CS 747, Autumn 2022: Lecture 24

Shivaram Kalyanakrishnan

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Autumn 2022

Navigation System

How to go from IIT Bombay to Marine Drive?

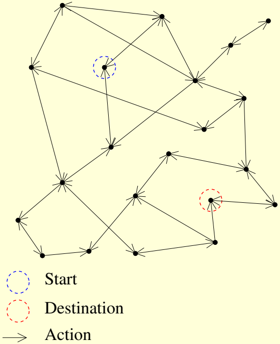
[1]

[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]



Start

Destination

Action

[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?

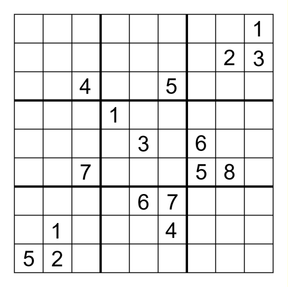
								1
							2	3
		4			5			
			1					
				3		6		
		7				5	8	
				6	7			
	1				4			
5	2							

Sudoku [1]

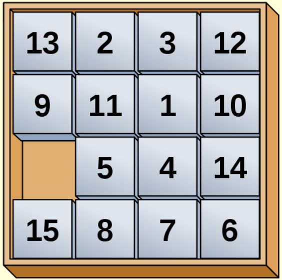
[1] https://upload.wikimedia.org/wikipedia/commons/e/eb/Sudoku_Puzzle_%28a_symmetrical_puzzle_with_17_clues%29.png.

Some Popular Puzzles

How to solve?



Sudoku [1]



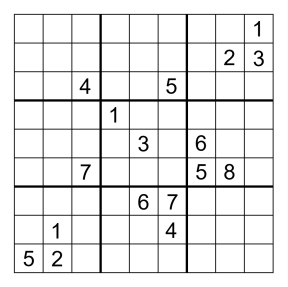
15-puzzle [2]

[1] https://upload.wikimedia.org/wikipedia/commons/e/eb/Sudoku_Puzzle_%28a_symmetrical_puzzle_with_17_clues%29.png.

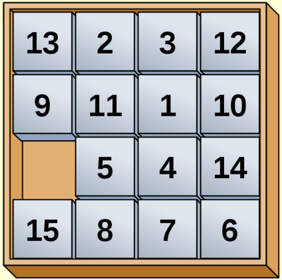
[2] https://commons.wikimedia.org/wiki/File:15-puzzle-solvable.svg.

Some Popular Puzzles

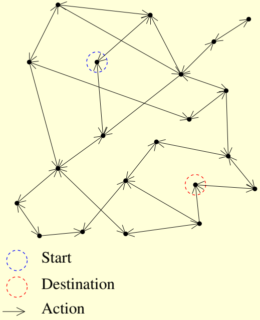
How to solve?



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

Classical Search

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

Classical Search

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

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

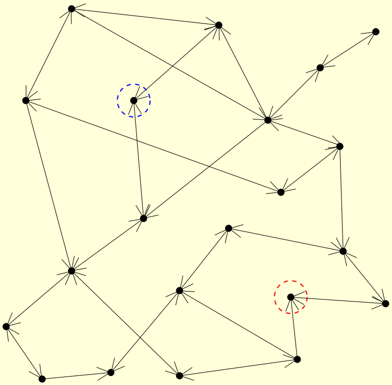
- Set of states, including designated start state.
- Set of actions available from each state.
- NextState(s, a) for each state s and action a.
- Cost(s, a) for each state s and action a (assumed ≥ 0).
- IsGoal(s) for each state s.

- **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 branching factor b.
- Length of optimal path to reach goal state is depth d of the search instance.

Problem Formulation: Navigation System



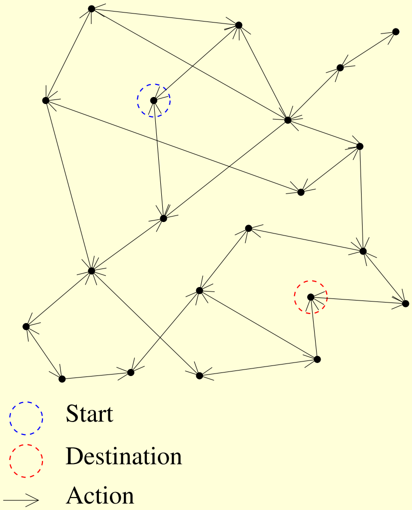
Start

Destination

Action

States?

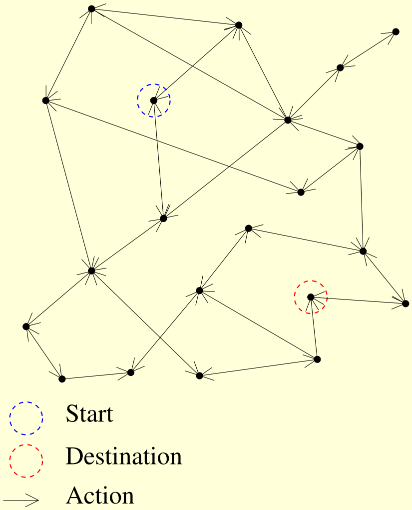
Problem Formulation: Navigation System



States?

Start state?

Problem Formulation: Navigation System

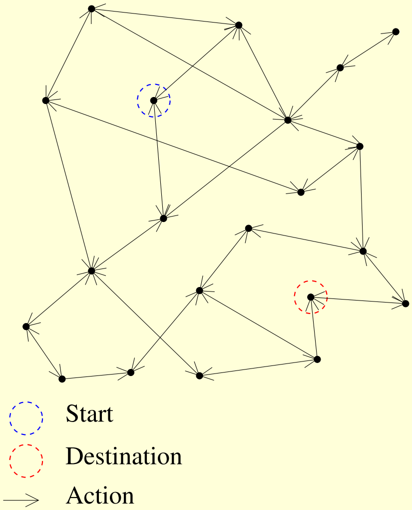


States?

Start state?

Actions?

Problem Formulation: Navigation System



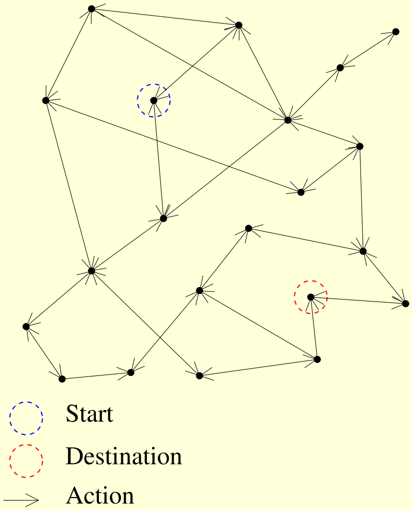
States?

Start state?

Actions?

NextState()?

Problem Formulation: Navigation System



States?

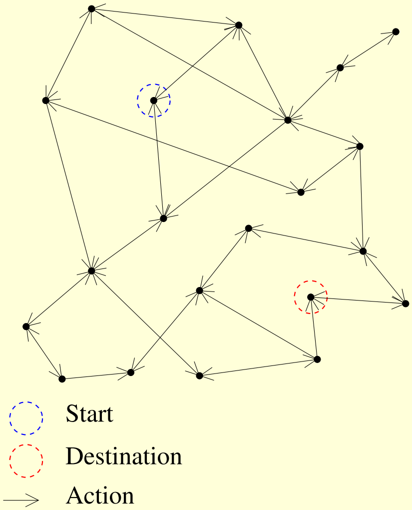
Start state?

Actions?

NextState()?

Cost()?

Problem Formulation: Navigation System



States?

Start state?

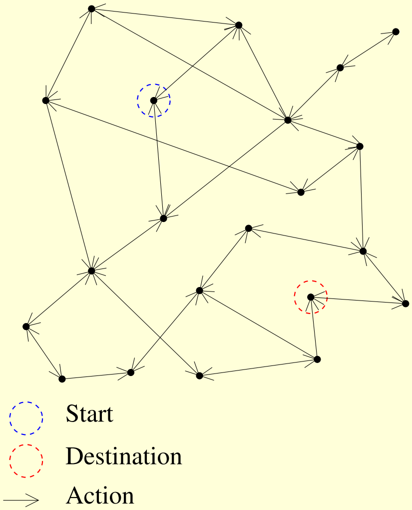
Actions?

NextState()?

Cost()?

IsGoal()?

Problem Formulation: Navigation System



States?

Start state?

Actions?

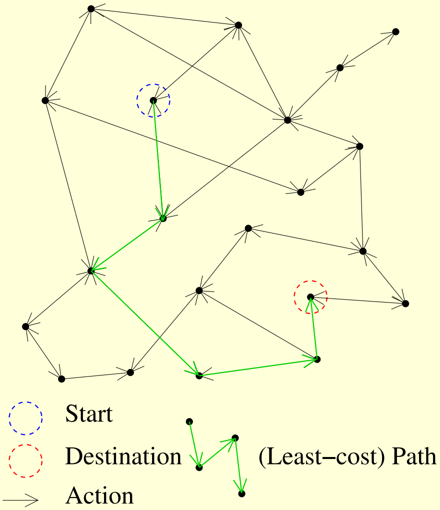
NextState()?

Cost()?

IsGoal()?

A solver needs to find the least-cost path.

Problem Formulation: Navigation System



States?

Start state?

Actions?

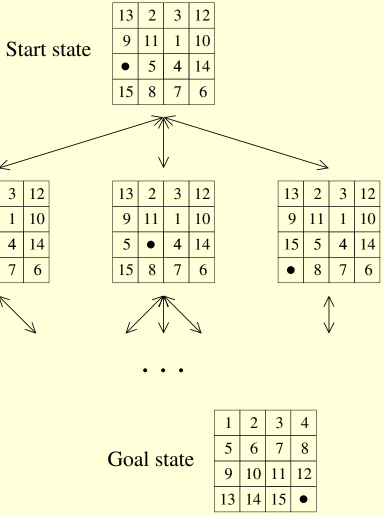
NextState()?

Cost()?

IsGoal()?

A solver needs to find the least-cost path.

Problem Formulation: 15 Puzzle



States?

Start state?

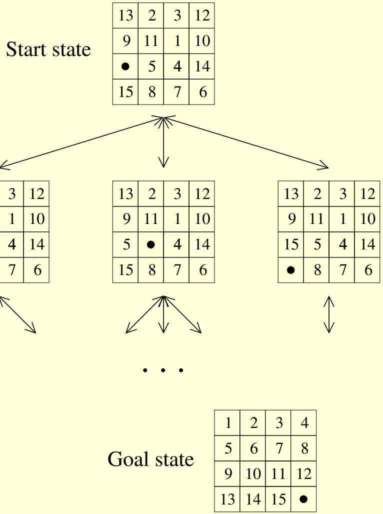
Actions?

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.

Classical Search

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

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.

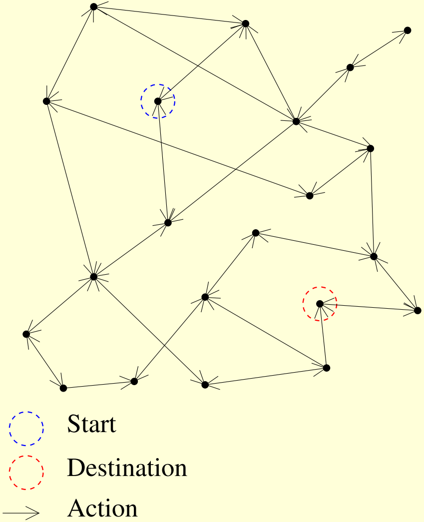
Generic Search Template: Pseudocode

$Frontier \leftarrow \{Node(startState, (startState), 0)\}$.
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.

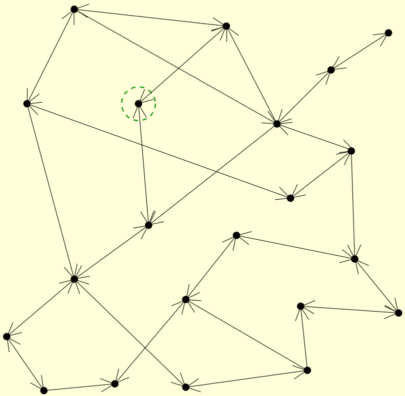
Generic Search Template: Pseudocode

```
Frontier ← {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 ← NextState(n.state, a).
        c ← Cost(n.state, a).
        n' ← 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

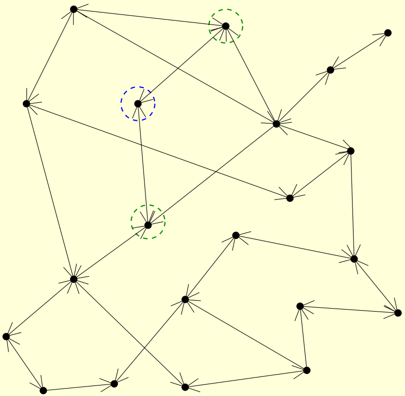


Explored

Frontier

Unexplored

Generic Search Template: Illustration

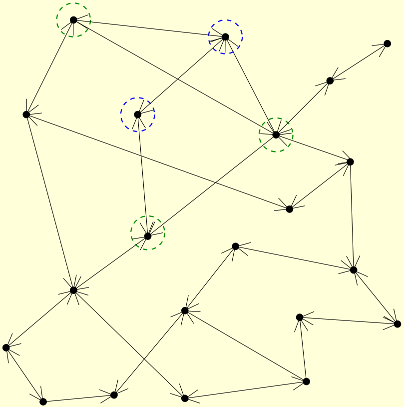


Explored

Frontier

Unexplored

Generic Search Template: Illustration

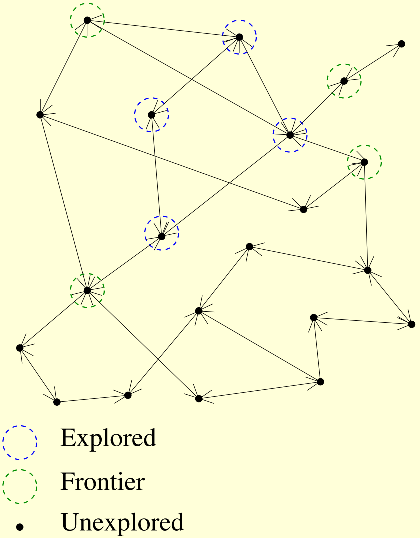


Explored

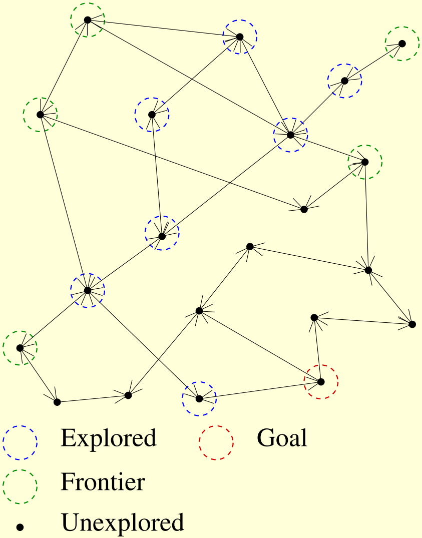
Frontier

Unexplored

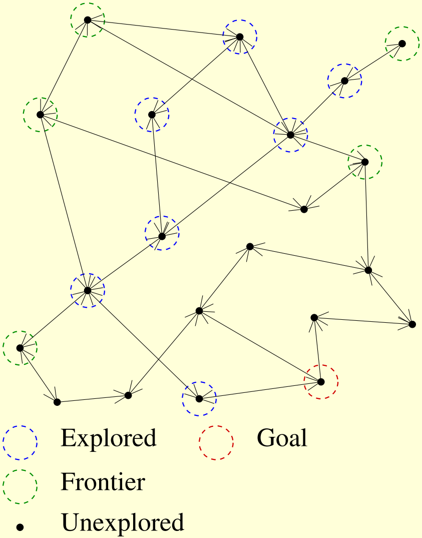
Generic Search Template: Illustration



Generic Search Template: Illustration



Generic Search Template: Illustration



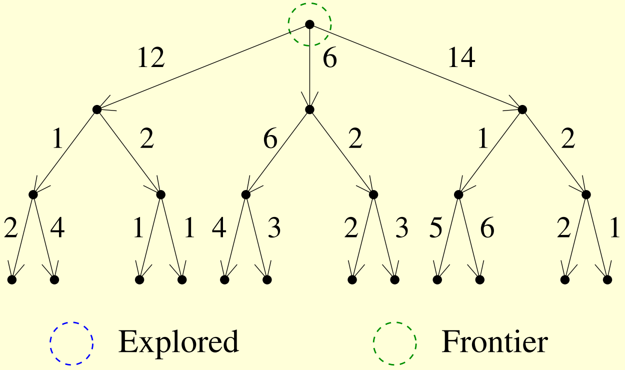
Which frontier node to expand?

Classical Search

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

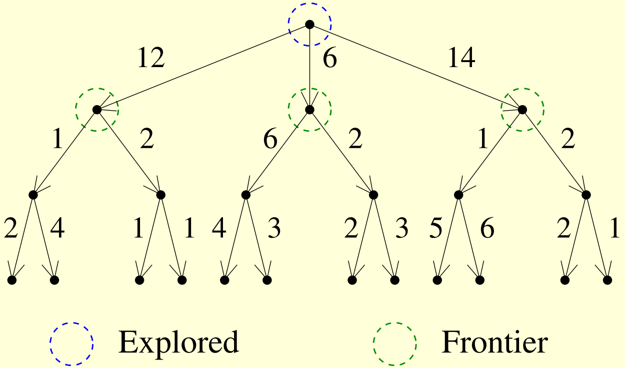
Depth-first Search (DFS)

Expand frontier node with longest path from start state.



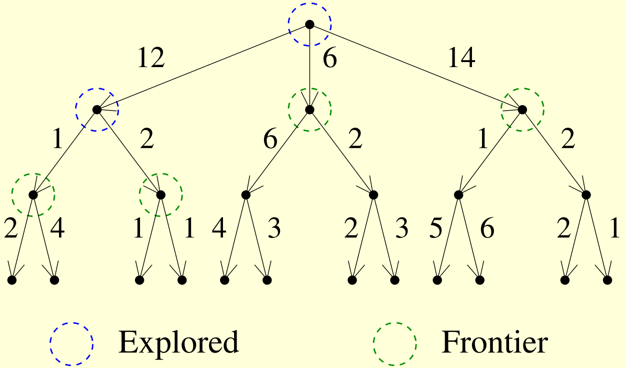
Depth-first Search (DFS)

Expand frontier node with longest path from start state.



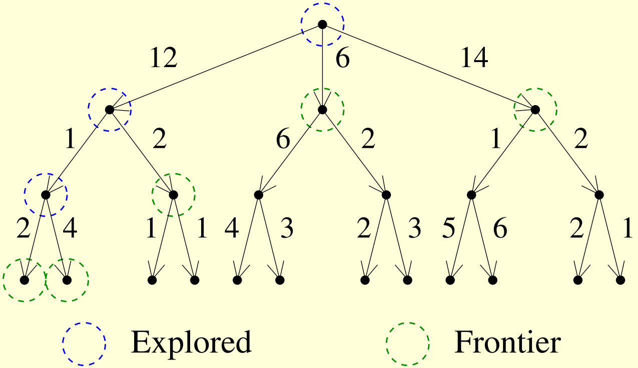
Depth-first Search (DFS)

Expand frontier node with longest path from start state.



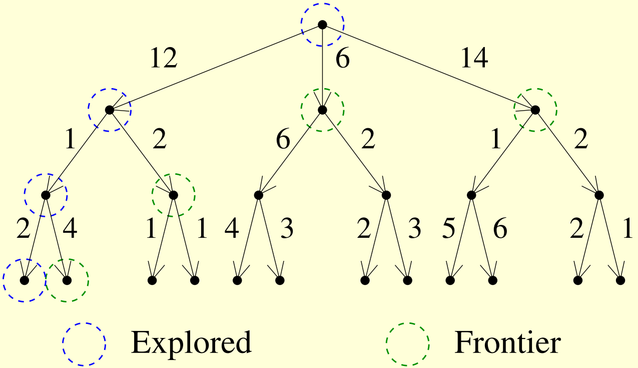
Depth-first Search (DFS)

Expand frontier node with longest path from start state.



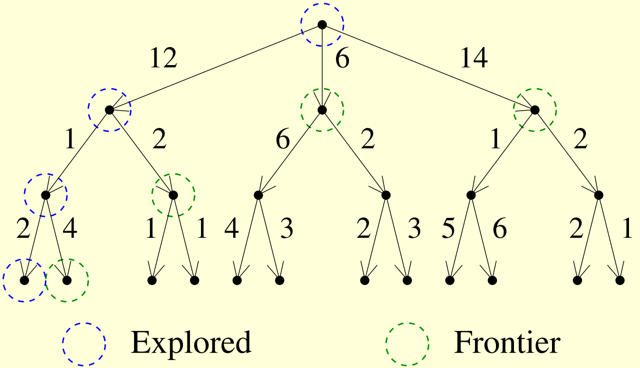
Depth-first Search (DFS)

Expand frontier node with longest path from start state.



Depth-first Search (DFS)

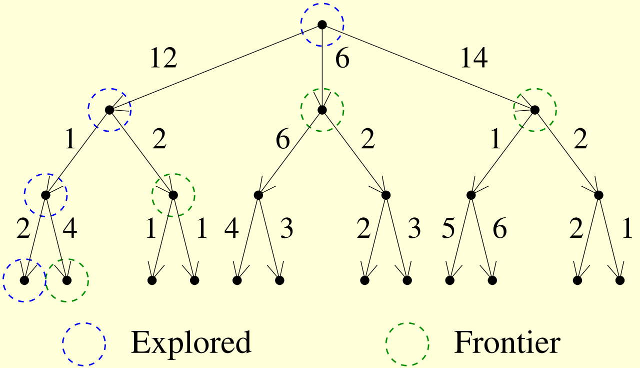
Expand frontier node with longest path from start state.



- Frontier treated like a stack (LIFO).

Depth-first Search (DFS)

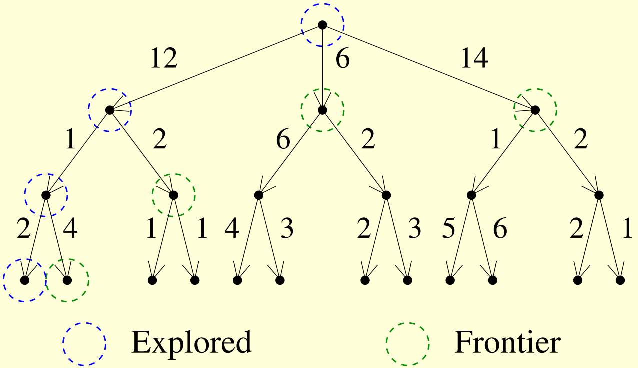
Expand frontier node with longest path from start state.



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

Depth-first Search (DFS)

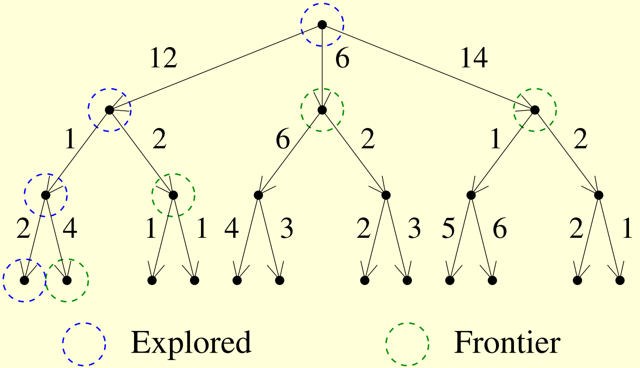
Expand frontier node with longest path from start state.



- 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)

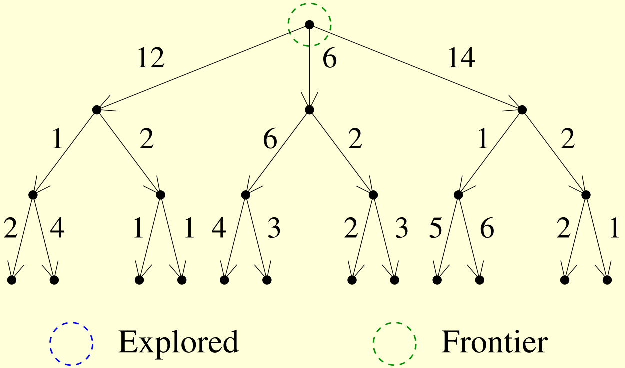
Expand frontier node with longest path from start state.



- 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.

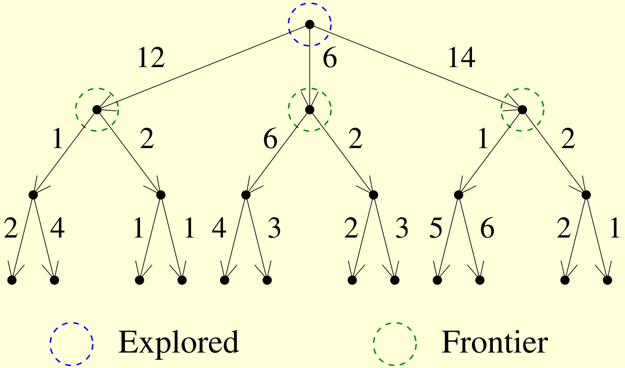
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



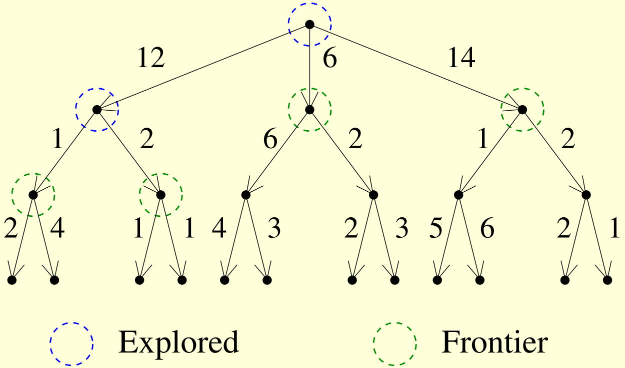
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



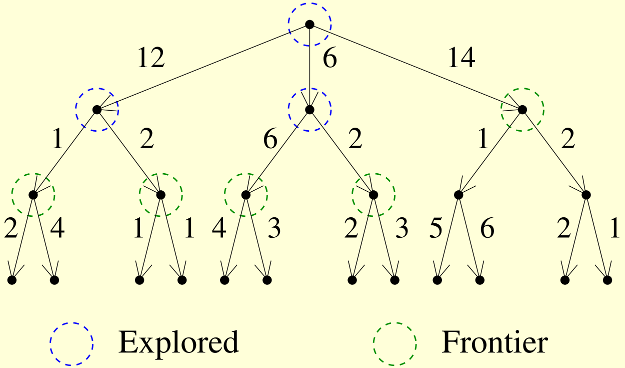
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



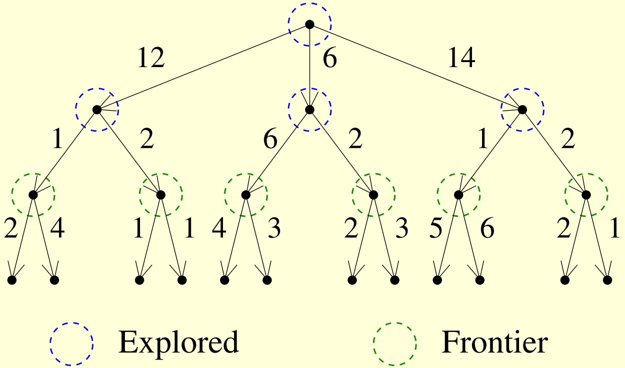
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



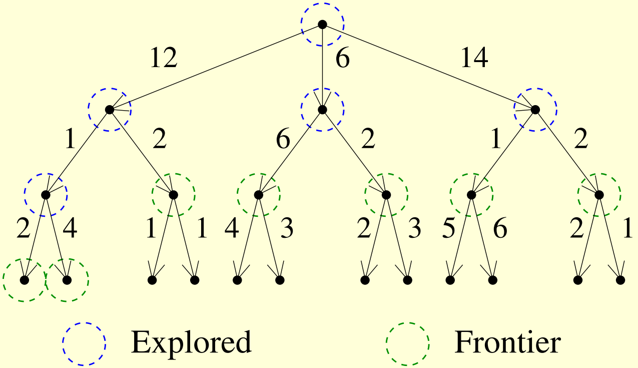
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



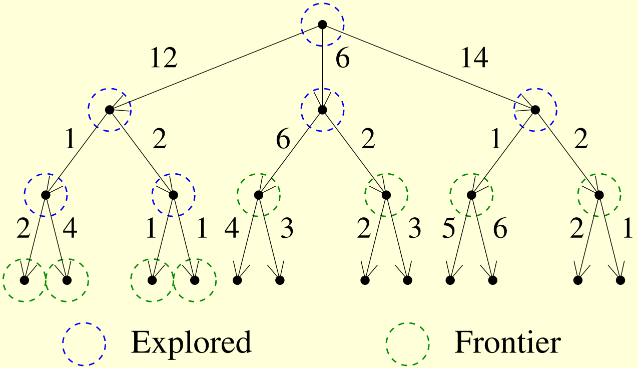
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



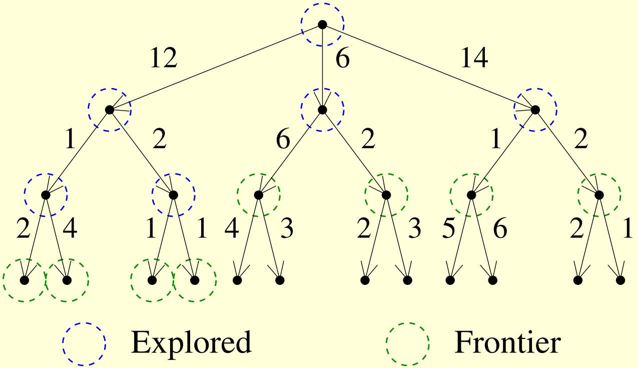
Breadth-first Search (BFS)

Expand frontier node with shortest path from start state.



Breadth-first Search (BFS)

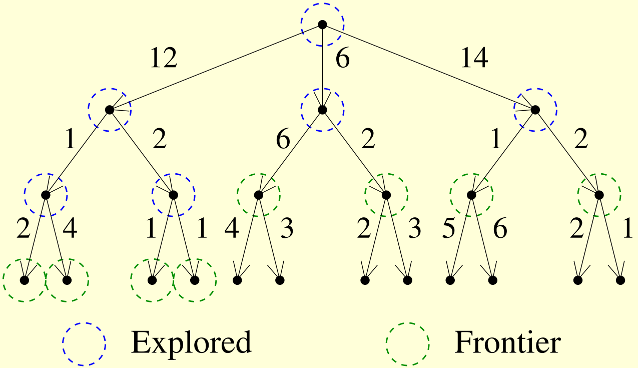
Expand frontier node with shortest path from start state.



- Frontier treated like a queue (FIFO).

Breadth-first Search (BFS)

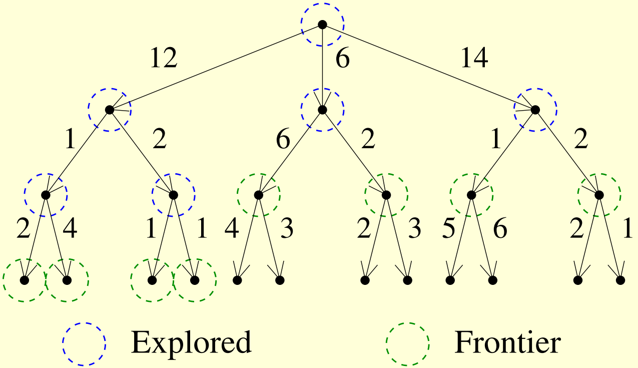
Expand frontier node with shortest path from start state.



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

Breadth-first Search (BFS)

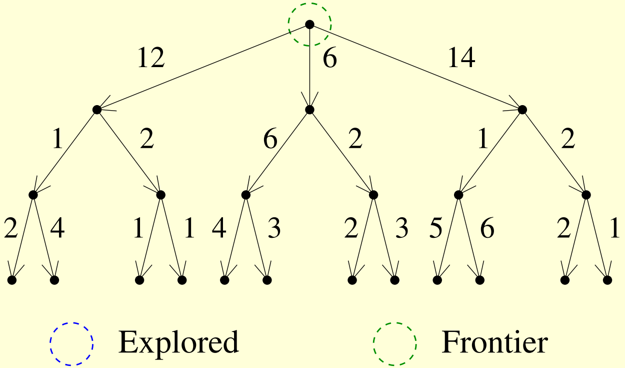
Expand frontier node with shortest path from start state.



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

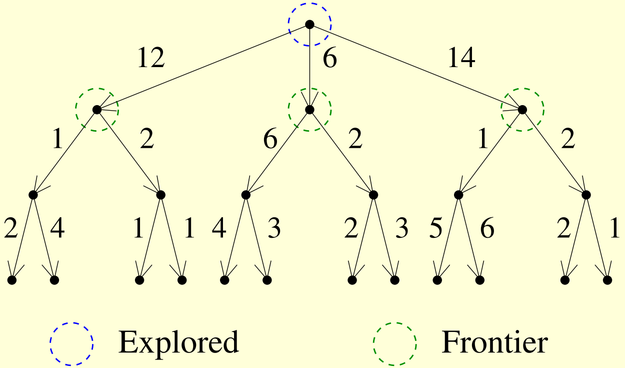
Lowest-cost-first Search (LCFS)

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



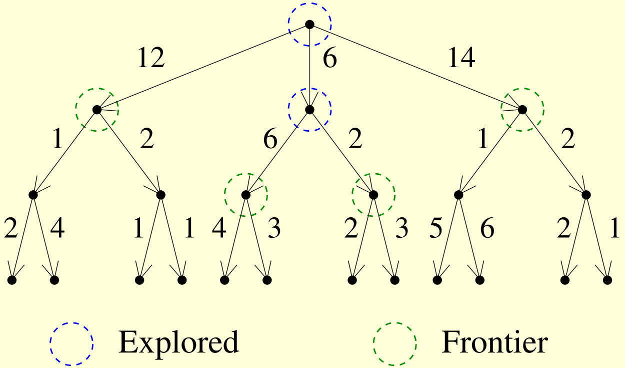
Lowest-cost-first Search (LCFS)

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



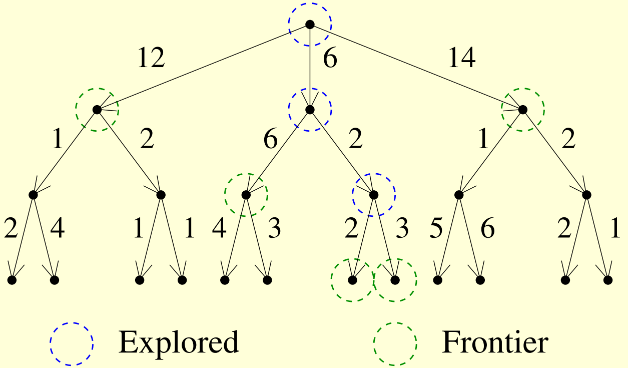
Lowest-cost-first Search (LCFS)

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



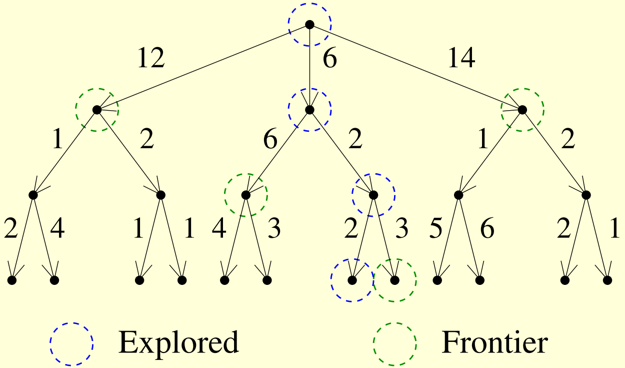
Lowest-cost-first Search (LCFS)

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



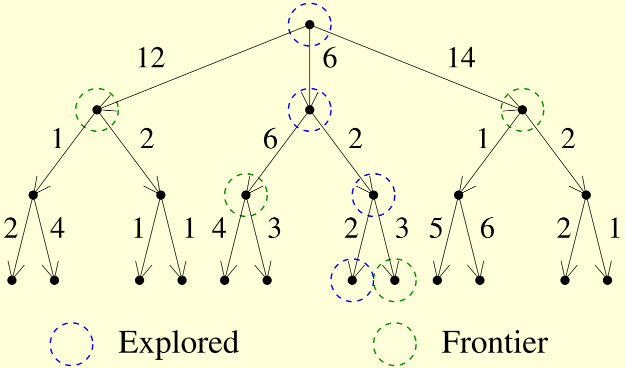
Lowest-cost-first Search (LCFS)

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



Lowest-cost-first Search (LCFS)

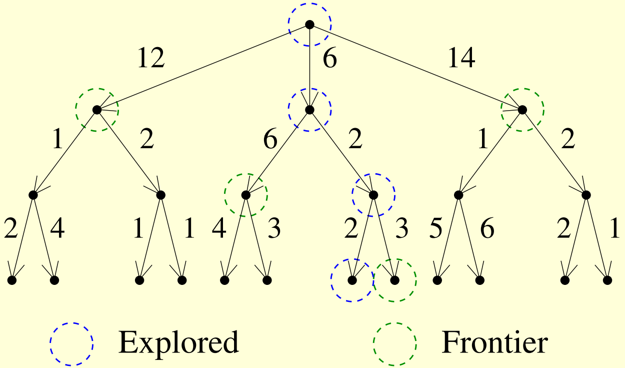
Expand frontier node with lowest path-cost from start state.



- 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)

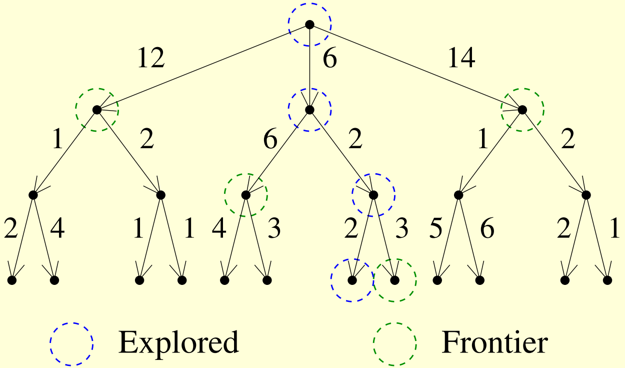
Expand frontier node with lowest path-cost from start state.



- 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.

Classical Search

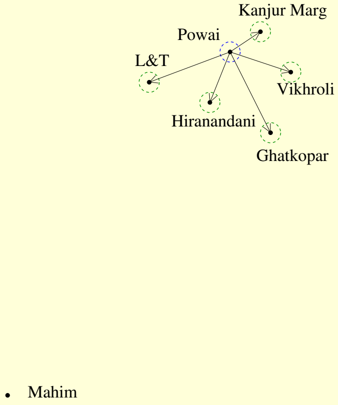
- Problem instances
- Generic search template
- Uninformed search
- Informed search (a.k.a. heuristic search)

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.

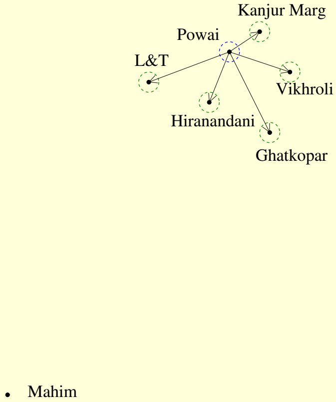
Incorporating Domain Knowledge into Search

- 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 $\operatorname{argmin}_{n \in \text{Frontier}} g(n)$.

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

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 $\text{argmin}_{n \in \text{Frontier}}\, g(n)$.
- 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).

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.

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.
- 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 $A^{\star}$ 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).

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,

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

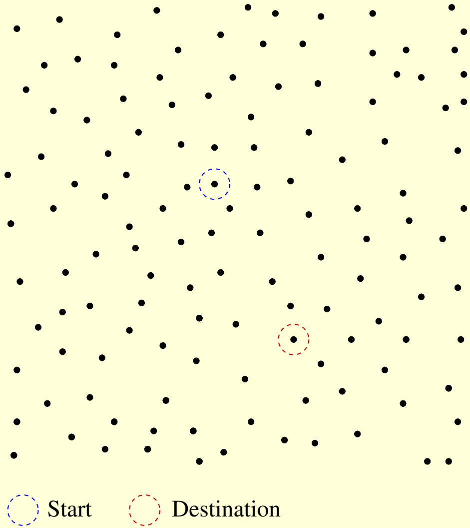
where $c^{\star}(n)$ is the optimal cost-to-goal of $n.state$.

- **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).

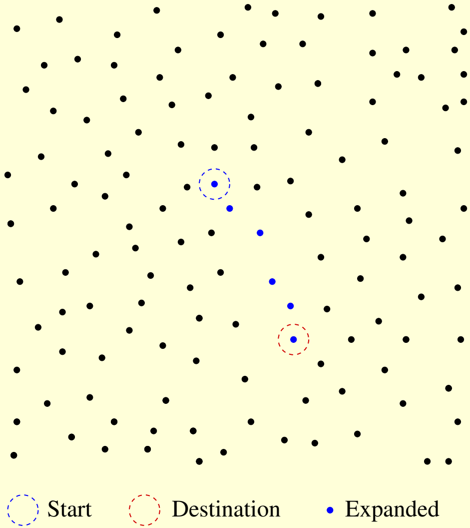
- Is straight line distance an admissible heuristic for navigation? Yes.

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

Effect of Heuristic

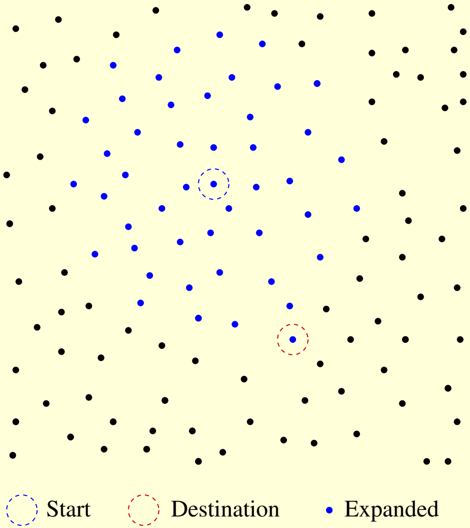


Effect of Heuristic



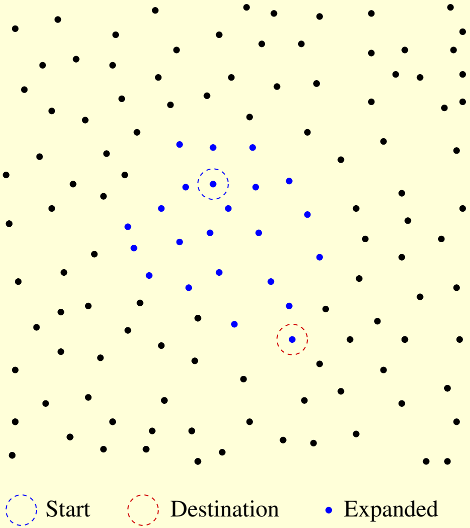
$h(n) = c^{\star}(n)$. Will only expand nodes along optimal path!
Unfortunately $c^{\star}(n)$ is not known!

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 class on search in games. But try to avoid.

Discussion

- Classical search a well-studied topic in AI.
- Compute time measured by number of nodes expanded.
- Heuristic guides search towards goal, improves efficiency.
- What if actions have stochastic outcomes?
- Studied as "decision-time planning" in MDPs.

  Technical problem: compute a near-optimal action for a particular "current" state in $o(|S|)$ time (that is, without visiting all states in the MDP).