Artificial Intelligence

3. Heuristic Search

Prof. Bojana Dalbelo Bašić
Assoc. Prof. Jan Šnajder

University of Zagreb
Faculty of Electrical Engineering and Computing

Academic Year 2019/2020

v3.10

Motivation

- Blind search strategies rely only on exact information (initial state, operators, goal predicate)
- They don't make use of additional **information about the nature of the problem** that might make the search more efficient
- If we have a (vague) idea in which direction to look for the solution, why not use this information to improve the search?

Outline

Outline

Heuristics

- **Heuristics** – rules about the nature of the problem, which are grounded in experience and whose purpose is to **direct** the search towards the goal so that it becomes more efficient

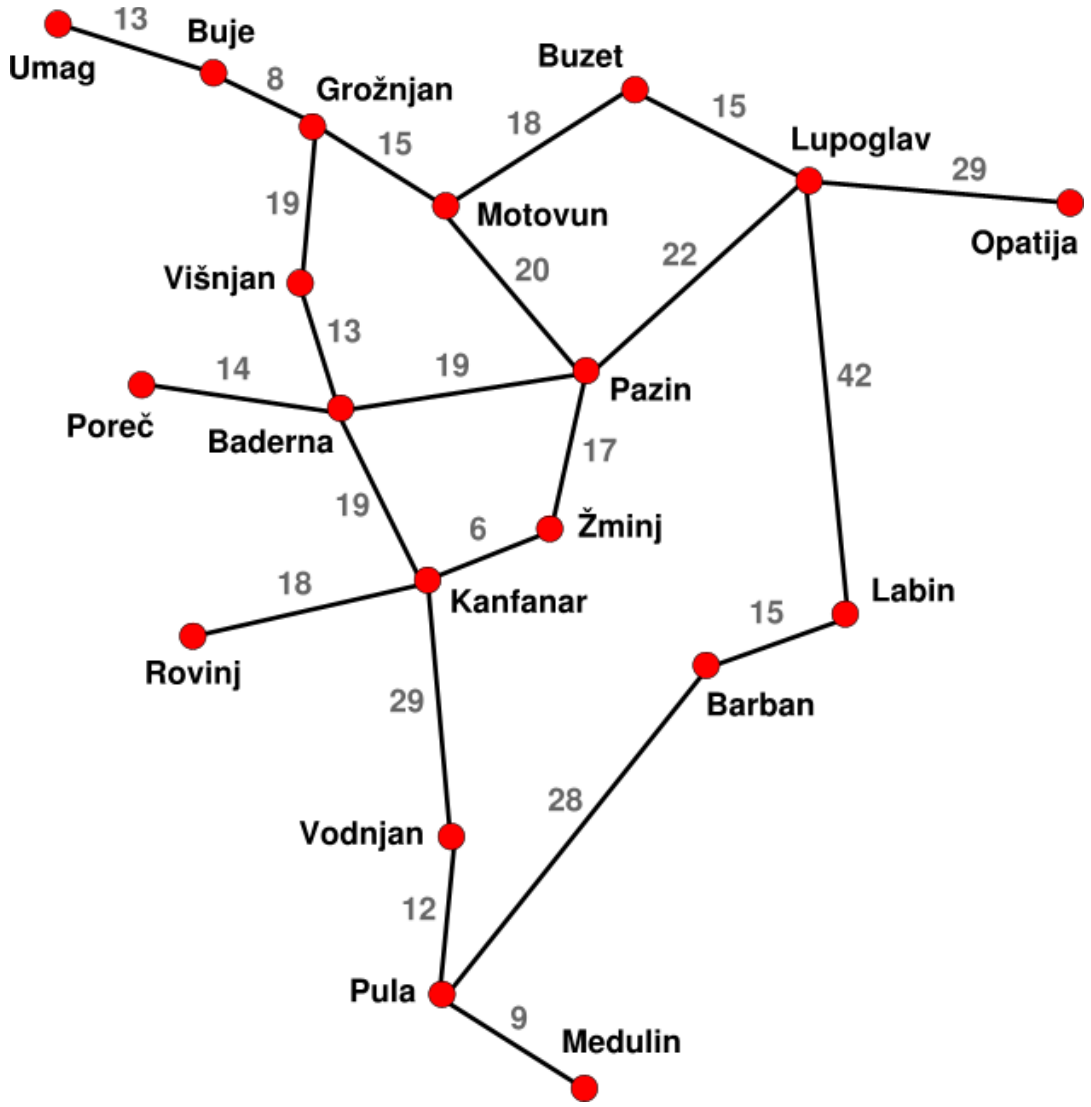
Heuristic function

Heuristic function $h : S \to \mathbb{R}^+$ assigns to each state $s \in S$ an estimate of the distance between that state and the goal state

The smaller the value $h(s)$, the closer is s to the goal state. If s is the goal state, then $h(s) = 0$

- Search strategies that use heuristics to narrow down the search are called **heuristic (informed, directed) search strategies**

An example: A journey through Istria



Air distances to Buzet:

Baderna	25
Barban	35
Buje	21
Grožnjan	17
Kanfanar	30
Labin	35
Lupoglav	13
Medulin	61
Motovun	12
Opatija	26
Pazin	17
Poreč	32
Pula	57
Rovinj	40
Umag	31
Višnjan	20
Vodnjan	47
Žminj	27

Another example: 8-puzzle

initial state:

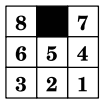
8		7
6	5	4
3	2	1

goal state:

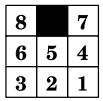
1	2	3
4	5	6
7	8	

What to use as heuristic function?

- Number of displaced squares:

  $h_1($  $) = 7$

- Sum of city-block (L1) distances between all squares and their final positions:

  $h_2($  $) = 21$

Notice that $h_2(s) \geq h_1(s)$

Heuristic search

We'll take a look at:

1. Greedy best-first search
2. Hill-climbing search
3. A* algorithm

A reminder: General search algorithm

General search algorithm

function search$(s_0, \text{succ}, \text{goal})$
  $open \leftarrow [\,\text{initial}(s_0)\,]$
  while $open \neq [\,]$ **do**
    $n \leftarrow \text{removeHead}(open)$
    if $\text{goal}(\text{state}(n))$ **then return** n
    for $m \in \text{expand}(n, \text{succ})$ **do**
      $\text{insert}(m, open)$
  return *fail*

Outline

Greedy best-first search

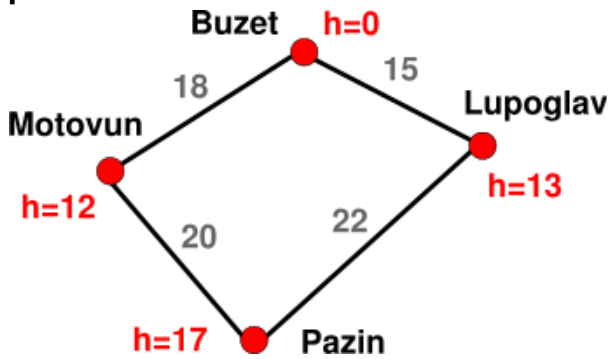
- Always expands the node with the best (lowest) heuristic value

Best-first search

function greedyBestFirstSearch(s_0, succ, goal, h)
  $open \leftarrow [\,\text{initial}(s_0)\,]$
  while $open \neq [\,]$ **do**
    $n \leftarrow \text{removeHead}(open)$
    if goal(state(n)) **then return** n
    for $m \in \text{expand}(n, \text{succ})$ **do**
      insertSortedBy(f, m, $open$)
  return fail
where $f(n) = h(\text{state}(n))$

Greedy search

- This is **greedy** best-first search
- **Greedy search**: the algorithm chooses the node that appears to be the closest to the goal, disregarding the total path cost accumulated so far
- The chosen path may not be optimal, but the algorithm doesn't backtrack to correct this. Hence, a greedy algorithm is **not optimal**
- **Q:** An example?



- The algorithm is **incomplete** (unless we use visited states list)
- **Time and space complexity:** $\mathcal{O}(b^m)$

Hill-climbing search

- Similar to greedy best-first algorithm, but doesn't bother to keep the generated nodes in the memory

Hill-climbing search

function $\text{hillClimbingSearch}(s_0, \text{succ}, h)$
  $n \leftarrow \text{initial}(s_0)$
  loop do
    $M \leftarrow \text{expand}(n, \text{succ})$
    if $M = \varnothing$ **then return** n
    $m \leftarrow \text{minimumBy}(f, M)$
    if $f(n) < f(m)$ **then return** n
    $n \leftarrow m$
where $f(n) = h(\text{state}(n))$

Hill-climbing search – properties

- **Not complete** and **not optimal**
- Easily trapped in so-called **local optima**
- Efficiency crucially depends on the choice of the heuristic function
- One typically employs the **random restart** technique
- **Time complexity**: $\mathcal{O}(m)$
- **Space complexity**: $\mathcal{O}(1)$

Outline

A* algorithm

- Similar to best-first algorithm, but takes into account both the heuristics and the path cost, i.e., the algorithm combines best-first and uniform cost search
- Path costs are updated when nodes are expanded (similarly as with uniform cost search):

function $\mathrm{expand}(n, \mathrm{succ})$
  return $\{ (s, g(n) + c) \mid (s, c) \in \mathrm{succ}(\mathrm{state}(n)) \}$

The total costs is computed as:

$g(n)$ – **true path cost** from the initial node to node n

$h(s)$ – **path cost estimate** from state s to the goal

$$f(n) = g(n) + h(\mathrm{state}(n))$$

A* search – implementation

A* search

function aStarSearch(s_0, succ, goal, h)
  $open \leftarrow [\,\text{initial}(s_0)\,]$
  $closed \leftarrow \varnothing$
  while $open \neq [\,]$ **do**
    $n \leftarrow \text{removeHead}(open)$
    if goal(state(n)) **then return** n
    $closed \leftarrow closed \cup \{\, n \,\}$
    for $m \in \text{expand}(n)$ **do**
      if $\exists m' \in closed \cup open$ **such that** $\text{state}(m') = \text{state}(m)$ **then**
        if $g(m') < g(m)$ **then continue**
        else $\text{remove}(m', closed \cup open)$
      $\text{insertSortedBy}(\text{f}, m, open)$
  return *fail*
where $f(n) = g(n) + h(\text{state}(n))$

A* search – example of execution

0. $open = [\,(Pula, 0)\,]$
   $closed = \varnothing$

1. $\text{expand}(Pula, 0) = \{(Vodnjan, 12), (Barban, 28), (Medulin, 9)\}$
   $open = [\,(Vodnjan, 12)^{59}, (Barban, 28)^{63}, (Medulin, 9)^{70}\,]$
   $closed = \{(Pula, 0)\}$

2. $\text{expand}(Vodnjan, 12) = \{(Kanfanar, 41), (Pula, 24)\}$
   $open = [\,(Barban, 28)^{63}, (Medulin, 9)^{70}, (Kanfanar, 41)^{71}\,]$
   $closed = \{(Pula, 0), (Vodnjan, 12)\}$

3. $\text{expand}(Barban, 28) = \{(Labin, 43), (Pula, 56)\}$
   $open = [\,(Medulin, 9)^{70}, (Kanfanar, 41)^{71}, (Labin, 43)^{78}\,]$
   $closed = \{(Barban, 28), (Pula, 0), (Vodnjan, 12)\}$

4. $\text{expand}(Medulin, 9) = \{(Pula, 18)\}$
   $open = [\,(Kanfanar, 41)^{71}, (Labin, 43)^{78}\,]$
   $closed = \{(Barban, 28), (Medulin, 9), (Pula, 0), (Vodnjan, 12)\}$

   $\vdots$

A* algorithm – properties

- **Time and space complexity:** $\mathcal{O}(\min(b^{d+1}, b|S|))$
  (especially space complexity is problematic in practice)
- **Completeness**: yes, because it accounts for path costs
- **Optimality**: yes, but provided the heuristic h is optimistic:

Optimistic heuristic

Heuristic function h is **optimistic** or **admissible** iff it never overestimates, i.e., its value is never greater than the true cost needed to reach the goal:

$$\forall s \in S.\ h(s) \leq h^*(s),$$

where $h^*(s)$ is the true path cost of reaching the goal state from state s

- If the heuristics is not optimistic, the search might bypass the optimal path because it seems more expensive than it really is
- **Q:** Are the heuristics from the previous examples optimistic?

An example: 8-puzzle

initial state:

8		7
6	5	4
3	2	1

goal state:

1	2	3
4	5	6
7	8	

Which heuristics are optimistic?

- $h_1(s) =$ number of displaced squares
- $h_2(s) =$ sum of city-block distances
- $h_3(s) = 0$
- $h_4(s) = 1$
- $h_5(s) = h^*(s)$
- $h_6(s) = \min(2, h^*(s))$
- $h_7(s) = \max(2, h^*(s))$

Quiz: Non-greedyness of A^*

The A^* is not greedy because it...

A uses a list of closed nodes

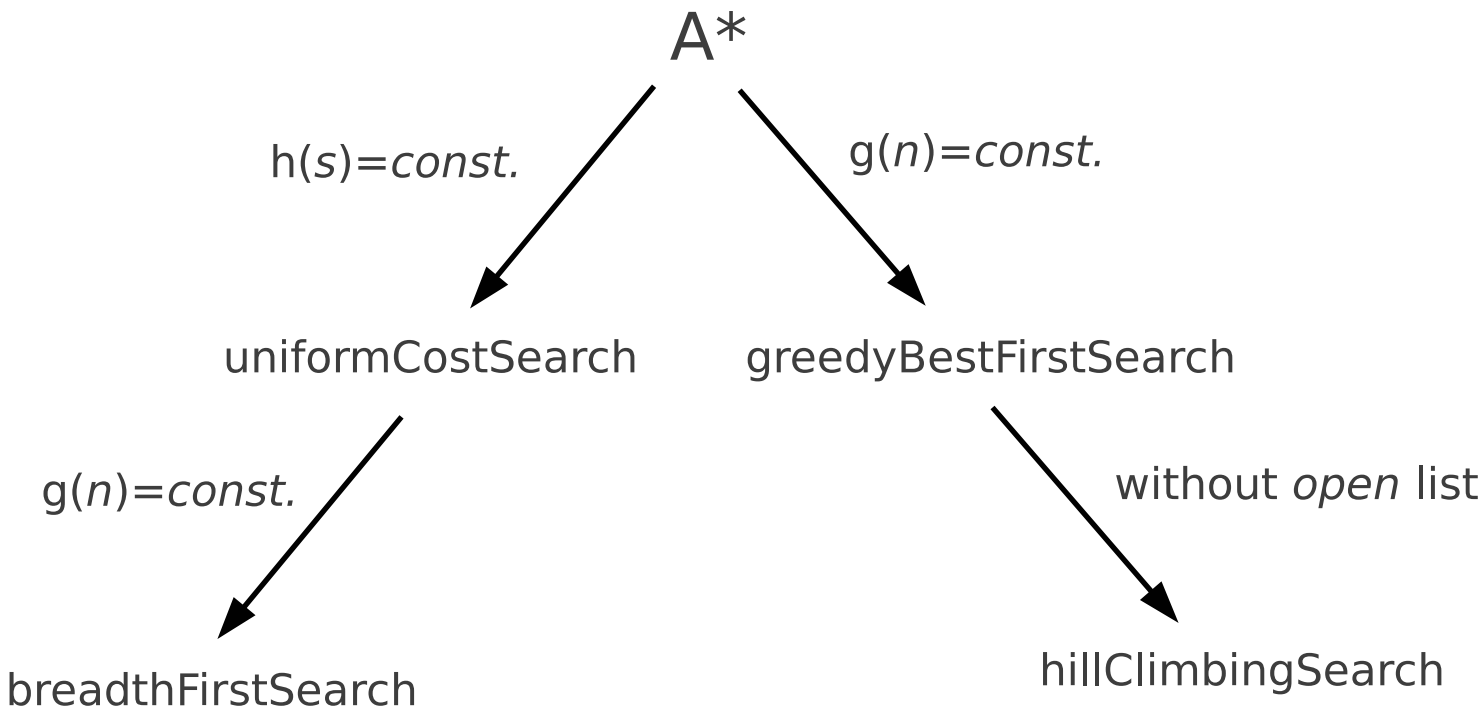
B uses a list of open nodes

C uses a heuristic function

D considers the cost from the initial node

E does not repeat the states already visited

Putting search algorithms in perspective



- A* dominates uniform cost search and breadth-first search

Outline

Consistent heuristics (1)

- Assuming optimistic heuristic, it holds
  $f(n) = g(n) + h(\text{state}(n)) \leq C^*$
  (the cost function has an upper bound)
- Along a path in the search tree, $f(n)$ may generally increase and decrease, but in the goal state it holds $f(n) = g(n) = C^*$
- It is desirable that $f(n)$ **monotonically increases**:

$$\forall n_2 \in \text{expand}(n_1) \implies f(n_2) \geq f(n_1)$$

  For perfect (oracle) heuristic h^*, we have $f(n_1) = f(n_2) = C^*$
- If $f(n)$ monotonically increases, then every node we **test (and close) for the first time for some state** will be the node with the **least path cost** to that particular state
- Thus, in case of repeated states, we don't need to check the path cost for already closed nodes (their cost is certainly less or equal)

Consistent heuristics (2)

- If $f(n)$ monotonically increases, then $\forall n_2 \in \text{expand}(n_1)$:

$$f(n_1) \leq f(n_2)$$

$$g(n_1) + h(\underbrace{\text{state}(n_1)}_{s_1}) \leq g(n_2) + h(\underbrace{\text{state}(n_2)}_{s_2})$$

$$g(n_1) + h(s_1) \leq \underbrace{g(n_1) + c}_{g(n_2)} + h(s_2)$$

$$h(s_1) \leq h(s_2) + c$$

Consistent heuristics

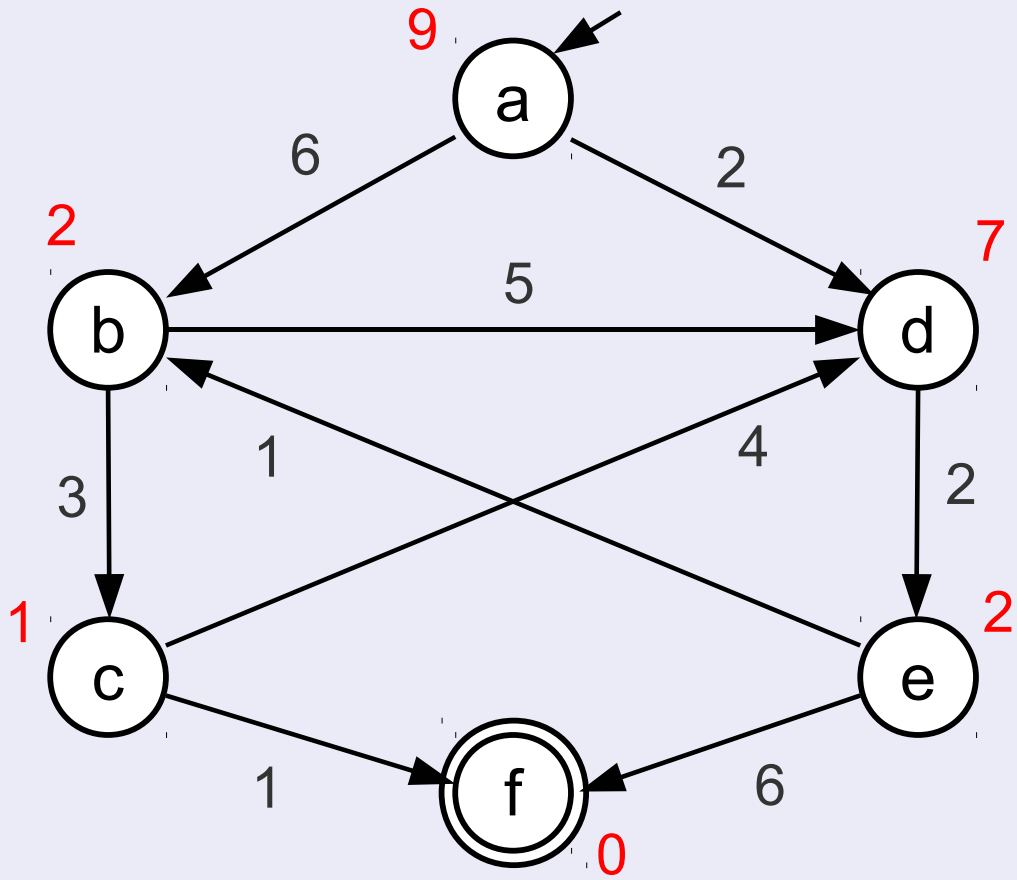
Heuristic h is **consistent** or **monotone** iff:

$$\forall (s_2, c) \in \text{succ}(s_1).\ h(s_1) \leq h(s_2) + c$$

- **NB:** A consistent heuristics is necessarily optimistic. On the other hand, in practice most optimistic heuristics are consistent

Quiz: Consistent heuristics

Is the following heuristics optimistic? Is it consistent?



A* algorithm – variants

The following variants of the A* algorithm are also **optimal**:

1. Without the use of a $closed$ list (but space and time inefficient because of repeated states)
2. With $closed$ list, but without re-opening of closed nodes, provided the heuristics is **consistent**
3. With $closed$ list and without re-opening, but using the **pathmax correction**:

$$f(n) = \max(f(\text{parent}(n)), g(n) + h(\text{state}(n)))$$

Domination property

Domination

Let A_1^* and A_2^* be optimal algorithms with *optimistic* heuristic functions h_1 and h_2. Algorithm A_1^* **dominates** algorithm A_2^* iff:

$$\forall s \in S.\ h_1(s) \geq h_2(s)$$

We also say that A_1^* is a **more informed** algorithm than A_2^*

- A more informed algorithm will generally search through a smaller state space than a less informed algorithm
- E.g., for 8-puzzle: $h^*(s) \geq h_2(s) \geq h_1(s)$,
  i.e., using city-block distance gives a more informed algorithm than when using the number of displaced squares
- The cost of computing the heuristics should also be taken into account!

Quiz: Heuristics

Let the current state s of the 8-puzzle be $[[1, 5, 2], [4, \square, 3], [7, 8, 6]]$, and the goal state be $[[1, 2, 3], [4, 5, 6], [7, 8, \square]]$. Assume h is an optimistic heuristics. Which of the following can be the most informed value of $h(s)$?

A 0

B 1

C 3

D 5

Good heuristic

- A good heuristic is:
  1. optimistic
  2. well informed
  3. simple to compute

Pessimistic heuristics?

- If we don't need an optimal solution, we can search for a "good enough" solution using a non-optimistic heuristic (one that for some states overestimates the true cost)
- Such a heuristic will additionally reduce the number of generated nodes
- Here we are trading off solution quality for computational efficiency

- How can one invent a good heuristics for a given problem?

Inventing heuristics

1. Problem relaxation
   - true cost of a **relaxed problem** gives an optimistic heuristic for the original problem
   - e.g. relaxing the 8-puzzle problem:
     the squares can move through other squares $\Rightarrow$ city-block distance
   - this yields a heuristic that is both optimistic and consistent (why?)
2. Combining several optimistic heuristics
   - If $h_1, h_2, \ldots, h_n$ are optimistic, we can combine them into a single dominant heuristic that will also be optimistic:

$$h(s) = \max\left(h_1(s), h_2(s), \ldots, h_n(s)\right)$$

3. Using sub-problem costs
   - pattern database containing precomputed true costs for a number of sub-problems
4. Learning heuristics
   - use of machine learning. E.g., learning of coefficients w_1 and w_2: $h(s) = w_1 x_1(s) + w_2 x_2(s)$, where x_1 are x_2 state describing features

Outline

Lab assignment: Path-finding on a terrain map

Write a program that uses the A* algorithm to find the optimal path between two points on a given terrain map.

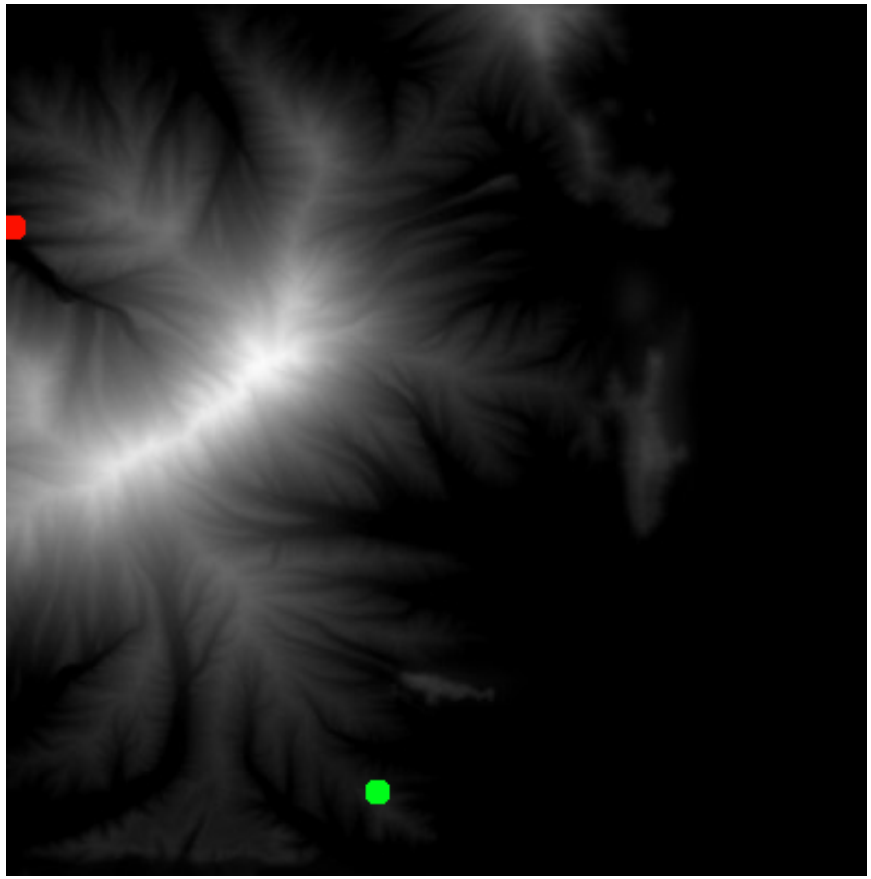
Let $\Delta a = a(x_2, y_2) - a(x_1, y_1)$ be the altitude difference. You can move from (x_1, y_1) to (x_2, y_2) if $|x_1 - x_2| \leq 1, |y_1 - y_2| \leq 1$ and $\Delta a \leq m$.

The cost of moving from position (x_1, y_1) to position (x_2, y_2) is

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \left(\frac{1}{2}\mathrm{sgn}(\Delta a) + 1\right) \cdot |\Delta a|$$

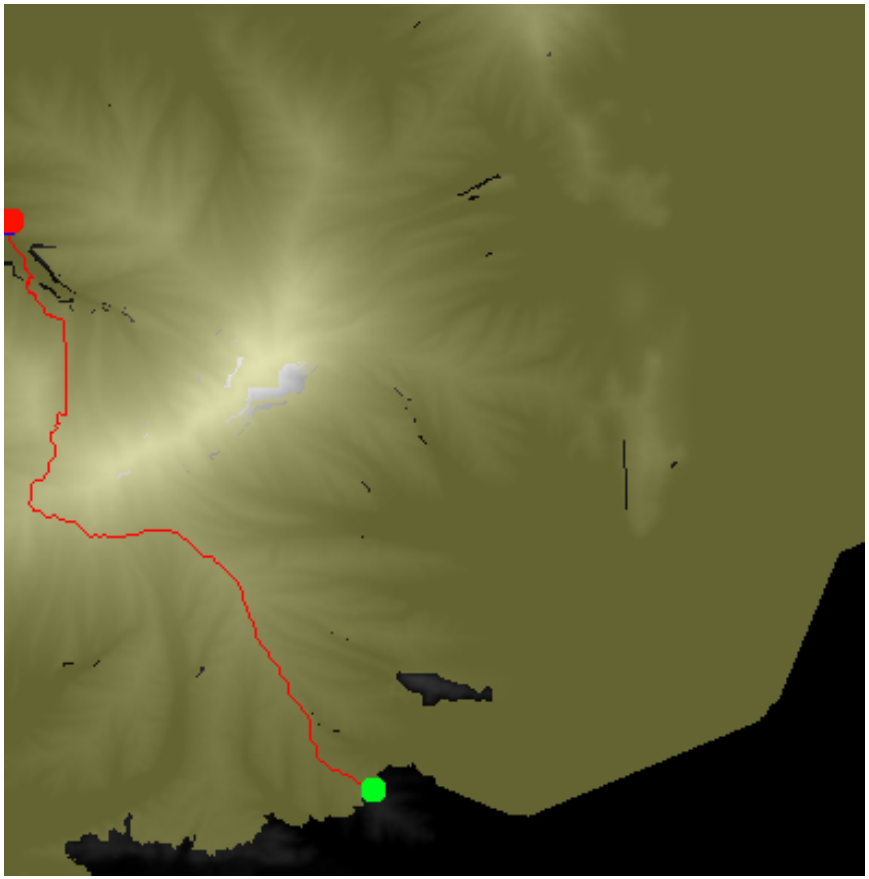
The program should read the map and the parameters from a specified file. It should generate an image of the height map with clearly indicated closed nodes, open nodes and the path found. The program should also print out the length of the path found and the number of iterations of the algorithm. You should propose at least three different heuristics and try them out on the map.

Path-finding on a terrain map (1)



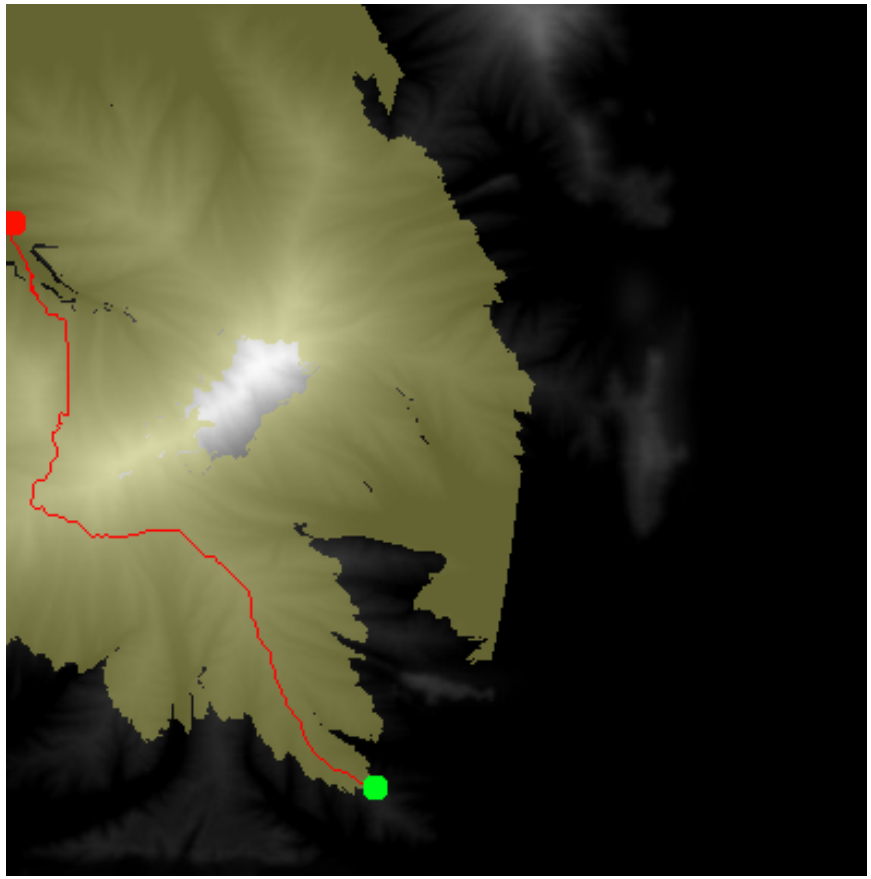
Terrain map
(brighter pixels correspond to positions at higher altitudes)
red: initial state; green: goal state

Path-finding on a terrain map (2)



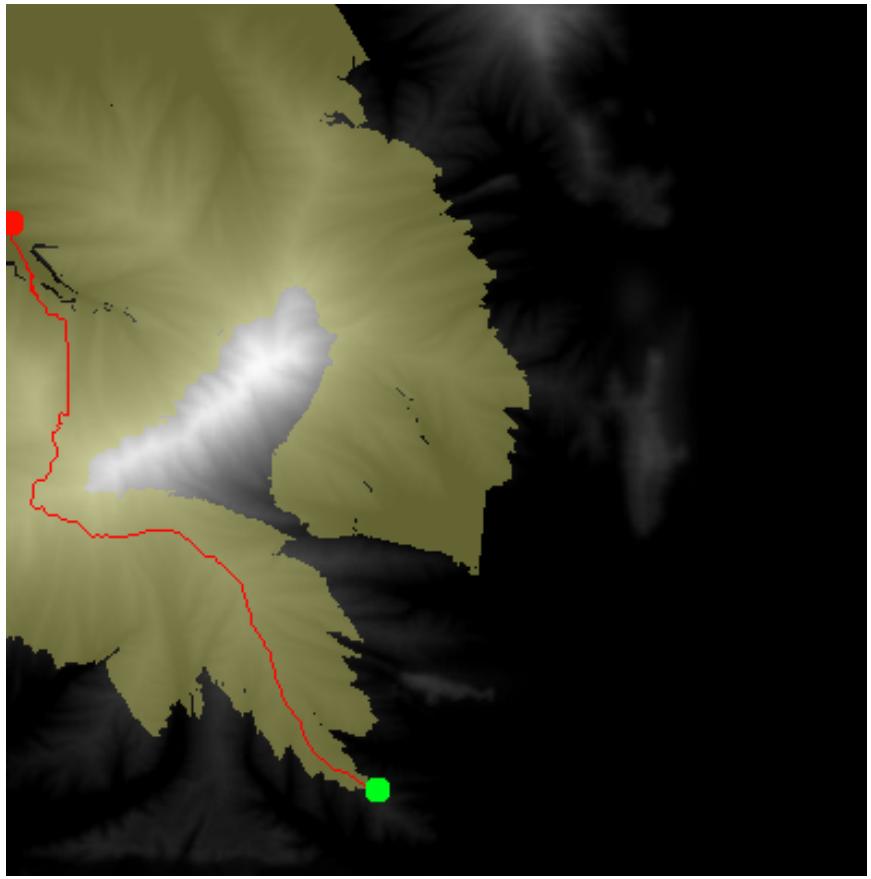
Uniform cost search
(red: path found; yellow: states visited)
number of closed nodes: 140580; path length: 740.58

Path-finding on a terrain map (3)



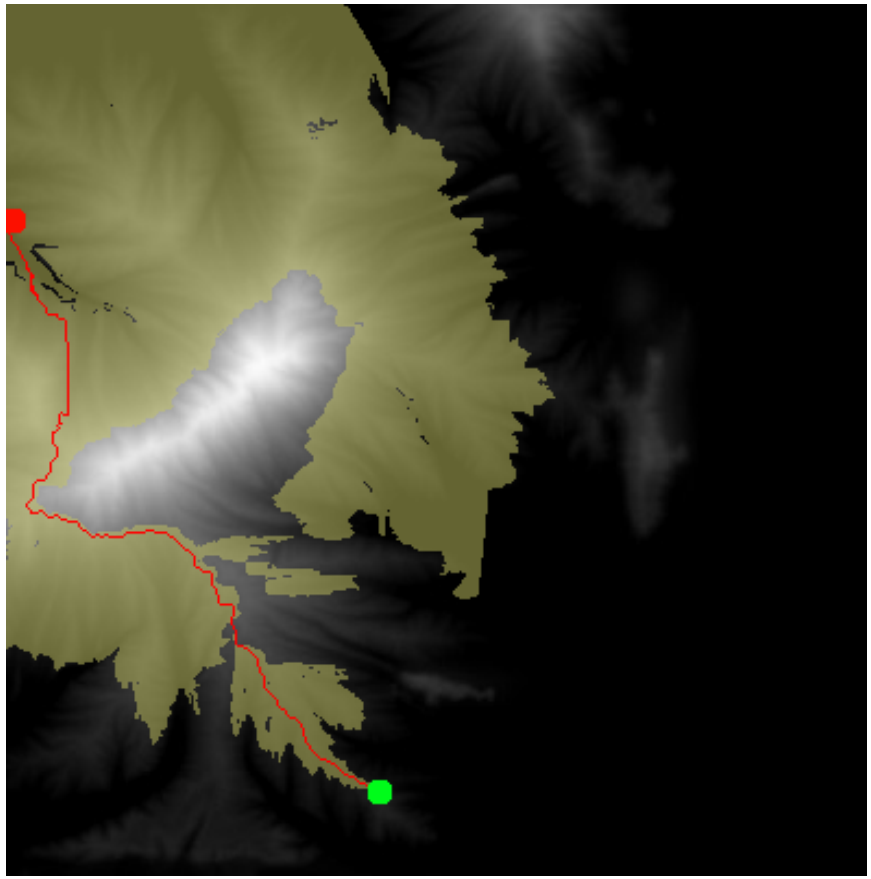
A* algorithm
(heuristics: air distance)
number of closed nodes: 64507; path length: 740.58

Path-finding on a terrain map (4)



A* algorithm
(heuristics: air distance + $\frac{1}{2}\Delta a$)
number of closed nodes: 56403; path length: 740.58

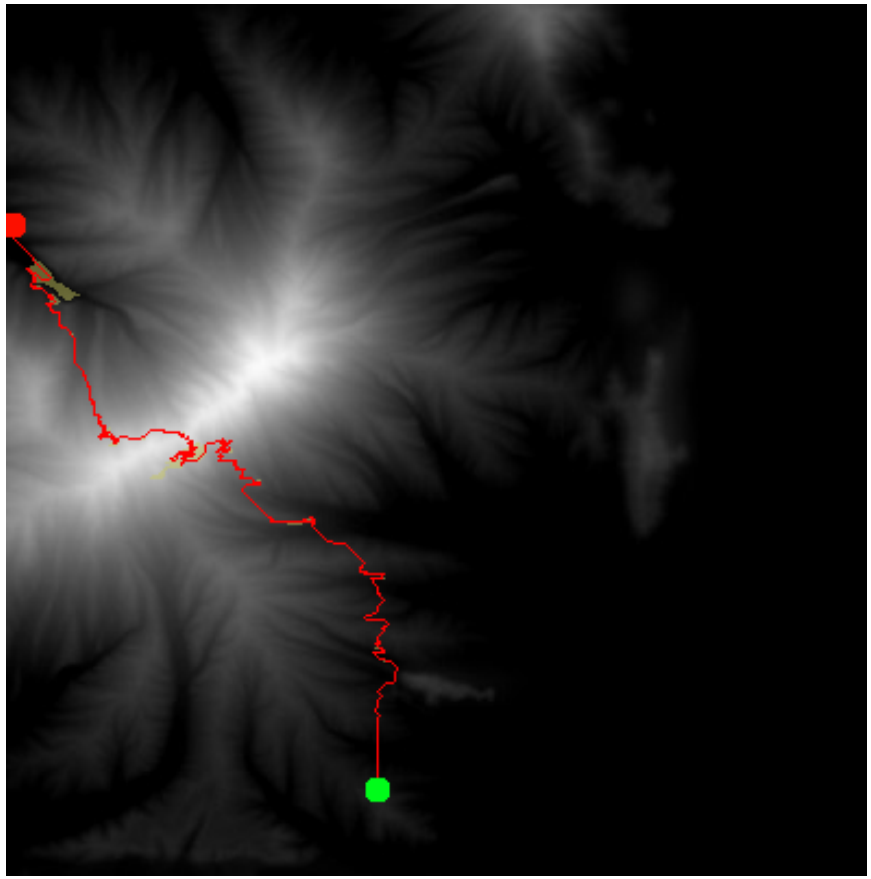
Path-finding on a terrain map (5)



A* algorithm
(heuristics: air distance + $|\Delta a|$)
number of closed nodes: 52099; path length: 755.16

This heuristic is not optimistic!

Path-finding on a terrain map (6)



Greedy best-first search
(heuristics: air distance; with visited states list)
number of closed nodes: 822; path length: 1428.54

Outline

Wrap-up

- Heuristic function **guides the search**, making it more efficient
- Heuristic function **estimates the distance** between the given state and the goal state. The smaller the distance, the closer we are to the goal state
- Heuristic should be **optimistic**, it may be **consistent**, and it should be **informed** as much as possible in order to make the search even more efficient
- Best-first and hill-climbing algorithms are **greedy** and therefore not optimal
- A* search is both **complete and optimal**

Next topic: Game playing