 제외한 나머지 속성들의 값의 범위를 $[-h_{Mk},h_{Mk}]$로 정하였다. 여기서 h_{Mk}는 블록의 최대 높이를 의미한다. 이렇게 설정한 이유는 각column의 높이 차가 h_{Mk} 이상일 경우와 h_{Mk}일 경우는 action 선택에 같은 영향을 미치기 때문이다.

위와 같이 problem을 재정의 하였을 경우 problem size는 $(2b+1)^{n-1}\times5$의 크기를 가지므로, 20×4의 필드를 가지는 melax 테트리스의 경우 $5^3\times5$의 problem size를 가진다.

테트리스의 쌓여있는 블록 중간 중간에 비어있는 칸을 무시하여 problem을 결정하므로 테트리스의 action a는 블록을 어느 위치에서 얼마만큼 회전시켜서 수직으로 떨어뜨리는 지만 결정하면 된다. 그러므로 하나의 action a는

$a=$ <블록의 column의 위치, 회전 횟수>

와 같이 블록의 column위치와 회전 횟수를 속성으로 가지는 벡터로 표현 할 수 있다. 이때 블록이 떨어지게 될 column의 위치 값은, 최대 값이 필드의 넓이를 넘을 수 없으므로 $[1,n]$의 값을 가지고, 블록의 회전 횟수는, 4번의 회전은 0번의 회전과 같으므로 [0,3]의 값을 가진다.

CBDT의 유사도 함수 σ는 다음과 같은 두 problem 이 있다고 하면,

$$p_1=<p_{11},p_{12},\cdots,p_{1n}>$$
$$p_2=<p_{21},p_{22},\cdots,p_{2n}>$$

일반적인 좌표의 distance를 구하는 방법을 사용하여 다음과 같이 구현하였다.

$$s=\sqrt{(p_{11}-p_{12})^2+(p_{12}-p_{22})+...+(p_{1(n-1)}-p_{2(n-1)})},$$

$$\sigma(p_1,p_2)=\begin{cases}0 & ,\text{if } p_{1n}\neq p_{2n}\\ \dfrac{S-s}{S} & ,otherwise\end{cases}$$

여기서 $p_{1n}\in B$과 $p_{2n}\in B$은 각각 각 problem의 현재 블록의 모양을 의미하고, S는 s의 최대 값을 나타낸다.

테트리스에서는 효용 값을 수행 즉시 알아낼 수 있으므로 결과를 저장하여 그 결과의 효용 값을 효용 함수를 통하여 알아내는 과정을 생략하고, 사례화 할 때 결과 대신 수행 후 알 수 있는 효용 값을 직접 저장하여 사례화 하였다. 사례에 저장되는 결과 r을 대신하는 효용 값은 다음과 같다.

효용 값 = (삭제한 줄 수)^2 + 2
+ (이전 필드 최고 블록의 높이)
- (현재 필드 최고 블록의 높이)

이렇게 정의한 problem과 action, 유사도 함수, 효용 함수를 통하여 테트리스 상에서 기존의 CBDT와 Dynamic CBDT와의 성능 차를 실험을 통하여 분석하였다.

4.2 실험 및 분석

성능 차를 분석하기에 앞서, CBDT와 Dynamic CBDT의 최적 threshold값을 찾기 위하여 threshold로 주어질 수 있는 여러 값들을 적용하여 실험하였다. round는 100번의 게임을 의미하고, score는 1 round 동안의 게임당 (삭제한 줄 수)^{2}의 평균값을 의미한다.

Figure 5와 6은 각각 CBDT와 Dynamic CBDT에 여러 threshold값을 설정하여 실험한 그래프로 세로축은 3000 round까지의 평균 score를 나타내고 가로축은 threshold값을 나타낸다. Figure 5를 통하여 CBDT는 threshold가 1일 때 가장 좋은 성능을 보여줌을 알 수 있고, Figure 6을 통하여, Dynamic CBDT는 9일 때 가장 좋은 성능을 보여줌을 알 수 있다. Dynamic CBDT는 $V_{p_{new}}$값을 계산할 때 식 (5)을 이용하여 기존의 효용 값에 지금까지 알고 있는 $\max_{a\in A}V_{p_{next}}(a)$을 더한 값을 weighted sum 하기 때문에 식 (2)를 통하여 계산하는 기존의 CBDT의 $V_{p_{new}}$값보다 큰 값을 가진다. CBDT는 threshold가 2 이상의 값을 가질 때, Dynamic CBDT는 17 이상의 값을 가질 경우 최악의 성능을 보임을 알 수 있다. 이것은 threshold값이 계산된 최대 V값보다 큰 값을 가지므로 exploration만을 수행하게 되기 때문이다. 실험을 통하여 threshold가 CBDT는 2 이상일 때, Dynamic CBDT는 17이상일 때 두 알고리즘이 같은 성능을 보임을 알 수 있었다.

Figure 7과 8은, 앞서 설명한 melax 테트리스 상에서, 각각 가장 좋은 성능을 내는 threshold값을 가질 때, 본 논문에서 제안한 Dynamic CBDT와 CBDT와 성능 비교를 측정한 그래프이다. 앞선 실험을 통하여 가장 좋은 성능을 보이는 값인, Dynamic CBDT의 threshold는 9, CBDT는 1로 설정하였다. Dynamic CBDT의 discounter factor는 0.85를 사용하였고 ϵ의 값은 0.05를 사용하였다. Figure 7과 8의 가로축은 round를 나타내고 세로축은 한 round(100 game)의 평균 score를 나타낸다.

Figure 7을 통하여, 연속된 의사결정 하에서, Dynamic CBDT가 CBDT보다 약 2.5배의 성능 차이를 보임을 알 수 있다. Figure 8은 Figure 7의 부분 그래프로 1~100 round 사이의 그래프를 확대한 것이다. CBDT가 초반에는 앞서지만 40 round정도에서부터 Dynamic CBDT가 CBDT와 비슷한 성능을 보임을 확인 할 수 있고, Figure 7을 통하여 CBDT는 성능이

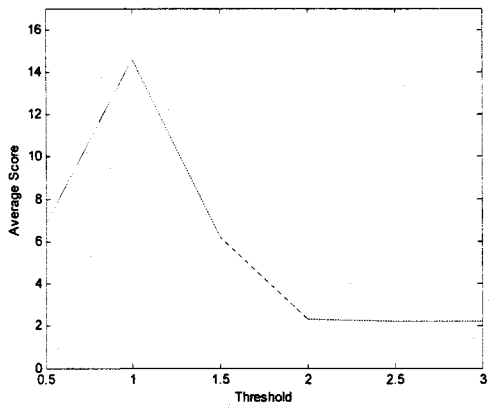

Figure 5. CBDT의 threshold에 따른 평균 score

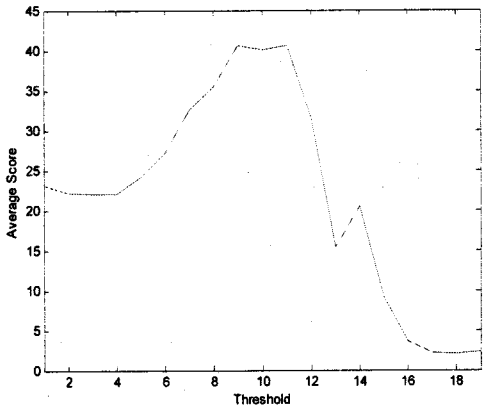

Figure 6. Dynamic CBDT의 threshold에 따른 평균 score

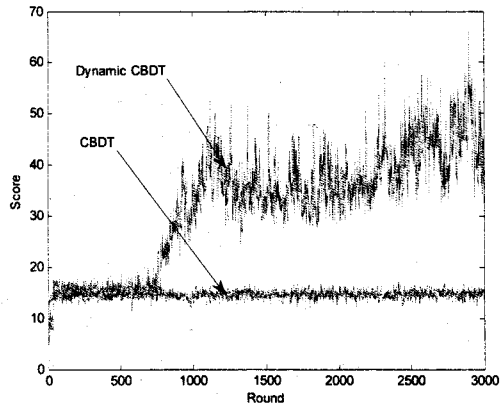

Figure 7. CBDT와 Dynamic CBDT와의 비교 그래프

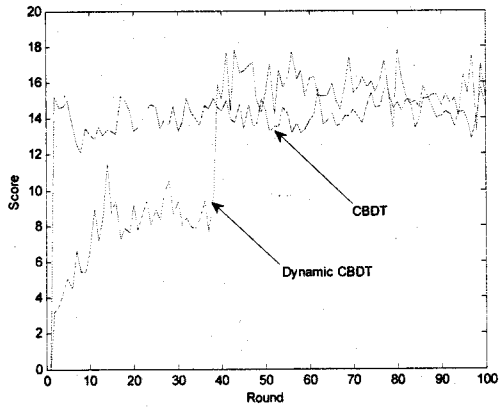

Figure 8. Figure 7의 부분그래프

유지되지만 Dynamic CBDT는 700 round에서는 성능이 더 증가하는 것을 볼 수 있다. CBDT는 현재 문제만을 고려한 최선의 선택을 하기 때문에 초반에는 앞서지만, 다음 문제까지 고려하기 위하여 확장한, Dynamic CBDT가 더 좋은 성능을 보인다.

이 실험을 통하여, 다음 문제에 대하여 고려하지 않고 의사결정을 하는 CBDT보다, Dynamic CBDT가 연속적인 의사 결정을 해야 하고 현재의 action이 다음 문제에 영향을 크게 미치는, 테트리스와 같은 환경에서 보다 좋은 학습 방법임을 확인 할 수 있다.

5. 결론 및 향후 연구 내용

지금까지 살펴본 바와 같이 불확실한 환경에 적합한 CBDT 알고리즘[1]을 연속된 의사결정 환경에서 보다 좋은 성능을 보일 수 있도록 확장하는 방법에 대하여 알아보았다. 앞에서 살펴본 바와 같이 CBDT가 사용하는 효용함수에 다음 problem의 최대 V값을 추가하여 다음 문제까지 고려한 효용함수로 재정의 하여 의사 결정을 함으로서 연속된 의사 결정 문제에서 기존의 CBDT보다 많은 성능 향상을 가져왔음을 알 수 있다. 앞서 threshold값에 따른 성능 비교 그래프를 통하여 threshold가 CBDT 알고리즘의 성능을 결정하는데 아주 큰 비중을 차지함을 알 수 있다. 이는 threshold가 Exploration과 Exploitation을 결정하는 기준이기 때문이다. 향후 CBDT의 threshold값을 performance에 따라 dynamic하게 변화시켜 줌으로서 CBDT의 성능을 개선하는 연구를 진행할 예정이다.

CBDT의 경우 유사도 함수가 정확할수록 의사 결정과정에서 더 강력한 모습을 보이기 때문에 이번 실험에서 사용한 일반적인 distance를 구하는 방법을 사용한 유사도 함수가 아닌 더 정확한 유사도 함수를 사용한다면 더 좋은 성능을 보여줄 것으로 예상된다.

참 고 문 헌

[1] Itzhak Gilboa; David Schmeidler "Case-based decision theory," *The Quarterly Journal of Economics*, Vol. 110, No.3, pp. 605-639, 1995.

[2] I. Gilboa "Case-based optimization," *Games and Economic Behavior*, Vol 15, pp. 1-26, 1996.

[3] L. P. Kaelbling; Michael L. Littman; Andrew W. Moore "Reinforcement learning: A survey," *Journal of Articial Intelligence Research*, Vol. 4, pp. 237-285, 1996.

[4] J. V. Neumann; O. Morgenstern "Theory of Games and Economic Behaviour," *Princeton Univ. Press*, 1944.

[5] R. Brafmann; M. Tennenholtz "On the foundations of qualitative decision theory," *in Proc. 13th Nat. Conf. Artificial Intelligence (AAAI-96)*, pp. 1291-1296, 1996.

[6] H. Fargier; J. Lang; T. Schiex "Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge," *in Proc. 13th Nat. Conf. Artificial Intelligence (AAAI-96)*, pp. 175-180, 1996.

[7] B. Bonet and H. Geffner, "Arguing for decisions: A qualitative model for decision making," *in Proc. Uncertainty Artificial Intelligence (UAI-96)*, pp. 98-105, 1996.

[8] D. B. Leake "Case-Based Reasoning: Experiences, Lessons, and Future Directions," *AAAI Press/MIT Press*, 1996.

[9] E. Hüllermeier "Experience-based decision making: a satisficing decision tree approach," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 35, NO. 5, pp. 641-653, 2005

[10] J. G. Marc; Herbert A. Simon "Organizations," *Blackwell Publishers*, 1993.

[11] R. S. Sutton; Andrew G. Barto "Reinforcement Learning: An Introduction," *MIT Press, Cambridge*, 1998

[12] E. D. Demaine; S. Hohenberger; H. J. Hoogeboom; W. A. Kosters; D. Liben-Nowell "Tetris is hard, even to approximate," *International Journal of Computational Geometry and Applications*, Vol. 14, No. 1, pp. 41, 2004.

[13] S. Melax "Reinforcement learning tetris example," 1998. URL http://www.melax.com/tetris/.