Lecture 12 - Planning & Learning with Tabular Methods (Prioritized Sweeping & MCTS)

Instructor: GuangBing Yang, PhD

On-policy Prediction with Approximation

- Outline
  - Prioritized Sweeping
  - Expected vs. Sample Updates
  - Trajectory Sampling
  - Real-time Dynamic Programming
  - Monte Carlo Tree Search

Prioritized Sweeping

- In Dyna and Dyna+Q, the simulated transitions of the state-action pairs are chosen uniformly from all previously experienced pairs.
- A uniform selection is not the best because it comes with highest uncertainty and inefficient in practical applications.
- A more efficient way is that the update of the simulated transitions are focused on particular state-action pairs (with priorities)
- In the maze example, if only select the state-action pairs with positive return value, ignore those having zero value because zero value has no contribution to the overall rewards.
- However, if simulated transitions are generated uniformly, then many wasteful updates will be made before stumbling onto one of these useful ones

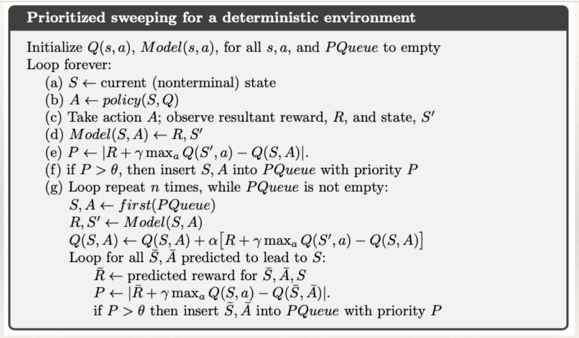
Prioritized Sweeping

- This is even worse in those particular much larger problems since the number of states is so large that an unfocused search would be extremely inefficient.
- A natural thing is that the predecessor pairs of those that have changed a lot are more likely to also change a lot.
- In a stochastic environment, estimated transition with high variations also contribute to more variations in the urgency with which pairs need to be updated.
- It is natural to prioritize the updates according to a measure of their urgency, and perform them in order of priority.
- This is the idea behind prioritized sweeping.

Algorithm of Prioritized Sweeping

- In general, to prioritize the selection of the state-action pairs, a backward approach from any state whose value has changed including the special case ‘goal state’ can be considered as the search strategies.
- If the values of these actions are updated, then the values of the predecessor states may change in turn. Then follow the actions update as well as their predecessor states.
- Hence, the algorithm can work backward from arbitrary states that have changed in value
- Which is called `backward focusing` of planning computations.

Algorithm of Prioritized Sweeping



Prioritized sweeping for a deterministic environment

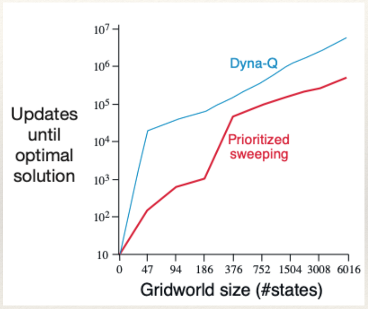
```
Initialize $Q(s,a)$, $Model(s,a)$, for all $s, a$, and $PQueue$ to empty
Loop forever:
  (a) $S \leftarrow$ current (nonterminal) state
  (b) $A \leftarrow policy(S, Q)$
  (c) Take action $A$; observe resultant reward, $R$, and state, $S'$
  (d) $Model(S, A) \leftarrow R, S'$
  (e) $P \leftarrow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

Algorithm of Prioritized Sweeping

- In this algorithm, a prioritized queue is maintained of every state-action pair.
- The priority is determined based on the size of the change of the state-action pair.
- When the top pair in the queue is updated, the effect on each of its predecessor pairs is computed.
- If the effect is greater than some small threshold, then the pair is inserted in the queue with the new priority.
- Doing this way, the effects of changes are efficiently propagated backward until quiescence.

Example: Prioritized Sweeping on Mazes

- This approach increases the speed of discovering the optimal solutions in complex maze problems.
- A typical example is shown to the right
- Prioritized sweeping maintained a decisive advantage over unprioritized Dyna-Q.



Adapted from Peng and Williams (1993)

Expected vs. Sample Updates

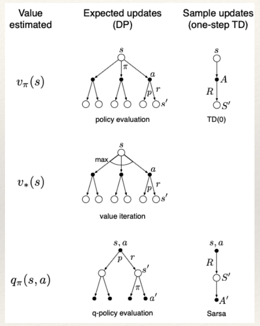
- **Prioritized sweeping** is just one way of distributing computations to improve planning efficiency, and probably not the best way.
- What the prioritized sweeping uses expected updates limits its efficiency because many of computations are wasted in the stochastic environment.
- In contrast, sample updates can get closer to the true value function with less computation despite the variance introduced by sampling.
- All kinds of state-space planning can be viewed as sequences of value updates. Only matter is the difference in the type of update, expected or sample, large or small, and in the order in which the updates are done.

Expected vs. Sample Updates

- **Backward focusing** is one strategy of planning improving;
- **Forward focusing** is another one that focus on states according to how easily they can be reached from the states that are visited frequently under the current policy
- Value update involves several dimensions (or factors); either update state values or action values, or with estimation of optimal policy or an arbitrary given policy.
- These two dimensions give rise to four classes of updates for approximating the four value functions q_*, v_*, q_π, v_π.

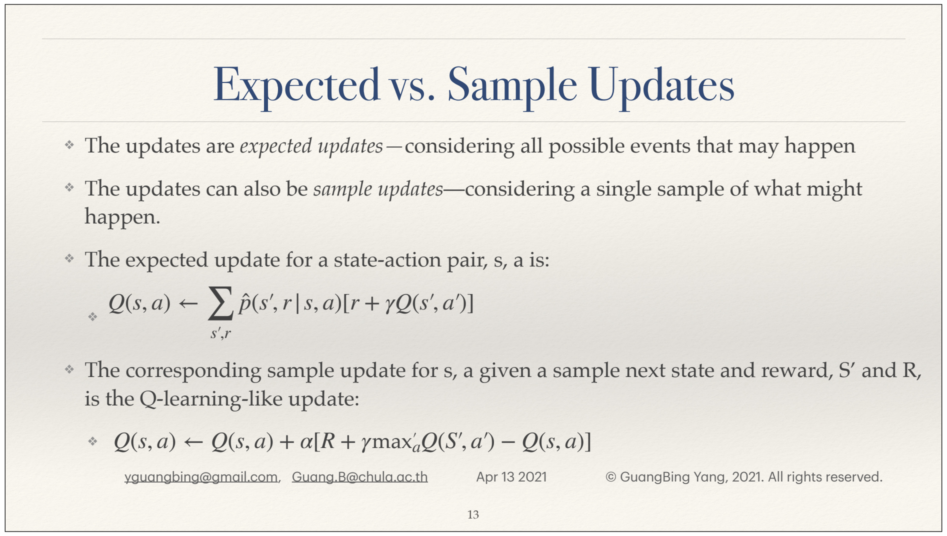
Expected vs. Sample Updates

- These three binary dimensions (v_π, v_*, q_π) give rise to eight cases (2^3).
- Seven of which correspond to specific algorithms, as shown in the figure to the right
- Dyna-Q uses q_* updates either *expected* or *sample* q_π
- Prioritized sweeping uses *expected* update.
- DP uses *expected* approach;
- TD(0) and one-set Sarsa use *sample* approach.



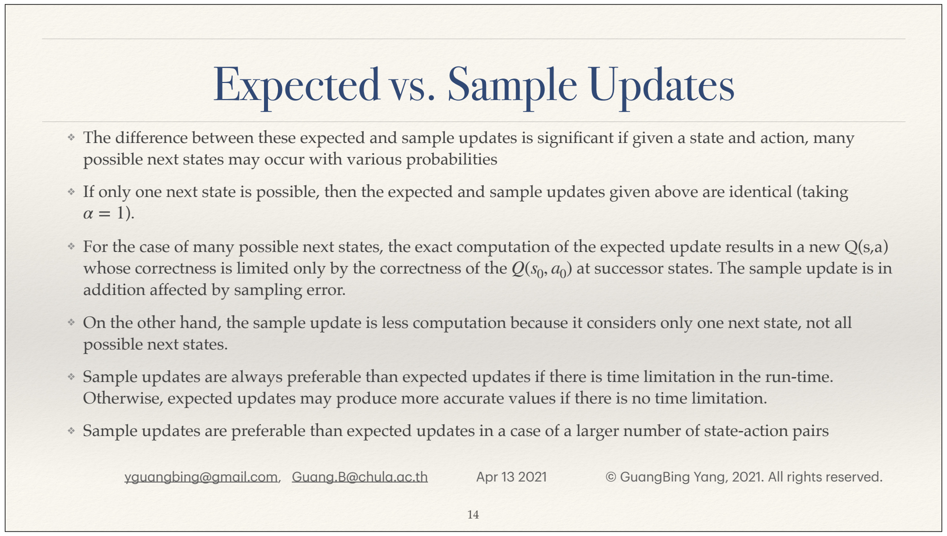
Expected vs. Sample Updates

- Without a distribution model, expected updates are not possible,
- But sample updates can be done using sample transitions from the environment or a sample model.
- Expected updates certainly produce a better estimation because the sample errors are averaged out.
- In contrast, expected updates need more computational resources, but these resources are often limit in planning.



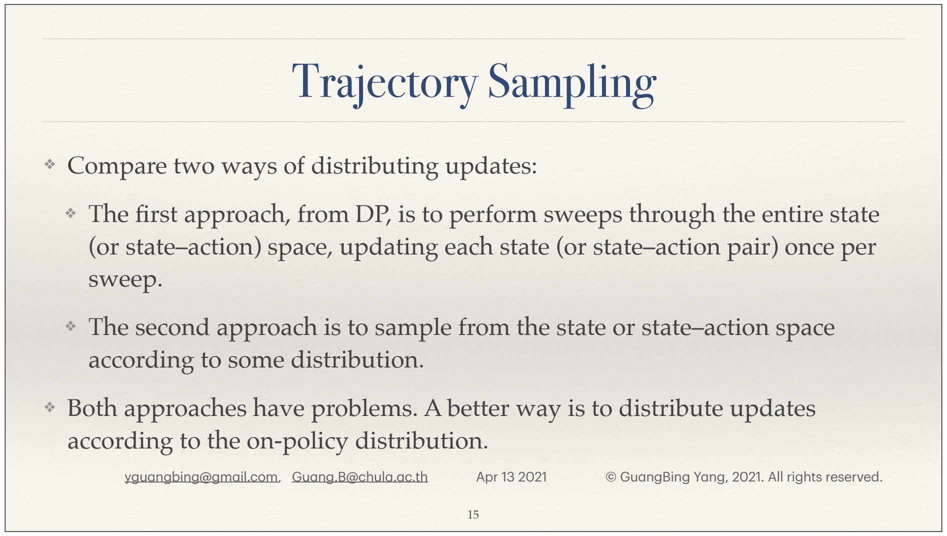
Expected vs. Sample Updates

- The updates are *expected updates*—considering all possible events that may happen
- The updates can also be *sample updates*—considering a single sample of what might happen.
- The expected update for a state-action pair, s, a is:
  - $Q(s,a) \leftarrow \sum_{s',r} \hat{p}(s',r|s,a)[r + \gamma Q(s',a')]$
- The corresponding sample update for s, a given a sample next state and reward, S′ and R, is the Q-learning-like update:
  - $Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_a' Q(S',a') - Q(s,a)]$



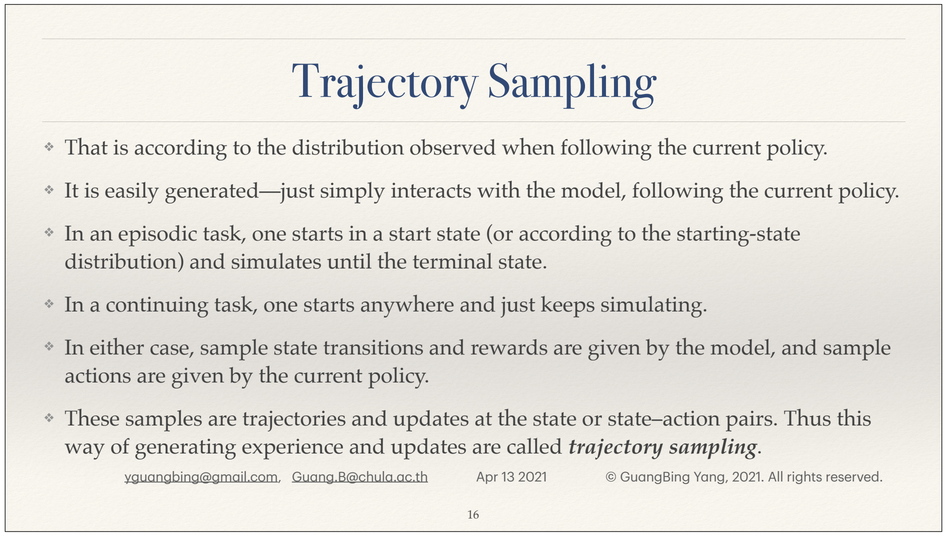
Expected vs. Sample Updates

- The difference between these expected and sample updates is significant if given a state and action, many possible next states may occur with various probabilities
- If only one next state is possible, then the expected and sample updates given above are identical (taking $\alpha = 1$).
- For the case of many possible next states, the exact computation of the expected update results in a new Q(s,a) whose correctness is limited only by the correctness of the $Q(s_0, a_0)$ at successor states. The sample update is in addition affected by sampling error.
- On the other hand, the sample update is less computation because it considers only one next state, not all possible next states.
- Sample updates are always preferable than expected updates if there is time limitation in the run-time. Otherwise, expected updates may produce more accurate values if there is no time limitation.
- Sample updates are preferable than expected updates in a case of a larger number of state-action pairs



Trajectory Sampling

- Compare two ways of distributing updates:
  - The first approach, from DP, is to perform sweeps through the entire state (or state–action) space, updating each state (or state–action pair) once per sweep.
  - The second approach is to sample from the state or state–action space according to some distribution.
- Both approaches have problems. A better way is to distribute updates according to the on-policy distribution.



Trajectory Sampling

- That is according to the distribution observed when following the current policy.
- It is easily generated—just simply interacts with the model, following the current policy.
- In an episodic task, one starts in a start state (or according to the starting-state distribution) and simulates until the terminal state.
- In a continuing task, one starts anywhere and just keeps simulating.
- In either case, sample state transitions and rewards are given by the model, and sample actions are given by the current policy.
- These samples are trajectories and updates at the state or state–action pairs. Thus this way of generating experience and updates are called ***trajectory sampling***.

Trajectory Sampling

- The trajectory sampling is probably the only efficient way of distributing updates according to the on-policy distribution.
- Even known an explicit on-policy distribution is still suffering the computational cost of sweeping all state-action pairs.
- However, in real cases, knowing the on-policy distribution in an explicit form is unlikely.
- In addition, the distribution changes whenever the policy changes, and computing the distribution requires computation comparable to a complete policy evaluation.
- Therefore, consideration of such other possibilities makes trajectory sampling seem both efficient and elegant.

Trajectory Sampling

- Trajectory sampling is better than the uniform distribution.
- In addition, the on-policy distribution has significant advantages when function approximation is used.
- In the uniform case, all state–action pairs need to be updated in each place
- In the on-policy case, for the simulated episodes, each state–action pair needs to be updated under the current ϵ-greedy policy (ϵ=0.1).

Real-time Dynamic Programming

- Real-time dynamic programming, RTDP, is an on-policy trajectory-sampling version of the value-iteration algorithm of DP.
- RTDP updates the values of states visited in actual or simulated trajectories by means of expected tabular value-iteration updates
- RTDP is an example of an asynchronous DP algorithm as described in Chapter 4 Section 5.
- asynchronous DP algorithm updates state values in any order using any values of other states happen to be available.
- In RTDP, the update order is dictated by the order states are visited in real or simulated trajectories.

Real-time Dynamic Programming

- For the prediction problem, given a policy, then on-policy trajectory sampling allows the algorithm to completely skip states that cannot be reached by the given policy from any of the start states: such states are irrelevant to the prediction problem.
- For the control problem, finding an optimal policy need not to specify optimal actions for these irrelevant states because there might well be states that cannot be reached by any optimal policy from any of the start states.
- Hence, only need is an optimal ***partial*** policy, meaning a policy that is optimal for the relevant states but can specify arbitrary actions, or even be undefined, for the irrelevant states.

Real-time Dynamic Programming

- However, finding such an optimal partial policy with an on-policy trajectory-sampling control method, such as Sarsa.
- In general this process requires visiting all state–action pairs—even those that will turn out to be irrelevant—an infinite number of times.
- This can be done, for example, by using exploring starts.
- This is the case for RTDP as well:
    - for episodic tasks with exploring starts, RTDP is an asynchronous value-iteration algorithm that converges to optimal polices for discounted finite MDPs.
    - and an optimal ***partial*** policy is required and any irrelevant states and actions can be ignored.
    - RTDP is guaranteed to find a policy that is optimal on the relevant states without visiting every state
    - This is a great advantage for problems with very large state sets, where even a single sweep may not be feasible

Planning at Decision Time

- Planning can be used in at least two ways
- DP and Dyna use planning to gradually improve a policy or value function on the basis of simulated experience obtained from a model (either a sample or a distribution model)
- If planning is not focused on the current state, this planning is called ***background planning***
- In contrast, if planning is to begin and complete after encountering a new state, this planning is called ***decision-time planning*** — it focuses on a particular state.

Planning at Decision Time

- There are two ways of thinking about planning
    - using simulated experience to gradually improve a policy or value function,
    - or using simulated experience to select an action for the current state
- They can combine together in natural and interesting ways.
- For planning at decision time, it can still be viewed as proceeding from simulated experience to updates and values, and ultimately to a policy
- It is just that the values and policy are specific to the current state and the action choices available there
- Decision-time planning is most useful in applications in which fast responses are not required

Heuristic Search

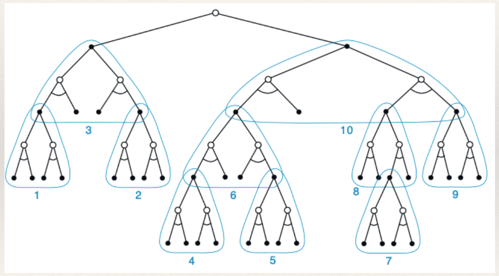
- A collection of decision-time planning methods in AI is known as Heuristic search.
- In heuristic search, a tree structure is considered for each state encountered.
- The approximate value function is applied to the leaf nodes.
- The current state is the true root, and tree travel back up from the leaf nodes toward the root — the current state.

Heuristic Search

- The backing up within the search tree is just the same as in the expected updates with maxes (those for v_* and q_*)
- Once the backed-up values of these nodes are computed, the best of them is chosen as the current action, and then all backed-up values are discarded.
- Thus, heuristic search can be viewed as an extension of the idea of a greedy policy beyond a single step
- Searching deeper than one step is to obtain better action selections
- On the other hand, the deeper the search, the more computation is required, usually resulting in a slower response time

Heuristic Search

- One-step heuristic search updates (shown here outlined in blue) backing up values from the leaf nodes toward the root.



The ordering shown here is for a selective **depth-first** search.

Rollout Algorithms

- Rollout algorithms are decision-time planning algorithms based on Monte Carlo control applied to simulated trajectories that all begin at the current environment state
- The term "rollout" comes from estimating the value of a backgammon position in game Backgammon
- The goal of a rollout algorithm is to estimate action values for each current state and for a given policy usually called the rollout policy based on Monte Carlo estimates
- It produces the action-value estimates when the action (or one of the actions) have the highest estimated value.
- As decision-time planning algorithms, rollout algorithms make immediate use of these action-value estimates, then discard them

Rollout Algorithms

- Based on the policy improvement theorem, given two policies π and π'
  - $\pi = \pi' \forall s$, except s, with $\pi'(s) = a \neq \pi(s)$
  - If $q_\pi(s, a) \geq v_\pi(s)$, then policy π' is as good as, or better than π.
- In rollout algorithms, s is the current state, and π is the rollout policy.
- Averaging the returns of the simulated trajectories produces estimates of $q_\pi(s, a') \forall \ a' \in A(s)$ for each action
- Then the policy that selects an action in s that maximizes these estimates and thereafter follows π is a good candidate for a policy that improves over π
- This process is similar to the policy-iteration algorithm of DP.
- Thus, the aim of a rollout algorithm is to improve upon the rollout policy; not to find an optimal policy

Monte Carlo Tree Search

- **Monte Carlo Tree Search (MCTS)** is a new and very successful example of decision-time planning.
- Basically, it is a rollout algorithm but enhanced for accumulating value estimates obtained from the Monte Carlo simulations.
- MCTS is the major improvement in computer Go from a weak amateur level in 2005 to a grandmaster level (6 dan or more) in 2015, and it is the main driver in AlphaGo.

Monte Carlo Tree Search

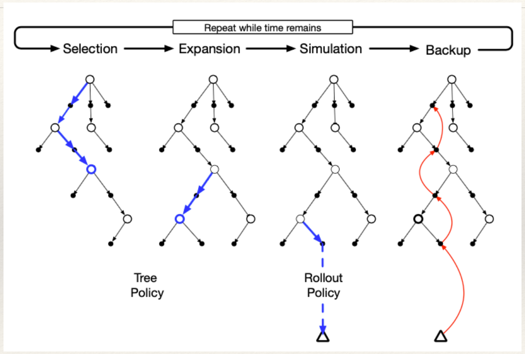
- MCTS randomly selects each new state, and then select the agent's action for that state;
- Then, it is executed again to select the action for the next state, and so on.
- As in a rollout algorithm, each execution is an iterative process that simulates many trajectories starting from the current state and running to a terminal state
- MCTS uses multiple simulations starting at the current state by extending the initial portions of trajectories with high evaluations from earlier simulations.
- MCTS does not keep approximate value functions or policies from one action selection to the next

Monte Carlo Tree Search

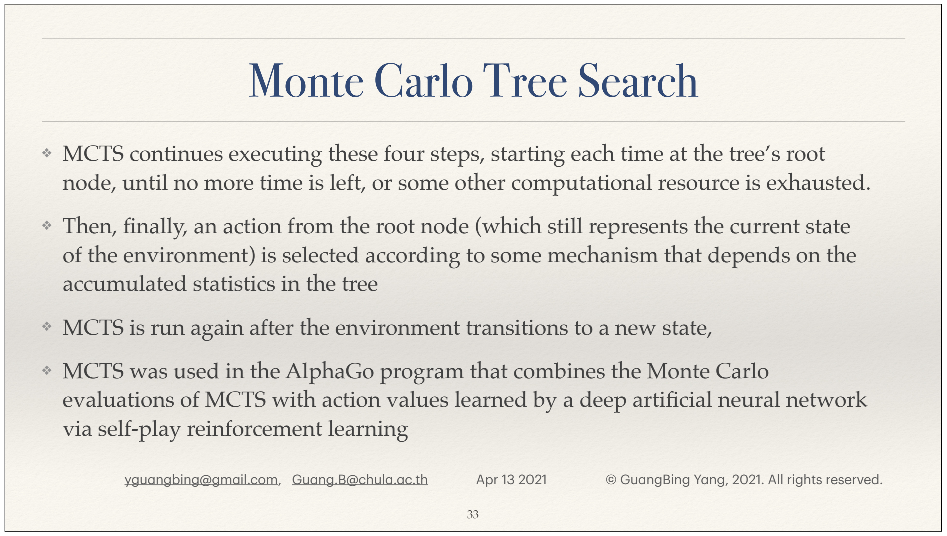
- Generally, a rollout policy from simpler rollout algorithms is used for generating the actions in the simulated trajectories
- Many simulated trajectories can be generated in a short period of time since the rollout policy is simple enough
- A tree rooted at the current state is constructed to maintain the Monte Carlo value estimates from the subset of state-action pairs
- MCTS adds notes that represent states from the results of the simulated trajectories to the tree.
- The policy used in MCTS is called *tree policy* that again balances exploration and exploitation

Monte Carlo Tree Search

- Generally, MCTS executes as many iterations as possible before an action needs to be selected
- Extend a tree whose root node represents the current state
- Each iteration consists of the four operations
  - Selection,
  - Expansion
  - Simulation, and
  - Backup

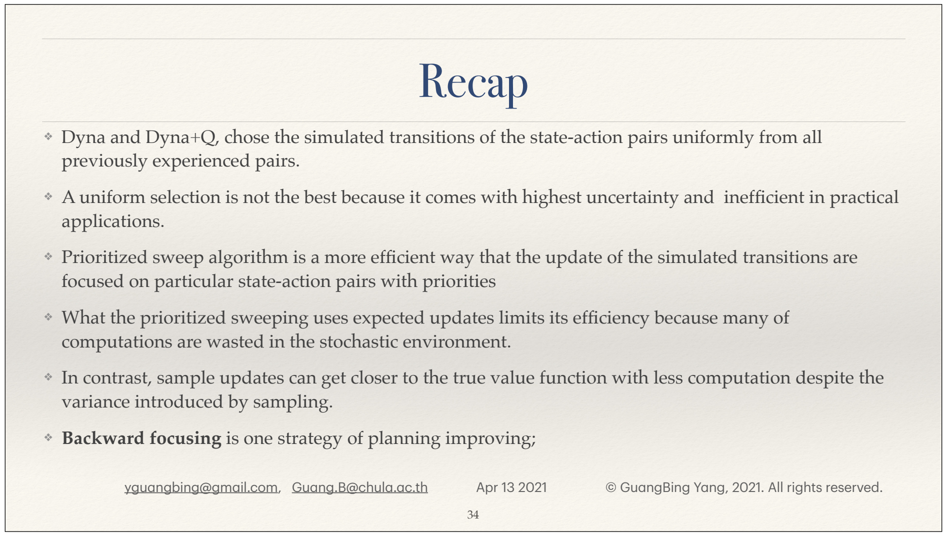


MCTS—Adapted from Chaslot, Bakkes, Szita, and Spronck (2008).



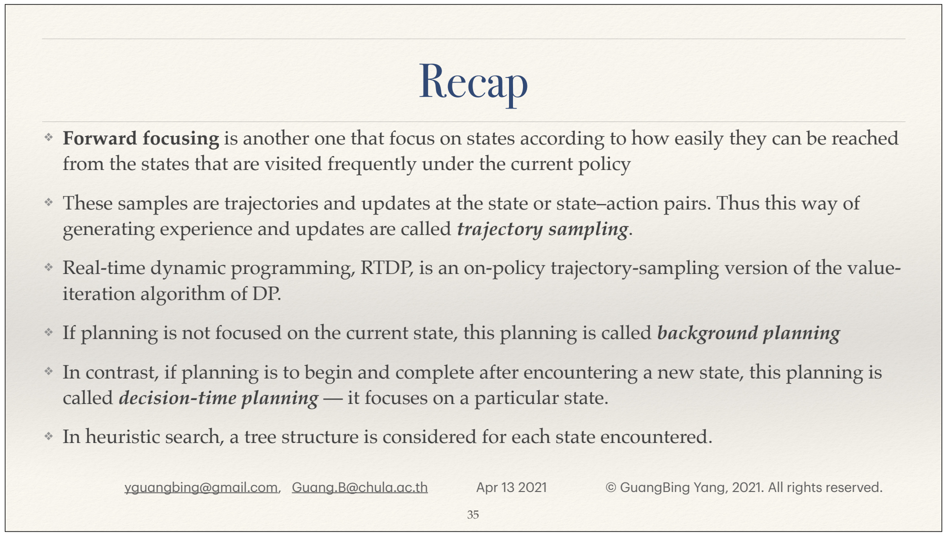
Monte Carlo Tree Search

- MCTS continues executing these four steps, starting each time at the tree's root node, until no more time is left, or some other computational resource is exhausted.
- Then, finally, an action from the root node (which still represents the current state of the environment) is selected according to some mechanism that depends on the accumulated statistics in the tree
- MCTS is run again after the environment transitions to a new state,
- MCTS was used in the AlphaGo program that combines the Monte Carlo evaluations of MCTS with action values learned by a deep artificial neural network via self-play reinforcement learning



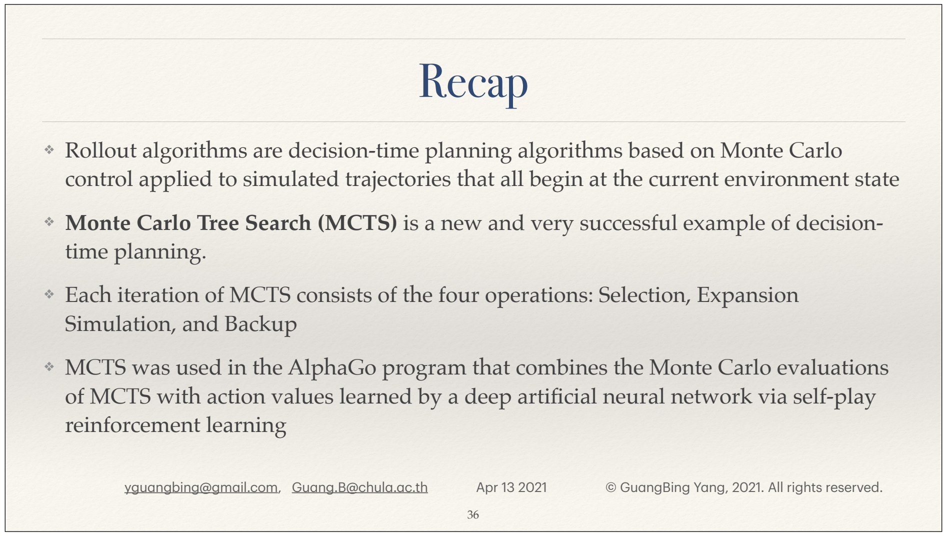
Recap

- Dyna and Dyna+Q, chose the simulated transitions of the state-action pairs uniformly from all previously experienced pairs.
- A uniform selection is not the best because it comes with highest uncertainty and inefficient in practical applications.
- Prioritized sweep algorithm is a more efficient way that the update of the simulated transitions are focused on particular state-action pairs with priorities
- What the prioritized sweeping uses expected updates limits its efficiency because many of computations are wasted in the stochastic environment.
- In contrast, sample updates can get closer to the true value function with less computation despite the variance introduced by sampling.
- **Backward focusing** is one strategy of planning improving;



Recap

- **Forward focusing** is another one that focus on states according to how easily they can be reached from the states that are visited frequently under the current policy
- These samples are trajectories and updates at the state or state–action pairs. Thus this way of generating experience and updates are called ***trajectory sampling***.
- Real-time dynamic programming, RTDP, is an on-policy trajectory-sampling version of the value-iteration algorithm of DP.
- If planning is not focused on the current state, this planning is called ***background planning***
- In contrast, if planning is to begin and complete after encountering a new state, this planning is called ***decision-time planning*** — it focuses on a particular state.
- In heuristic search, a tree structure is considered for each state encountered.



Recap

- Rollout algorithms are decision-time planning algorithms based on Monte Carlo control applied to simulated trajectories that all begin at the current environment state
- **Monte Carlo Tree Search (MCTS)** is a new and very successful example of decision-time planning.
- Each iteration of MCTS consists of the four operations: Selection, Expansion Simulation, and Backup
- MCTS was used in the AlphaGo program that combines the Monte Carlo evaluations of MCTS with action values learned by a deep artificial neural network via self-play reinforcement learning

Assignment 4

- Assignment 4 worth 15%, and is about DynaQ+ and Prioritized Sweeping Algorithms. Two simple programming questions with one bonus survey (Critical Thinking Test). Each of them worths 5 points.
- Assigned from MS Team, you can check out it from there or directly from shared G drive.
- same as the assignment 1, 2, & 3, copy and download my Colab to your Google drive (Important note: Don't modify my Colab notebook, otherwise other classmates will see your work.)
- Working on your copy of the Colab notebook. Don't forget to add your name and student id in it.
- After finishing it, share it (the Notebook not the actual Python script) with me (only me, do not share your work with others.)
- All programming exercises MUST be running correctly in Colab without any errors and exceptions. If your code cannot run at all, and I cannot see any kind of outputs, you receive no grade points for that part.
- Before you submit your **Colab notebook**, make sure to leave the outputs (results) of the functions in the notebook. I ONLY review the outputs of your functions or the final results.
- The assignment due at May 3rd, 2021. It is an individual assignment.
- Just remind you to beware of academic integrity and responsible behaviour.

Questions and Lab