Chapter 8. Planning and Learning with Tabular Methods

강인호

https://jay.tech.blog/2016/12/29/planning-and-learning-with-tabular-methods/

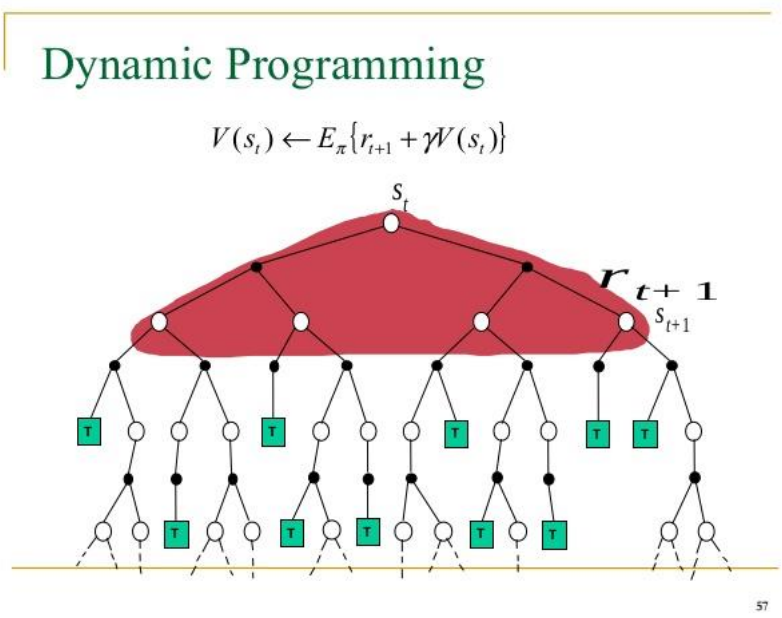
https://jay.tech.blog/2017/01/01/heuristic-search-mcts/

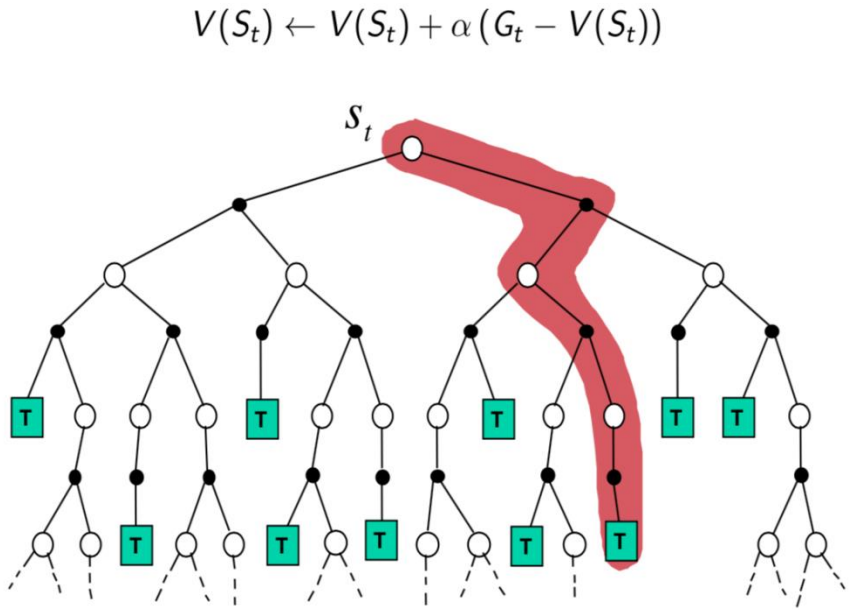
들어가기 전에

- Model based -> planning
  - Dynamic programming, Heuristic search
- Model Free -> learning
  - Monte Carlo, Temporal difference

8.1 Models and Planning

- Distribution models -> DP에 사용
- Sample models -> 5장의 블랙잭

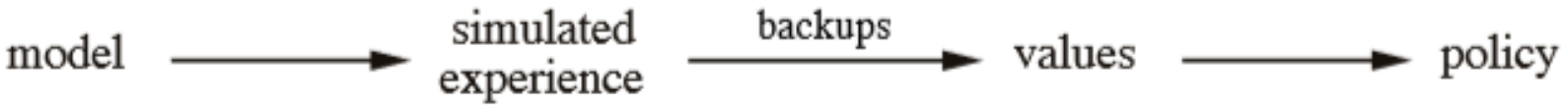




8.1 Models and Planning

Planning의 2가지 접근법:

1. State-space planning -> 이걸 다룸
2. Plan-space planning



Random-sample one-step tabular Q-planning

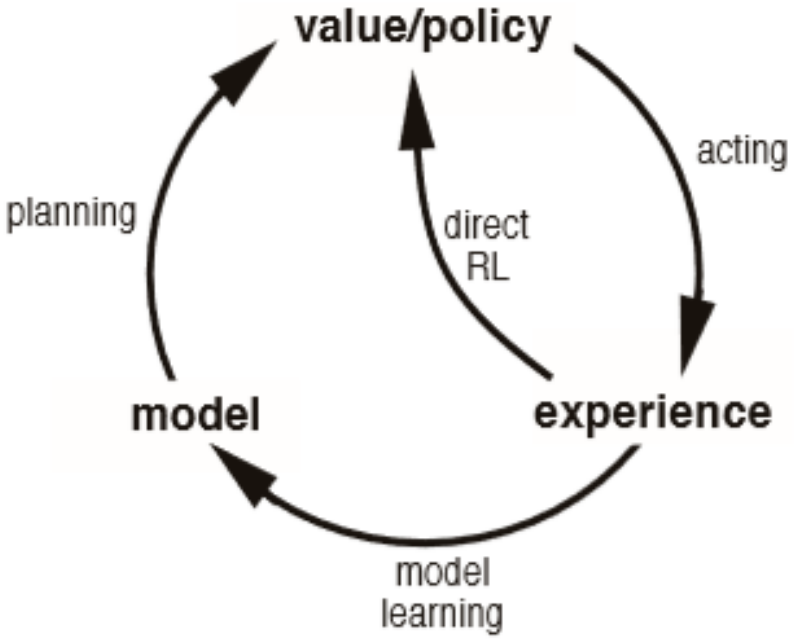
Loop forever:

1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain
   a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
   $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

8.2 Dyna: Integrated Planning, Acting, and Learning

Planning agent에서 real experience의 역할

1. 모델을 개선할 때 사용 : 환경에 더 잘 맞는 모델을 만들기 위함
   -> Model Learning
2. 강화학습 방법을 사용 : value function과 policy를 개선
   -> Direct Reinforcement learning



장단점

Indirect Method
: 환경과의 상호작용량이 적어도 더 나은 policy를 얻을 수 있음

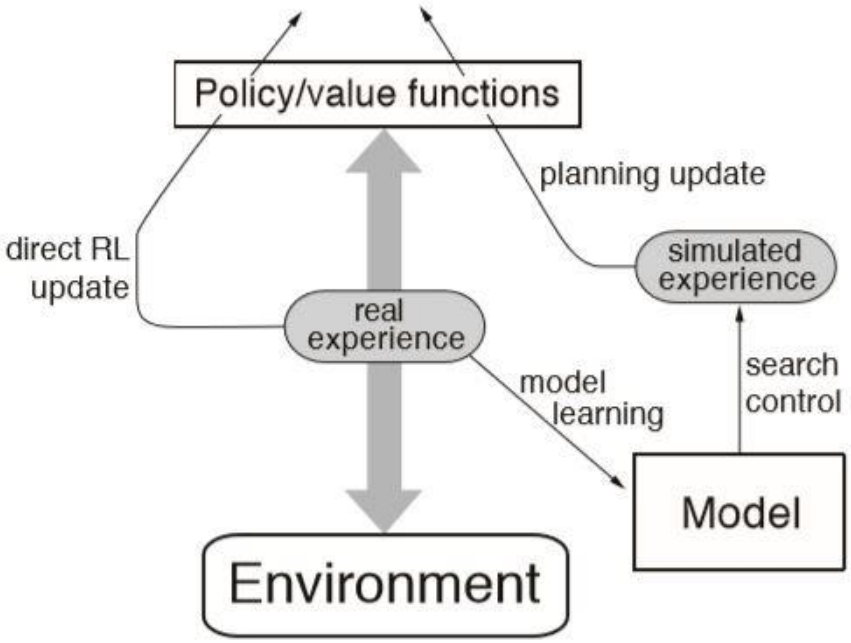
Direct Method
: 간단하며 Model의 영향을 받지 않음

8.2 Dyna: Integrated Planning, Acting, and Learning

Dyna-Q Learning

: planning, action, model-learning, direct RL이 연속적으로 일어남

-> planning method는 앞서 말한 Q planning



8.2 Dyna: Integrated Planning, Acting, and Learning

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \varepsilon$-greedy(S,Q)
- (c) Take action A; observe resultant reward, R, and state, S'
- (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
- (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Loop repeat n times:
  - $S \leftarrow$ random previously observed state
  - $A \leftarrow$ random action previously taken in S
  - $R, S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Step d: Direct Reinforcement Learning
Step e: Model-learning
Step f: planning
만약 e, f가 빠진다면 one-step tabular Q-learning과 동일한 알고리즘

8.3 When the Model Is Wrong

Real experience로 Model을 만들었을 경우

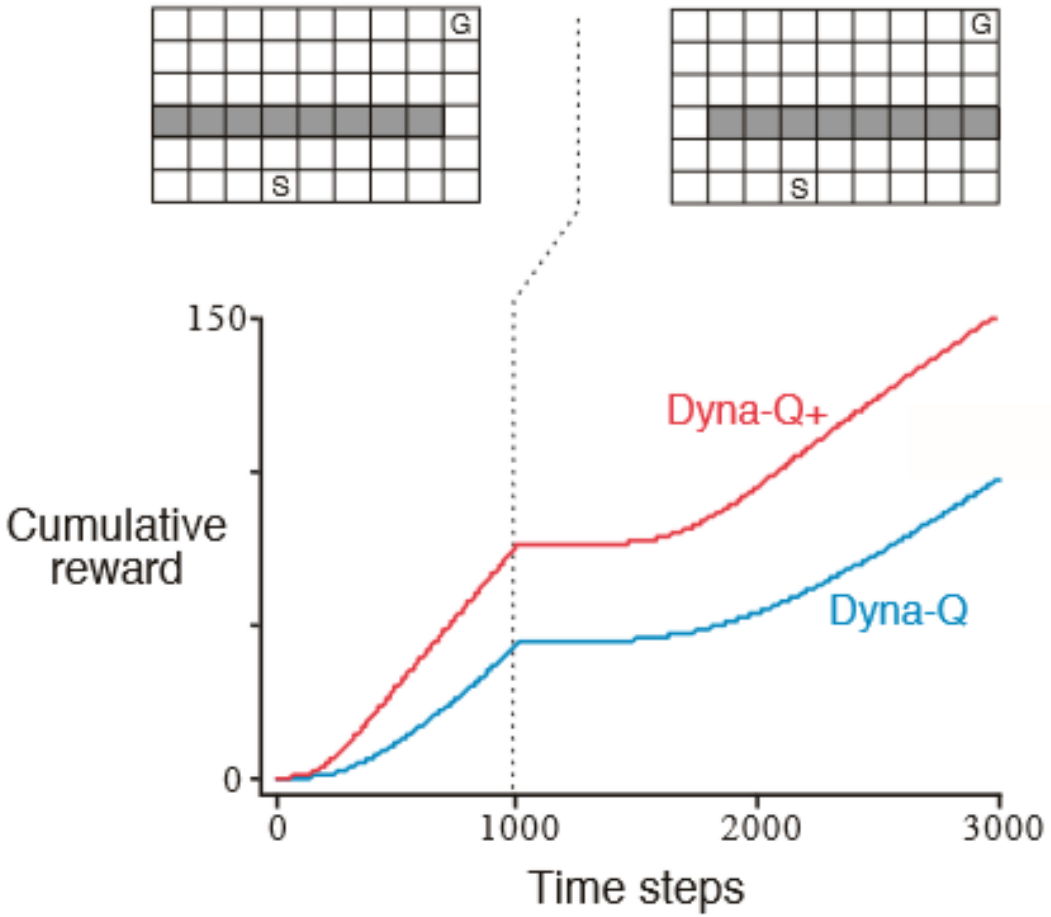
1. Environment가 stochastic하다.

2. 한정된 sample만 관찰되었다.

3. Model이 function approximation을 사용하였기에 완벽하지 않게 generalized되었다.

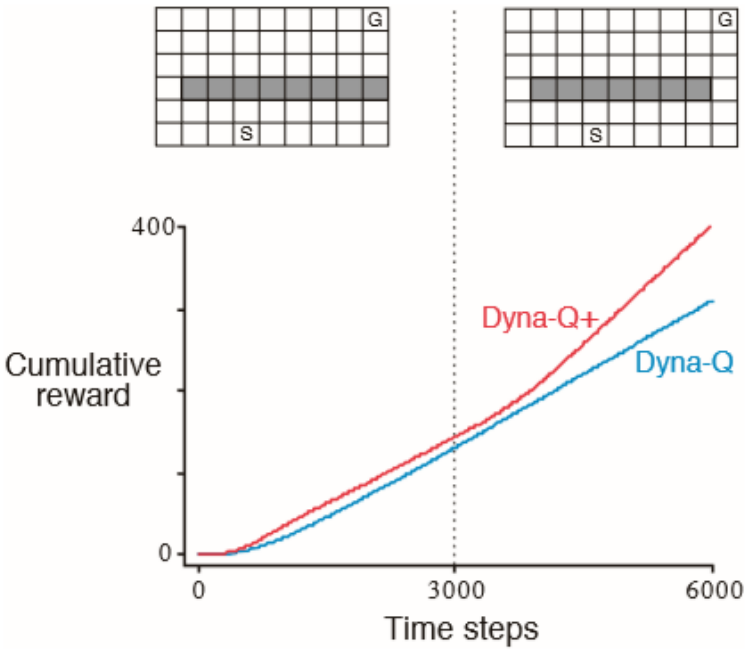
8.3 When the Model Is Wrong

Example1. Block Maze



8.3 When the Model Is Wrong

Example2. Shortcut Maze



Exploration: model을 더 개선시키려는 action 선택

Exploitation: 현재의 model에서 optimal한 action을 선택

Dyna-Q+는 짧은 길 찾는 방법을 Heuristic을 사용

8.4 Prioritized Sweeping

Backward focusing:
Value의 변화가 일어난 state로 부터 시작하여
Backward로 update가 이루어지는 아이디어

Prioritized sweeping의 배경:
Backup 긴급성에 기반하여 backup의 우선순위를
정하는 것

8.4 Prioritized Sweeping

Prioritized sweeping for a deterministic environment

Initialize $Q(s,a)$, $Model(s,a)$, for all s, a, and $PQueue$ to empty
Loop forever:
- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow policy(S,Q)$
- (c) Take action A; observe resultant reward, R, and state, S'
- (d) $Model(S,A) \leftarrow R, S'$
- (e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$.
- (f) if $P > \theta$, then insert S, A into $PQueue$ with priority P
- (g) Loop repeat n times, while $PQueue$ is not empty:
  - $S, A \leftarrow first(PQueue)$
  - $R, S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
  - Loop for all $\bar{S}, \bar{A}$ predicted to lead to S:
    - $\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$
    - $P \leftarrow |\bar{R} + \gamma \max_a Q(S,a) - Q(\bar{S},\bar{A})|$.
    - if $P > \theta$ then insert $\bar{S}, \bar{A}$ into $PQueue$ with priority P

8.5 Expected vs Sample Updates

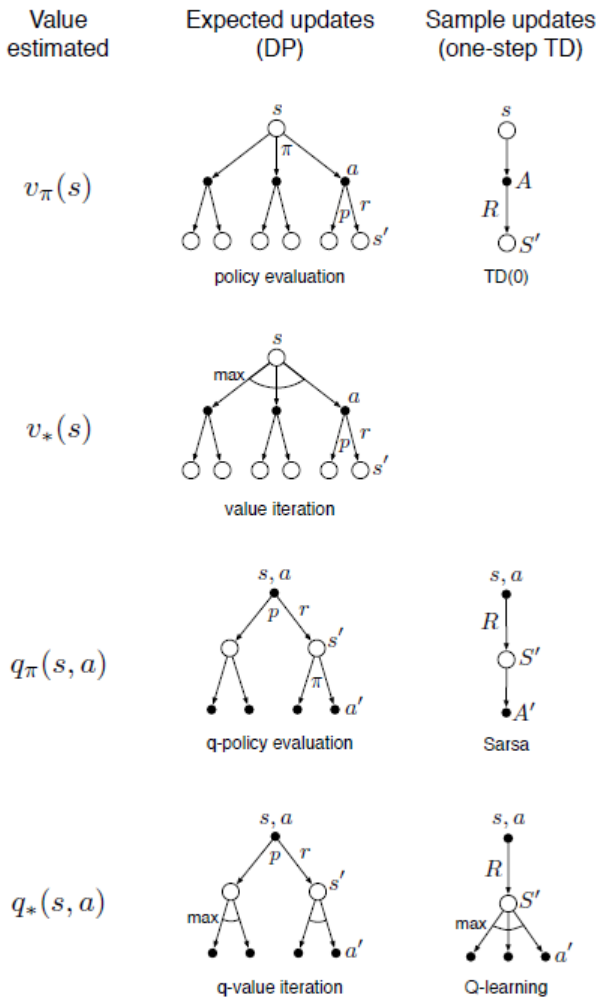


Figure 8.6: Backup diagrams for all the one-step updates considered in this book.

Any of these one-step updates can be used in planning methods.

Expected와 Sample updates의 상대적 장점을 평가하려면 서로 다른 컴퓨터 요구 사항을 제어해야 함.

8.5 Expected vs Sample Updates

Expected updates

$$Q(s,a) \leftarrow \sum_{s',r} \hat{p}(s',r|s,a)\Big[r + \gamma \max_{a'} Q(s',a')\Big].$$

Sample updates

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[R + \gamma \max_{a'} Q(S',a') - Q(s,a)\Big],$$

두 개의 차이점은 환경이 얼마나 확률적인지, 정확한지와 다음 states가 얼마나 많은지에 따라 중요도가 달라짐.

State-action pairs가 많은 상황이 대다수기 때문에 적은 수의 쌍보다 많은 수의 쌍을 확인하는 sample updates가 실생활에 좋음.

Given a unit of computational effort, is it better devoted to a few expected updates or to b times as many sample updates?

8.5 Expected vs Sample Updates

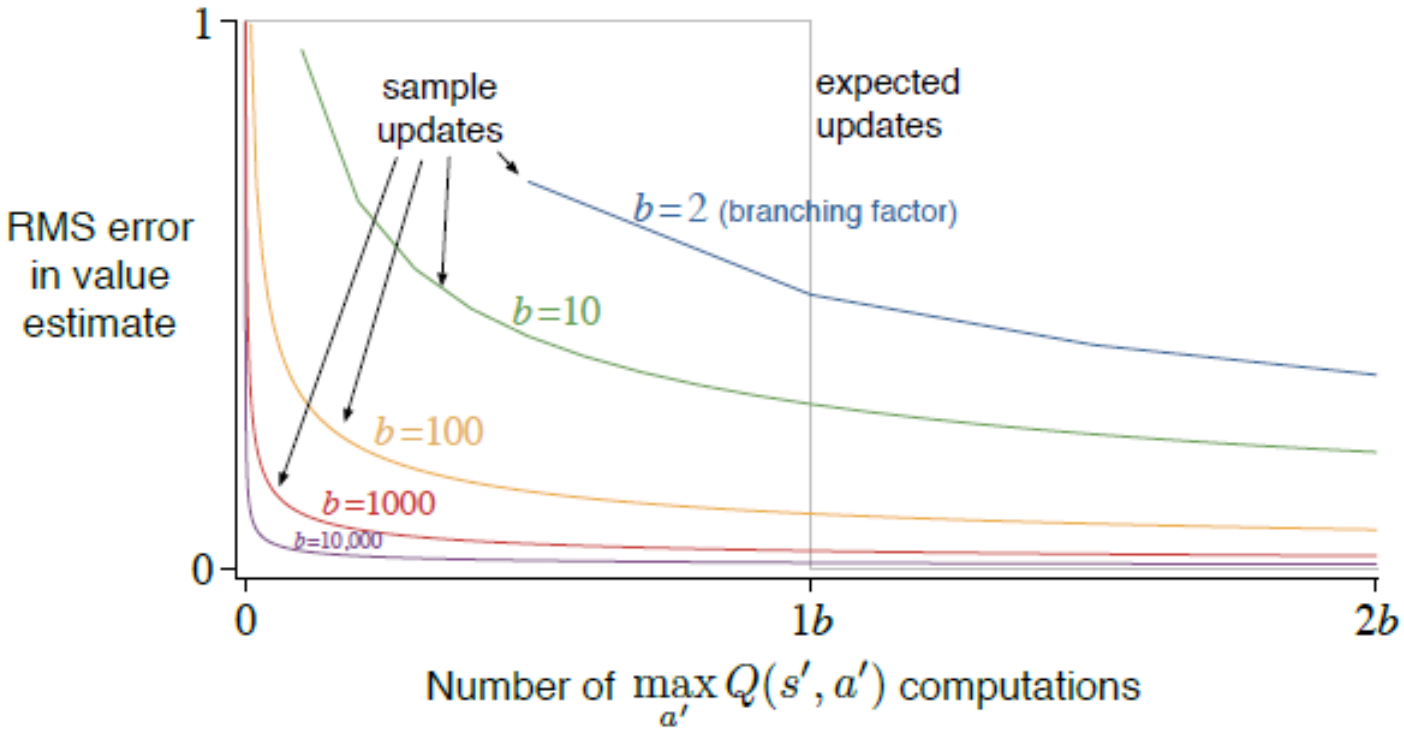


Figure 8.7: Comparison of efficiency of expected and sample updates.

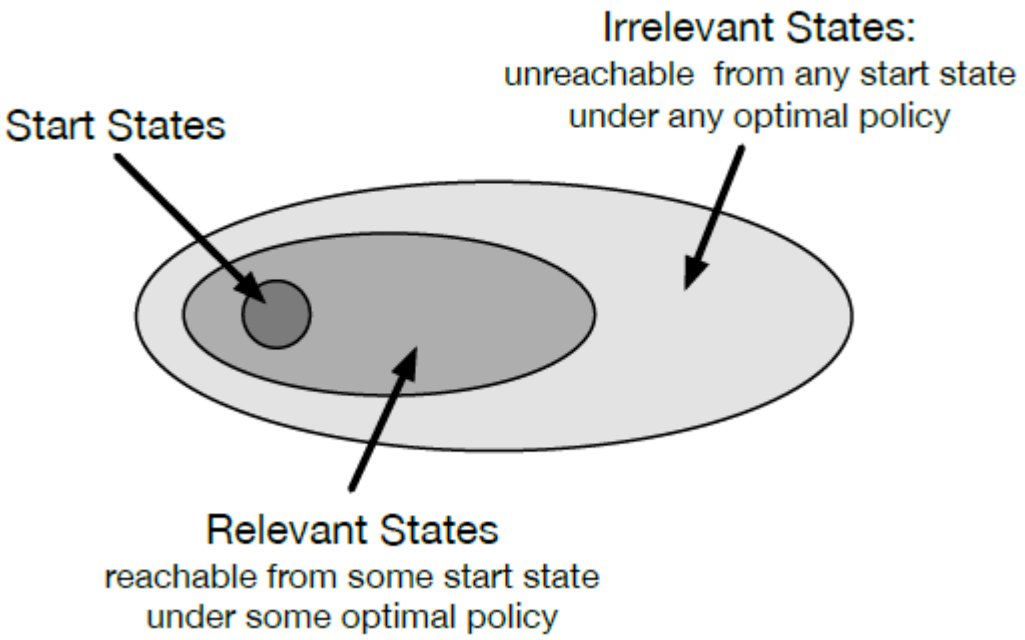
8.6 Trajectory Sampling

One simulates explicit individual trajectories and performs updates at the state or state–action pairs encountered along the way.

We call this way of generating experience and updates trajectory sampling.

8.7 Real-time Dynamic Programming

Real-time dynamic programming, or RTDP, is an on-policy trajectory-sampling version of the value-iteration algorithm of dynamic programming (DP).



8.8 Planning Decision Time

Background planning :
Well before an action is selected for any current state St, planning has played a part in improving the table entries, or the mathematical expression, needed to select the action for many states, including St.

Decision-time planning:
Well before an action is selected for any current state St, planning has played a part in improving the table entries, or the mathematical expression, needed to select the action for many states, including St.

8.9 Heuristic Search

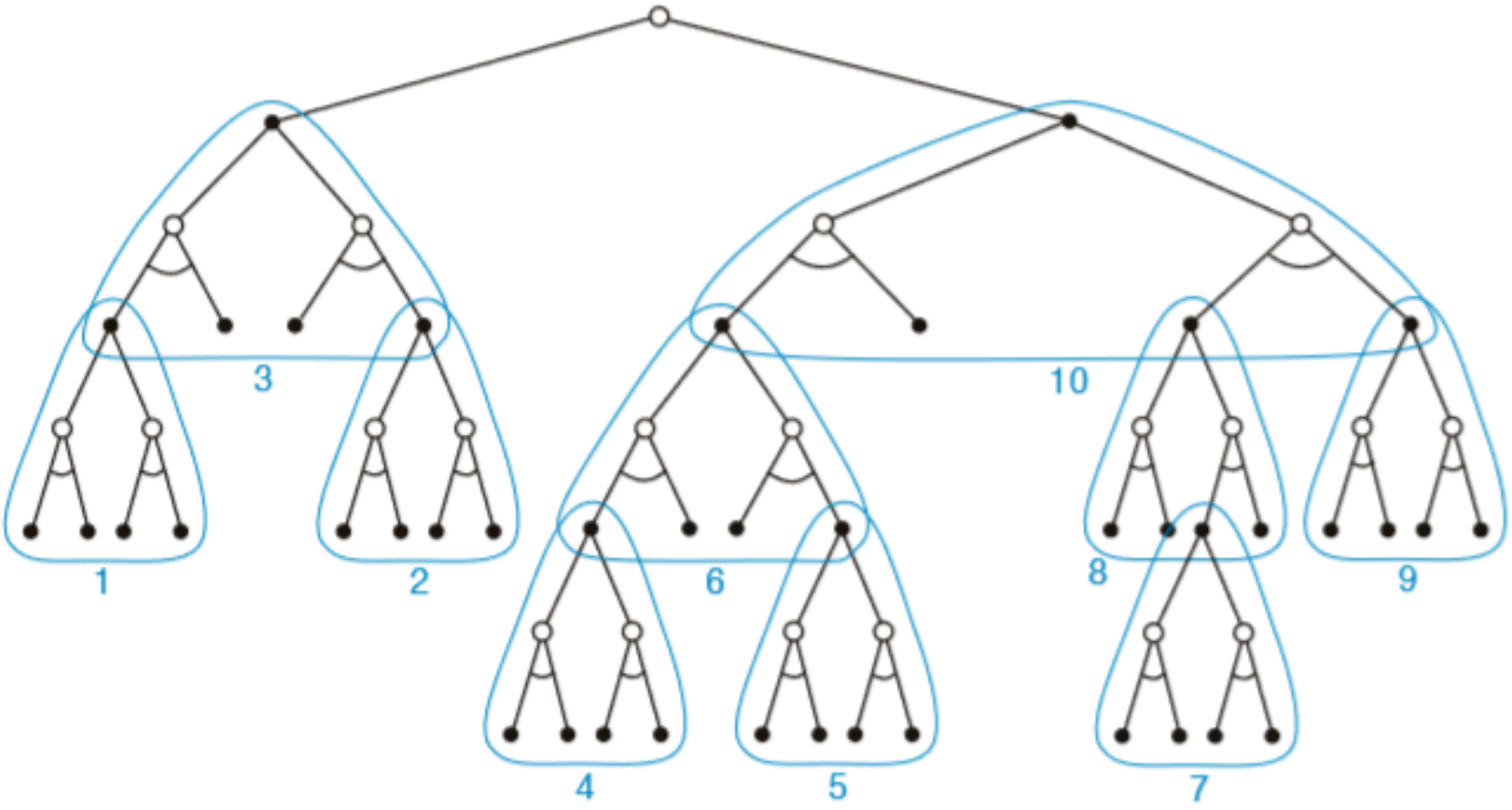
Heuristic search에선 모든 state로부터 다양한 가능성의 나뭇가지가 자라는 형태의 구조가 고려됨.

Value function은 leaf node에 적용된 후 root node에 있는 현재 State로 backup이 이루어짐.

Backup이 이루어지는 여러 노드 중에 최선의 것만 현재의 action으로 선택이 된 후 모든 backup value는 폐기됨.

Heuristic search가 가지는 효율성의 대부분은 현재의 state 바로 다음 선택되는 state와 action에 집중된 search tree로 기인한 것임.

8.9 Heuristic Search



8.10 Rollout Algorithm

Rollout algorithms are decision-time planning algorithms

based on Monte Carlo control applied to simulated trajectories

that all begin at the current environment state.

They produce Monte Carlo estimates of action values only

for each current state and for a given policy.

8.11 Monte Carlo Tree Search

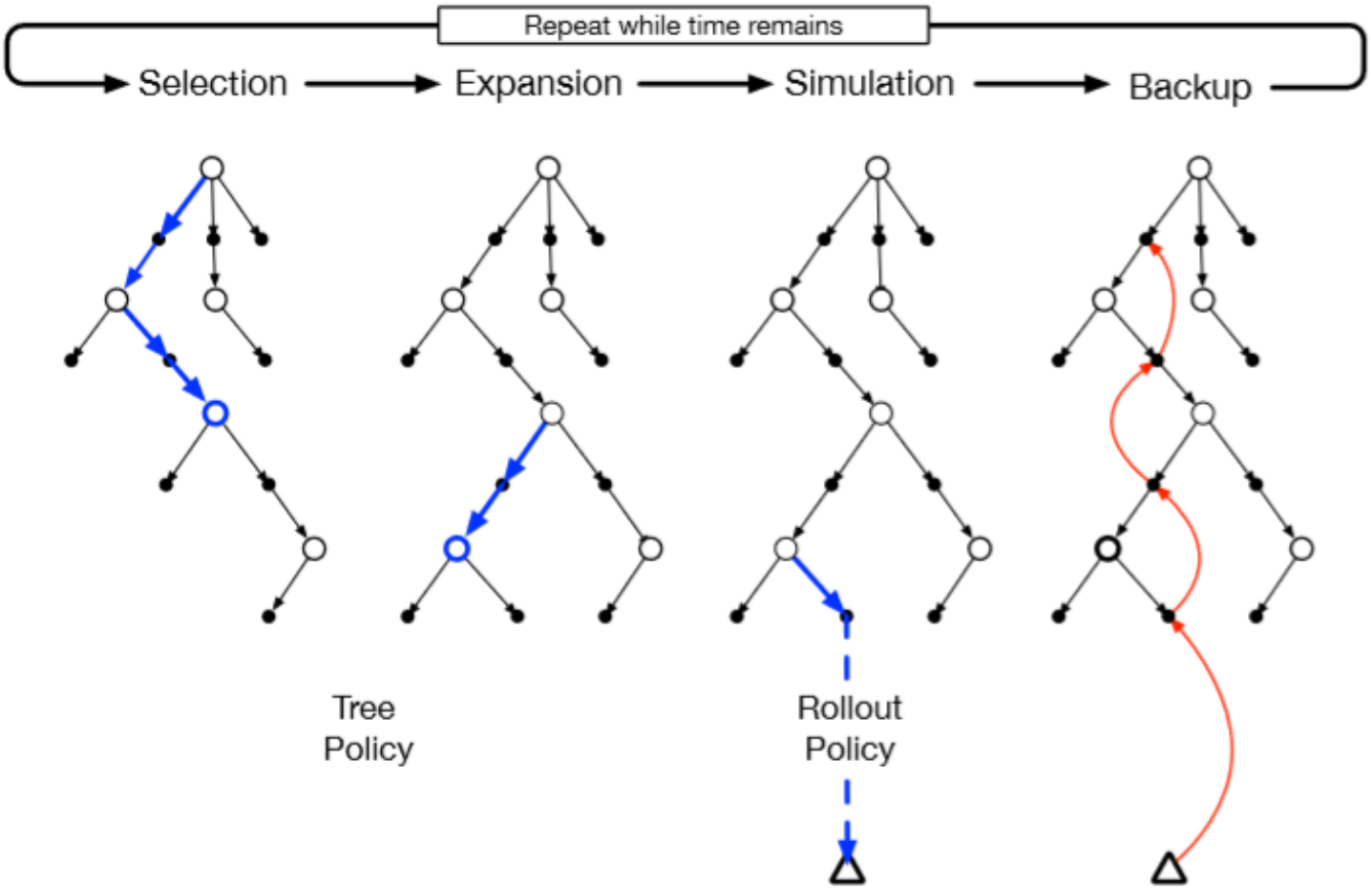
Monte Carlo Tree Search(MCTS)
: policy의 일부분으로서의 planning이 간단한 예

일반적으로 한 time step기준으로 정보가 저장되고
폐기되는 형식으로 value function이나 policy를 추정하려고
하지 않음.

한 step동안 현재 sate로부터 시작해서 많은 simulated
Trajectory가 생성되어 terminal state까지 진행됨.
대부분의 경우 simulated trajectory의 action은 default
Policy(균등확률)라고 부르는 simple policy를 사용하여 선택.

8.11 Monte Carlo Tree Search

MCTS의 각 iteration은 4단계를 거쳐 진행



8.11 Monte Carlo Tree Search

1. Selection:
주어진 정보를 이용해서 추가적으로 탐색할 가치가 있는 가장 가능성이 있는 하나의 node가 현재 tree에서 선택.

2. Expansion:
선택된 node로부터 가지를 치는 방식으로 한 개 또는 몇 개의 node를 추가하는 방식으로 tree를 확장해 나감. 새 가지는 game tree의 leaf node가 되는데, 그 동안 tree를 만드는 과정에서 한번도 시도되지 않은 node임.

3. Simulation:
주어진 정보를 이용해서 leaf node들 중 하나를 선택해서 simulation의 시작점으로 삼음. 이로부터 종료시점까지 *rollout policy*를 사용하여 roll out을 함.

4. Backpropagation:
Simulated game의 결과는 partial game tree의 링크와 관련된 state 통계의 update를 위해 backup됨.