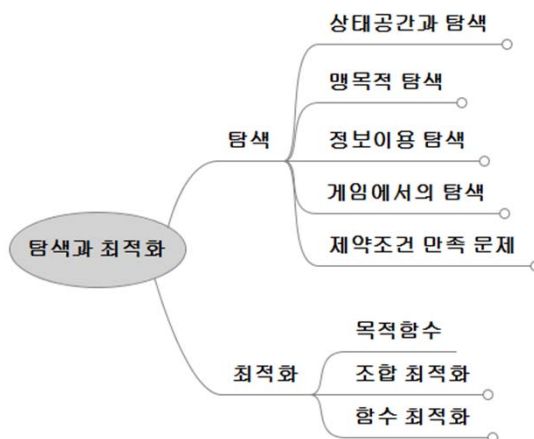


탐색과 최적화 - I

충북대학교 소프트웨어학과
이건명



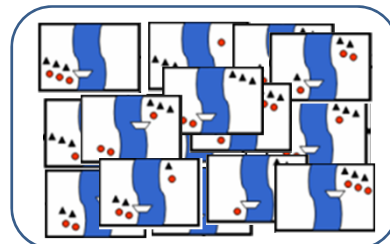
1. 상태 공간과 탐색

- ❖ **탐색 (探索, search)**
 - 문제의 **해(solution)**이 될 수 있는 것들의 **집합을 공간(space)**으로 간주하고, 문제에 대한 **최적의 해**를 찾기 위해 공간을 **체계적으로 찾아 보는 것**
- ❖ **탐색문제의 예**
 - 선교사-식인종 강건너기 문제
 - 틱-택-토(tic-tac-toe)
 - 8-퍼즐 문제
 - 순회 판매자 문제 (traveling salesperson problem, TSP)
 - 해(解, solution)
 - 일련의 동작으로 구성되거나 하나의 상태로 구성



상태 공간과 탐색

- ❖ **상태(state)**
 - 특정 시점에 문제의 세계가 처해 있는 모습
- ❖ **세계(world)**
 - 문제에 포함된 대상들과 이들의 상황을 포괄적으로 지칭
- ❖ **상태 공간(state space)**
 - 문제 해결 과정에서 초기 상태에서부터 도달할 수 있는 모든 상태들의 집합
 - 문제의 해가 될 가능성이 있는 모든 상태들의 집합
- 초기 상태(initial state)
 - 문제가 주어진 시점의 시작 상태
- 목표 상태(goal state)
 - 문제에서 원하는 최종 상태

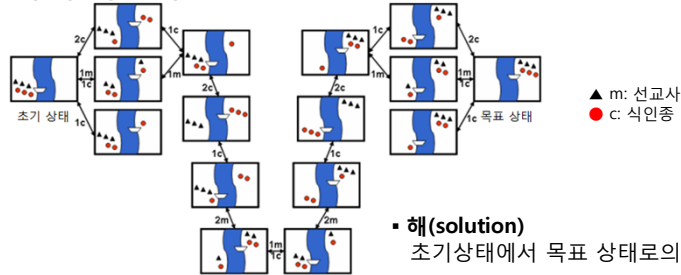


상태 공간과 탐색

❖ 상태 공간 그래프(state space graph)

- 상태공간에서 각 행동에 따른 상태의 변화를 나타낸 그래프
 - 노드 : 상태
 - 링크 : 행동

▪ 선교사-식인종 문제



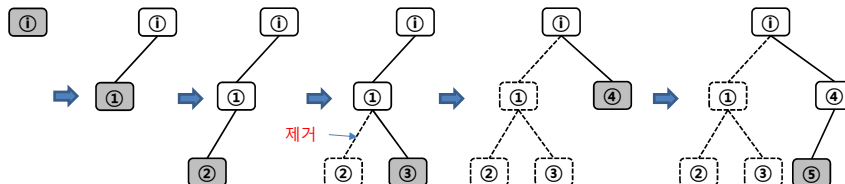
- 일반적인 문제에서는 상태공간이 매우 큼
 - 미리 상태 공간 그래프를 만들기 어려움
 - 탐색과정에서 그래프 생성

image : Gillian Mosley

2. 맹목적 탐색

❖ 맹목적 탐색(blind search)

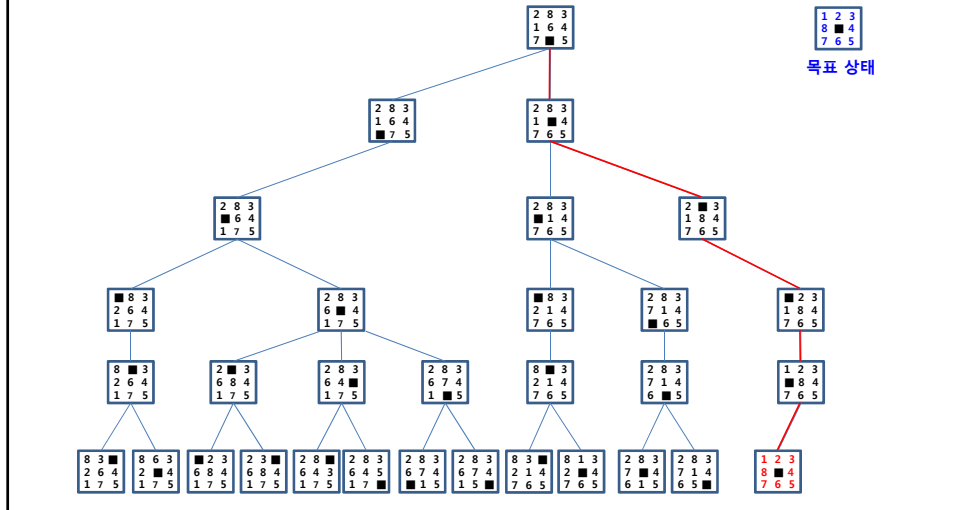
- 정해진 순서에 따라 상태 공간 그래프를 점차 생성해 가면서 해를 탐색하는 방법
- 깊이 우선 탐색(depth-first search, DFS)
 - 초기 노드에서 시작하여 깊이 방향으로 탐색
 - 목표 노드에 도달하면 종료
 - 더 이상 진행할 수 없으면, 백트래킹(backtracking, 되짚어가기)
 - 방문한 노드는 재방문하지 않음



맹목적 탐색

❖ 8-퍼즐 문제의 깊이 우선 탐색 트리

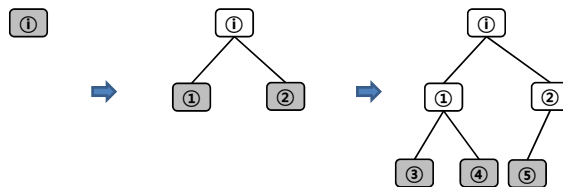
- 루트 노드에서 현재 노드까지의 경로 하나만 유지



맹목적 탐색

❖ 너비 우선 탐색(breadth-first search, BFS)

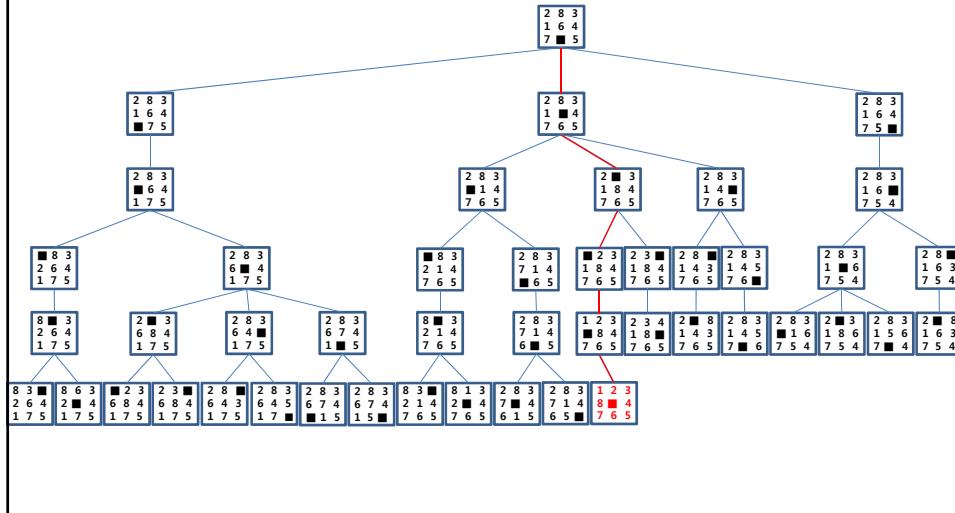
- 초기 노드에서 시작하여 모든 자식 노드를 확장하여 생성
- 목표 노드가 없으면 단말노드에서 다시 자식 노드 확장



맹목적 탐색

❖ 8-퍼즐 문제의 너비 우선 탐색 트리

- 전체 트리를 메모리에서 관리



맹목적 탐색

❖ 반복적 깊이심화 탐색(iterative-deepening search)

- 깊이 한계가 있는 깊이 우선 탐색을 반복적으로 적용

ⓐ

깊이 0: ⓐ

맹목적 탐색

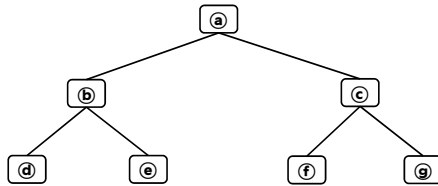
- ❖ 반복적 깊이심화 탐색(iterative-deepening search)
 - 깊이 한계가 있는 깊이 우선 탐색을 반복적으로 적용



깊이 0: a
깊이 1: a,b,c

맹목적 탐색

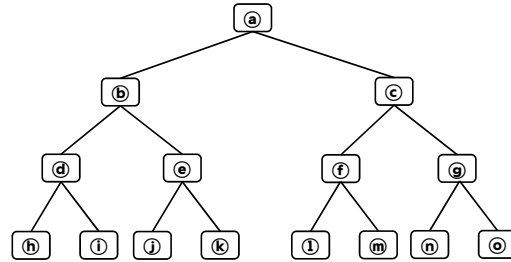
- ❖ 반복적 깊이심화 탐색(iterative-deepening search)
 - 깊이 한계가 있는 깊이 우선 탐색을 반복적으로 적용



깊이 0: a
깊이 1: a,b,c
깊이 2: a,b,d,e,c,f,g

맹목적 탐색

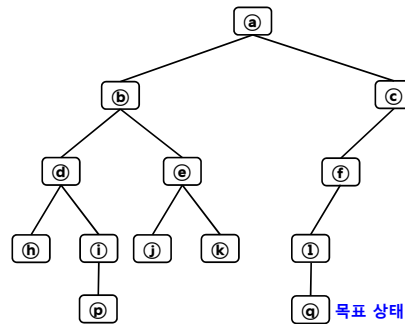
- ❖ 반복적 깊이심화 탐색(iterative-deepening search)
 - 깊이 한계가 있는 깊이 우선 탐색을 반복적으로 적용



깊이 0: a
 깊이 1: a, b, c
 깊이 2: a, b, d, e, c, f, g
 깊이 3: a, b, d, h, i, e, j, k, c, f, l, m, g, n, o

맹목적 탐색

- ❖ 반복적 깊이심화 탐색(iterative-deepening search)
 - 깊이 한계가 있는 깊이 우선 탐색을 반복적으로 적용



깊이 0: a
 깊이 1: a, b, c
 깊이 2: a, b, d, e, c, f, g
 깊이 3: a, b, d, h, i, e, j, k, c, f, l, m, g, n, o
 깊이 4: a, b, d, h, i, p, e, j, k, c, f, l, q

맹목적 탐색

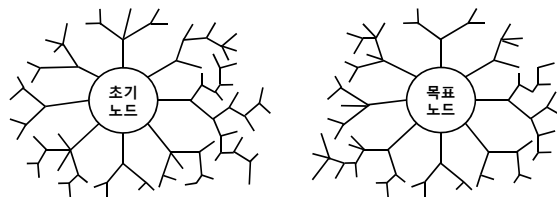
❖ 맹목적 탐색 방법의 비교

- 깊이 우선 탐색
 - 메모리 공간 사용 효율적
 - 최단 경로 해 탐색 보장 불가
- 너비 우선 탐색
 - 최단 경로 해 탐색 보장
 - 메모리 공간 사용 비효율
- 반복적 깊이심화 탐색
 - 최단 경로 해 보장
 - 메모리 공간 사용 효율적
 - 반복적인 깊이 우선 탐색에 따른 비효율성
 - 실제 비용이 크게 늘지 않음
 - 각 노드가 10개의 자식노드를 가질 때, 너비 우선 탐색 대비 약 11%정도 추가 노드 생성
 - 맹목적 탐색 적용시 우선 고려 대상

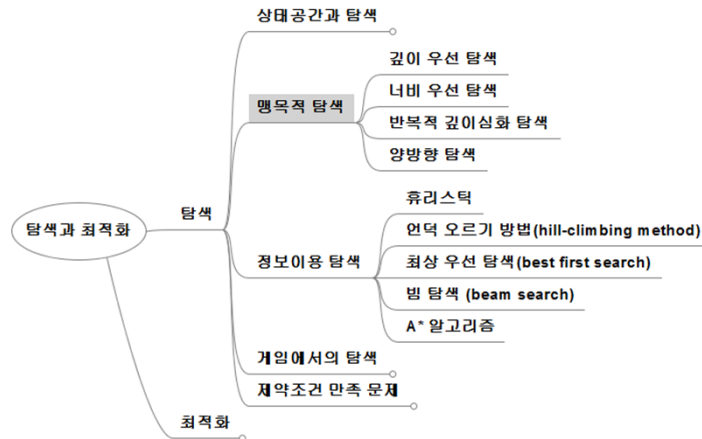
맹목적 탐색

❖ 양방향 탐색(bidirectional search)

- 초기 노드와 목표 노드에서 동시에 너비 우선 탐색을 진행
- 중간에 만나도록 하여 초기 노드에서 목표 노드로의 최단 경로를 찾는 방법



맹목적 탐색



3. 정보이용 탐색

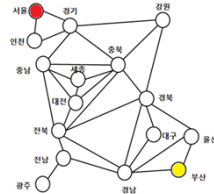
- ❖ 정보이용 탐색(informed search)
 - 휴리스틱 탐색(heuristic search)
 - 언덕 오르기 방법, 최상 우선 탐색, 빔 탐색, A* 알고리즘 등
 - **휴리스틱**(heuristic)
 - 그리스어 Εὐρίσκω (Eurisko, 찾다, 발견하다)
 - 시간이나 정보가 불충분하여 합리적인 판단을 할 수 없거나, 굳이 체계적이고 합리적인 판단을 할 필요가 없는 상황에서 **신속하게 어림짐작하는 것**
 - 예.
 - 최단 경로 문제에서 목적지까지 남은 거리
 - » 현재 위치에서 목적지(목표 상태)까지 지도상의 직선 거리

정보이용 탐색

❖ 휴리스틱 비용 추정의 예

▪ 최단경로 문제

- 현재 위치에서 목적지까지 직선 거리



▪ 8-퍼즐 문제

- 제자리에 있지 않는 타일의 개수

2	8	3
1	6	4
7		5

현재 상태

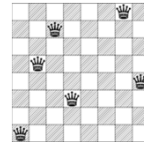
1	2	3
1	4	
7	6	5

목표 상태

추정비용 : 4

▪ 8-퀸 문제

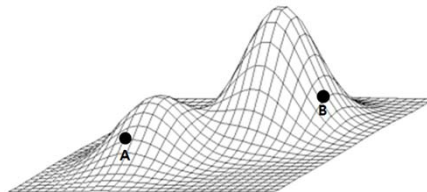
- 충돌하는 회수



정보이용 탐색

❖ 언덕 오르기 방법(hill climbing method)

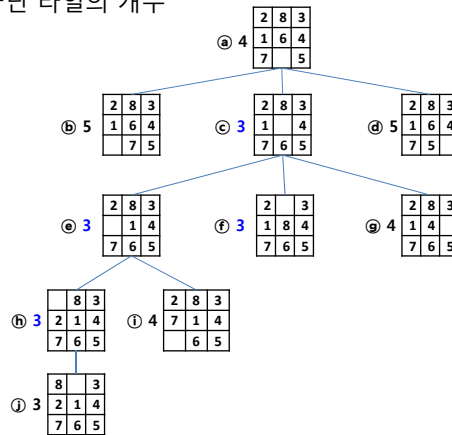
- 지역 탐색(local search), 휴리스틱 탐색(heuristic search)
- 현재 노드에서 휴리스틱에 의한 평가값이 가장 좋은 이웃 노드 하나를 확장해 가는 탐색 방법
- 국소 최적해(local optimal solution)에 빠질 가능성



정보이용 탐색

❖ 최상 우선 탐색(best-first search)

- 확장 중인 노드들 중에서 목표 노드까지 남은 거리가 가장 짧은 노드를 확장하여 탐색
- 남은 거리를 정확히 알 수 없으므로 휴리스틱 사용
 - 제자리가 아닌 타일의 개수



정보이용 탐색

❖ 빔 탐색(beam search)

- 휴리스틱에 의한 평가값이 우수한 일정 개수의 확장 가능한 노드만을 메모리에 관리하면서 최상 우선 탐색을 적용

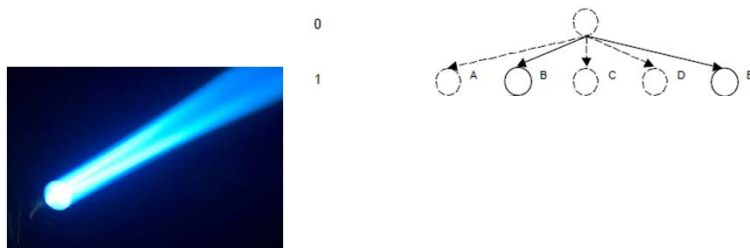


Image : america.pink

정보이용 탐색

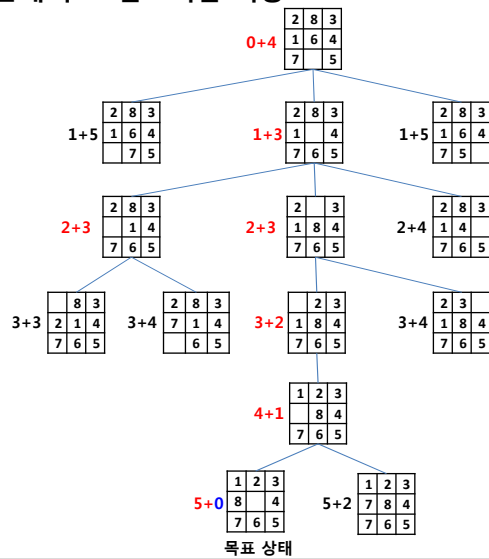
❖ A* 알고리즘

- 추정한 전체 비용 $\hat{f}(n)$ 을 최소로 하는 노드를 확장해 가는 방법
- $f(n)$: 노드 n 을 경유하는 전체 비용
 - 현재 노드 n 까지 이미 투입된 비용 $g(n)$ 과 목표 노드까지의 남은 비용 $h(n)$ 의 합
 - $f(n) = g(n) + h(n)$
- $h(n)$: 남은 비용의 정확한 예측 불가
 - $\hat{h}(n)$: $h(n)$ 에 대응하는 휴리스틱 함수(heuristic function)
- $\hat{f}(n)$: 노드 n 을 경유하는 추정 전체 비용
 - $\hat{f}(n) = g(n) + \hat{h}(n)$

정보이용 탐색

❖ 8-퍼즐 문제의 A* 알고리즘 적용

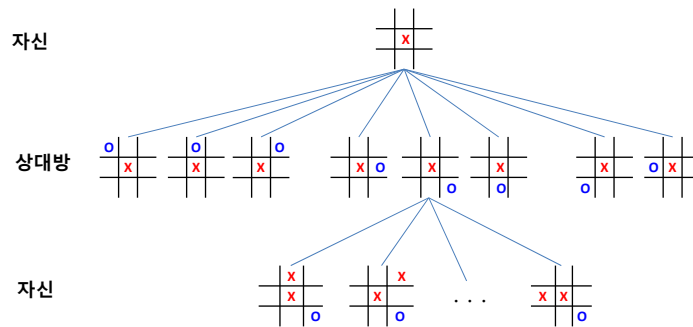
$$f(n) = g(n) + h(n)$$



4. 게임에서의 탐색

❖ 게임 트리(game tree)

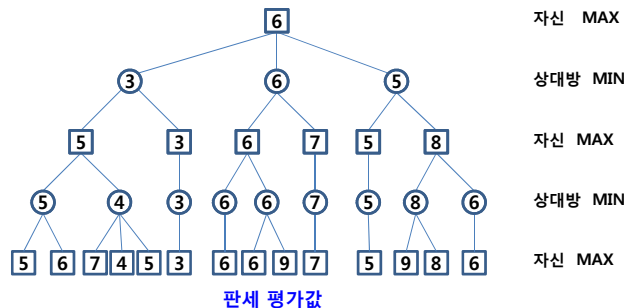
- **상대가 있는 게임**에서 자신과 상대방의 가능한 게임 상태를 나타낸 트리
 - 틱-택-톡(tic-tac-toe), 바둑, 장기, 체스 등
- 게임의 결과는 마지막에 결정
- 많은 수(**lookahead**)를 볼 수록 유리



게임에서의 탐색

❖ mini-max 알고리즘(mini-max algorithm)

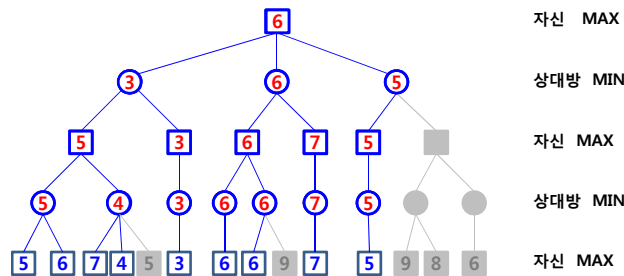
- **MAX 노드**
 - 자신에 해당하는 노드로 자기에게 유리한 최대값 선택
- **MIN 노드**
 - 상대방에 해당하는 노드로 최소값 선택
- 단말 노드부터 위로 올라가면서 최소(**minimum**)-최대(**maximum**) 연산을 반복하여 자신이 선택할 수 있는 방법 중 가장 좋은 것은 값을 결정



게임에서의 탐색

❖ α - β 가지치기 (pruning)

- 검토해 볼 필요가 없는 부분을 탐색하지 않도록 하는 기법
- 깊이 우선 탐색으로 제한 깊이까지 탐색을 하면서, MAX 노드와 MIN 노드의 값 결정
 - α -자르기(cut-off) : MIN 노드의 현재값이 부모노드의 현재 값보다 같거나 작으면, 나머지 자식 노드 탐색 중지
 - β -자르기 : MAX 노드의 현재값이 부모노드의 현재 값보다 같거나 크면, 나머지 자식 노드 탐색 중지

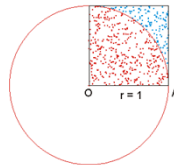


간단한 형태의 α - β 가지치기 예

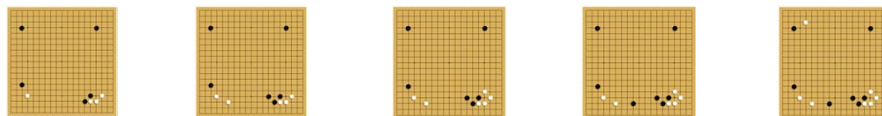
게임에서의 탐색

❖ 몬테카를로 시뮬레이션 (Monte Carlo Simulation)

- 특정 확률 분포로부터 무작위 표본(random sample)을 생성하고,
- 이 표본에 따라 행동을 하는 과정을 반복하여 결과를 확인하고,
- 이러한 결과확인 과정을 반복하여 최종 결정을 하는 것



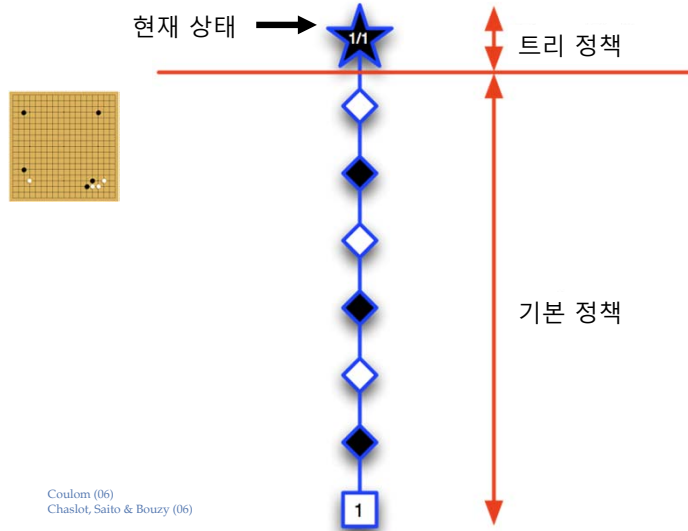
$$\frac{\text{원 안의 샘플 개수}}{\text{전체 샘플의 개수}} \rightarrow \frac{\pi}{4}$$



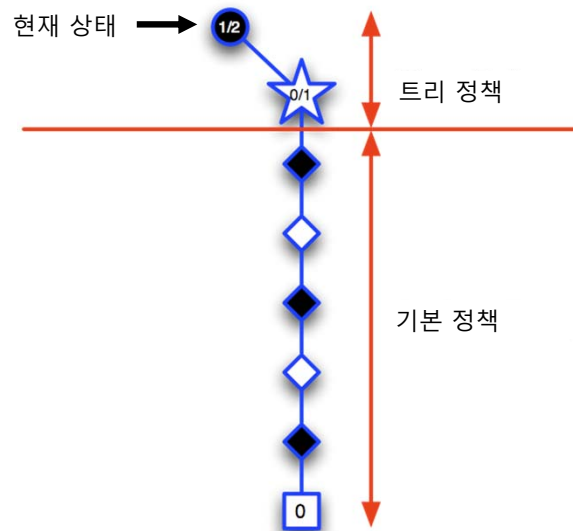
images: www.fansshare.com, www.casinonewsdaily.com

몬테카를로 트리 검색

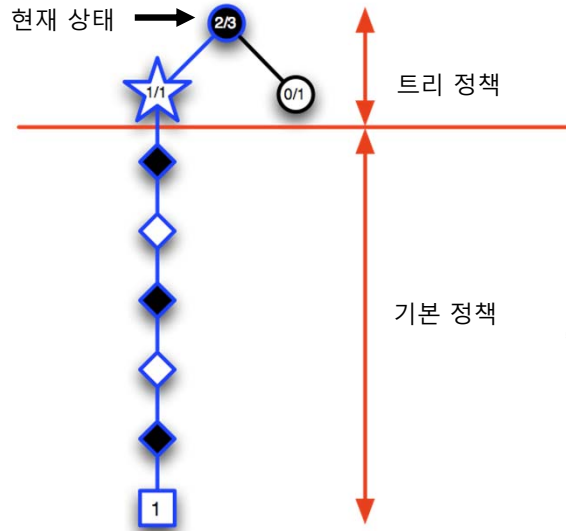
❖ 몬테카를로 트리 탐색(Monte Carlo Tree Search, MCTS)



몬테카를로 트리 검색

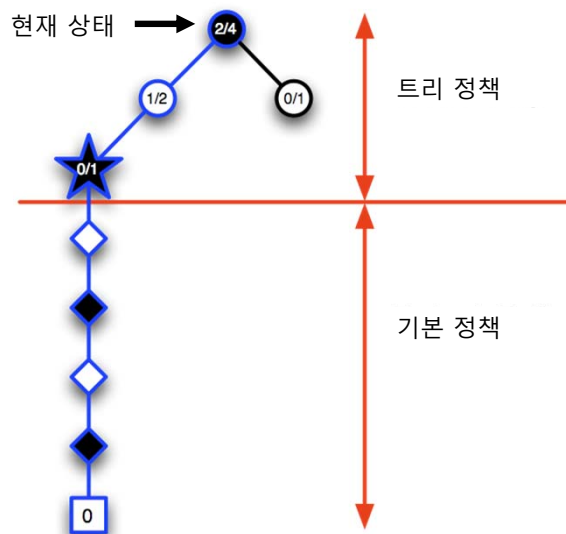


몬테카를로 트리 검색



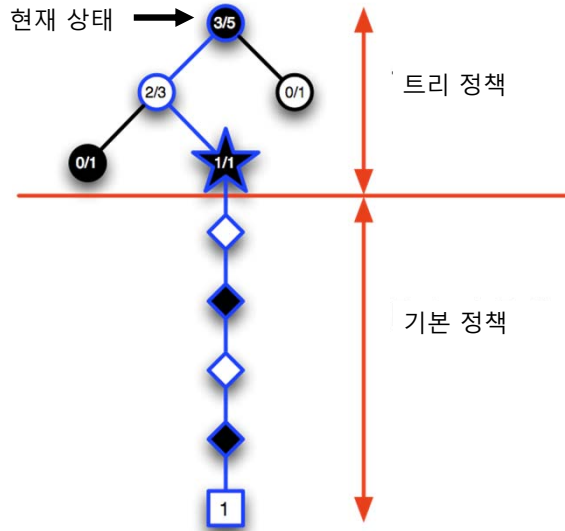
Slide from Sylvain Gelly

몬테카를로 트리 검색



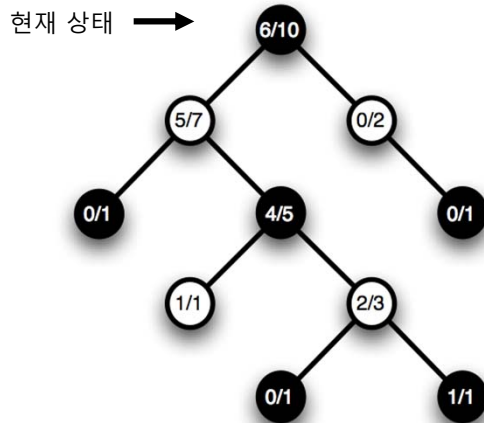
Slide from Sylvain Gelly

몬테카를로 트리 검색



Slide from Sylvain Gelly

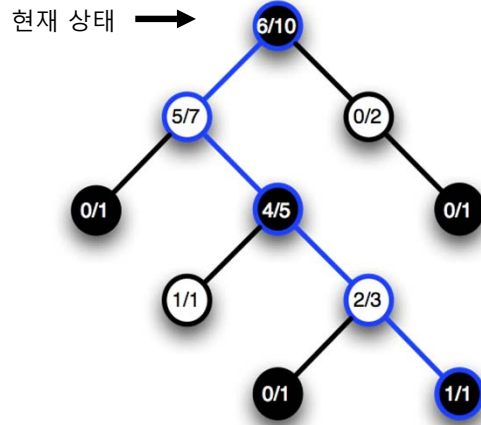
몬테카를로 트리 검색



Kocsis & Szepesvari (06)

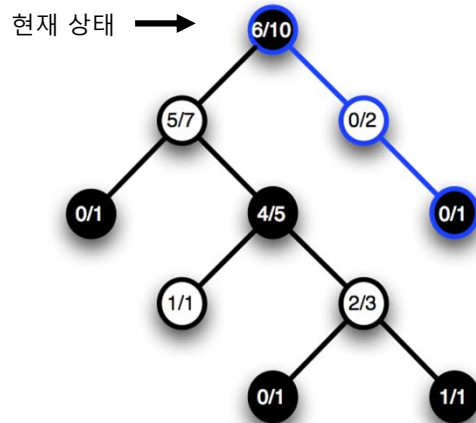
Slide from Sylvain Gelly

몬테카를로 트리 검색



Slide from Sylvain Gelly

몬테카를로 트리 검색

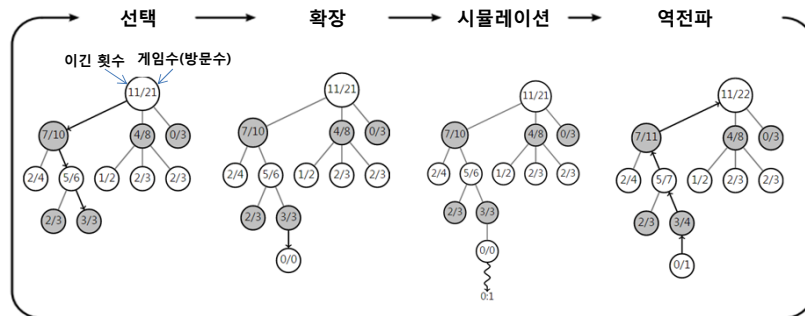


Slide from Sylvain Gelly

게임에서의 탐색

❖ 몬테카를로 트리 탐색(Monte Carlo Tree Search, MCTS)

- 탐색 공간(search space)을 무작위 표본추출(random sampling)을 하면서, 탐색트리를 확장하여 가장 좋아 보이는 것을 선택하는 휴리스틱 탐색 방법
- 4개 단계를 반복하여 시간이 허용하는 동안 트리 확장 및 시뮬레이션
 - 선택(selection)
 - 확장(expansion)
 - 시뮬레이션(simulation) : 몬테카를로 시뮬레이션
 - 역전파(back propagation)



게임에서의 탐색

❖ 몬테카를로 트리 탐색 - cont.

- 선택(selection) : 트리 정책(tree policy) 적용
 - 루트노드에서 시작
 - 정책에 따라 자식 노드를 선택하여 단말노드까지 내려 감
 - 승률과 노드 방문횟수 고려하여 선택
 - UCB(Upper Confidence Bound) 정책 : UCB가 큰 것 선택

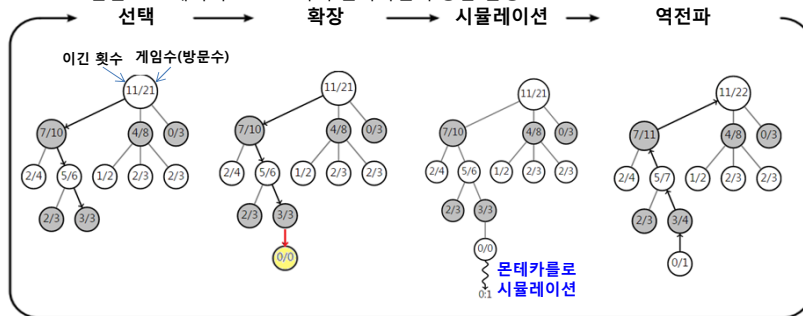
$$\frac{Q(v')}{N(v')} + c \sqrt{\frac{2 \ln N(v)}{N(v')}} \quad \begin{array}{l} v: \text{부모노드}, v': \text{자식노드} \\ N(v): \text{방문 횟수} \\ Q(v): \text{점수 (이진 횟수)} \end{array}$$



게임에서의 탐색

❖ 몬테카를로 트리 탐색 - cont.

- **확장(expansion)**
 - 단말노드에서 트리 정책에 따라 노드 추가
 - 예. 일정 횟수이상 시도된 수(move)가 있으면 해당 수에 대한 노드 추가
- **시뮬레이션(simulation)**
 - 기본 정책(default policy)에 의한 몬테카를로 시뮬레이션 적용
 - 무작위 선택(random moves) 또는 약간 똑똑한 방법으로 게임 끝날 때까지 진행
- **역전파(backpropagation)**
 - 단말 노드에서 루트 노드까지 올라가면서 승점 반영

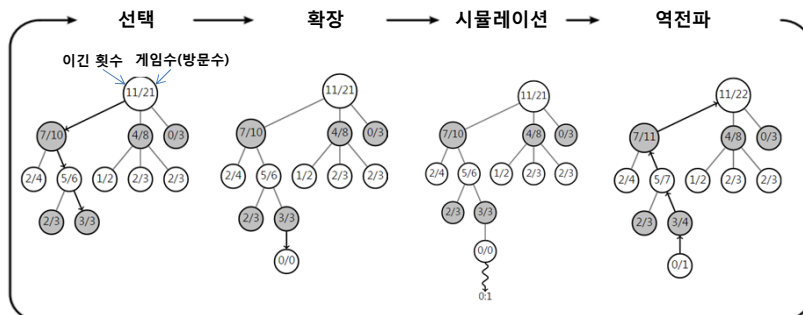


몬테카를로 트리 검색

❖ 몬테카를로 트리 탐색 - cont.

- **동작 선택 방법**
 - 가장 승률이 높은 루트의 자식 노드 선택
 - 가장 빈번하게 방문한 루트의 자식 노드 선택
 - 승률과 빈도가 가장 큰 루트의 자식 노드 선택
없으면, 조건을 만족하는 것이 나올 때까지 탐색 반복
 - 자식 노드의 **confidence bound값의 최소값이 가장 큰**, 루트의 자식 노드 선택

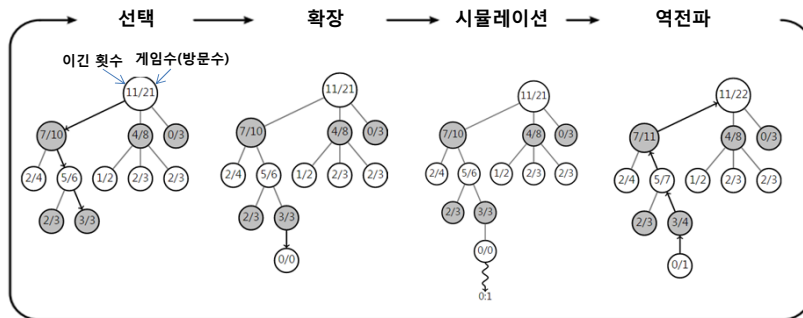
$$\frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v)'}}$$



몬테카를로 트리 검색

❖ 몬테카를로 트리 검색 - cont.

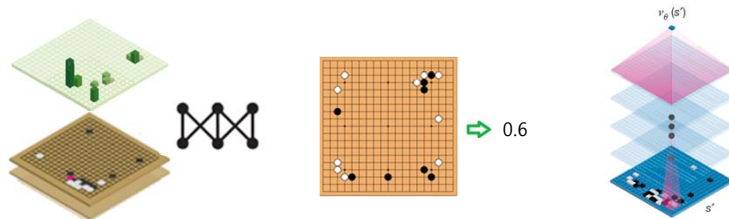
- 판의 형세판단을 위해 휴리스틱을 사용하는 대신, 가능한 많은 수의 몬테카를로 시뮬레이션 수행
- 일정 조건을 만족하는 부분은 트리로 구성하고, 나머지 부분은 몬테카를로 시뮬레이션
 - 가능성이 높은 수(move)들에 대해서 노드를 생성하여 트리의 탐색 폭을 줄이고, 트리 깊이를 늘리지 않기 위해 몬테카를로 시뮬레이션을 적용
- 탐색 공간 축소



알파고의 탐색

❖ 알파고의 몬테카를로 트리 검색

- 바둑판 형세 판단을 위한 한가지 방법으로 몬테카를로 트리 검색 사용
- 무작위로 바둑을 두는 것이 아니라, 프로 바둑기사들을 기보를 학습한 확장 정책망(rollout policy network)이라는 간단한 계산모델 사용



정책망: 가능한 착수(着手)들에 대한 선호 확률분포

가치망: 바둑판의 형세 값을 계산하는 계산모델

- 확률에 따라 착수를 하여 몬테카를로 시뮬레이션을 반복하여 해당 바둑판에 대한 형세판단값 계산
- 별도로 학습된 딥러닝 신경망인 가치망(value network)을 사용하여 형세판단값을 계산하여 함께 사용

Image : Nature

알파고의 탐색

❖ 알파고의 몬테카를로 트리 검색

- 많은 수의 몬테카를로 시뮬레이션과 딥러닝 모델의 신속한 계산을 위해 다수의 CPU와 GPU를 이용한 분산처리

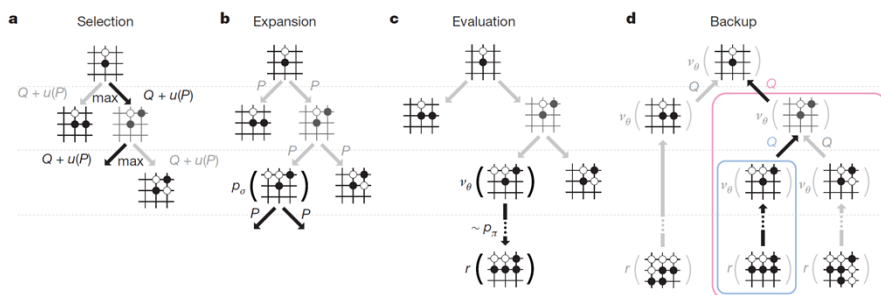


Image : Nature

게임에서의 탐색

