CSE 411: Artificial Intelligence (Elective Course #6)

400 Level, Mechatronics Engineering
2^{nd} Term 2016/2017, Lecture #5

Hazem Shehata

Dept. of Computer & Systems Engineering
Zagazig University

March 27^{th}, 2017

Credits to Dr. Mohamed El Abd for the slides

Adminstrivia

Notes

- Assignment #2:
  - To be released on Thursday (Due 1 week after).
  - Constructing search trees on paper.

Course Info:

- Website: http://hshehata.github.io/courses/zu/cse411/
- Office hours: Sunday 11:30am - 12:30pm

Adminstrivia

Notes

- Assignment #2:
  - To be released on Thursday (Due 1 week after).
  - Constructing search trees on paper.
- Assignment #3:
  - To be released next week.
  - Implementing search algorithms in Python.

Course Info:

- Website: http://hshehata.github.io/courses/zu/cse411/
- Office hours: Sunday 11:30am - 12:30pm

Search

Types of search algorithms

- **Uninformed Search:**
  Only has the information provided by the problem formulation (initial state, available actions, transition model, goal test, and step/path cost).

Search

Types of search algorithms

- **Uninformed Search:**
  Only has the information provided by the problem formulation (initial state, available actions, transition model, goal test, and step/path cost).
  - Algorithms: **BFS**, **DFS**, **DLS**, **IDS**, **UCS**.

Search

Types of search algorithms

- **Uninformed Search:**
  Only has the information provided by the problem formulation (initial state, available actions, transition model, goal test, and step/path cost).
  - Algorithms: **BFS**, **DFS**, **DLS**, **IDS**, **UCS**.
- **Informed Search:**
  Has additional information that allows it to judge the promise of an action, *i.e.*, the estimated cost from a state to a goal.

Search

Types of search algorithms

- **Uninformed Search:**
  Only has the information provided by the problem formulation (initial state, available actions, transition model, goal test, and step/path cost).
  - Algorithms: **BFS**, **DFS**, **DLS**, **IDS**, **UCS**.

- **Informed Search:**
  Has additional information that allows it to judge the promise of an action, *i.e.*, the estimated cost from a state to a goal.
  - Algorithms: **GBFS**, **A***.

Outline

1. Informed Search

2. Requirements & Reading Material

Outline

Informed Search

Introduction

- Uninformed search algorithms are systematic but inefficient.

Informed Search

Introduction

- Uninformed search algorithms are systematic but inefficient.
- They just repeat a cycle of:
  - Choosing the next node to expand.
  - Check whether it is the goal.
  - If it is not, expand the node.

Informed Search

Introduction

- Uninformed search algorithms are systematic but inefficient.
- They just repeat a cycle of:
  - Choosing the next node to expand.
  - Check whether it is the goal.
  - If it is not, expand the node.
- Nodes are chosen in a specific order (imposed by a search strategy), which is not necessarily the ”best” order in terms of getting closer/faster to the goal.

Informed Search

Introduction

- Would like to use additional knowledge so that ”better” nodes are expanded/explored first.

Informed Search

Introduction

- Would like to use additional knowledge so that ”better” nodes are expanded/explored first.
- In terms of pseudo-code, we want to order the frontier so that ”better” nodes get popped first.

Informed Search

Introduction

- Would like to use additional knowledge so that ”better” nodes are expanded/explored first.
- In terms of pseudo-code, we want to order the frontier so that ”better” nodes get popped first.
- We will introduce an **evaluation function** $f(n)$ that indicates the desirability of considering node n next for exploration and expansion.

Informed Search

Introduction

- Would like to use additional knowledge so that ”better” nodes are expanded/explored first.
- In terms of pseudo-code, we want to order the frontier so that ”better” nodes get popped first.
- We will introduce an **evaluation function** $f(n)$ that indicates the desirability of considering node n next for exploration and expansion.
- Nodes with a better $f(n)$ are always considered first. This approach is called **best-first search**.

Informed Search

Introduction

- Would like to use additional knowledge so that ”better” nodes are expanded/explored first.
- In terms of pseudo-code, we want to order the frontier so that ”better” nodes get popped first.
- We will introduce an **evaluation function** $f(n)$ that indicates the desirability of considering node n next for exploration and expansion.
- Nodes with a better $f(n)$ are always considered first. This approach is called **best-first search**.
- How should we compute $f(n)$?

Informed Search

Introduction

- Uniform-cost search orders the frontier according to the path cost $g(n)$:
  - Path cost is distance from root to state n.

Informed Search

Introduction

- Uniform-cost search orders the frontier according to the path cost $g(n)$:
  - Path cost is distance from root to state n.
- So, uniform-cost search uses an evaluation function $f(n) = g(n)$.

Informed Search

Introduction

- Uniform-cost search orders the frontier according to the path cost $g(n)$:
  - Path cost is distance from root to state n.
- So, uniform-cost search uses an evaluation function $f(n) = g(n)$.
- The path cost $g(n)$ only accounts for cost to reach n. Hence, uniform-cost search is not goal directed/oriented.

Informed Search

Introduction

- Would also like to consider cost from node n to the goal.

Informed Search

Introduction

- Would also like to consider cost from node n to the goal.
- More generally:
  We want the search algorithm to be able to *estimate* the path cost from the current node to the goal.

Informed Search

Introduction

- Would also like to consider cost from node n to the goal.
- More generally:
  We want the search algorithm to be able to *estimate* the path cost from the current node to the goal.
- This estimate is called a **heuristic function**.

Informed Search

Introduction

- Would also like to consider cost from node n to the goal.
- More generally:
  We want the search algorithm to be able to *estimate* the path cost from the current node to the goal.
- This estimate is called a **heuristic function**.
- Cannot be done based on problem formulation:
  Need to add additional information.

Informed Search

Introduction

- Path cost $g(n)$ is an exact value.

Informed Search

Introduction

- Path cost $g(n)$ is an exact value.
- Heuristic function $h(n)$:
  - $h(n)$: estimated cost from node n to goal.
  - $h(n1) < h(n2)$ means it is probably cheaper to get to the goal from $n1$.
  - $h(goal) = 0$.

Informed Search

Introduction

- Path cost $g(n)$ is an exact value.
- Heuristic function $h(n)$:
  - $h(n)$: estimated cost from node n to goal.
  - $h(n1) < h(n2)$ means it is probably cheaper to get to the goal from $n1$.
  - $h(goal)$ = 0.
- Evaluation function $f(n)$:
  - $f(n) = g(n)$: Uniform-cost search.
  - $f(n) = h(n)$: Greedy best-first search.
  - $f(n) = g(n) + h(n)$: A* search.

Informed Search

Greedy best-first search

- GBFS algorithm always explores and expands the node judged to be closest to goal.

Informed Search

Greedy best-first search

- GBFS algorithm always explores and expands the node judged to be closest to goal.
- It uses $f(n) = h(n)$ and ignores the path cost $g(n)$ entirely.

Informed Search

Greedy best-first search

- GBFS algorithm always explores and expands the node judged to be closest to goal.
- It uses $f(n) = h(n)$ and ignores the path cost $g(n)$ entirely.
- Consider it to be the complement of uniform-cost search.

Informed Search

Greedy best-first search

- GBFS algorithm always explores and expands the node judged to be closest to goal.
- It uses $f(n) = h(n)$ and ignores the path cost $g(n)$ entirely.
- Consider it to be the complement of uniform-cost search.
- GBFS algorithm is identical to UNIFORM-COST-SEARCH except that h is used instead of g.

Search Algorithms

Greedy best-first search algorithm (tree version)

function GREEDY-BEST-FIRST-SEARCH(*problem*) **returns** a solution, or failure
  node ← node with STATE=*problem*.INITIAL-STATE, PATH-COST=0
  frontier ← priority queue ordered by **heuristic** *h*, with *node* as only element
  loop do
    if *frontier*.EMPTY?() **then return** failure
    node ← *frontier*.POP()   /* choose lowest-cost node in frontier */
    if *problem*.GOAL-TEST(*node*.STATE) **then return** *node*.SOLUTION()
    for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
      child ← *node*.CHILD-NODE(*problem*, *action*)
      frontier.INSERT(*child*, *h*(*child*))

Search Algorithms

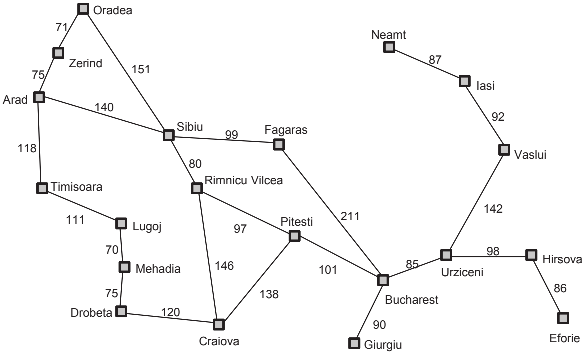
Greedy best-first search algorithm (graph version)

```
function GREEDY-BEST-FIRST-SEARCH(problem) returns a solution, or failure
  node ← node with STATE=problem.INITIAL-STATE, PATH-COST=0
  frontier ← priority queue ordered by [heuristic h], with node as only element
  explored ← an empty set
  loop do
    if frontier.EMPTY?() then return failure
    node ← frontier.POP()          /* choose lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return node.SOLUTION()
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← node.CHILD-NODE(problem, action)
      if child.STATE is not in explored and not in frontier then
        frontier.INSERT(child, [h(child)])
      else if child.STATE is in frontier with higher [h-value] then
        replace that frontier node with child
```

Informed Search

GBFS example - Romania map

Find a route from Arad to Bucharest.

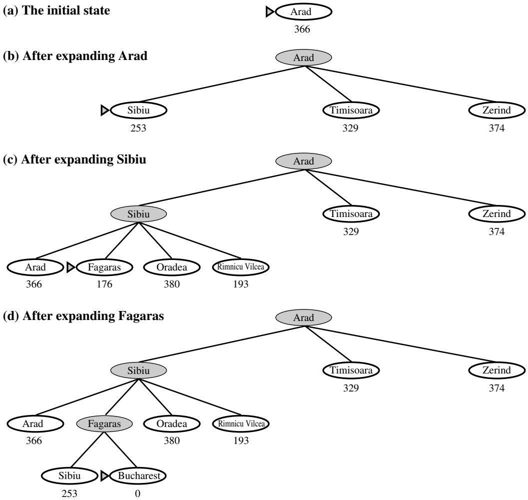


straight-line distances to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Informed Search

GBFS example - Romania map

Find a route from Arad to Bucharest.



Informed Search

Greedy best-first search

- Not optimal.

Informed Search

Greedy best-first search

- Not complete (unless m is finite; uncommon in trees).
  - May get stuck in an infinite loop (*e.g.,* reaching a deadend through a reversible action).
- Not optimal.

Informed Search

Greedy best-first search

- Not complete (unless m is finite; uncommon in trees).
  - May get stuck in an infinite loop (*e.g.*, reaching a deadend through a reversible action).
- Not optimal.
- Time complexity = $O(b^m)$.

Informed Search

Greedy best-first search

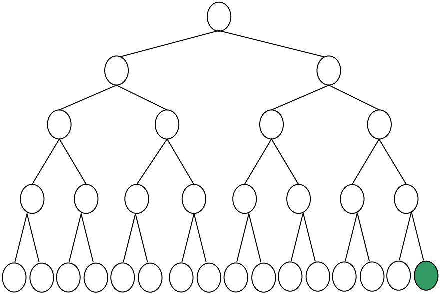
- Not complete (unless m is finite; uncommon in trees).
  - May get stuck in an infinite loop (*e.g.*, reaching a deadend through a reversible action).
- Not optimal.
- Time complexity = $O(b^m)$.
- Space complexity = $O(b^m)$.

Informed Search

Greedy best-first search

GBFS properties:

- Upper-bound case: goal is last node of the tree:
  - Number of nodes generated: b nodes for each node of m levels (entire tree).
  - Time and space complexity: all generated nodes $O(b^m)$.



Informed Search

A* search

- Uniform cost search orders the queue according to the path cost $g(n)$:
  - Optimal, complete, but inefficient in time and space.

Informed Search

A* search

- Uniform cost search orders the queue according to the path cost $g(n)$:
  - Optimal, complete, but inefficient in time and space.
- Greedy best first search orders the queue using the heuristic cost $h(n)$:
  - Not optimal, not complete but efficient and directed (with good heuristic).

Informed Search

A* search

- Idea behind A* is to combine the two strategies:

Informed Search

A* search

- Idea behind A* is to combine the two strategies:
  - Use an evaluation function $f(n) = g(n) + h(n)$ to order the nodes to be explored.

Informed Search

A* search

- Idea behind A* is to combine the two strategies:
  - Use an evaluation function $f(n) = g(n) + h(n)$ to order the nodes to be explored.
  - $f(n)$ measures the cheapest total estimated cost from the initial state to the goal state passing through the current state n.

Informed Search

A* search

- Idea behind A* is to combine the two strategies:
  - Use an evaluation function $f(n) = g(n) + h(n)$ to order the nodes to be explored.
  - $f(n)$ measures the cheapest total estimated cost from the initial state to the goal state passing through the current state n.
- The resulting search is both optimal and complete assuming certain conditions on the heuristic cost $h(n)$.

Informed Search

A* search

- Idea behind A* is to combine the two strategies:
  - Use an evaluation function $f(n) = g(n) + h(n)$ to order the nodes to be explored.
  - $f(n)$ measures the cheapest total estimated cost from the initial state to the goal state passing through the current state n.
- The resulting search is both optimal and complete assuming certain conditions on the heuristic cost $h(n)$.
- A* algorithm is identical to UNIFORM-COST-SEARCH except that $g + h$ is used instead of g.

Search Algorithms

A* search algorithm (tree version)

function A-STAR-SEARCH(*problem*) **returns** a solution, or failure
  node ← node with STATE=*problem*.INITIAL-STATE, PATH-COST=0
  frontier ← priority queue ordered by **evaluation** *f*, with *node* as only element
  loop do
    if *frontier*.EMPTY?() **then return** failure
    node ← *frontier*.POP()   /* choose lowest-cost node in frontier */
    if *problem*.GOAL-TEST(*node*.STATE) **then return** *node*.SOLUTION()
    for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
      child ← *node*.CHILD-NODE(*problem*, *action*)
      frontier.INSERT(*child*, *f*(*child*))

Search Algorithms

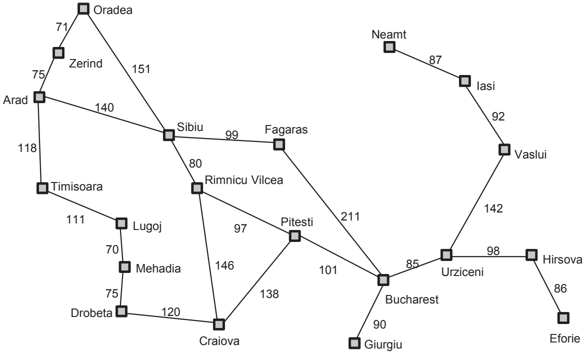
A* search algorithm (graph version)

function A-STAR-SEARCH(*problem*) **returns** a solution, or failure
  node ← node with STATE=*problem*.INITIAL-STATE, PATH-COST=0
  frontier ← priority queue ordered by **evaluation *f***, with *node* as only element
  explored ← an empty set
  loop do
    if *frontier*.EMPTY?() **then return** failure
    node ← *frontier*.POP()   /* choose lowest-cost node in frontier */
    if *problem*.GOAL-TEST(*node*.STATE) **then return** *node*.SOLUTION()
    add *node*.STATE to *explored*
    for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
      child ← *node*.CHILD-NODE(*problem*, *action*)
      if *child*.STATE is not in *explored* and not in *frontier* **then**
        frontier.INSERT(*child*, *f*(*child*))
      else if *child*.STATE is in *frontier* with higher *f*-value **then**
        replace that frontier node with *child*

Informed Search

A* example - Romania map

Find a route from Arad to Bucharest.

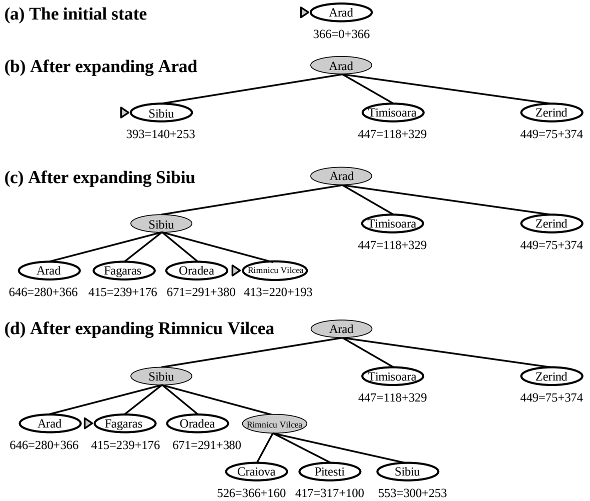


straight-line distances to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Informed Search

A* example - Romania map

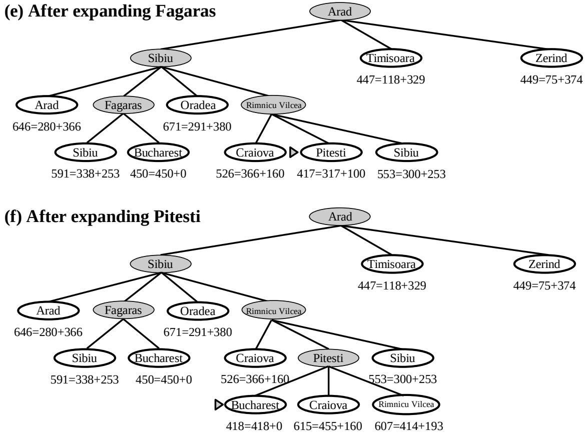
Find a route from Arad to Bucharest.



Informed Search

A* example - Romania map

Find a route from Arad to Bucharest.



Informed Search

A* search

- Optimal, given an admissible heuristic.

Informed Search

A* search

- Complete (if b and ε are finite).
  - Reason: number of nodes with cost $\leq C^*$ is finite.
- Optimal, given an admissible heuristic.

Informed Search

A* search

- Complete (if b and ε are finite).
  - Reason: number of nodes with cost $\leq C^*$ is finite.
- Optimal, given an admissible heuristic.
- Time and space complexity: not straightforward!
  - Number of nodes explored depends on the difference between h and h^* (true cost).
  - If $h = h^*$, A* expands only the nodes on the optimal solution path(s).
  - If $h = 0$, A* consumes as much (time/space) resources as UCS.

Informed Search

A* search

- A heuristic $h(n)$ is admissible if for every node n:
$$h(n) \leq h^*(n),$$
where $h^*(n)$ is the true cost to reach goal state from n.

Informed Search

A* search

- A heuristic $h(n)$ is admissible if for every node n:
$$h(n) \leq h^*(n),$$
where $h^*(n)$ is the true cost to reach goal state from n.

- An admissible heuristic function $h(n)$ **never overestimates the true cost from n to goal**.

Informed Search

A* search

- A heuristic $h(n)$ is admissible if for every node n:
$$h(n) \leq h^*(n),$$
where $h^*(n)$ is the true cost to reach goal state from n.

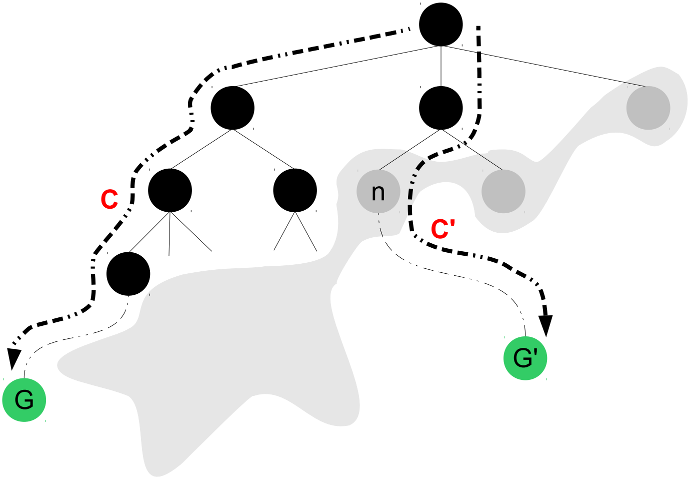
- An admissible heuristic function $h(n)$ **never overestimates the true cost from n to goal**.

- If $h(n)$ never overestimates, then $f(n)$ never overestimates the true cost to the goal through node n.

Informed Search

A* search

- Proving that "admissibility guarantees optimality":



Informed Search

A* search

- Proving that "admissibility guarantees optimality":
  - Suppose an A* search (that uses an admissible heuristic h) finds a goal G whose path cost of is C.
    - $h(G) = 0$, and hence: $f(G) = g(G) = C$
  - Let G' be another goal node whose path cost is C'.
  - Frontier contains a node n on the optimal path to G'.
    - h is admissible, and hence: $f(n) = g(n) + h(n) \leq C'$
  - Node G has been popped from frontier before node n.
    - $f(G) \leq f(n)$, and hence: $C \leq C'$.
  - Using an admissible heuristic, A* would always discover the lowest-cost (*i.e.*, optimal) solution.

Informed Search

A* search

- In graph-search algorithms, a new node is discarded if it's already in the *explored* set.

Informed Search

A* search

- In graph-search algorithms, a new node is discarded if it's already in the *explored* set.
- If the new node has a better path cost (g) than the old node, a shorter path has been ignored.

Informed Search

A* search

- In graph-search algorithms, a new node is discarded if it's already in the *explored* set.
- If the new node has a better path cost (g) than the old node, a shorter path has been ignored.
- This means that the graph version of A* search is not optimal any more!

Informed Search

A* search

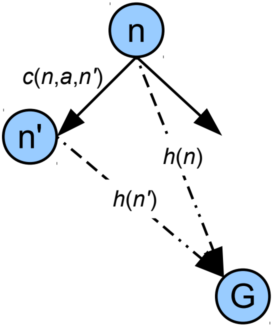
- In graph-search algorithms, a new node is discarded if it's already in the *explored* set.
- If the new node has a better path cost (g) than the old node, a shorter path has been ignored.
- This means that the graph version of A* search is not optimal any more!
- To ensure optimality, A* must use a **consistent** heuristic function h.
  - Consistency is a stronger condition than admissibility.

Informed Search

A* search

- **Consistency** (monotonicity) means that for every node n and child node n' reachable from n by action a, the estimated cost $h(n)$ is never greater than the estimated cost $h(n')$ plus the step cost of getting to n':

$$h(n) \leq h(n') + c(n, a, n')$$



Informed Search

A* search

- A consistent heuristic function $h(n)$ **never overestimates the true cost of action a taken from n to n'**:

$$h(n) - h(n') \leq c(n, a, n')$$

Informed Search

A* search

- A consistent heuristic function $h(n)$ **never overestimates the true cost of action a taken from n to n'**:

$$h(n) - h(n') \leq c(n, a, n')$$

- With some mathematical arrangements:

$$\begin{aligned} f(n) &= h(n) + g(n) \\ &\leq h(n') + c(n, a, n') + g(n) \\ &\leq h(n') + g(n') \\ &\leq f(n') \end{aligned}$$

So, $f(n)$ never decreases as we approach the goal.

Informed Search

A* search

- A consistent heuristic function $h(n)$ **never overestimates the true cost of action a taken from n to n'**:

$$h(n) - h(n') \leq c(n, a, n')$$

- With some mathematical arrangements:

$$\begin{aligned} f(n) &= h(n) + g(n) \\ &\leq h(n') + c(n, a, n') + g(n) \\ &\leq h(n') + g(n') \\ &\leq f(n') \end{aligned}$$

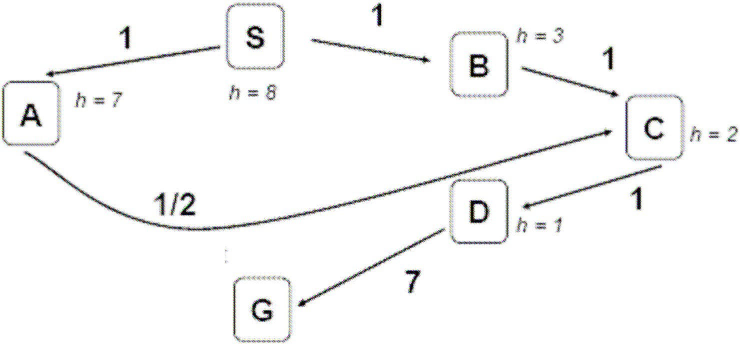
  So, $f(n)$ never decreases as we approach the goal.

- Consistency guarantees that states are always visited by the cheapest path first; no need to check if subsequent paths are better than the first.

Informed Search

A* example - consistency

Check whether h is consistent, and perform A* search.



Informed Search

Introduction

- Informed search algorithms use additional knowledge about the problem to direct search toward goal(s).

Informed Search

Introduction

- Informed search algorithms use additional knowledge about the problem to direct search toward goal(s).
- Such additional knowledge takes the form of a heuristic function h.
  - $h(n)$: *estimated* path cost from node n to closest goal.

Informed Search

Introduction

- Informed search algorithms use additional knowledge about the problem to direct search toward goal(s).
- Such additional knowledge takes the form of a heuristic function h.
  - $h(n)$: *estimated* path cost from node n to closest goal.
- Frontier nodes are ordered using an evaluation function f defined in terms of h.
  - GBFS: $f(n) = h(n)$.
  - A*: $f(n) = h(n) + g(n)$.

Informed Search

Introduction

- Informed search algorithms use additional knowledge about the problem to direct search toward goal(s).
- Such additional knowledge takes the form of a heuristic function h.
  - $h(n)$: *estimated* path cost from node n to closest goal.
- Frontier nodes are ordered using an evaluation function f defined in terms of h.
  - GBFS: $f(n) = h(n)$.
  - A*: $f(n) = h(n) + g(n)$.
- A* has the advantage of being optimal given that:
  - h is admissible (tree version).
  - h is consistent (graph version).

Informed Search

Effect of heuristic functions

- Uniform-cost search expands in circular cost contours $(h(n) = 0)$.

Informed Search

Effect of heuristic functions

- Uniform-cost search expands in circular cost contours $(h(n) = 0)$.
- A* search elongates & rotates contours towards goal:
  - More narrow/elongated, the better $h(n)$ is. More directed.

Informed Search

Quality of heuristic function

- One heuristic function might be better than another for a given problem.

Informed Search

Quality of heuristic function

- One heuristic function might be better than another for a given problem.

- **Informedness:**
  For two admissible heuristic functions, $h1$ and $h2$:

  if $h2(n) \geq h1(n)$
  $h2(n)$ is more informed than $h1(n)$

  Alternatively we say that $h2(n)$ dominates $h1(n)$.

Informed Search

Quality of heuristic function

- One heuristic function might be better than another for a given problem.
- **Informedness:**
  For two admissible heuristic functions, $h1$ and $h2$:

  if $h2(n) \geq h1(n)$
  $h2(n)$ is more informed than $h1(n)$

  Alternatively we say that $h2(n)$ dominates $h1(n)$.
- More informedness implies fewer expanded states.

Informed Search

Creating a heuristic function

- How to choose the heuristic function for a given problem?

Informed Search

Creating a heuristic function

- How to choose the heuristic function for a given problem?
- The heuristic function is usually chosen by means of ***problem relaxation***.

Informed Search

Creating a heuristic function

- How to choose the heuristic function for a given problem?
- The heuristic function is usually chosen by means of ***problem relaxation***.
- Problem relaxation means making the problem easier by dropping some constraints.

Informed Search

Creating a heuristic function

- How to choose the heuristic function for a given problem?
- The heuristic function is usually chosen by means of ***problem relaxation***.
- Problem relaxation means making the problem easier by dropping some constraints.
- Can also have different heuristics and always choose the best one:

  $h(n) = \max\{h_1(n), h_2(n), ..., h_m(n)\}$

Informed Search

Creating a heuristic function

- For the Romania map problem, the heuristic function was selected as the straight line distance between the current city and the goal.

Informed Search

Creating a heuristic function

- For the Romania map problem, the heuristic function was selected as the straight line distance between the current city and the goal.
- How is this a relaxed problem?

Informed Search

Creating a heuristic function

- For the Romania map problem, the heuristic function was selected as the straight line distance between the current city and the goal.
- How is this a relaxed problem?
  - By dropping the traveling on roads constraint.

Informed Search

Example: creating heuristic for 8-puzzle

- Consider the 8-puzzle problem.

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Informed Search

Example: creating heuristic for 8-puzzle

- Consider the 8-puzzle problem.
  - It would take at least *26 moves* to solve the problem instance shown below.

7	2	4
5		6
8	3	1

Start State

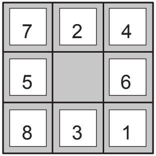
	1	2
3	4	5
6	7	8

Goal State

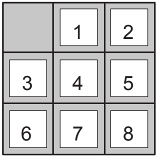
Informed Search

Example: creating heuristic for 8-puzzle

- Consider the 8-puzzle problem.
  - It would take at least *26 moves* to solve the problem instance shown below.
  - An admissible heuristic shouldn't overestimate that cost of 26 moves.
    - $h(Start) \leq 26$



Start State



Goal State

Informed Search

Example: creating heuristic for 8-puzzle

- Consider the 8-puzzle problem.
  - It would take at least *26 moves* to solve the problem instance shown below.
  - An admissible heuristic shouldn't overestimate that cost of 26 moves.
    - $h(Start) \leq 26$
  - Generally speaking, an admissible heuristic shouldn't overestimate the cost of solving the puzzle starting from any node.
    - $h(n) \leq$ minimum number of moves to get from n to $Goal$.

7	2	4
5		6
8	3	1

Start State

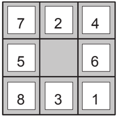
	1	2
3	4	5
6	7	8

Goal State

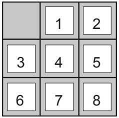
Informed Search

Example: creating heuristic for 8-puzzle

- Description of legal moves in 8-puzzle:
  A tile can move from location A to location B if A,B are adjacent and B is blank.



Start State

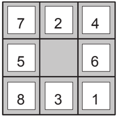


Goal State

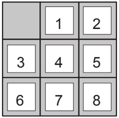
Informed Search

Example: creating heuristic for 8-puzzle

- Description of legal moves in 8-puzzle:
  A tile can move from location A to location B if A,B are adjacent and B is blank.
- Problem relaxation:



Start State

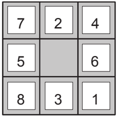


Goal State

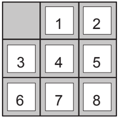
Informed Search

Example: creating heuristic for 8-puzzle

- Description of legal moves in 8-puzzle:
  A tile can move from location A to location B if A,B are adjacent and B is blank.
- Problem relaxation:
  - A tile can move from location A to location B:



Start State

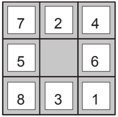


Goal State

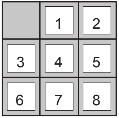
Informed Search

Example: creating heuristic for 8-puzzle

- Description of legal moves in 8-puzzle:
  A tile can move from location A to location B if A,B are adjacent and B is blank.
- Problem relaxation:
  - A tile can move from location A to location B:
    - $h_1(n) =$ number of tiles out of place.



Start State

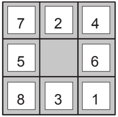


Goal State

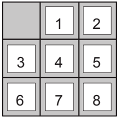
Informed Search

Example: creating heuristic for 8-puzzle

- Description of legal moves in 8-puzzle:
  A tile can move from location A to location B if A,B are adjacent and B is blank.
- Problem relaxation:
  - A tile can move from location A to location B:
    - $h_1(n)$ = number of tiles out of place.
    - $h_1(Start) = 1+1+1+1+1+1+1+1 = 8 \leq 26.$



Start State



Goal State

Informed Search

Example: creating heuristic for 8-puzzle

- Description of legal moves in 8-puzzle:
  A tile can move from location A to location B if A,B are adjacent and B is blank.
- Problem relaxation:
  - A tile can move from location A to location B:
    - $h_1(n) =$ number of tiles out of place.
    - $h_1(Start) = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 8 \leq 26.$
  - A tile can move from loc. A to loc. B if A,B are adjacent:

7	2	4
5		6
8	3	1

Start State

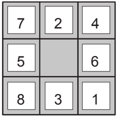
	1	2
3	4	5
6	7	8

Goal State

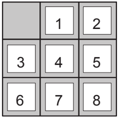
Informed Search

Example: creating heuristic for 8-puzzle

- Description of legal moves in 8-puzzle:
  A tile can move from location A to location B if A,B are adjacent and B is blank.
- Problem relaxation:
  - A tile can move from location A to location B:
    - $h_1(n) =$ number of tiles out of place.
    - $h_1(Start) = 1+1+1+1+1+1+1+1 = 8 \leq 26.$
  - A tile can move from loc. A to loc. B if A,B are adjacent:
    - $h_2(n) =$ sum of distances of tiles from goal locations.



Start State



Goal State

Informed Search

Example: creating heuristic for 8-puzzle

- Description of legal moves in 8-puzzle:
  A tile can move from location A to location B if A,B are adjacent and B is blank.
- Problem relaxation:
  - A tile can move from location A to location B:
    - $h_1(n) =$ number of tiles out of place.
    - $h_1(Start) = 1+1+1+1+1+1+1+1 = 8 \leq 26.$
  - A tile can move from loc. A to loc. B if A,B are adjacent:
    - $h_2(n) =$ sum of distances of tiles from goal locations.
    - $h_2(Start) = 3+1+2+2+2+3+3+2 = 18 \leq 26.$

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Informed Search

Example: creating heuristic for 8-puzzle

- Description of legal moves in 8-puzzle:
  A tile can move from location A to location B if A,B are adjacent and B is blank.
- Problem relaxation:
  - A tile can move from location A to location B:
    - $h_1(n) =$ number of tiles out of place.
    - $h_1(Start) = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 8 \leq 26$.
  - A tile can move from loc. A to loc. B if A,B are adjacent:
    - $h_2(n) =$ sum of distances of tiles from goal locations.
    - $h_2(Start) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18 \leq 26$.
- Notice that h_1 and h_2 are admissible, and $h_1(n) \leq h_2(n)$.

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Informed Search

A* search - 8-puzzle search costs

Data are averaged over 100 instances for the 8-puzzle problem across various solution lengths.

Optimal Solution Length (d)	Search Cost (nodes generated)		
	IDS	A* using h_1	A* using h_2
4 steps	112	13	12
8 steps	6384	39	25
12 steps	3644035	227	73
16 steps	-	1301	211
20 steps	-	7276	676
24 steps	-	39135	1641

Outline

1. Informed Search

2. Requirements & Reading Material

Requirements

What do I need from you

- When given a certain problem you should be able to:

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Build the search tree up to a given depth.

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Build the search tree up to a given depth.
  - Traverse the search tree according to a given strategy.

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Build the search tree up to a given depth.
  - Traverse the search tree according to a given strategy.
  - Propose a good heuristic function for the problem.
- Answer descriptive questions.

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Build the search tree up to a given depth.
  - Traverse the search tree according to a given strategy.
  - Propose a good heuristic function for the problem.
  - Indicate whether a given heuristic is admissible/consistent or not.
- Answer descriptive questions.

Reading Material

Which parts of the textbook are covered

- Russell-Norvig, Chapters 3:
  - Pages 92 - 98.
  - Pages 102 - 106.