□기술해설□

설명 기반 학습(EBL)의 원리와 응용

부산대학교 류 광 렬*

목 차

1. 서 론

학습을 할 수 있는 시스템은 별도의 프로그래밍에 의존함이 없이도 자신의 문제 해결 능력을 스스로 향상시킬 수 있다. 여기서, 문제 해결 능력의 향상이란 의미는 두가지의 서로 다른 측면에서 설명되어야 한다. 첫째는 이전에 해결하지 못하던 문제를 새로 학습한 지식을 이용하여 풀 수 있게 되는 것이고, 둘째는 이전에도 해결할 수 있었던 문제일지라도 경험이 축적됨에 따라 그것을 보다 빨리 효율적으로 풀 수 있게 되는 것이다. 전자의 경우는 원래 시스템이 가지고 있지 못하던 지식을 학습을 통해 획득함으로써 가능해 지는데 비해, 후자의 경우는 원래 시스템이 가지고 있던 지식을 단지 적용하기에 보다 편리한 형태로 변환시켜 줌으로써 가능해 지는 것이다. 흔히 이들 두 경우를 구분하여 전자와 관련된 학습을 지식 수준 학습(knowledge level learning), 그리고 후자와 관련된 학습을 부호 수준 학습(symbol level learning)이라 부르고 있다[1].

*정회원

지식 수준 학습에 의하면 어떤 목표 개념(target concept)을 배우기 위해 그 개념의 실사례(example) 및 반사례(counter example)에 해당하는 다량의 데이타를 분석함으로써 그 개념의 공통적인 속성 및 상이 개념과의 차별성등을 귀납적(inductive)으로 찾아 낸다. 이렇게 찾은 공통성 및 차별성은 어떠한 임의의 새로운 사례가 주어질 경우 그것이 목표 개념에 속하는 것인지의 여부를 판별하는 조건이 되므로, 이를 근거로 목표 개념 인식을 위한 판단 규칙(decision rule)이 만들어 진다. 귀납적 학습에 의해 유도된 규칙은 많은 데이타의 관찰 결과에 다름아니므로 정확성(correctness)의 보장은 없다. 예를 들어 자동차 고장 진단용 전문가 시스템을 개발할 때 전문가로부터 직접 지식 획득이 어려울 경우, 많은 고장 사례를 모아 둔 데이타베이스만 있다면 그로부터 귀납적 학습을 통해 고장 증세와 고장 원인을 연결하는 규칙을 추출함으로써 필요한 전문 지식을 유도해 낼 수 가 있다. 그러나 이렇게 데이타로부터 추출된 규칙이 모든 고장 사례에 대해 항상 정확한 진단을 내려주리란 보장은 없는 것이다.

부호 수준 학습은 이와는 대조적으로, 어떤 임의의 사례가 하나 주어졌을 때 그것이 특정 목표 개념에 속하는 것인지의 여부를 이미 시스템이 스스로 가지고 있는 지식을 활용하여 추론함으로써 판별할 수 있다는 가정 하에서, 차후에 유사한 사례가 주어지면 추론 시간을 단축할 수 있도록 하려는데 그 목적이 있다. 이를 위해서, 주어진 사례가 목표 개념에 속하는 것으로 판단하는 과정에 어떠한 지식들이 동원되어 어떤 추론 고리(reasoning chain)를 거쳤는지를 분석 설명한 후, 그 분석 결과를 바탕으로 추론 고리에 개입되었던 지식들을 컴파일(compile)된 형태로 변환하여 기억시킴으로써 차후에 목표 개념에 속하는 사례의 인식을 신속하게 할 수 있게 한다. 이러한 학습 방법은 분석과 설명을 바탕으로 한다 하여 주로 설명 기반 학습(EBL : Explanation-Based Learning)이라 불린다[2, 3]. EBL에 의해 유도되는 지식은 연역적(deductive) 추론 과정을 통한 것이므로 항상 정확성(correctness)이 보장되고 학습에 필요한 사례의 수가 하나라는 점에서도 귀납적 학습과 대조된다. 귀납적 학습은 기존 데이타에 내재하는 규칙성(regularity)을 이끌어 내어 지식으로 활용하려는 목적에 주로 쓰이는 반면, EBL 방식은 일반적으로 문제풀이(Problem Solving)나 계획(Planning) 등을 행하는 수행 시스템(performance system)의 효율을 향상시키는데 주로 응용되고 있다. 시스템의 효율이 향상되면 일차적으로는 프로그램의 수행 속도가 빨라지는 것이지만 그에 따라 시스템의 문제 해결 능력도 개선되는 효과가 있다. 제한된 자원을 가지고 탐색을 수행해야 하는 시스템의 입장에서는 탐색 제어가 효과적으로 이루어지면 그에 따라 절약되는 노력을 보다 나은 답을 찾는데 투입할 수 있으므로 결과적으로 해(solution)의 질적 수준도 향상시킬 수 있게 된다.

본 논문은 EBL의 방법론과 문제점 및 해결해야 할 과제 등을 적용 사례와 함께 소개한다. 2장에서는 EBL의 학습 방법을 소개하고 그것이 시스템의 효율 향상을 위해 적용되는 분야로서 탐색 제어 규칙의 학습과 macro operator 학습을 설명한다. 3장에서는 EBL의 문제점으로 알려져 있는 사항들 중 설명에 필요한 domain theory가 완벽하지 못할 경우의 문제, 학습된 규칙들이 시스템의 속도를 오히려 저하시키게 되는 현상인 utility 문제, 그리고 학습된 규칙의 역기능 중 또 다른 것으로서 시스템이 찾아내는 해답의 질을 떨어 뜨리는 현상인 masking 효과를 설명한다. 4장에서는 최근의 연구 동향을 간략히 정리하고, 마지막으로 5장에서 결론을 맺는다.

2. EBL의 방법론

EBL은 서론에서 언급하였듯이 주로 문제풀이 시스템의 성능 개선, 특히 문제를 푸는데 걸리는 시간의 단축을 위한 학습 방법(speedup learning)으로 많이 응용된다. 문제를 푼다는 것은, 상태 공간(state space) 내에서 주어진 초기 상태(initial state)로부터 목표 상태(goal state)까지 상태 변환(state transition)을 시켜줄 수 있는 일련의 operator를 찾아내는 탐색(search)작업이라 볼 수 있다. 이러한 탐색을 수행하는 시스템은 탐색도중 매 분기점(branching point)에서 어떤 경로를 선택해 탐색을 계속할 것인지를 결정해야 한다. 이때 가장 유망한 경로를 선택해 낼 수 있는 제어 전략(control strategy)을 갖춘 시스템일수록 탐색 제어(search control)를 효율적으로 수행하여 좋은 성능을 나타내게 된다. 그런데, 수행 시스템의 설계 단계에서 일반적으로 널리 적용되는 탐색 제어 규칙(search control rule)들을 최대한 많이 확보하여 미리 준비해 두더라도, 그것들 만으로 특정한 상황마다 매번 효과적인 제어를 해 내기는 지극히 어렵다. 그 이유는, 문제마다 탐색 제어를 해야 할 상황들이 매우 복잡하고 다양하게 전개되므로 이들을 사전에 일일이 예측하여 대비할 수 있는 규칙을 모두 확보하는 것이 불가능 하기 때문이다.

이에, 많은 연구가들은 시스템 스스로가 각종 탐색 제어 상황을 겪어 나감에 따라 적절한 제어 규칙을 자동으로 배우게 함으로써 문제 풀이의 경험의 축적에 비례하여 효율이 점차 개선되게 하는 방법을 고안하게 되었다[4, 5, 6, 7]. 이들 방법에 의하면 시스템이 자기 스스로가 어떻게 탐색을 수행하는가에 대한 지식을 갖추

게 하여, 탐색시 상황마다 제어를 어떻게 한 것이 어떤 결과(성공이든 실패든)를 초래하였는지를 분석하게 하고 그로부터 얻은 유용한 결론을 다음의 유사한 상황에서 활용할 수 있도록 EBL을 적용 하고 있다. 탐색 제어 규칙의 학습에는 이외에도 귀납적 학습(inductive learning) 방법을 사용한 연구 사례도 발표되고 있다[8, 9, 10].

이상과 같이 탐색시 대안들(alternatives)을 모색해 가는 과정에서 대안 선택의 결정을 적절하고 신속하게 할 수 있도록 규칙을 학습하는 방법과 더불어 또 한가지 탐색 효율을 높이는 방안으로는, 탐색 결과 찾아낸 operator sequence들을 조합하여 macro operator들을 만들어 기억시킴으로써 차후에 문제풀이 시스템이 이들을 사용하여 탐색 공간내에서 한번에 큰 도약을 할 수 있게 하는 EBL 방법이 있다[11]. 규칙 기반 시스템(rule-based system)에서 규칙들을 모아 결합한 macro 규칙, 자연어 이해 시스템에서 사용하는 schema나 script, 그리고 심리학이나 인지과학에서 이야기하는 chunk 등이 사실 모두 macro operator류에 속하는 것으로 볼 수 있다.

2.1 EBL 알고리즘의 적용예

본 절에서는 EBL 알고리즘을 보다 구체적으로 설명하기 위하여 [12]의 내용 중 일부를 인용한다.

그림 1은 EBL 알고리즘의 설명을 위해 EGGS 시스템[13]의 적용예를 보인 것이다. 그림의 윗쪽에는 EGGS가 주어진 문제를 풀기 위해 가지고 있는 문제 domain의 지식인 domain theory가 규칙의 형태로 나열되어 있는데 여기에 등장하는 ?x, ?y 등 물음표로 시작하는 부호들은 변수를 나타낸다. 이들 규칙의 의미는 (1) 어떤 사람이 누군가 친절한 사람을 알게되면 그를 좋아한다 ; (2) 살아 있는 객체는 스스로를 안다 ; (3) 사람은 살아 있는 객체다 ; (4) 우호적인 사람들은 친절하다 ; 그리고 (5) 행복한 사람들은 친절하다는 것으로 풀이될 수 있다. EGGS에게 주어진 문제는, John이 행복한 사람이라는 사실을 알고 있다는 가정하에서 John이 스스로를 좋아한다는 것을 증명하는 일이다. 이는

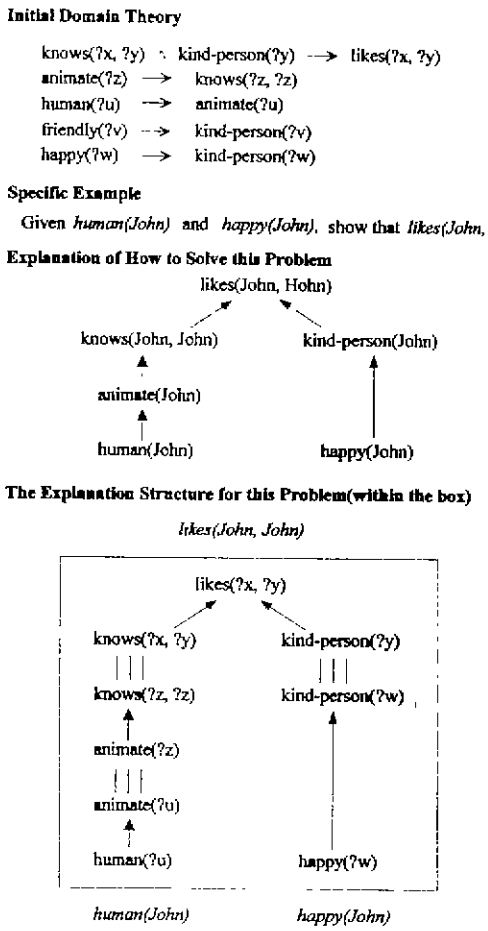


그림 1 EGGS의 EBL 알고리즘 적용예

그림 1의 윗쪽 트리 형태의 그래프에 보인 바와 같이 domain theory를 이용하여 쉽게 할 수 있다. 이 그래프는 문제풀이의 과정에 개입된 규칙들과 추론 고리를 보여주는 일종의 설명이라 볼 수 있다.

만약 이 문제를 푼 결과를 단순히 기억하려 한다면 "John이 사람이고 John이 행복하다면 John은 스스로를 좋아한다"라고하는 규칙을 만들게 되겠지만, John 이외의 사람들에게도 적용될 수 있게 하기 위해서는 이 규칙을 일반화(generalize)해야 한다. 이를 위해 EGGS는 문

제풀이 결과의 설명에 들어 있는 instantiate된 규칙들을 모두 원래의 변수를 가진 일반 규칙으로 대치함으로써 그림 1에서 점선으로 싸인 부분에 도시한 바와 같은 설명 구조(explanation structure)[2]라는 것을 만들어 낸다. 이 그림에서 삼중 실선은 한 규칙의 조건부가 다른 규칙의 결론부와 unify되어야 한다는 것을 표시한 것으로, 예를 든다면 *kind-person*(*?w*)가 *kind-person*(*?y*)와 unify된다는 것이다. 여기서 한가지 주의할 것은 문제의 구체적 목표인 *likes*(*John, John*)과 구체적 사실 데이타인 *human*(*John*) 및 *happy*(*John*)은 설명 구조 속에 포함되어 있지 않다는 점이다.

EGGS의 최종 목적은 설명 구조에 들어 있는 일반 규칙들이 이런식으로 서로 연결되기 위하여 규칙의 변수들이 따라야 할 최소한의 제약 조건을 찾는 것이다. 이는 삼중 실선으로 연결된 모든 쌍들에 대한 가장 일반적인 unifier를 찾음으로써 이루어 진다. 본 예에서는 모든 변수들이 동일해야 한다는 것이 바로 그 제약 조건이다.

EGGS는 설명 구조의 root와 leaf node들을 추출하여 "행복한 사람은 스스로를 좋아한다"라는 일반 규칙을 만든다. 이 규칙은 이제 John 이외의 사람들에 대해서도 스스로를 좋아함에 관한 추론을 효율적으로 하는데 적용될 수 있다. 학습한 규칙에 의한 속도의 개선은 문제풀이에 개입되는 중간 추론 과정을 뛰어 넘을 수 있게 됨으로 인해 이루어 지기도 하지만, 보다 중요한 것은 이 문제풀이에 개입되어 도움이 되지 않는 규칙들에 대한 탐색을 피할 수 있게 됨으로써도 이루어 진다는 점이다.

2.2 탐색 제어 규칙의 학습

문제풀이 시스템이 문제에 주어진 목표를 달성하기 위하여 탐색을 수행할 때 매 분기점(branching point)에서 만나게 되는 여러 대안들 중 가장 유망한 경로를 선별해 낼 수 있다면 매우 효율적인 탐색을 할 수 있게 될 것이다. 여기서 여러 대안들 중에서의 선택이란 보다 구체적으로, 탐색 트리(search tree)의 노드(node)들 중 다음에 전개(expand)할 노드를 선택하는 것, 선택된 노드 내에 들어 있는 목표(goal or subgosl)들 중에서 다음에 수행할 것을 선택하는 것, 선택된 목표의 달성을 가능하게 해 주는 opearator들 중 가장 적합한 것을 찾는 것, 그리고 선택된 operator내에 포함되어 있는 변수들의 binding을 결정하는 것 등을 말한다. 탐색 제어 규칙은 결국 이러한 대안들 중에서 어느 하나를 선택(select), 배제(reject), 또는 선호(prefer)하도록 결정을 해 주는 규칙을 의미한다. 별도의 탐색 제어 규칙이 없는 시스템은 무작위적으로 혹은 대안들의 발생 순서에 따라 탐색을 수행할 수 밖에 없을 것이다.

탐색 제어 규칙을 학습하기 위해서는 탐색 과정에서 생성되는 탐색 트리를 분석하여 어떤 대안을 선택한 것이 궁극적으로 어떤 결과를 초래하게 되었는지를 파악하는 것이 첫 순서다. 이를 위해서는 탐색 시스템이 탐색을 수행하는 방법에 대한 지식과 대상 문제의 성격에 관한 지식을 학습 시스템이 가지고 있어야 한다. PRODIGY 시스템[4]의 예를 보면, 그림 2에 일부 보인 것처럼 이들 지식을 탐색 시스템에 관한 architecture-level axiom과 대상 문제 영역에 관한 domain-level axiom으로 나누고 있다. 이 중 domain-level axiom은 대상 문제에 사용되는 operator의 precondition과 postcondition을 분석함으로써 얻은 것이다. PRODIGY의 학습 시스템이 탐색 트리를 분석한다는 것은 결국 탐색 트리의 각 부분 부분들이 이들 axiom 중 어떠한 것들과 match되는 예인지를 찾는 작업인 것이다. 탐색 트리의 한 부분과 일군의 axiom들 사이에 match가 이루어 지면 2.1절에서 언급한 바와 같은 설명 구조가 생성되고 그것으로 부터 변수화 과정을 거쳐 탐색 제어 규칙이 유도된다. PRODIGY의 경우, 탐색 가지(branch)가 성공으로 이어진 것으로 설명되면 그로부터 선호 규칙을, 실패 이면 배제 규칙을, 어떤 한 분기점의 대안들 중 어느 하나를 제외한 나머지 모두가 실패하면 선택 규칙을, 그리고 목표들 간의 수행 순서가 잘못 선택됨으로써 상호 간섭 현상이 발생하여 실패하면 그것을 피할 수 있는 순서를 선호하는 규칙을 각각 학습하게 된다.

```
Architecture-Level Axioms

(OPERATOR-SUCCEEDS op goal node)
if (AND (MATCHES-EFFECT goal op)
        (APPLICABLE op node))

(OPERATOR-SUCCEEDS op goal node)
if (AND (APPLICABLE next-op node)
        (OPERATOR-SUCCEEDS op goal child-node)
        (CHILD-NODE-AFTER-APPLYING-OP child-node next-op node))

Domain-Level Axioms

(APPLICABLE op node)
if (AND (MATCHES-OP op (POLISH obj time))
        (KNOWN node (AND (IS-OBJECT obj)
                         (OR (SHAPE obj RECTANGULAR)
                             (CLAMPABLE obj POLISHER))
                         (AVAILABLE obj time)
                         (IDLE POLISHER time))))
(APPLICABLE op node)
if (AND (MATCHES-OP op (CLAMP obj time machine))
        (KNOWN node (AND (HAS-CLAMP machine)
                         (TEMPERATURE obj COLD))))
(MATCHES-EFFECT effect op)
if (AND (MATCHES-OP op (POLISH obj time))
        (MATCHES effect (SURFACE-CONDITION obj POLISHED)))
```

그림 2 PRODIGY의 EBL에 사용되는 domain theory의 일부

2.3 Macro Operator의 학습

탐색의 결과 찾아낸 operator sequence들을 조합하여 macro operator를 만들어 둠으로써 차후에 유사한 상황이 발생하면 다시 탐색을 되풀이할 필요 없이 macro operator를 이용하여 바로 답을 제시하도록 하는 방안은 일견 사례 기반 추론(CBR : Case-Based Reasoning) 방식과 흡사해 보인다. 그러나, 이 두 방식은 문제 풀이의 결과를 저장하는 형태에 있어서나 그 저장된 결과를 차후에 사용하는 방법에 있어서 모두 현격한 차이를 보이고 있다. CBR 방식에 의하면 문제풀이의 결과를 그대로 변형없이 메모리에 사례(case)로 저장한다. 차후에 새로운 문제가 주어지면 복잡한 탐색 과정을 거치는 대신 사례 베이스(case base)로 부터 가장 유사해 보이는 사례를 가져온 후 그것을 새 문제에 맞게 변형시키는 adaptation 과정을 거침으로써 새 문제에 대한 답을 효율적으로 만들어 낸다. 따라서, CBR의 성공 여부는 유사 사례를 판별하여 찾아 오는 indexing 방법과 그 사례의 내용을 새 문제에 맞게 고치는 adaptaion 방법이 얼마나 좋으냐에 달려 있다고 볼 수 있다. 반면에, macro operator는 탐색 결과 찾아낸 operator sequence를 일반화함으로써 만들어 지는 것이다. 즉, 어떤 구체적인 문제 풀이의 결과인 operator sequence 내에 들어 있는 상수들을 2. 1절에서의 설명과 비슷한 방법으로 변수화함으로써 유사한 문제를 푸는데 사용될 수 있는 형태로 만들어져 메모리에 저장된다. Macro operator를 사용하여 새로운 문제를 풀 때는 macro operator의 변수화된 precondition이 새 문제와 부합하기만 하면 그것을 가져와서, macro operator 내의 변수들을 새 문제와의 matching 결과에 따라 상수화함으로써 새로운 답을 쉽게 만들어 낼 수 있다. CBR 방식과 비교하면, macro operator는 indexing이나

adaptation과 관련한 어려움은 없는 대신 그 적용 범위가 상당히 제한적이다. 예를 들어, 어떤 주어진 문제를 풀기 위해서 메모리 내의 한 macro operator를 그대로 적용해서는 안되고 그 sequence 내에 하나의 operator를 더 추가하기만 하면 되는 경우가 있더라도, adaptation 과정이 없기 때문에 그 macro operator의 적용 자체가 불가능 해진다.

본 절의 나머지에서는 다시 [12]의 일부를 인용하여 macro operator의 학습 과정을 보다 구체적으로 설명한다.

문제풀이 시스템이 문제의 description으로 주어진 어떤 초기 상태(initial state)를 문제의 해답인 최종 상태(final state)로 변환 시키기 위하여 이를 가능하게 해 주는 operator들을 찾을 경우, 각 상태 마다 적용 가능한 operator들이 여러 개 있을 수 있기 때문에 만족스런 operator sequence를 찾기 까지 탐색을 수행해야 한다. 이 경우 시스템의 속도를 높여주는 한 방법으로서, 이전에 풀었던 문제들과 그 해답들을 문제 description P와 해답 S의 쌍, 즉 $\langle P, S\rangle$의 형태로 기억시켜 둔다면 새로운 문제가 주어질 때 이미 풀었던 것을 다시 풀 필요가 없게 될 것이다.

그러나, 이 방식은 과거에 풀었던 문제와 완전히 동일한 문제를 만날 경우에만 속도 개선의 효과를 볼 수 있을 뿐이다. 보다 나은 접근 방법은 $\langle P, S\rangle$를 일반화 하여 상수를 변수로 바꾸고 불필요하게 상세한 세부 사항들을 없애 버림으로써 $\langle P^*(x), S^*(x)\rangle$로 만드는 것이다. 새로운 문제가 주어지면 먼저 $P^*(x)$와의 match를 시도하고 성공할 경우 match된 x의 값을 $S^*(x)$에 대입하여 답을 구할 수 있다. 이렇게 일반화된 문제/해답의 쌍 $\langle P^*(x), S^*(x)\rangle$가 바로 $P^*(x)$를 조건부로 하고 $S^*(x)$를 결과부로 하는 macro operator가 된다.

$\langle P, S\rangle$를 정확히 일반화하기 위해서는 문제 description P의 어떤 부분이 해답 S를 찾는데 중요한 부분이며, 또 P의 세부 내용과 S의 세부 내용 사이에 정확히 어떤 연관 관계가 있는지를 알아 내어야 한다. $\langle P, S\rangle$를 일반화 하는데 필요한 이러한 모든 정보는 P로 부터 S에 도달하기 까지의 전 과정에 걸친 문제풀이의 기록으로 부터 찾을 수 있다는 것이 EBL의 핵심이 된다. 이 기록은 결국 해답이 어떻게 구해졌는지에 대한 설명이라 볼 수 있고, EBL은 이 설명을 분석하여 2.1절에서 설명한 바와 같은 방법으로 일반화함으로써 새로운 macro operator인 $\langle P^*(x), S^*(x)\rangle$를 학습하게 된다.

3. EBL의 문제점

EBL 시스템은 문제풀이의 결과를 일반화하여 기억하거나 문제풀이의 과정에서 겪게 되는 탐색용 제어 규칙을 유도해 학습하는 방법으로 시스템의 성능을 개선할 수 있다고 생각하지만 실제로 반드시 그렇지만은 않다. 본 장에서는 EBL과 관련한 문제점으로 우선 불완전한(imperfect) domain theory가 학습에 미치는 영향을 설명하고, 이어서 EBL로 학습된 규칙들이 문제풀이의 시간 단축의 측면에서 뿐만 아니라 문제풀이의 결과로 얻게되는 해답의 질적 수준에 관해서도 오히려 악영향을 줄 수 있음을 지적하고 그에 대한 대책들을 소개한다.

3.1 불완전한 Domain Theory

EBL은 주어진 domain theory를 기초로 연역적 추론 과정을 거쳐 학습 대상 사례를 일반화하므로 그 일반화의 결과가 항상 정확(correct)하다는 보장이 있다. 그러나 이는 어디까지나 주어진 domain theory에 관한한 그렇다는 것일 뿐이다. 만약 domain theory 자체에 오류가 포함되어 있다면 학습 결과에도 오류가 있을 것은 당연한 일이다. 또한, domain theory에 오류가 없더라도 그것이 대상 문제의 제반 현상을 완벽하게 다 설명하지 못하는 것이라면 학습의 범위도 그만큼 제한적일 수 밖에 없다. 탐색 제어 규칙을 학습하는 시스템의 경우 문제풀이 시스템의 탐색 방식을 완벽하게 domain theory로 표현하지 못하면 학습 과정에서 일부 중요한 탐색 제어 규칙을 놓치게 된다. 실제 예로서, PRODIGY의 경우 목표들 간의 간섭 현상에 관한 domain theory가 완벽하지 못함으로 인해 목표 순서에 관한 규칙의 학습이 충분히 이루어 지지 못하는데 비해, PAL 시스템[6]의 경우에는 간섭 현상을 보다 철저히 분석함

으로써 PRODIGY가 학습하지 못하는 순서 규칙을 학습할 수 있게 되어 있다.

Domain theory가 불충분함으로써 생길 수 있는 또 한가지 문제는 학습된 규칙의 조건부가 충분히 일반화 되지 못하여(over-specific) 그 규칙의 활용도가 떨어지게 되는 것이다. 그림 3과 그림 4는 동일한 하나의 문제로 부터 PRODIGY가 학습한 규칙과 PAL이 학습한 규칙을 각각 보이고 있다. PRODIGY의 규칙은 PAL의 규칙에 비해 불필요하게 많은 조건들을 가지고 있어서 그 규칙을 사용하려 할 때 matching 과정에서 많은 시간이 소요될 뿐 아니라 매우 제한적인 상황에서만 규칙이 적용될 수 있도록 되어 있다. 반면에 PAL의 규칙을 보면 해당 두 목표의 경우 무조건적으로 순서가 결정되어야 하는 것으로 되어 있어서 규칙의 활용도가 매우 높다.

불완전한 domain theory의 문제는 현실적인 대규모의 문제에서 거의 언제나 발생한다고 보아야 한다. [2]는 이 문제를 세 가지 경우로 나누어 설명하고 있다. 첫째는 domain theory가 incomplete할 경우로, 주가 예측이나 일기예보와 같이 그 현상이 너무 복잡하여 완전한 파악이 어려울 때 발생한다. 이 때는 주어진 theory만 가지고 plausible한 설명을 이끌어 내는 방안을 강구해야 할 것이고, 또 theory 자체를 학습에 의해 보충하여 점차 완전해 지도록 하는 방안이 연구되어야 한다. 둘째는 domain theory가 intractable할 경우로, 어떤 현상에 대한 완전한 이론이 이미 정립되어 있더라도 실제 현상을 설명하는 과정이 계산적으로(computationally) intractable할 때를 말한다. 이 경우에는 현상을 개략적으로 설명할 수 있는 approximate theory를 만들어 냄으로써 설명 과정을

```
(lhs
 (and (current-node <n>)
      (candidate-goal <n> (surface-condition <x> POLISHED))
      (known <n> (has-clamp POLISHER))
      (known <n> (temperature <x> COLD))
      (known <n> (is-object <x>))
      (known <n> (last-scheduled <x> <t1>))
      (known <n> (later <t2> <t1>))
      (in-goal-exp <n> (joined <y> <x> <or-1>))
      (known <n> (is-object <y>))
      (known <n> (can-be-welded <y> <x> <or-1>))
      (known <n> (last-scheduled <y> <t3>))
      (known <n> (composite-object <yx> <or-1> <y> <x>))
      (known <n> (later <t4> <t3>))
      (known <n> (later <t4> <t2>))
      (forall (<m-1>)
              (Known <n> (scheduled <w> <m-1> <t2>))
              (not-equal <m-1> POLISHER))
      (or (not-equal <y> <x>)
          (not-equal <t3> <t1>))
      (forall (<m-2>)
              (Known <n> (scheduled <z> <m-2> <t4>))
              (not-equal <m-2> WELDER))
      (candidate-goal <n> <other-goal>)
      (not-equal (surface-condition <x> POLISHED) <other-goal>)))
(rhs
 (prefer goal (surface-condition <x> POLISHED) <other-goal>))
```

그림 3 PRODIGY에 의해 유도된 지나치게 제한적인 규칙의 예

```
(lhs
 (and (current-node <n>)
      (candidate-goal <n> (surface-condition <x> POLISHED))
      (candidate-goal <n> (joined <y> <x> <or>))))
(rhs (prefer goal (surface-condition <x> POLISHED)
                  (joined <y> <x> <or>)))
```

그림 4 그림 3과 동일한 문제로부터 유도된 PAL의 규칙

tractable하게 하는 방안이 유력해 보인다. 셋째는 domain theory가 inconsistent할 경우로, 어떤 현상에 대해 서로 상충되는 설명이 만들어 질 때 생길 수 있는 문제다.

3.2 Utility 문제

EBL 시스템은 학습된 규칙들을 활용함으로써 새로운 문제를 빨리 풀 수 있어야 한다. 그러나, 만약 주어진 문제를 푸는데 도움이 되는 규칙을 메모리로 부터 하나도 발견하지 못하는 경우가 발생한다면 규칙을 찾는데 소모한 시간은 낭비밖에 되지 않는다. 도움이 되는 규칙을 발견하는 경우라도 학습된 규칙이 저장된 메모리가 방대하여 도움되는 규칙을 찾는데 걸리는 시간이 그냥 문제를 푸는 것 보다 오히려 더 길다면 학습된 규칙을 사용하는 것이 무의미해진다. Utility 문제란 학습에 의해 규칙이 누적되어 감에 따라 이렇게 시스템의 속도가 늦어지는 현상을 말한다.

[14]에 의하면 PRODIGY는 다음과 같은 비용/혜택(cost/benefit)의 식으로 utility를 표현하여 학습된 제어 규칙의 속도 개선 효과에 관한 척도로 삼는다.

Utility=(*AvrSavings ApplicFreq*)-*AvrMatchCost*

여기서 *AvrSavings*는 규칙이 적용 가능(applicable)할 때 탐색을 피할 수 있게 됨으로 인해 생기는 시간 절감의 평균치이고, *ApplicFreq*은 규칙이 match되었을 때 그 규칙이 적용 가능한 것으로 판명될 확률이며, *AvrMatchCost*는 규칙을 match하는데 소요되는 시간의 평균치이다.

하나의 제어 규칙이 학습되고 나면 PRODIGY는 그 규칙을 유도하게 했던 사례 문제를 기초로 일단 그 규칙의 utility 값을 추정한다. 즉, 규칙을 match하는데 소요된 시간과 그 규칙의 사용으로 절감된 탐색 시간을 비교하는 것이다. 절감 시간이 소요 시간보다 클 경우에만 그 규칙을 기억 저장하게 함으로써 이 추정 단계에서 명백히 비효과적 이라고 판단되는 규칙들을 미리 제거해 버린다. 이렇게 함으로써 얻게 되는 한 가지 중요한 부수적 이득은 이들 비효과적인 규칙들이 차후 다른 규칙 학습시의 설명 과정에 개입하는 것을 막을 수 있게 되는 것이다. 학습을 위해 어떤 탐색 과정의 결과를 분석하는데, 만약 그 탐색 과정 중에 비효과적인 규칙이 개입되어 포함되어 있다면 분석과 설명 자체가 비효과적인 면을 그대로 반영한 것일 수 밖에 없고, 그렇게 되면 그 설명으로 부터 유도되는 규칙은 더욱 더 비효과적인 것이 된다. 이런 과정이 되풀이 되면 걷잡을 수 없이 점점 비효과적인 규칙이 양산될 위험이 있다.

PRODIGY에 의하면 규칙이 시스템에 저장되고 난 후에도 utility의 초기 추정치를 실험적으로 검증하여 utility가 음의 값이 되는 규칙이 제거되도록 하고 있다. *AvrMatchCost*와 *ApplicFreq*은 바로 측정에 의해 구할 수 있으나 *AvrSavings*를 구하는 것은 쉽지 않다. 이를 위해서는 각 문제마다 규칙이 있을 때와 없을 때를 구분하여 시스템을 수행시켜 보아야 하고, 그 작업을 또 개개의 규칙에 대해 모두 실시해야 한다. PRODIGY에서는 이렇게 시간 소모적인 과정을 생략하고, 대신 규칙이 학습되었던 사례 문제 하나만을 기초로 AvrSavings 값의 추정치를 구하여 사용한다.

이상의 utility 측정 방법이 가지는 문제점들 중의 하나는 각 규칙의 utility를 독립적으로 생각한다는 것이다. 이렇게 할 경우 최종적으로 남는 규칙들이 반드시 최고의 utility를 갖는 규칙 집단(rule set)이 되리라는 보장이 없다. 일반적으로 규칙들 간에는 서로 상호작용이 있

다. 예를 들어, 하나의 선택 규칙이 추가될 경우 그 전부터 존재하던 배제 규칙의 *ApplicFreq*이 영향을 받을 수 있는 것이다. 하나의 집단으로서 규칙들의 utility를 극대화 하기 위해서는 규칙들 간의 상호 작용을 고려할 수 있는 보다 고도의 세련된 기법이 필요하다.

이 외에도 utility 문제의 해결을 위해서는 여러 가지 다양한 방법들이 연구되었다. 학습 단계에서 문제풀이의 결과를 선택적으로 일반화하여 저장함으로써 규칙의 수가 폭발적으로 증가하는 것을 막는다든지[15,16], 학습되는 규칙의 조건부의 표현성(expressiveness)에 제약을 가함으로써 지나치게 비대하여 match 비용이 많이 드는 규칙들을 제거해 버리기도 하며[17], 학습된 규칙들 중의 일부를 무작위적으로 선택하여 삭제해 버리는 식의 매우 단순한 방법으로 시스템의 성능을 만족스런 수준으로 되돌려 놓을 수 있음을 확인한 연구 결과도 있다[18].

3.3 Masking 효과

과거의 문제풀이 결과를 일반화 하여 학습하는 시스템은 새로운 문제가 주어 졌을 때 그에 대한 해답을 신속히 구할 수는 있지만, 새로 구한 해답의 질(solution quality)은 경우에 따라 학습 결과에 의존하지 않을 때보다 오히려 못할 수 있다. 학습된 지식을 사용함으로 인해, 그렇지 않을 경우 찾아 낼 수 있을지 모르는 보다 나은 대안을 탐색할 기회가 막혀 버리기 때문으로, 이런 현상을 학습의 masking 효과라 한다.

Masking 효과를 보다 구체적으로 살펴 보기 위하여 이하에서는 [19]의 내용을 일부 인용하되 Groundworld 문제의 예를 중심으로 설명한다. Groundworld는 2차원의 multi-agent 시뮬레이션 domain의 문제이다. 본 예에서는 침입자(envasion agent)와 감시자(persuit agent)의 두 agent가 은폐물인 벽들 사이에서 아리 저리 서로 좇고 좇기는 상황을 가정한다. 침입자의 임무는 출발점으로 부터 최종 목적지 까지 감시자에게 잡히지 않고 최대한 빨리 가는 것이다. 감시자의 임무는 물론 침입자를 잡는 일이다. 두 agent는 모두 제한된 시야(range of vision)를 가지고 있는데, 서로가 서로의 시야에 들어 오면 감시자는 좇기 시작하고 침입자는 벽 뒤로 도망가 숨어서 다시 최종 목적지 까지 갈 계획을 세우려 한다.

그림 5 (a)는 Groundworld에서 일어날 수 있는 상황의 한 예를 보여 주고 있다. 굵은 실선은 벽을 표시하며, 두 agent는 서로의 시야 안에 있다. 침입자는 잡히지 않기 위하여 그림에서 점선으로 표시된 것처럼 벽 뒤로 숨을 계획을 수립한다. 이 계획은 차후의 유사 상황에서 활용하기 위해 규칙의 형태로 학습 저장된다. 그림 5 (b)는 이 학습된 계획이 적용되어 숨을 장소를 정할 수 있는 하나의 유사 상황이다. 그러나, 이 학습된 규칙에 의존함으로 인해 침입자는 보다 가까운 은신처(그림에서 X로 표시)를 놓치고 만다. 만약 침입자가 학습된 규칙 없이 그림 5 (b)의 상황에 직면했다면 바로 그 가

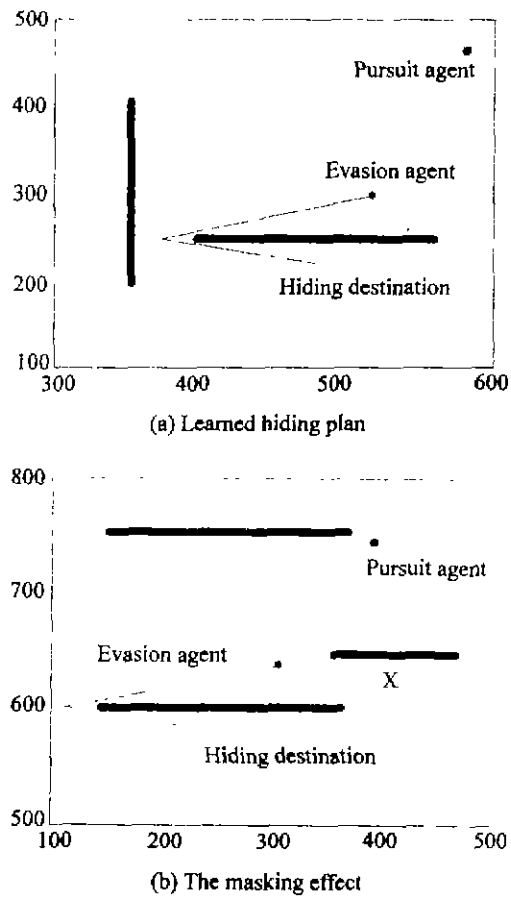


그림 5 Masking 효과를 보여주는 Groundworld 문제의 예

까운 은신처로 가는 길을 찾았을 수도 있다. 학습된 규칙 때문에 침입자가 좋지 못한 계획을 따르게 된 것이다. 그 좋지 못한 계획에 따라서도 성공적으로 숨을 수는 있지만 숨는데 상당한 시간이 걸리고 그 만큼 최종 목적지 까지 도달하는 시간이 지연된다.

Masking 효과에 의해 좋지 못한 계획이 만들어 지는 것이 항상 나쁘다고만 할 수는 없다. 예를 들어 실시간(real time) 상황에서는 저 품질의 해(low quality solution)라도 제한된 시간 내에 구해 지기만 한다면 받아 들일 수도 있을 것이다. 그러나, 그 외의 상황에서는 고 품질의 해가 훨씬 높은 우선 순위를 가질 것이고 그렇게 되면 masking 효과를 피할 수 있는 방법의 개발이 중요해 진다.

Masking 효과가 발생하는 것은 규칙을 학습할 때 고 품질의 해를 유도하는 데 필요한 관련 지식을 모두 확보하지 못한 때문이다. 이런 일이 생기는 이유는 예를 들어 필요 지식이 문제 풀이와 묵시적(implicit) 관계만 있다든지 아니면 그 지식을 모두 확보한다는 것 자체가 intractable하기 때문일 수 있다. 어떤 경우든, 학습된 규칙에는 적용시 고 품질의 해를 구할 수 있게 되는 상황에 대한 정확한 지식이 빠져 있게 된다. 따라서, 그렇게 학습된 규칙이 저 품질의 해를 유도하게 되는 상황에도 적용되는 것이다.

이는 분명히 overgenerality의 한 예라고 볼 수 있는 것이지만, 이 overgenerality는 단지 해의 품질이 좋으냐 나쁘냐에 관한 것이지 해 자체가 맞느냐 틀리느냐의 관점에서 본 것은 아니다. 다시 말해서 학습된 규칙이 적어도 실패를 유도하지는 않는다는 말이다. 그림 5 (b)의 예에서도 학습된 규칙이 보다 가까운 은신처를 놓치게 하는 masking 효과를 유발하기는 해도 숨는데 실패하게 한 것은 아니다.

실패 사례로 부터 학습된 규칙은 masking 효과를 유발하지 않는다. 실패 사례로 부터 학습된 규칙은 새로운 문제에서 유사한 실패 상황을 만나면 과거의 실패를 되풀이 하지 않게 해 주므로, 시스템은 실패하는 방향으로의 탐색을 피하고 신속히 문제의 해를 구할 수 있게 된다. 이 때 해의 품질이 좋고 나쁨에 그 규칙이 직접 관여되지는 않는다. 반면에, 성공 사례로 부터 학습된 규칙은 새로운 문제의 유사 상황에서 바로 성공할 수 있는 해를 제시하므로 그 해를 시스템이 받아 들일 경우 보다 나은 해를 탐색하려는 시도를 하지 않게되어 masking 효과가 발생하는 것이다.

Masking 효과에 대한 해결책으로는 학습시에 미리 고 품질의 해를 유도하는 상황에만 규칙이 적용될 수 있도록 필요한 모든 지식을 확보하는 방안이 제안된 바 있으나[20], 경우에 따라 규칙의 조건부가 지나치게 복잡해 져서 2절에서 설명한 이유로 utility가 떨어지게 되고, 일반적으로 복잡한 문제 domain에서는 그 모든 지식을 확보하는 것이 intractable하다. 따라서, [19]에서는 approximation과 filter의 개념을 도입하여 학습된 규칙이 사용되는 순간 그 규칙이 부적절한 경우에 대한 경고를 함으로써 시스템이 보다 나은 해를 구하도록 하고 있다.

4. 연구 동향

본 장에서는 이상의 설명에서 참조한 것들 외에 최근 보고된 주요 연구 결과들을 간략히 정리해 본다.

Utility 문제의 해결책으로 3.2절에서 소개한 PRODIGY의 방법은 학습된 규칙이 가져다 주는 탐색 시간 절감 효과를 그 규칙이 학습된 사례 하나만 보고 추정하였고, 규칙들 사이의 상호 작용이 개개 규칙의 사용 빈도에 미치는 영향을 무시하였다. 이로 인해, 효용 가치가 높은 규칙 집단을 확보하기가 어려웠다. [21]과 [22]는 이러한 한계를 극복하기 위하여 확률 및 통계적 접근 방법을 동원함으로써 보다 정확한 utility 값의 추정이 가능하게 하였다. 이들 방법에 의하면, 어느 정도 학습 과정의 overhead가 커지기는 하지만 PRODIGY에 비해 학습된 규칙에 의한 수행 시스템의 성능 향상이 훨씬 두드러짐을 실험적으로 확인할 수 있었다. [23]은 utility의 개념을 일반적으로 확장하여 EBL 뿐만 아니라 귀납적 학습에서도 공히 발생하는 문제라고 보았다. Decision tree나 neural network 학습과 같은 귀납적 학습에서 과도한 학

습으로 인해 규칙이 충분히 일반화 되지 못하고 학습에 사용된 사례 데이타에 지나치게 충실해 지게 되는 overfitting 문제도, 결국 시스템의 성능을 저하시킨다는 측면에서(이 경우는 속도가 아니라 규칙 적용시 error rate의 관점에서) EBL의 utility 문제와 같은 성격의 것이라고 해석하였다. 이러한 관점에서 [23]은 학습의 양을 조절하여 시스템의 성능 저하를 막을 수 있는 방안을 제시하고 있다.

복잡한 실 세계 domain에서 동작하는 계획 시스템은 완벽한 domain theory를 가질 수 없기 때문에 계획의 결과가 실 세계와 완전히 합치되지 않는다. [24]는 이 문제의 해결을 위해 permissive planning이라는 기법을 제시하고 있는데, 이에 의하면 실패의 경험을 분석하고 학습을 통하여 원래 주어진 domain axiom들과 결합하여 추론할 수 있게 하였다. 시스템의 지식을 보완하기 위한 또 다른 대책으로는, 실제 전문가의 문제 해결 과정, 특히 문제를 풀어나가는 단계의 순서를 관찰하여 중요한 정보를 학습하도록 하는 접근 방법이 [25]에 의해 제안 되었다. [26]은 귀납적인 neural network 학습과 EBL을 결합함으로써 순수한 귀납적 학습 보다 필요한 사례 데이타의 수를 훨씬 줄이면서도 domain theory가 가진 오류에 대해서 robust한 성능을 보일 수 있는 알고리즘을 제시하였다.

이 외에도 EBL을 적용한 주요 연구 결과로서, [27]은 부분 순서 계획(partial order planning)에서 EBL로 탐색 제어 규칙을 학습하는 방법을 제시하였고, [28]은 의사 결정 시스템에 관한 것으로 자신의 의사 결정 이유를 설명해야 하는 전투기 조종 시뮬레이션 프로그램이 학습을 통해 스스로 설명 능력을 향상시킬 수 있음을 보였다.

5. 결 론

EBL은 시스템이 스스로 이미 가지고 있는 지식, 즉 domain theory를 기반으로 새로 부딪히는 상황을 해석하여 규칙을 유도하고 기억함으로써 성능을 향상시킬 수 있게 하는 학습 방법이다. 주로 응용되는 대상으로는 시스템이 탐색시 가장 유망한 경로를 선택할 수 있게 해 주는 탐색 제어 지식의 학습과, 탐색의 여러 단계를 한꺼번에 뛰어 넘을 수 있게 해 주는 macro operator의 학습 등이 있다.

그러나, EBL을 주의 깊게 적용하지 않으면 시스템의 성능이 오히려 저하되는 결과를 초래할 수도 있다. EBL은 설명을 기반으로 하는 학습 방법인데 설명의 근거가 되는 domain theory가 완벽하지 못하다면 학습의 결과도 만족스럽지 못하게 된다. 현실적인 대규모의 문제들에 대하여 완벽한 domain theory를 갖추기는 어려우므로 불완전한 domain theory로 부터라도 plausible한 학습 결과를 이끌어 내는 방안이 연구되어야 할 것이며, domain theory 자체를 또 다른 학습을 통해 점차 보완해 나갈 수 있는 방법의 개발도 필요할 것이다. Utility 문제는 학습을 통해 규칙을 많이 확보하여 저장해 둔다고 반드시 시스템의 수행 속도가 빨라지지는 않는다는 점을 지적한 것이다. 문제 공간(problem space)의 탐색이 절약되는 것보다 그를 위해 규칙을 찾는데 더 많은 시간을 낭비해야 한다면 시스템의 수행 속도는 오히려 학습을 하기 전보다 느려지게 된다. 이 문제의 해결을 위해서는 규칙이 제공하는 혜택과 규칙을 찾는데 소요되는 비용을 모두 고려하는 방안들이 활발히 연구되고 있다. 비교적 최근에 지적되고 있는 EBL의 또 하나의 문제로는 학습 결과의 활용이 최적해의 탐색에 장애 요인으로 작용하게 된다는 masking 효과가 있다. 시스템이 직면한 어떤 상황에서 과거 유사 상황의 학습 결과를 적용한다면 그 문제의 해답을 신속히 구할 수는 있지만, 그 해답의 질은 학습 결과에 의존하지 않을 때보다 오히려 못할 수 있다. 학습 결과부터 사용함으로 인해, 그러지 않고 탐색을 수행하여 더 좋은 해답을 찾을 기회가 차단되는 것이다. Masking 효과에 대처하는 방안에 관한 연구는 아직 초기단계에 머물고 있다.

학습 능력은 시스템이 지능적 행동(intelligent behavior)을 하기 위해 기본적으로 갖추어야 할 요건 중의 하나임은 분명하다. 앞으로는, 귀납적 학습 방법과 EBL을 통합한 [26]의 연구 결과에서 보는 바와 같이 다양한 학습 방법을 상호 보완적으로 결합하는 방안에 관한 연구가

활발해 질 것으로 보이며, 더 나아가 궁극적으로는 현재까지 개발된 여러가지 학습 방법을 하나의 큰 테두리 내에서 설명할 수 있는 통일된 이론으로 정립해 가는 방향으로의 연구가 필요할 것으로 생각된다.

참 고 문 헌

[1] Dietterich, T. G., Learning at the Knowledge Level, Machine Learning, 1 : 287-316, 1986.

[2] Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T., *Explanation-Based* Generalization : a Unifying View, Machine Learning, 1 : 47-80, 1986.

[3] *DeJong*, G. F. and Mooney, R., Explanation-Based Learning : an Alternative View, Machine Learning, 1 : 145-176, 1986.

[4] Minton, S. and Carbonell, J. t., Strategies for Learning Search Control Rules : an Explanation-Based Approach, *Proceedings* of the Tenth International Joint Conference on Artificial Intelligence, pp. 228-235, Milan, Italy, 1987.

[5] Laird, J. E., Rosenbloom, P. S., and Newell, A., Chunking in Soar : The Anatomy of a General Learning Mechanism, Machine Learning, 1 : 11-46, 1986.

[6] Ryu, K. R. and Irani, K. B., Learning from Goal Interactions in Planning : Goal Stack Analysis and Generalization, *Proceedings of the Tenth National Conference* on Artificial Intelligence, pp. 401-407, San Jose, California, 1992.

[7] Sussman, G. J., A Computer Model of Skill Acquisition, American Elsevier, New York, 1975.

[8] Samuel, A. L., Some Studies in Machine Learning Using the Game of Checkers, *IBM Journal*, Vol. 3, No. 3, 1959.

[9] Mitchell, T. M., Utgoff, P. E., and Banerji, R., Learning by Experimentation : Acquiring and Refining Problem-Solving Heuristics, Machine Learning : An Artificial Intelligence Approach, Morgan Kaufmann, San Mateo, CA., 1983.

[10] Grefenstette, J. J., Credit Assignment in *Rule Discovery Systems Based on Genetic* Algorithms, Machine Learning, 3 : 225-245, 1988.

[11] Fikes, R. E., Hart, P. E., and Nilsson, N. J., Learning and Executing Generalized Robot Plans, Artificial Intelligence, 3 : 251-288, 1972.

[12] Shavlik, J. W. and Dietterich, T. G., Eds., *Readings in Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1990.

[13] Mooney, R. J., A General Explanation-Based Learning Mechanism and Ite Application to Narrative Understanding, Pitman, London, 1990.

[14] Minton, S., Quantitative Results Concerning the Utility of Explanation-Based Learning, *Artificial Intelligence*, 42 : 363-392, 1990.

[15] Minton, S., Selectively Generalizing Plans for Problem-Solving, Proceedings of the *Ninth International Joint* Conference on Artificial Intelligence, 1985.

[16] Iba, G. A., A Heuristic Approach to the Discovery of Macro Opeartors, Machine Learning, 3 : 285-317, 1989.

[17] Tambe, M. and Rosenbloom, P., *Eliminating* Expensive Chunks by Restricting Expressiveness, Proceedings of the Eleventh *International Joint Conference* on Artificial Intelligence, pp. 731-737, Detroit, MI, 1989.

[18] Markovitch, S. and Scott, P. D., The Role of Forgetting in Learning, Proceedings of the Fifth International *Conference on* Machine Learning, pp. 459-465, Ann Arbor, MI, 1988.

[19] *Tambe*, M. and Rosenbloom, P., On the Masking Effect, Proceedings of the Eleventh National Conference on Artificial Intelligence, pp. 526-533, Washington, DC, 1993.

[20] Clark, P. and Holte, R., Lazy Partial Evaluation : An Integration of Explanation-Based Generalization and Partial Evaluation, Proceedings of the International *Conference* on Machine Learning, pp. 82-91, 1992.

[21] Gratch, J. and DeJong, G., COMPOSER : A Probabilistic Solution to the Utility Problem in Speed-up Learning, Proceedings of the Tenth National Conference on Artificial Intelligence, pp. 235-240, San Jose, California, 1992.

[22] Greiner, R. and Jurisica, I., A Statistical Approach to Solving the EBL Utility Problem, Proceedings of the Tenth National Conference on Artificial Intelligence, pp. 241-248, San Jose, California, 1992.

[23] Holder, L. B., Empirical Analysis of the General Utility Problem in Machine Learning, Proceedings of the Tenth National Conference on Artificial Intelligence, pp. 249-254, San Jose, California, 1992.

[24] DeJong, G. and Bennett, S., Permissive Planning : A Machine Learning Approach to Linking Internal and External Worlds, Proceedings of the Eleventh National Conference on Artificial Intelligence, pp. 508-513, Washington, DC, 1993.

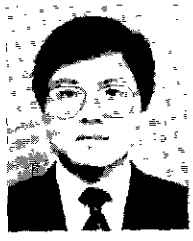
[25] Donoho, S. K. and Wilkins, D. C., Exploiting the Ordering of Observed Problem-solving Steps for Knowledge Base Refinement : an Apprenticeship Approach, Proceedings of the Twelfth National Conference on Artificial Intelligence, pp. 569-575, Seattle, Washington, 1994.

[26] Thrun, S. B. and Mitchell, T. M., Integrating Inductive Neural Network Learning and Explanation-Based Learning, Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, pp. 930-936, Chambery, France, 1993.

[27] Katukam, S. and Kambhampati, S., Learning Explanation-Based Search Control Rules for Partial Order Planning, Proceedings of the Twelfth National Conference on Artificial Intelligence, pp. 582-587, Seattle, Washington, 1994.

[28] Johnson, W. L., Agents that Learn to Explain Themselves, Proceedings of the Twelfth National Conference on Artificial Intelligence, pp. 1257-1263, Seattle, Washington, 1994.

류 광 렬



1979 서울대학교 전자공학과 학사
1981 서울대학교 전자공학과 석사
1983~1984 충북대학교 전자계산기공학과 전임강사
1992 University of Michigan 컴퓨터 공학 박사
1992~1993 Ford Motor Company, Scientific Research Lab. 선임연구원
1993~현재 부산대학교 컴퓨터공학과, 조교수
관심분야 : 인공지능, 기계학습, 사례기반추론, Genetic Algorithms, 최적화, 인공지능의 생산시스템 응용 등임