CS 748, Spring 2021: Week 2, Lecture 1

Shivaram Kalyanakrishnan

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Spring 2021

Navigation System

How to go from IIT Bombay to Marine Drive?

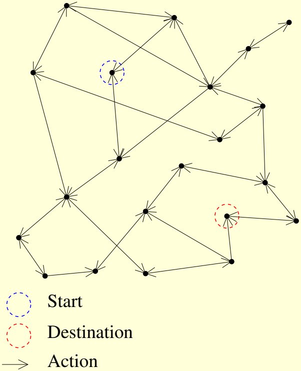
[1]

[1] https://www.flickr.com/photos/nat507/16088993607. CC image courtesy of Nathan Hughes Hamilton on Flickr licensed under CC BY 2.0.

Navigation System

How to go from IIT Bombay to Marine Drive?

[1]



[1] https://www.flickr.com/photos/nat507/16088993607. CC image courtesy of Nathan Hughes Hamilton on Flickr licensed under CC BY 2.0.

Some Popular Puzzles

How to solve?

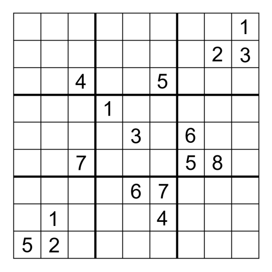
								1
							2	3
		4			5			
			1					
				3		6		
		7				5	8	
				6	7			
	1				4			
5	2							

Sudoku [1]

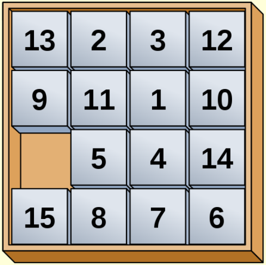
[1] https://upload.wikimedia.org/wikipedia/commons/e/eb/Sudoku_Puzzle_%28a_symmetrical_puzzle_with_17_clues%29.png. CC image courtesy of LithiumFlash on WikiCommons licensed under CC-BY-SA-4.0.

Some Popular Puzzles

How to solve?



Sudoku [1]



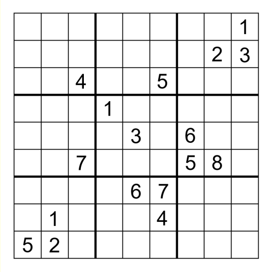
15-puzzle [2]

[1] https://upload.wikimedia.org/wikipedia/commons/e/eb/Sudoku_Puzzle_%28a_symmetrical_puzzle_with_17_clues%29.png. CC image courtesy of LithiumFlash on WikiCommons licensed under CC-BY-SA-4.0.

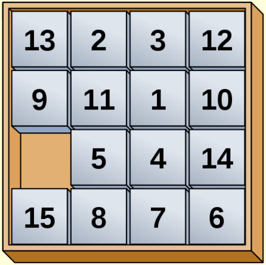
[2] https://commons.wikimedia.org/wiki/File:15-puzzle-solvable.svg. CC image courtesy of Stannic on WikiMedia Commons licensed under CC-BY-SA-3.0

Some Popular Puzzles

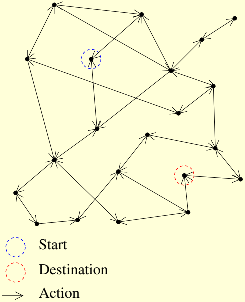
How to solve?



Sudoku [1]



15-puzzle [2]



Same abstraction?

[1] https://upload.wikimedia.org/wikipedia/commons/e/eb/Sudoku_Puzzle_%28a_symmetrical_puzzle_with_17_clues%29.png. CC image courtesy of LithiumFlash on WikiCommons licensed under CC-BY-SA-4.0.

[2] https://commons.wikimedia.org/wiki/File:15-puzzle-solvable.svg. CC image courtesy of Stannic on WikiMedia Commons licensed under CC-BY-SA-3.0

Search

- Classical search
  - Problem instances
  - Generic search template
  - Uninformed search
  - Informed search (a.k.a. heuristic search)
  - Minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Discussion

Search

- Classical search
  - Problem instances
  - Generic search template
  - Uninformed search
  - Informed search (a.k.a. heuristic search)
  - Minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Discussion

Elements of a Search Problem Instance

Elements of a Search Problem Instance

- Set of states, including designated start state.
- Set of actions available from each state.
- NextState(s, a) for each state s and action a.
- Cost(s, a) for each state s and action a (assumed ≥ 0).
- IsGoal(s) for each state s.

Elements of a Search Problem Instance

- Set of states, including designated start state.
- Set of actions available from each state.
- NextState(s, a) for each state s and action a.
- Cost(s, a) for each state s and action a (assumed ≥ 0).
- IsGoal(s) for each state s.

- **Expected output**: a sequence of actions, which when applied from start state:
  - reaches a goal state, and
  - (optionally) has minimum path-cost.

Elements of a Search Problem Instance

- Set of states, including designated start state.
- Set of actions available from each state.
- NextState(s, a) for each state s and action a.
- Cost(s, a) for each state s and action a (assumed ≥ 0).
- IsGoal(s) for each state s.

- **Expected output**: a sequence of actions, which when applied from start state:
  - reaches a goal state, and
  - (optionally) has minimum path-cost.

  Note: Sometimes there might be no solution!

Elements of a Search Problem Instance

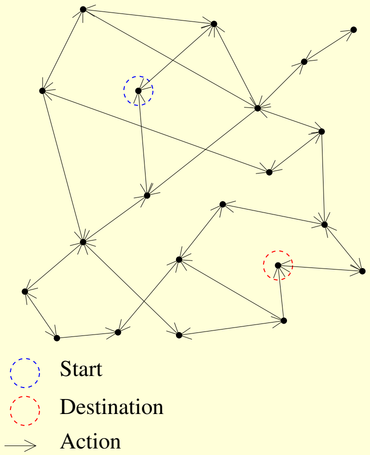
- Set of states, including designated start state.
- Set of actions available from each state.
- NextState(s, a) for each state s and action a.
- Cost(s, a) for each state s and action a (assumed ≥ 0).
- IsGoal(s) for each state s.

- **Expected output**: a sequence of actions, which when applied from start state:
  - reaches a goal state, and
  - (optionally) has minimum path-cost.

  Note: Sometimes there might be no solution!

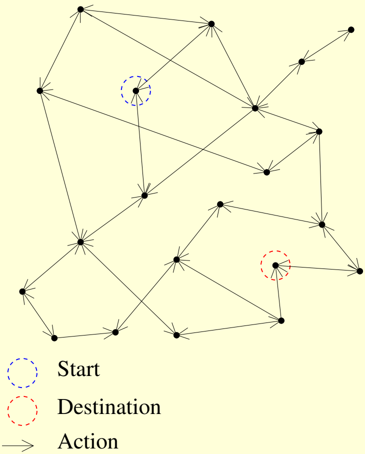
- Number of available actions in each state is called the branching factor b.
- Length of optimal path to reach goal state is called the depth d of the search instance.

Problem Formulation: Navigation System



States?

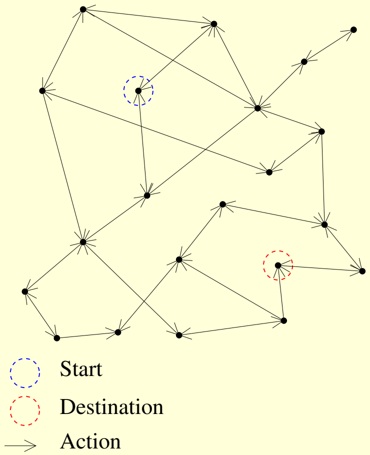
Problem Formulation: Navigation System



States?

Start state?

Problem Formulation: Navigation System

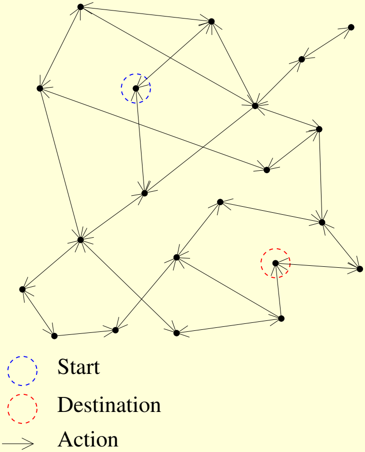


States?

Start state?

Actions?

Problem Formulation: Navigation System



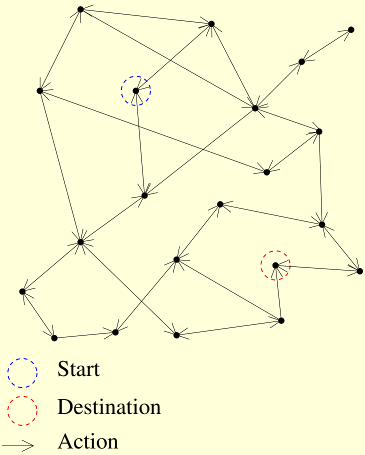
States?

Start state?

Actions?

NextState()?

Problem Formulation: Navigation System



States?

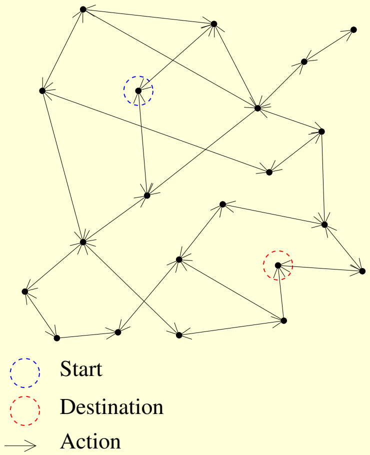
Start state?

Actions?

NextState()?

Cost()?

Problem Formulation: Navigation System



States?

Start state?

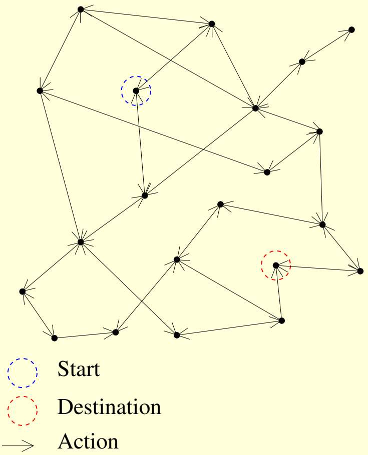
Actions?

NextState()?

Cost()?

IsGoal()?

Problem Formulation: Navigation System



States?

Start state?

Actions?

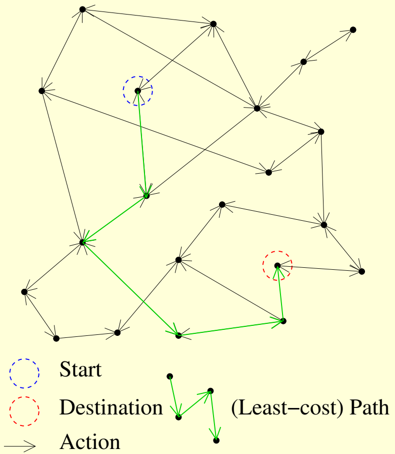
NextState()?

Cost()?

IsGoal()?

A solver needs to find the least-cost path.

Problem Formulation: Navigation System



States?

Start state?

Actions?

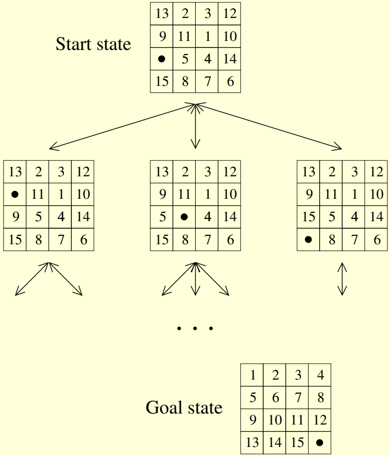
NextState()?

Cost()?

IsGoal()?

A solver needs to find the least-cost path.

Problem Formulation: 15 Puzzle



States?

Start state?

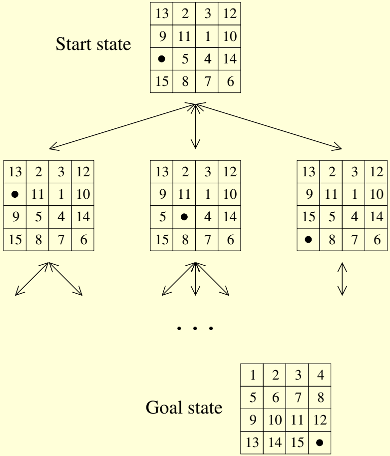
Actions?

NextState()?

Cost()?

IsGoal()?

Problem Formulation: 15 Puzzle



States?

Start state?

Actions?

NextState()?

Cost()?

IsGoal()?

A solver needs to find the shortest path to goal state.

Search

- Classical search
  - Problem instances
  - Generic search template
  - Uninformed search
  - Informed search (a.k.a. heuristic search)
  - Minimax search

- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search

- Discussion

Generic Search Template: Pseudocode

- Primary data element is a Node, which a tuple of the form

$$(state, pathFromStartState, pathCost).$$

Generic Search Template: Pseudocode

- Primary data element is a Node, which a tuple of the form

$$(state, pathFromStartState, pathCost).$$

- At every stage of the search,
- some states have been explored
- some states remain unexplored, and
- The *Frontier* is a set of nodes due for imminent expansion.

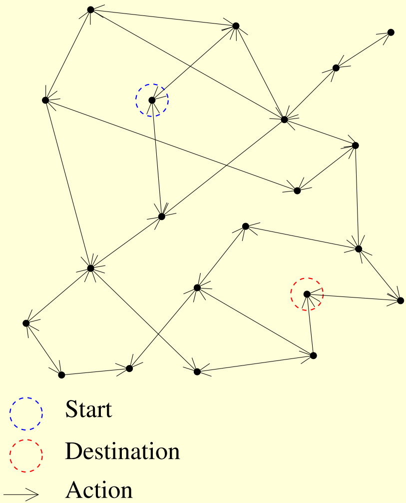
Generic Search Template: Pseudocode

$Frontier \leftarrow \{Node(startState, (startState), 0)\}$.
Repeat for ever:
    Select a node n from $Frontier$.
    //Expand n.
    If $isGoal(n.state)$:
        Return n.
    For each action a available from $n.state$:
        $s \leftarrow NextState(n.state, a)$.
        $c \leftarrow Cost(n.state, a)$.
        $n' \leftarrow Node(s, n.path + (a, s), n.pathCost + c)$.
        Merge n' with $Frontier$.//Typically insertion; might also allow deletions.

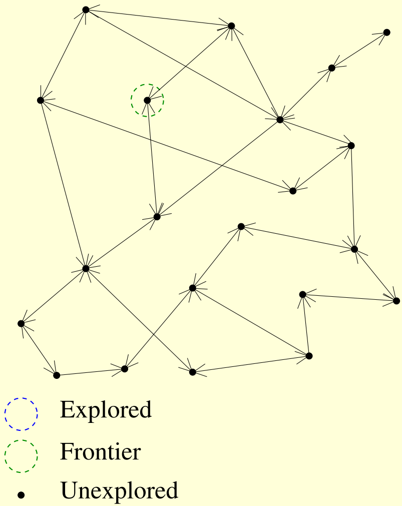
Generic Search Template: Pseudocode

$Frontier \leftarrow \{Node(startState, (startState), 0)\}$.
Repeat for ever:
    Select a node n from $Frontier$.//Which one?
    //Expand n.
    If $isGoal(n.state)$:
        Return n.
    For each action a available from $n.state$:
        $s \leftarrow NextState(n.state, a)$.
        $c \leftarrow Cost(n.state, a)$.
        $n' \leftarrow Node(s, n.path + (a, s), n.pathCost + c)$.
        Merge n' with $Frontier$.//Typically insertion; might also allow deletions.

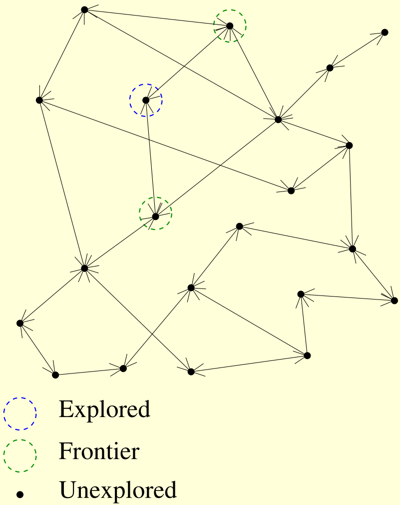
Generic Search Template: Illustration



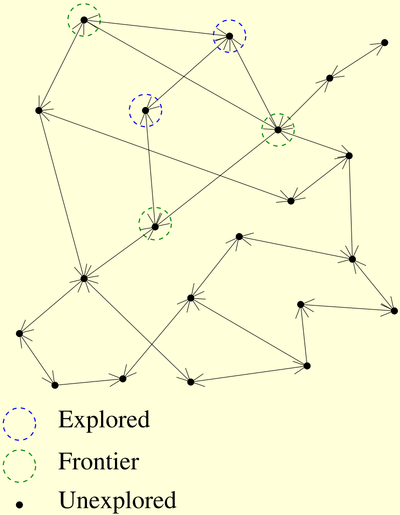
Generic Search Template: Illustration



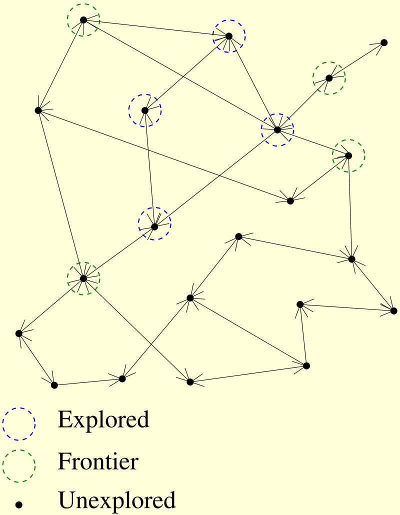
Generic Search Template: Illustration



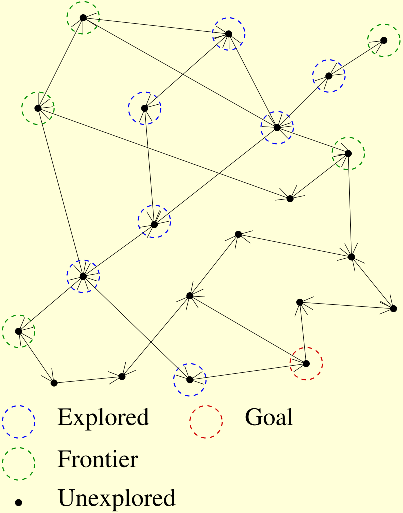
Generic Search Template: Illustration



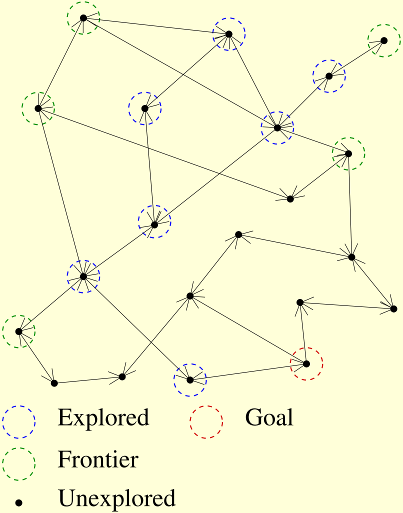
Generic Search Template: Illustration



Generic Search Template: Illustration



Generic Search Template: Illustration



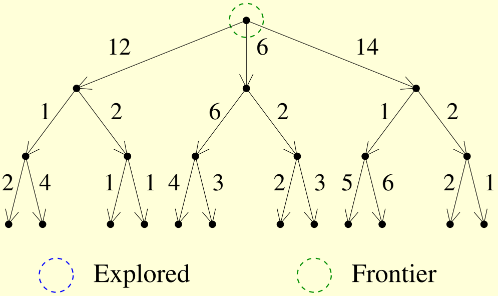
How did we decide which frontier nodes to expand?

Search

- Classical search
  - Problem instances
  - Generic search template
  - Uninformed search
  - Informed search (a.k.a. heuristic search)
  - Minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Discussion

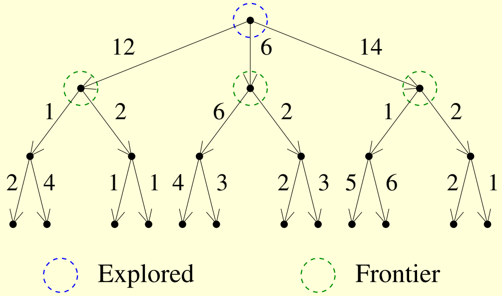
Depth-first Search (DFS)

Expand frontier node with longest path from start state.



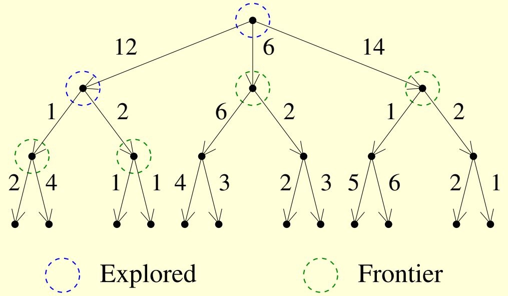
Depth-first Search (DFS)

Expand frontier node with longest path from start state.



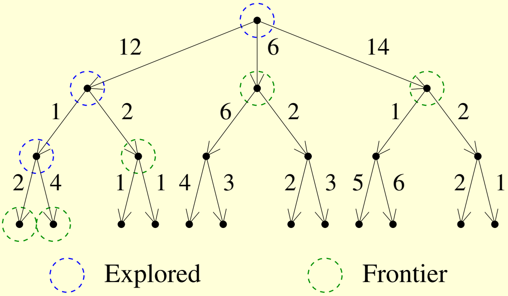
Depth-first Search (DFS)

Expand frontier node with longest path from start state.



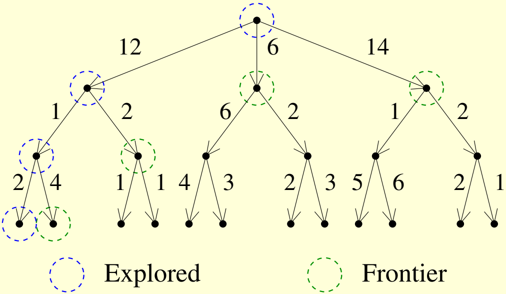
Depth-first Search (DFS)

Expand frontier node with longest path from start state.



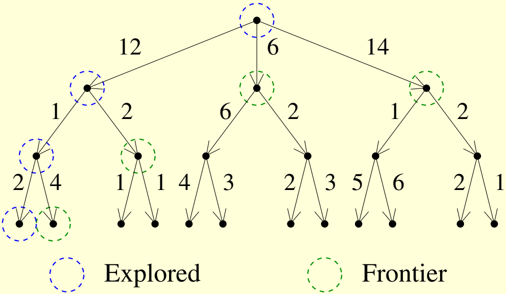
Depth-first Search (DFS)

Expand frontier node with longest path from start state.



Depth-first Search (DFS)

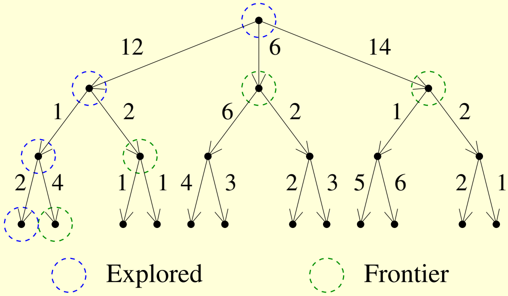
Expand frontier node with longest path from start state.



- Frontier treated like a stack (LIFO).

Depth-first Search (DFS)

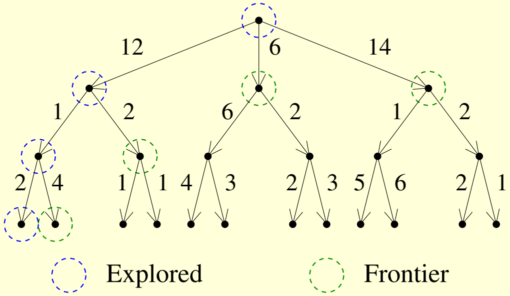
Expand frontier node with longest path from start state.



- Frontier treated like a stack (LIFO).
- No need to explicitly maintain frontier (construct on-line).

Depth-first Search (DFS)

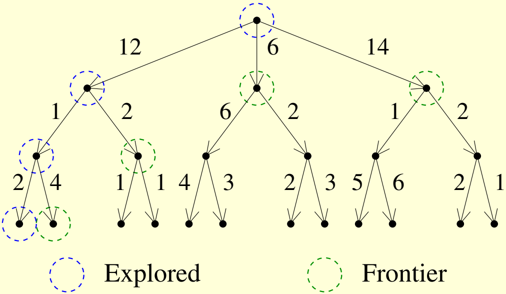
Expand frontier node with longest path from start state.



- Frontier treated like a stack (LIFO).
- No need to explicitly maintain frontier (construct on-line).
- Guaranteed to terminate on finite search instances.

Depth-first Search (DFS)

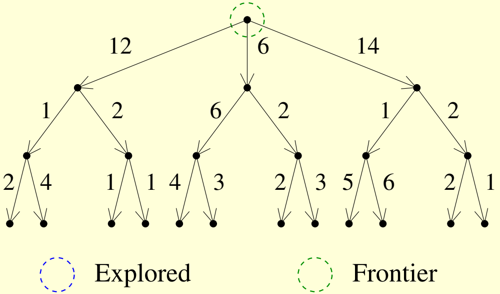
Expand frontier node with longest path from start state.



- Frontier treated like a stack (LIFO).
- No need to explicitly maintain frontier (construct on-line).
- Guaranteed to terminate on finite search instances.
- Memory requirement linear in depth d.

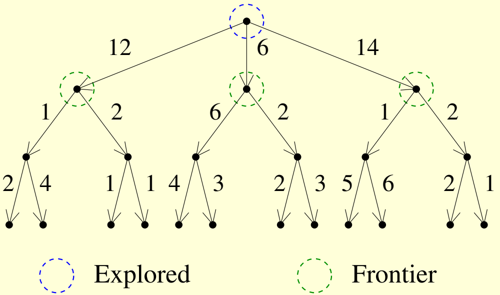
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



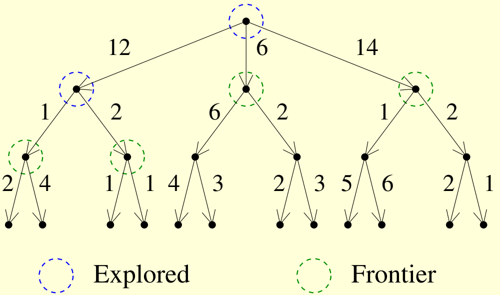
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



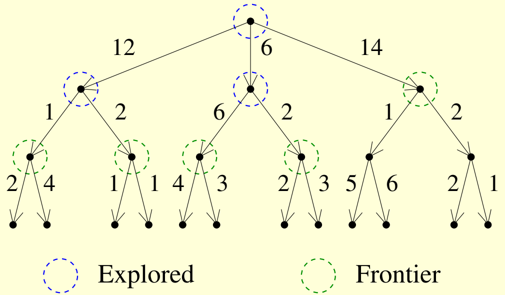
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



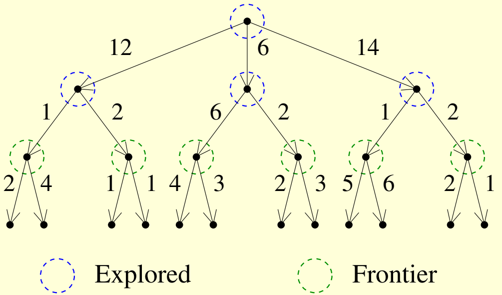
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



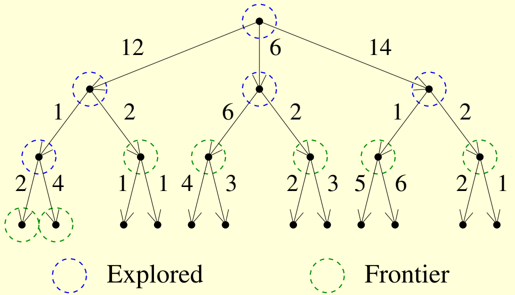
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



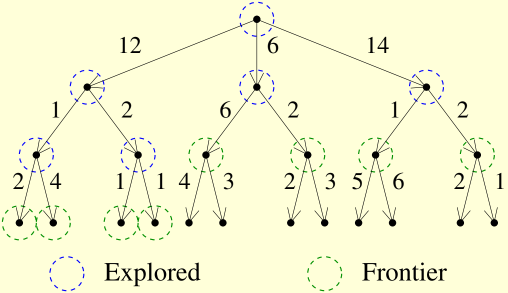
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



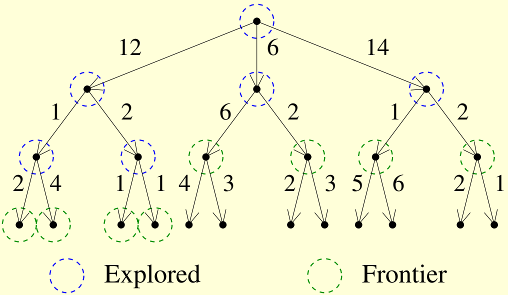
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



Breadth-first Search (BFS)

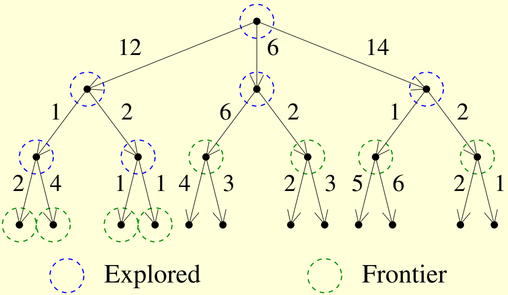
Expand frontier node with shortest path from start state.



- Frontier treated like a queue (FIFO).

Breadth-first Search (BFS)

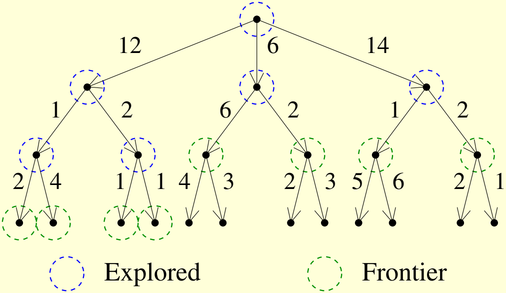
Expand frontier node with shortest path from start state.



- Frontier treated like a queue (FIFO).
- Guaranteed to terminate if search depth is finite.

Breadth-first Search (BFS)

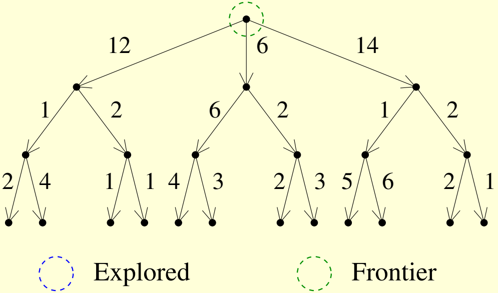
Expand frontier node with shortest path from start state.



- Frontier treated like a queue (FIFO).
- Guaranteed to terminate if search depth is finite.
- Memory requirement $O(b^d)$.

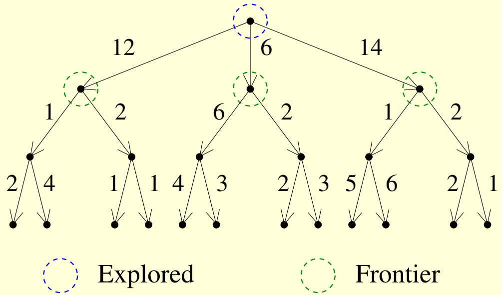
Lowest-cost-first Search (LCFS)

Expand frontier node with lowest path-cost from start state.



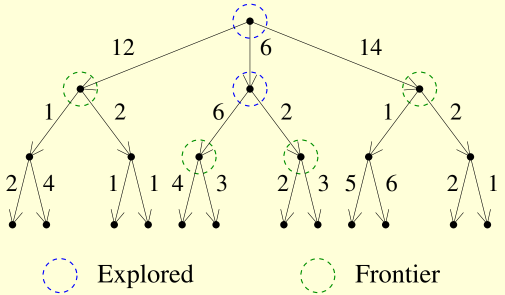
Lowest-cost-first Search (LCFS)

Expand frontier node with lowest path-cost from start state.



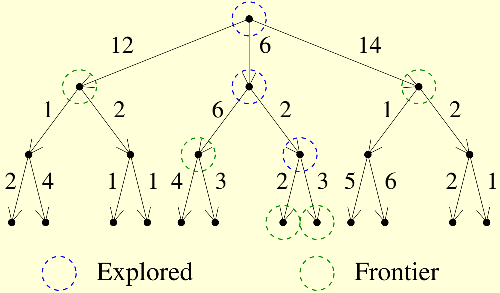
Lowest-cost-first Search (LCFS)

Expand frontier node with lowest path-cost from start state.



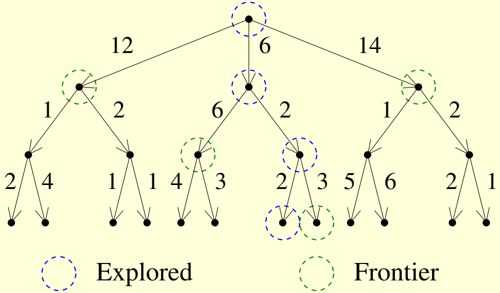
Lowest-cost-first Search (LCFS)

Expand frontier node with lowest path-cost from start state.



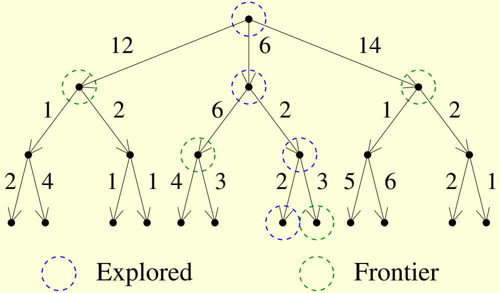
Lowest-cost-first Search (LCFS)

Expand frontier node with lowest path-cost from start state.



Lowest-cost-first Search (LCFS)

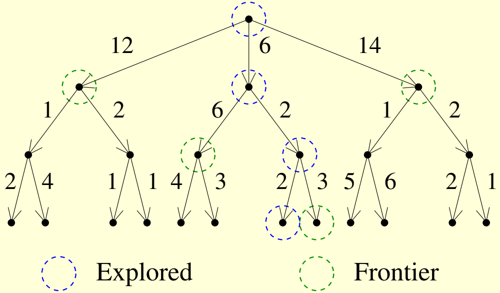
Expand frontier node with lowest path-cost from start state.



- For node n, denote path-cost from start state $g(n)$. Frontier treated as priority queue based on $g(n)$.

Lowest-cost-first Search (LCFS)

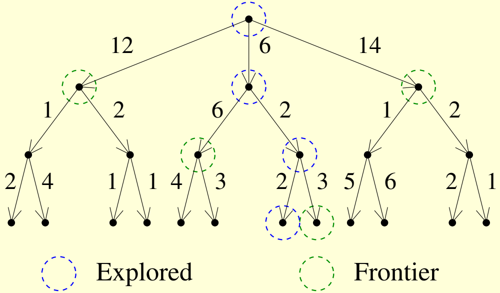
Expand frontier node with lowest path-cost from start state.



- For node n, denote path-cost from start state $g(n)$. Frontier treated as priority queue based on $g(n)$.
- Guaranteed to terminate if search depth is finite and each cost exceeds $\epsilon > 0$.

Lowest-cost-first Search (LCFS)

Expand frontier node with lowest path-cost from start state.



- For node n, denote path-cost from start state $g(n)$. Frontier treated as priority queue based on $g(n)$.
- Guaranteed to terminate if search depth is finite and each cost exceeds $\epsilon > 0$.
- Memory requirement depends heavily on instance.

Search

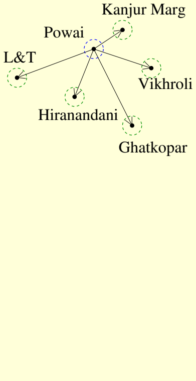
- Classical search
  - Problem instances
  - Generic search template
  - Uninformed search
  - Informed search (a.k.a. heuristic search)
  - Minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Discussion

Incorporating Domain Knowledge into Search

- Have to travel from Powai to Mahim.

Incorporating Domain Knowledge into Search

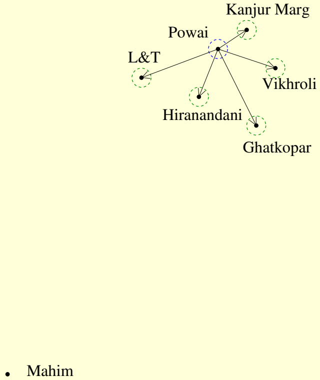
- Have to travel from Powai to Mahim.



- First you expand the Powai node.

Incorporating Domain Knowledge into Search

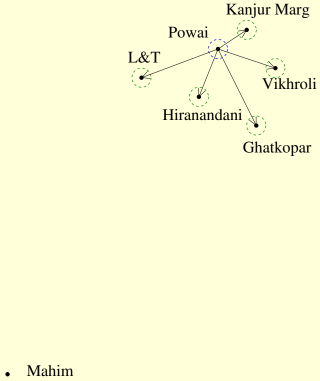
- Have to travel from Powai to Mahim.



- First you expand the Powai node.
- Which node will you expand next?

Incorporating Domain Knowledge into Search

- Have to travel from Powai to Mahim.



- First you expand the Powai node.
- Which node will you expand next?
- L&T and Hiranandani are geographically closer to Mahim: should that count?

Heuristic Functions and A* Search Algorithm

- A heuristic function $h(n)$ is a guess of $c^\star(n)$, the optimal path-cost-to-goal of (the state in) node n.

Heuristic Functions and A* Search Algorithm

- A heuristic function $h(n)$ is a guess of $c^\star(n)$, the optimal path-cost-to-goal of (the state in) node n.
- $h(n)$ is usually easy to compute. On the previous slide, we implicitly used straight line distance:

$$h(n) = \sqrt{(n.state.x - Mahim.x)^2 + (n.state.y - Mahim.y)^2}.$$

Heuristic Functions and A* Search Algorithm

- A heuristic function $h(n)$ is a guess of $c^\star(n)$, the optimal path-cost-to-goal of (the state in) node n.
- $h(n)$ is usually easy to compute. On the previous slide, we implicitly used straight line distance:

$$h(n) = \sqrt{(n.state.x - Mahim.x)^2 + (n.state.y - Mahim.y)^2}.$$

- Recall that in LCFS, we expand $\text{argmin}_{n \in \text{Frontier}}\, g(n)$.

Heuristic Functions and A* Search Algorithm

- A heuristic function $h(n)$ is a guess of $c^\star(n)$, the optimal path-cost-to-goal of (the state in) node n.
- $h(n)$ is usually easy to compute. On the previous slide, we implicitly used straight line distance:

$$h(n) = \sqrt{(n.state.x - Mahim.x)^2 + (n.state.y - Mahim.y)^2}.$$

- Recall that in LCFS, we expand $\mathrm{argmin}_{n \in \text{Frontier}}\, g(n)$.
- In A* search, we expand $\mathrm{argmin}_{n \in \text{Frontier}}\, (g(n) + h(n))$.

Heuristic Functions and A* Search Algorithm

- A heuristic function $h(n)$ is a guess of $c^\star(n)$, the optimal path-cost-to-goal of (the state in) node n.
- $h(n)$ is usually easy to compute. On the previous slide, we implicitly used straight line distance:

$$h(n) = \sqrt{(n.state.x - Mahim.x)^2 + (n.state.y - Mahim.y)^2}.$$

- Recall that in LCFS, we expand $\operatorname{argmin}_{n \in \text{Frontier}} g(n)$.
- In A* search, we expand $\operatorname{argmin}_{n \in \text{Frontier}} (g(n) + h(n))$.

- $g(n)$ summarises the past (known); $h(n)$ anticipates the future (unknown).

Heuristic Functions and A* Search Algorithm

- A heuristic function $h(n)$ is a guess of $c^\star(n)$, the optimal path-cost-to-goal of (the state in) node n.
- $h(n)$ is usually easy to compute. On the previous slide, we implicitly used straight line distance:

$$h(n) = \sqrt{(n.state.x - Mahim.x)^2 + (n.state.y - Mahim.y)^2}.$$

- Recall that in LCFS, we expand $\text{argmin}_{n \in \text{Frontier}}\, g(n)$.
- In A* search, we expand $\text{argmin}_{n \in \text{Frontier}}\, (g(n) + h(n))$.

- $g(n)$ summarises the past (known); $h(n)$ anticipates the future (unknown).
- The addition of $h(n)$ makes A* an informed or heuristic search algorithm.

Heuristic Functions and A* Search Algorithm

- A heuristic function $h(n)$ is a guess of $c^\star(n)$, the optimal path-cost-to-goal of (the state in) node n.
- $h(n)$ is usually easy to compute. On the previous slide, we implicitly used straight line distance:

$$h(n) = \sqrt{(n.state.x - Mahim.x)^2 + (n.state.y - Mahim.y)^2}.$$

- Recall that in LCFS, we expand $\operatorname{argmin}_{n \in \text{Frontier}} g(n)$.
- In A* search, we expand $\operatorname{argmin}_{n \in \text{Frontier}} (g(n) + h(n))$.

- $g(n)$ summarises the past (known); $h(n)$ anticipates the future (unknown).
- The addition of $h(n)$ makes A* an informed or heuristic search algorithm.
- A* search originally conceived for robotic path planning.

Admissible Heuristics

- A heuristic h is admissible if for all nodes n,

$$0 \leq h(n) \leq c^\star(n),$$

where $c^\star(n)$ is the optimal cost-to-goal of $n.state$.

Admissible Heuristics

- A heuristic h is admissible if for all nodes n,

$$0 \leq h(n) \leq c^\star(n),$$

  where $c^\star(n)$ is the optimal cost-to-goal of *n.state*.

- **Key result**. If A* search is run using an admissible heuristic (and some minor technical conditions hold), then the first goal node it expands will have optimal path-cost from the start state (and the algorithm can terminate).

Admissible Heuristics

- A heuristic h is admissible if for all nodes n,

$$0 \leq h(n) \leq c^\star(n),$$

  where $c^\star(n)$ is the optimal cost-to-goal of $n.state$.

- **Key result**. If A* search is run using an admissible heuristic (and some minor technical conditions hold), then the first goal node it expands will have optimal path-cost from the start state (and the algorithm can terminate).

- Is straight line distance an admissible heuristic for navigation?

Admissible Heuristics

- A heuristic h is admissible if for all nodes n,

$$0 \leq h(n) \leq c^{\star}(n),$$

where $c^{\star}(n)$ is the optimal cost-to-goal of $n.state$.

- **Key result**. If A* search is run using an admissible heuristic (and some minor technical conditions hold), then the first goal node it expands will have optimal path-cost from the start state (and the algorithm can terminate).

- Is straight line distance an admissible heuristic for navigation? Yes.

Admissible Heuristics

- A heuristic h is admissible if for all nodes n,

$$0 \leq h(n) \leq c^{\star}(n),$$

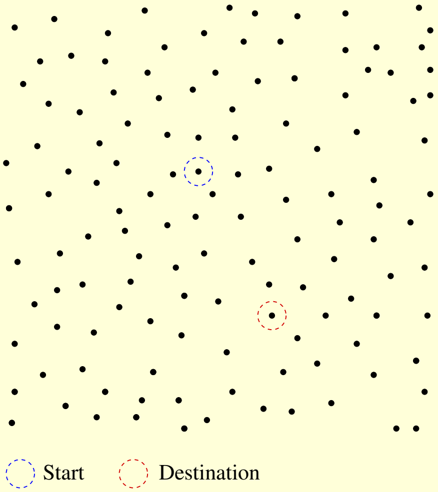
  where $c^{\star}(n)$ is the optimal cost-to-goal of $n.state$.

- **Key result**. If $A^{\star}$ search is run using an admissible heuristic (and some minor technical conditions hold), then the first goal node it expands will have optimal path-cost from the start state (and the algorithm can terminate).

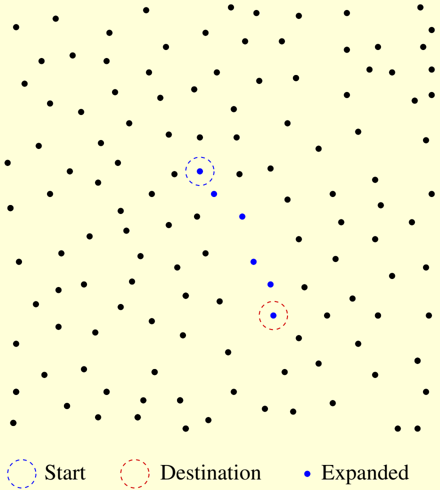
- Is straight line distance an admissible heuristic for navigation? Yes.

- For a given task, which is the best heuristic function to use?

Effect of Heuristic

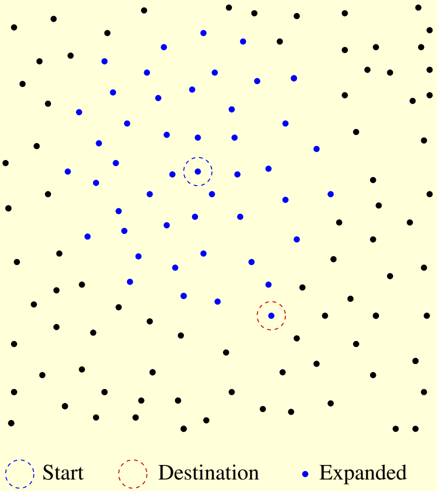


Effect of Heuristic



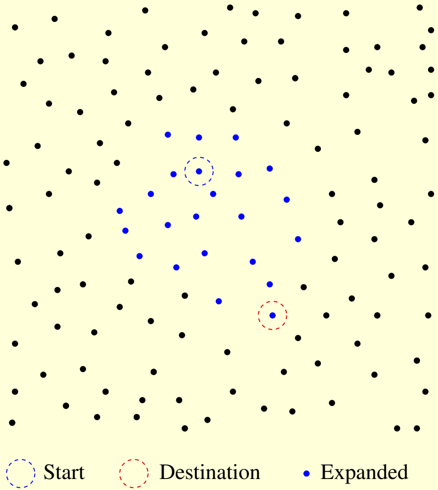
$h(n) = c^{\star}(n)$. Will only expand nodes along optimal path!
Unfortunately $c^{\star}(n)$ is not known!

Effect of Heuristic



$h(n) = 0$. Identical to LCFS.

Effect of Heuristic



Intermediate/typical $h(n)$ expands fewer nodes than LCFS.

Admissible Heuristics

- How to design an effective admissible heuristic for a task?

Admissible Heuristics

- How to design an effective admissible heuristic for a task? For many tasks people have already done so. A general strategy is to solve the task with relaxed constraints.

Admissible Heuristics

- How to design an effective admissible heuristic for a task? For many tasks people have already done so. A general strategy is to solve the task with relaxed constraints.

- What's a good heuristic for 15-puzzle?

Start state

13	2	3	12
9	11	1	10
•	5	4	14
15	8	7	6

- - - - - - - - - - ->

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	•

Goal state

Admissible Heuristics

- How to design an effective admissible heuristic for a task? For many tasks people have already done so. A general strategy is to solve the task with relaxed constraints.

- What's a good heuristic for 15-puzzle?

Start state

13	2	3	12
9	11	1	10
•	5	4	14
15	8	7	6

------------->

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	•

Goal state

Sum of Manhattan distances between each number's position in start state and its position in goal state.

Admissible Heuristics

- How to design an effective admissible heuristic for a task? For many tasks people have already done so. A general strategy is to solve the task with relaxed constraints.

- What's a good heuristic for 15-puzzle?

Start state

13	2	3	12
9	11	1	10
•	5	4	14
15	8	7	6

------------->

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	•

Goal state

Sum of Manhattan distances between each number's position in start state and its position in goal state.

- Can we make do with inadmissible heuristics?

Admissible Heuristics

- How to design an effective admissible heuristic for a task? For many tasks people have already done so. A general strategy is to solve the task with relaxed constraints.

- What's a good heuristic for 15-puzzle?

Start state

13	2	3	12
9	11	1	10
•	5	4	14
15	8	7	6

------------->

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	•

Goal state

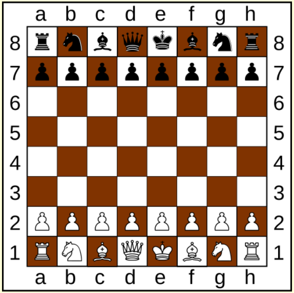
Sum of Manhattan distances between each number's position in start state and its position in goal state.

- Can we make do with inadmissible heuristics? Yes—example coming up in next section. But try to avoid.

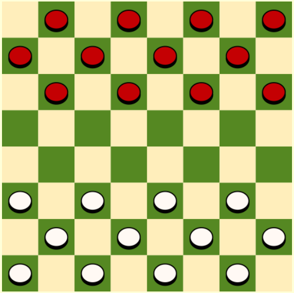
Search

- Classical search
  - Problem instances
  - Generic search template
  - Uninformed search
  - Informed search (a.k.a. heuristic search)
  - Minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Discussion

Search and Games



[1]

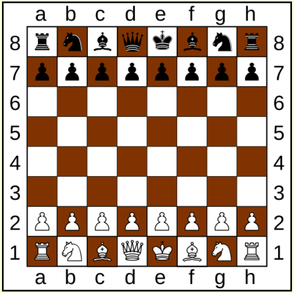


[2]

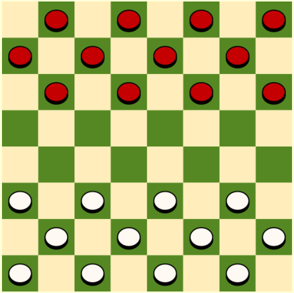
[1] https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg. CC image courtesy of ILA-boy on WikiMedia Commons licensed under CC-BY-SA-3.0.

[2] https://commons.wikimedia.org/wiki/File:Draughts.svg.

Search and Games



Chess [1]

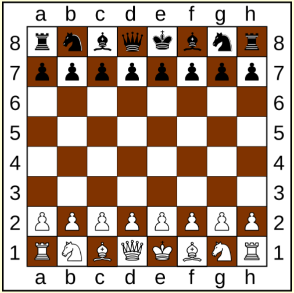


Checkers/Draughts [2]

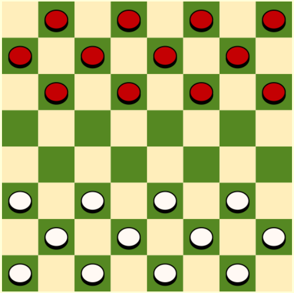
[1] `https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg`. CC image courtesy of ILA-boy on WikiMedia Commons licensed under CC-BY-SA-3.0.

[2] `https://commons.wikimedia.org/wiki/File:Draughts.svg`.

Search and Games



Chess [1]



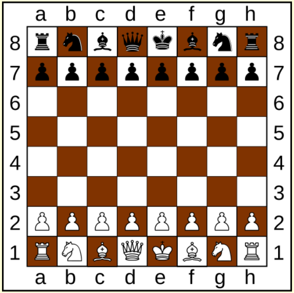
Checkers/Draughts [2]

- Winning at chess/checkers: a search problem?

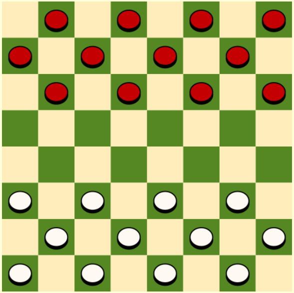
[1] https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg. CC image courtesy of ILA-boy on WikiMedia Commons licensed under CC-BY-SA-3.0.

[2] https://commons.wikimedia.org/wiki/File:Draughts.svg.

Search and Games



Chess [1]



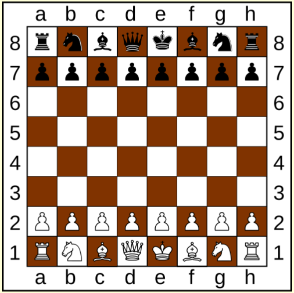
Checkers/Draughts [2]

- Winning at chess/checkers: a search problem?
- What's the main difference from our previous examples?

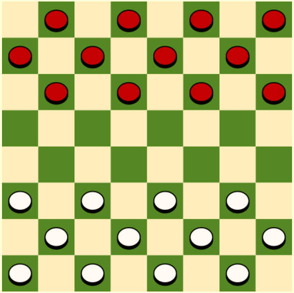
[1] https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg. CC image courtesy of ILA-boy on WikiMedia Commons licensed under CC-BY-SA-3.0.

[2] https://commons.wikimedia.org/wiki/File:Draughts.svg.

Search and Games



Chess [1]



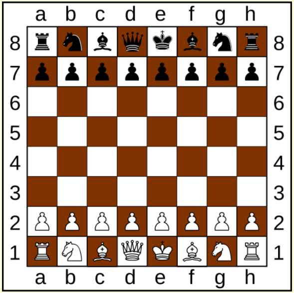
Checkers/Draughts [2]

- Winning at chess/checkers: a search problem?
- What's the main difference from our previous examples?
  There's another player!

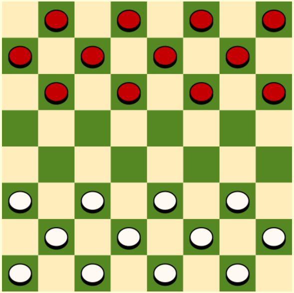
[1] https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg. CC image courtesy of ILA-boy on WikiMedia Commons licensed under CC-BY-SA-3.0.

[2] https://commons.wikimedia.org/wiki/File:Draughts.svg.

Search and Games



Chess [1]



Checkers/Draughts [2]

- Winning at chess/checkers: a search problem?
- What's the main difference from our previous examples?
  There's another player!
- Instances of turn-based two player zero-sum games.

[1] https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg. CC image courtesy of ILA-boy on WikiMedia Commons licensed under CC-BY-SA-3.0.

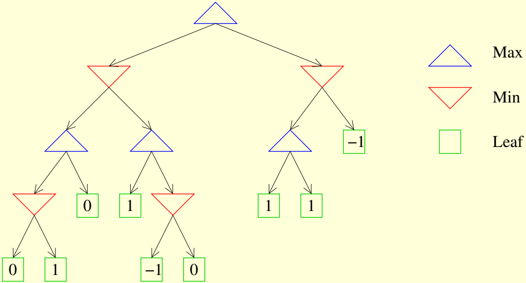
[2] https://commons.wikimedia.org/wiki/File:Draughts.svg.

Game Tree

- Assume turn-taking zero sum game played by Max and Min.
- Action costs usually taken as 0, but leaves have value

  -1 (Max loses), 0 (draw), 1 (Max wins).
- Value of Max node is maximum of values of children.

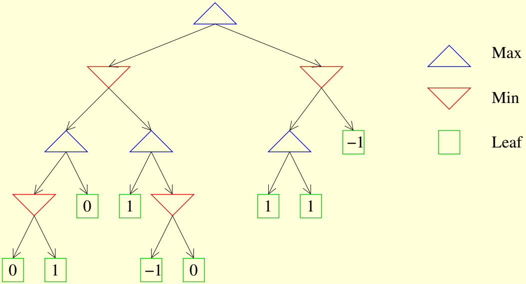
  Value of Min node is minimum of values of children.



Game Tree

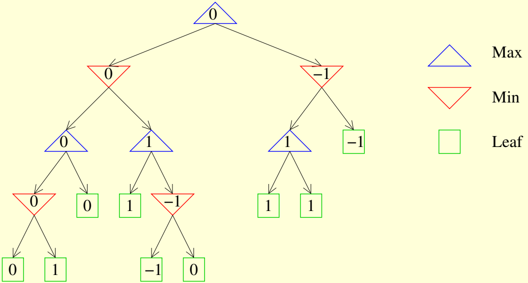
- Assume turn-taking zero sum game played by Max and Min.
- Action costs usually taken as 0, but leaves have value

  -1 (Max loses), 0 (draw), 1 (Max wins).
- Value of Max node is maximum of values of children.
  Value of Min node is minimum of values of children.
- What is the value of the root node?



Game Tree

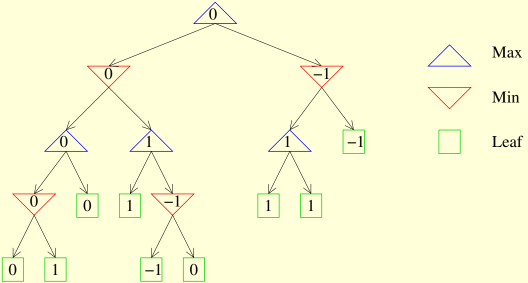
- Assume turn-taking zero sum game played by Max and Min.
- Action costs usually taken as 0, but leaves have value
  -1 (Max loses), 0 (draw), 1 (Max wins).
- Value of Max node is maximum of values of children.
  Value of Min node is minimum of values of children.
- What is the value of the root node?



Game Tree

- Assume turn-taking zero sum game played by Max and Min.
- Action costs usually taken as 0, but leaves have value

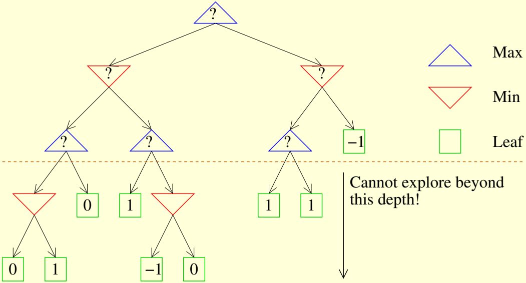
  -1 (Max loses), 0 (draw), 1 (Max wins).
- Value of Max node is maximum of values of children.
  Value of Min node is minimum of values of children.
- What is the value of the root node?



- In 2007, a massive, long-running computation concluded that the value of the root node for Checkers is 0 (draw).

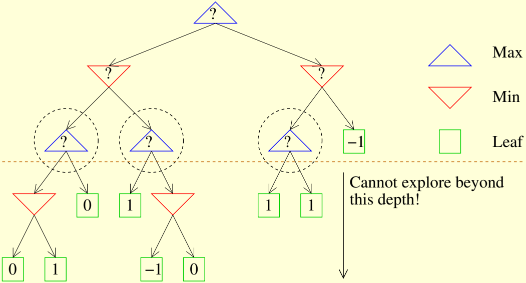
Evaluation Function

- The Checkers tree has $\approx 10^{40}$ nodes; Chess has $\approx 10^{120}$.
  Infeasible to solve!



Evaluation Function

- The Checkers tree has $\approx 10^{40}$ nodes; Chess has $\approx 10^{120}$. Infeasible to solve!



- At some depth d from current node, estimate node value using features.
- For example, in Chess, set evaluation as

$$w_1 \times \text{Material diff.} + w_2 \times \text{King safety} + w_3 \times \text{pawn strength} + \ldots.$$

- Weights $w_1, w_2, w_3, \ldots$ are tuned or learned.

Search

- Classical search
  - Problem instances
  - Generic search template
  - Uninformed search
  - Informed search (a.k.a. heuristic search)
  - Minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Discussion

Decision-time planning: Problem

- So far we have assumed that an agent's learning algorithm produces π or Q as output. While acting on-line, the agent just needs a "look up" or associative "forward pass" from any state s to obtain its action.

Decision-time planning: Problem

- So far we have assumed that an agent's learning algorithm produces π or Q as output. While acting on-line, the agent just needs a "look up" or associative "forward pass" from any state s to obtain its action.
- Sometimes π or Q might be difficult to learn in compact form, but a model $M = (T, R)$ (given or learned, exact or approximate) might be available.

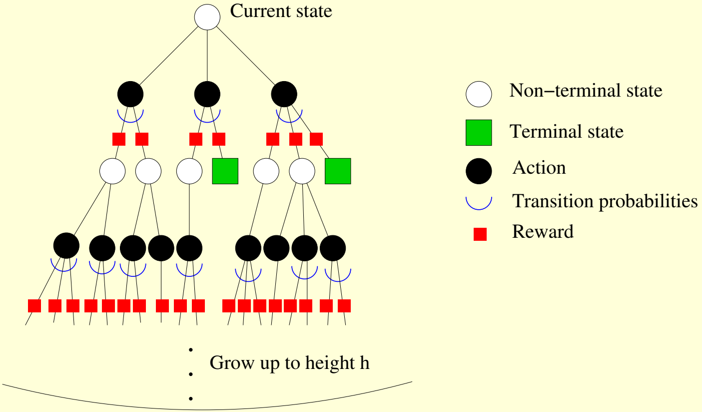
Decision-time planning: Problem

- So far we have assumed that an agent's learning algorithm produces π or Q as output. While acting on-line, the agent just needs a "look up" or associative "forward pass" from any state s to obtain its action.
- Sometimes π or Q might be difficult to learn in compact form, but a model $M = (T, R)$ (given or learned, exact or approximate) might be available.
- In decision-time planning, at every time step, we "imagine" possible futures emanating from the current state by using M, and use the computation to decide which action to take.

Decision-time planning: Problem

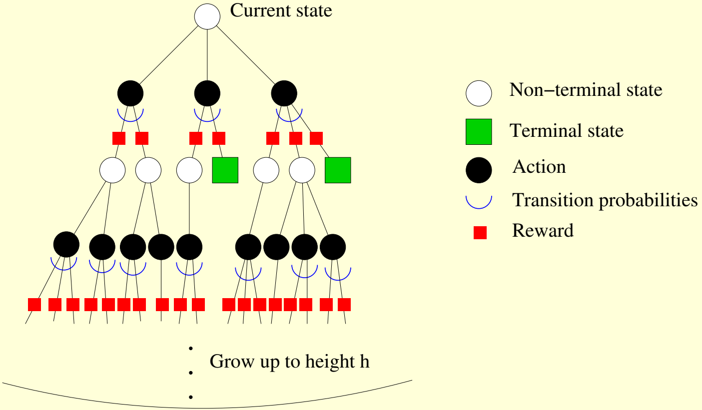
- So far we have assumed that an agent's learning algorithm produces π or Q as output. While acting on-line, the agent just needs a "look up" or associative "forward pass" from any state s to obtain its action.
- Sometimes π or Q might be difficult to learn in compact form, but a model $M = (T, R)$ (given or learned, exact or approximate) might be available.
- In decision-time planning, at every time step, we "imagine" possible futures emanating from the current state by using M, and use the computation to decide which action to take.
- How to rigorously do so?

Tree Search on MDPs



- Expectimax calculation. Set $Q^h \leftarrow \mathbf{0}$ //Leaves.
  For $d = h-1, h-2, \ldots, 0$://Bottom-up calculation.
  $V^d(s) \leftarrow \max_{a \in A} Q^{d+1}(s, a)$;
  $Q^d(s, a) \leftarrow \sum_{s' \in S} T(s, a, s')\{R(s, a, s') + \gamma V^d(s')\}$.

Tree Search on MDPs



- Need $h = \theta(\frac{1}{1-\gamma})$ for sufficient accuracy.
- With branching factor b, tree size is $\theta(b^h)$. Expensive!
- Often M is only a sampling model (not distribution model).
- Can we avoid expanding (clearly) inferior branches?

Search

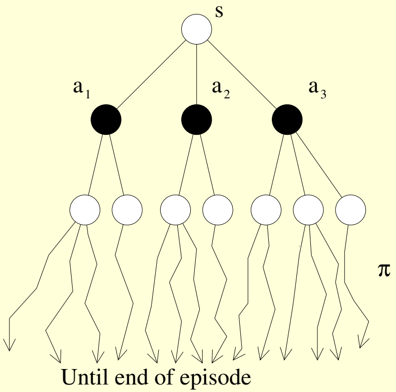
- Classical search
  - Problem instances
  - Generic search template
  - Uninformed search
  - Informed search (a.k.a. heuristic search)
  - Minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Discussion

Rollout Policies

- Suppose we have a (look-up) policy π.
- Let policy π' satisfy $\pi'(s) = \max_{a \in A} Q^{\pi}(s, a)$ for $s \in S$.
- By the policy improvement theorem, we know $\pi' \succeq \pi$.

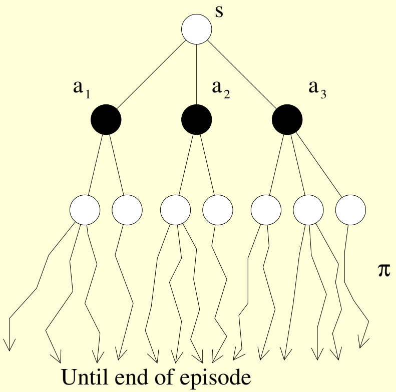
Rollout Policies

- Suppose we have a (look-up) policy π.
- Let policy π' satisfy $\pi'(s) = \max_{a \in A} Q^{\pi}(s, a)$ for $s \in S$.
- By the policy improvement theorem, we know $\pi' \succeq \pi$.
- We implement π' using Monte Carlo rollouts (through M).



Rollout Policies

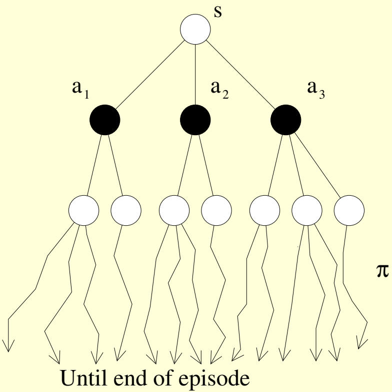
- Suppose we have a (look-up) policy π.
- Let policy π' satisfy $\pi'(s) = \max_{a \in A} Q^{\pi}(s, a)$ for $s \in S$.
- By the policy improvement theorem, we know $\pi' \succeq \pi$.
- We implement π' using Monte Carlo rollouts (through M).



- From current state s, for each action $a \in A$, generate N trajectories by taking a from s and thereafter following π.

Rollout Policies

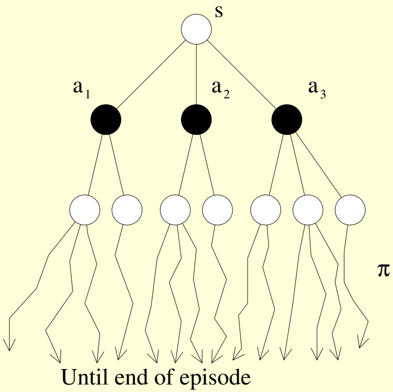
- Suppose we have a (look-up) policy π.
- Let policy π' satisfy $\pi'(s) = \max_{a \in A} Q^{\pi}(s, a)$ for $s \in S$.
- By the policy improvement theorem, we know $\pi' \succeq \pi$.
- We implement π' using Monte Carlo rollouts (through M).



- From current state s, for each action $a \in A$, generate N trajectories by taking a from s and thereafter following π.
- Set $\hat{Q}^{\pi}(s, a)$ as average of episodic returns.

Rollout Policies

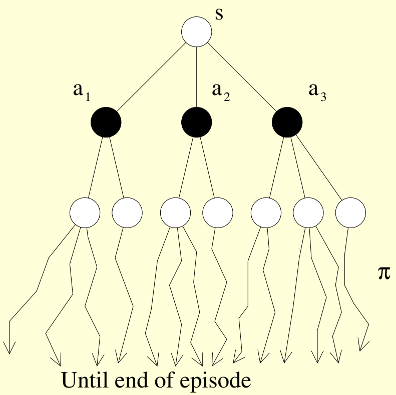
- Suppose we have a (look-up) policy π.
- Let policy π' satisfy $\pi'(s) = \max_{a \in A} Q^{\pi}(s, a)$ for $s \in S$.
- By the policy improvement theorem, we know $\pi' \succeq \pi$.
- We implement π' using Monte Carlo rollouts (through M).



- From current state s, for each action $a \in A$, generate N trajectories by taking a from s and thereafter following π.
- Set $\hat{Q}^{\pi}(s, a)$ as average of episodic returns.
- Take action $\pi'(s) = \text{argmax}_{a \in A} \hat{Q}^{\pi}(s, a)$.

Rollout Policies

- Suppose we have a (look-up) policy π.
- Let policy π' satisfy $\pi'(s) = \max_{a \in A} Q^{\pi}(s, a)$ for $s \in S$.
- By the policy improvement theorem, we know $\pi' \succeq \pi$.
- We implement π' using Monte Carlo rollouts (through M).



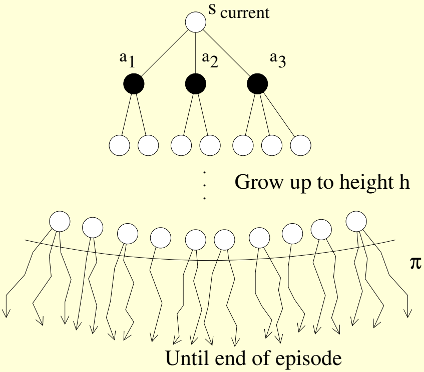
- From current state s, for each action $a \in A$, generate N trajectories by taking a from s and thereafter following π.
- Set $\hat{Q}^{\pi}(s, a)$ as average of episodic returns.
- Take action $\pi'(s) = \text{argmax}_{a \in A} \hat{Q}^{\pi}(s, a)$.
- Repeat same process from next state s'.

Search

- Classical search
  - Problem instances
  - Generic search template
  - Uninformed search
  - Informed search (a.k.a. heuristic search)
  - Minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Discussion

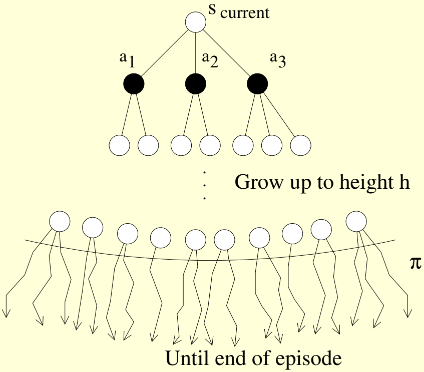
Monte Carlo Tree Search (UCT Algorithm)

- Build out a tree up to height h (say 5–10) from current state s_{current}. "Data" for the tree are samples returned by M.
- For (s, a) pairs reachable from s_{current} in $\leq h$ steps, maintain
  - $Q(s, a)$: average of returns of rollouts passing through (s, a).
  - $ucb(s, a) = Q(s, a) + C_p \sqrt{\frac{\ln(t)}{\text{visits}(s,a)}}$.



Monte Carlo Tree Search (UCT Algorithm)

- Build out a tree up to height h (say 5–10) from current state s_{current}. "Data" for the tree are samples returned by M.
- For (s, a) pairs reachable from s_{current} in $\leq h$ steps, maintain
  - $Q(s, a)$: average of returns of rollouts passing through (s, a).
  - $ucb(s, a) = Q(s, a) + C_p \sqrt{\frac{\ln(t)}{\text{visits}(s,a)}}$.

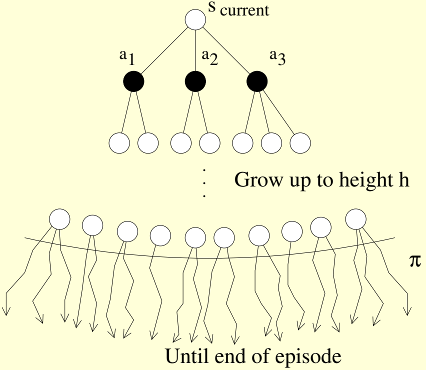


Repeat N times from s_{current}:

1. Generate trajectory by calling M. From stored state s, "take" $\text{argmax}_{a \in A}\, ucb(s, a)$; from leaf follow rollout policy π until end of episode.

Monte Carlo Tree Search (UCT Algorithm)

- Build out a tree up to height h (say 5–10) from current state s_{current}. "Data" for the tree are samples returned by M.
- For (s, a) pairs reachable from s_{current} in $\leq h$ steps, maintain
  - $Q(s, a)$: average of returns of rollouts passing through (s, a).
  - $ucb(s, a) = Q(s, a) + C_p \sqrt{\frac{\ln(t)}{\text{visits}(s,a)}}$.



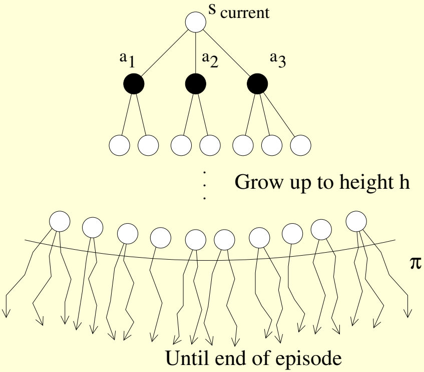
Repeat N times from s_{current}:

1. Generate trajectory by calling M. From stored state s, "take" $\text{argmax}_{a \in A}\, ucb(s, a)$; from leaf follow rollout policy π until end of episode.

2. Update Q, ucb for (s, a) pairs visited in trajectory.

Monte Carlo Tree Search (UCT Algorithm)

- Build out a tree up to height h (say 5–10) from current state s_{current}. "Data" for the tree are samples returned by M.
- For (s, a) pairs reachable from s_{current} in $\leq h$ steps, maintain
  - $Q(s, a)$: average of returns of rollouts passing through (s, a).
  - $ucb(s, a) = Q(s, a) + C_p \sqrt{\frac{\ln(t)}{\text{visits}(s,a)}}$.



Repeat N times from s_{current}:

1. Generate trajectory by calling M. From stored state s, "take" $\text{argmax}_{a \in A}\, ucb(s, a)$; from leaf follow rollout policy π until end of episode.

2. Update Q, ucb for (s, a) pairs visited in trajectory.

Take $\text{argmax}_{a \in A}\, ucb(s, a)$.

Monte Carlo Tree Search (UCT Algorithm)

- Main parameters of UCT: rollout policy π, search tree height h, number of rollouts N.
- π typically an associative/look-up policy, often even a random policy.
- Better guarantees as h is increased (if $N = \infty$).
- In practice N limited by available "think" time.

Monte Carlo Tree Search (UCT Algorithm)

- Main parameters of UCT: rollout policy π, search tree height h, number of rollouts N.
- π typically an associative/look-up policy, often even a random policy.
- Better guarantees as h is increased (if $N = \infty$).
- In practice N limited by available "think" time.
- C_p in the UCB formula needs to be large to deal with nonstationarity (from changes downstream).

Monte Carlo Tree Search (UCT Algorithm)

- Main parameters of UCT: rollout policy π, search tree height h, number of rollouts N.
- π typically an associative/look-up policy, often even a random policy.
- Better guarantees as h is increased (if $N = \infty$).
- In practice N limited by available “think” time.
- C_p in the UCB formula needs to be large to deal with nonstationarity (from changes downstream).
- In general there could be multiple paths to any particular stored (s, a) pair starting from s_{current}.

Monte Carlo Tree Search (UCT Algorithm)

- Main parameters of UCT: rollout policy π, search tree height h, number of rollouts N.
- π typically an associative/look-up policy, often even a random policy.
- Better guarantees as h is increased (if $N = \infty$).
- In practice N limited by available “think” time.
- C_p in the UCB formula needs to be large to deal with nonstationarity (from changes downstream).
- In general there could be multiple paths to any particular stored (s, a) pair starting from s_{current}.
- UCT focuses attention on rewarding regions of state space.
- Rollouts can easily be parallelised.
- Extremely successful algorithm in practice.

Search

- Classical search
  - Problem instances
  - Generic search template
  - Uninformed search
  - Informed search (a.k.a. heuristic search)
  - Minimax search

- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search

- Discussion

Search in AI/ML

- Heuristic search ("problem-solving") is among the earliest topics studied in AI.
- Applications: Theorem-proving, constraint satisfaction problems/integer programming, robotic path planning, logistics, Video games (movement of characters).
- $A^\star$ search (and variants such as $IDA^\star$) used widely in practice.
- Search is different from learning, although these two attributes of intelligence often come together.
- Main technical challenge: large (exponentially growing) number of states in most practical tasks.

Search in On-line Decision Making

- Key requirement: simulator (model).
- More computationally expensive than lookup of π or Q.
- MCTS with rollout policies an effective approach to handle stochasticity as well as large state spaces.
- Learning (say an evaluation function) can also help solution quality of search in practice.
- Proof of all these claims: AlphaGo!

Search in On-line Decision Making

- Key requirement: simulator (model).
- More computationally expensive than lookup of π or Q.
- MCTS with rollout policies an effective approach to handle stochasticity as well as large state spaces.
- Learning (say an evaluation function) can also help solution quality of search in practice.
- Proof of all these claims: AlphaGo!

- We'll cover more model-based methods, as well as AlphaGo, in upcoming lectures.