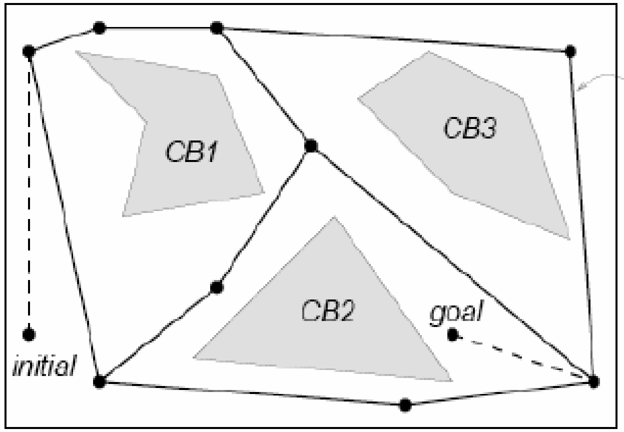
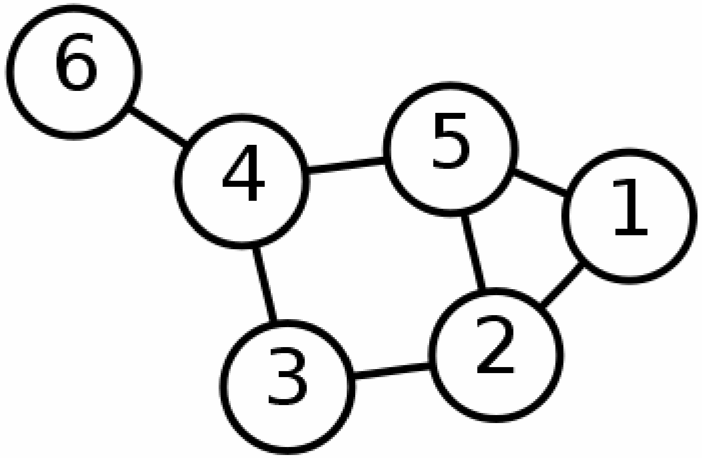
Topological maps

Planning in topological maps

- Topological map: simplified map with only relationship between points. It can be represented as a graph:
  - nodes are real positions
  - edges join positions in the free space, they include the distance
- It is easy to find a path in a topological map. How to build a topological map?
  - Visibility graph
  - Voronoi diagram
- How to solve the graph?
  - A* algorithm



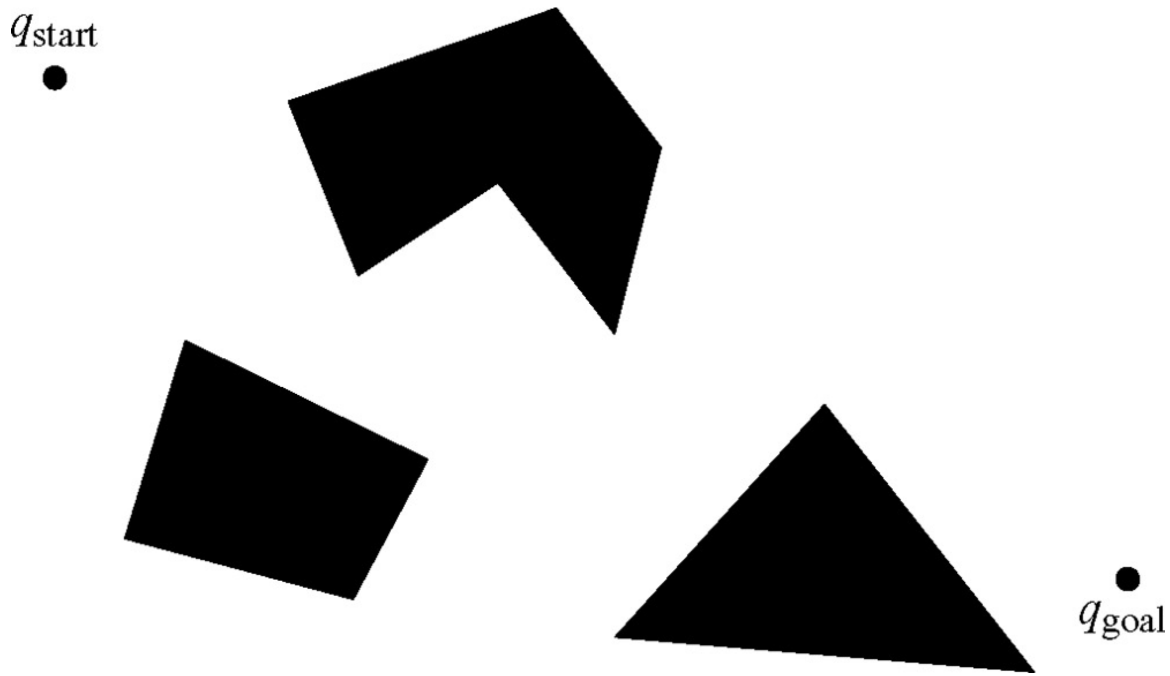


Top. Maps: Visibility Graph

Defined for a 2D polygonal configuration space

- The nodes v_i of the visibility graph include the start location, the goal location, and all the vertices of the configuration space obstacles.
- The graph edges e_{ij} are straight-line segments that connect two line-of-sight nodes ν_i and ν_j, i.e.,

$$e_{ij} \neq \emptyset \iff s v_i + (1-s) v_j \in \mathrm{cl}(\mathcal{Q}_{\mathrm{free}}) \ \forall s \in [0, 1].$$



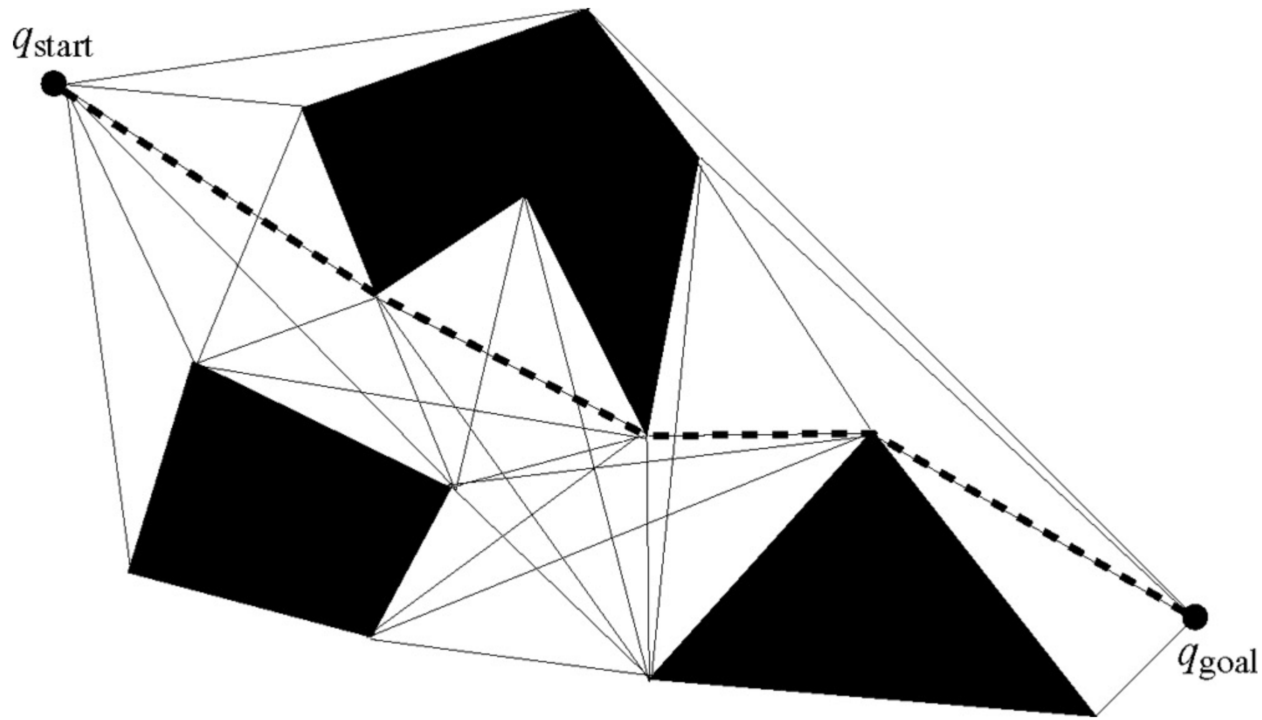
Top. Maps: Visibility Graph

- Construction of the visibility graph with n nodes has complexity n^3

  for all nodes; for all potential edges; for all obstacle edges

  wich can be reduced with the Rotational Plane Sweep Algorithm ($n^2 \log n$).

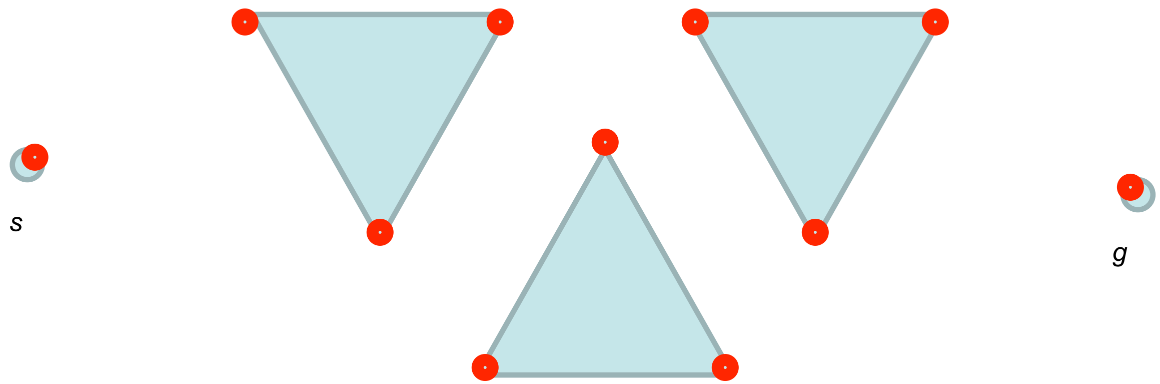
- Using the euclidean distance, the graph can be searched to find the shortest distance.



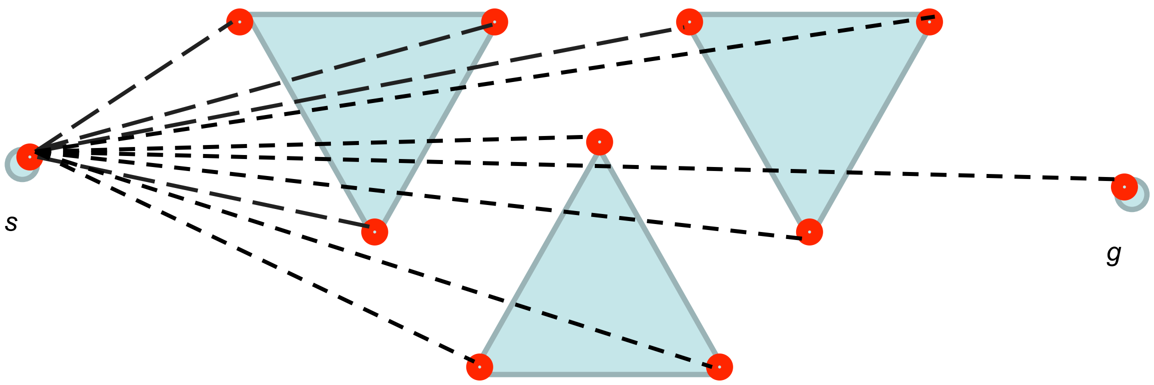
Top. Maps: Visibility Graph

Visibility graph construction with brute force

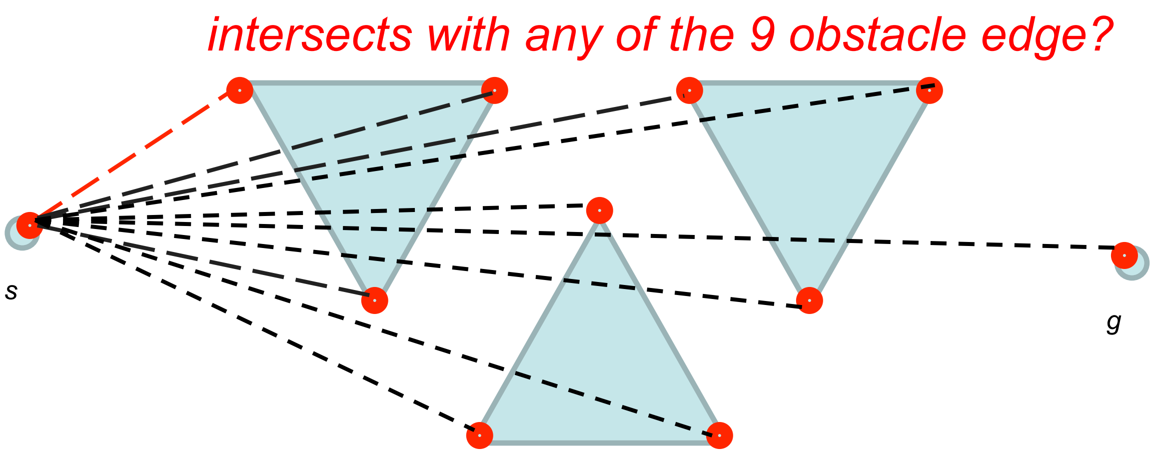
Top. Maps: Visibility Graph



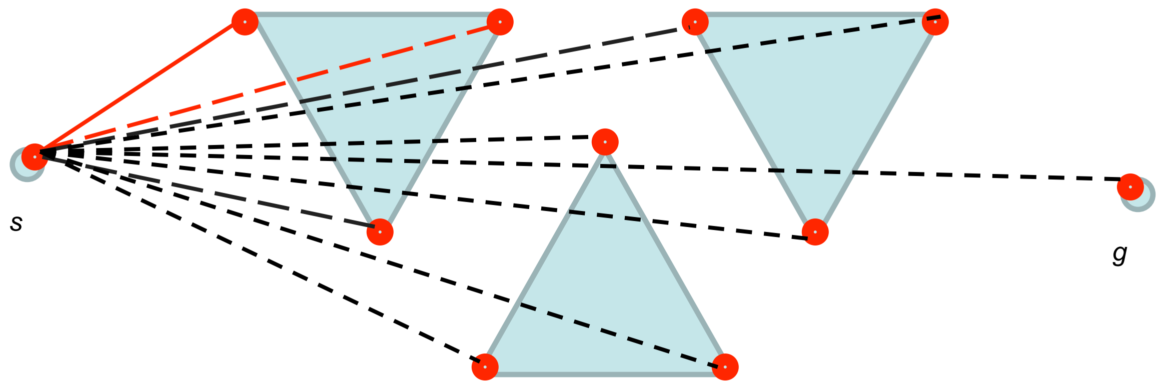
Top. Maps: Visibility Graph



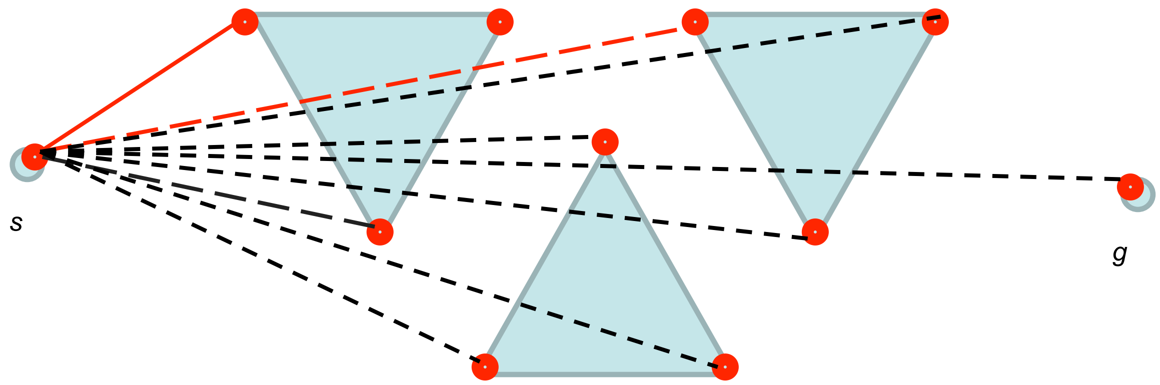
Top. Maps: Visibility Graph



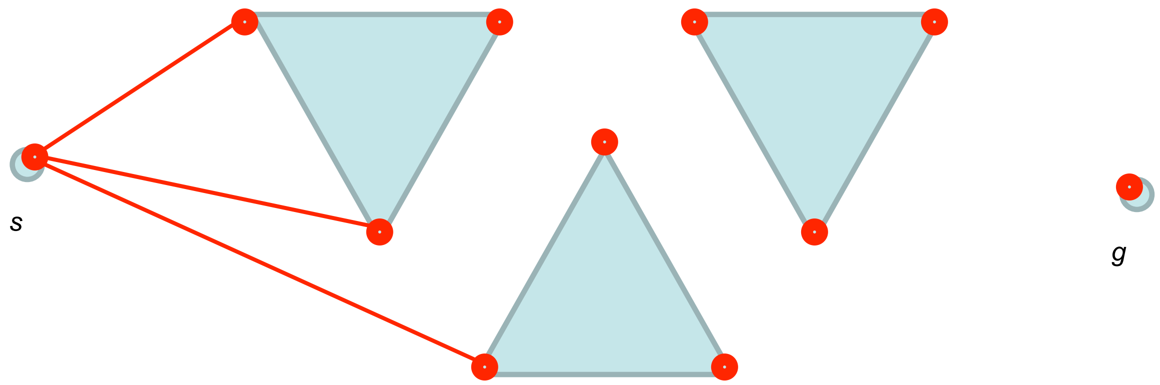
Top. Maps: Visibility Graph



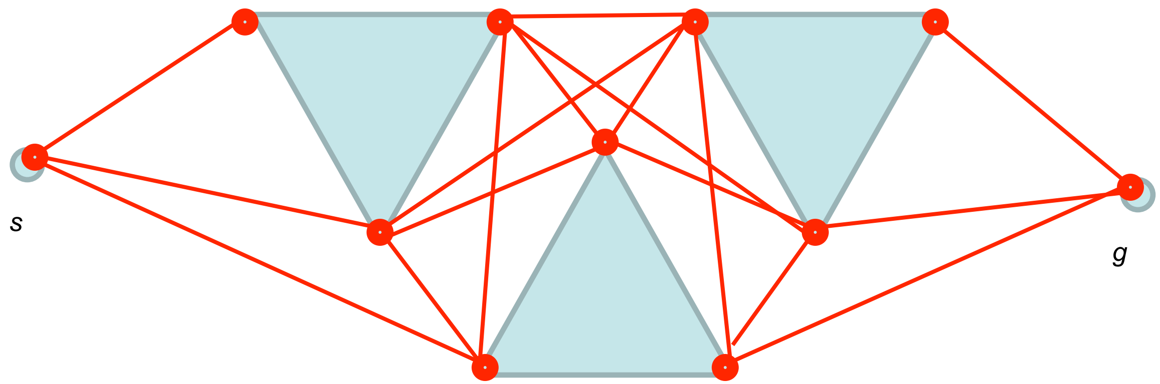
Top. Maps: Visibility Graph



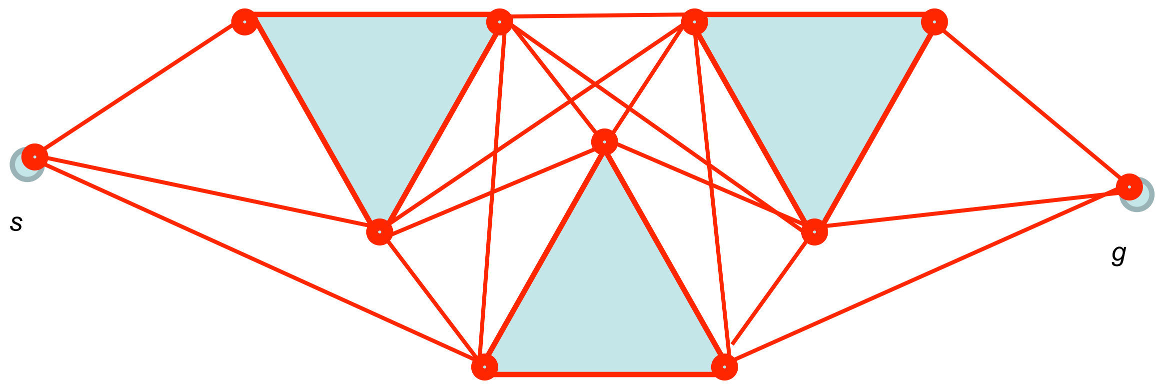
Top. Maps: Visibility Graph



Top. Maps: Visibility Graph



Top. Maps: Visibility Graph



Rotational plane sweep algorithm

Algorithm for building the visibility graph in a total time complexity of $n^2 \log n$:

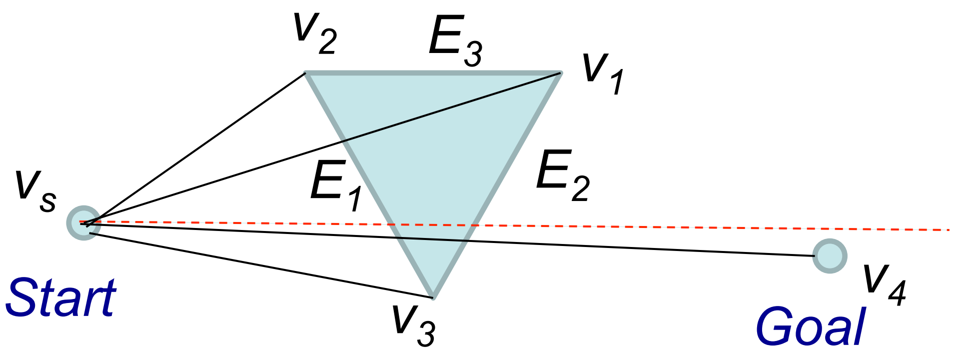
- A rotating half-line emanating from any vertex will be used to determine the vertices which are visible.
- The half-line has to stop only in the directions in which there is a vertex.
- At each vertex angle, a list of edges which intersect the beam will be updated (list S).
- Since the line rotates following the sorted list of vertex angles, list ε, the updating of the S list consists only on adding or removing the edges that contain the candidate vertex.
- Then, to determine if the vertex is visible, only intersection with lines contained in the S list, that are closer than the candidate vertex, have to be checked.

Rotational plane sweep algorithm

Algorithm 5: Rotational Plane Sweep Algorithm

Input: A set of vertices $\{v_i\}$ (whose edges do not intersect) and a vertex v
Output: A subset of vertices from $\{v_i\}$ that are within line of sight of v
1: For each vertex v_i, calculate α_i, the angle from the horizontal axis to the line segment vv_i.
2: Create the vertex list $\mathcal{E}$, containing the α_i 's sorted in increasing order.
3: Create the active list $\mathcal{S}$, containing the sorted list of edges that intersect the horizontal half-line emanating from v.
4: **for all** α_i **do**
5:   **if** v_i is visible to v **then**
6:     Add the edge (v, v_i)to the visibility graph.
7:   **end if**
8:   **if** v_i is the beginning of an edge, E, not in $\mathcal{S}$ **then**
9:     Insert the E into $\mathcal{S}$.
10:   **end if**
11:   **if** v_i is the end of an edge in $\mathcal{S}$ **then**
12:     Delete the edge from $\mathcal{S}$.
13:   **end if**
14: **end for**

Rotational plane sweep algorithm

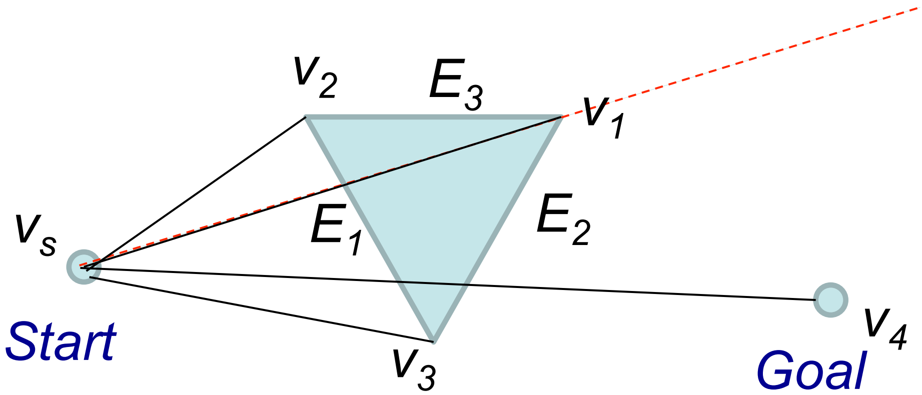


$\varepsilon=\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$

Initialization:

$S=\{E_1,E_2\}$

Rotational plane sweep algorithm



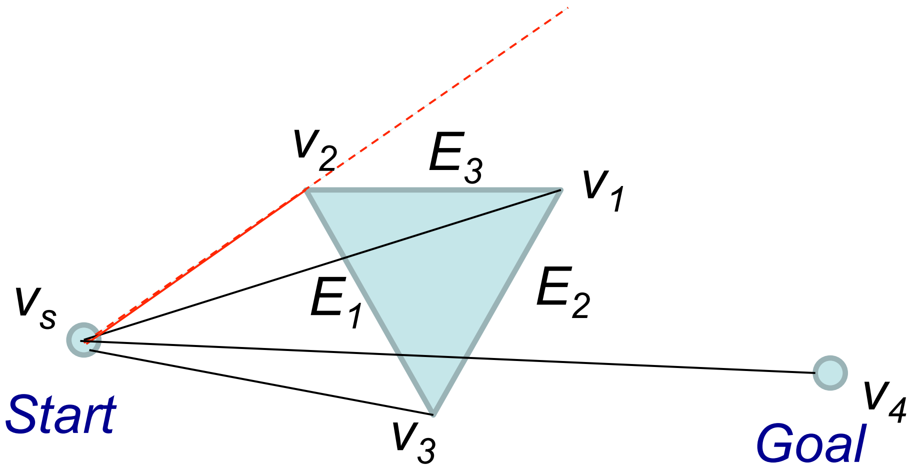
$\varepsilon=\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$

Iteration 1, stop at α_1 :

$S=\{E_1,E_3\}$

V_sV_1 intersects with E_1!

Rotational plane sweep algorithm



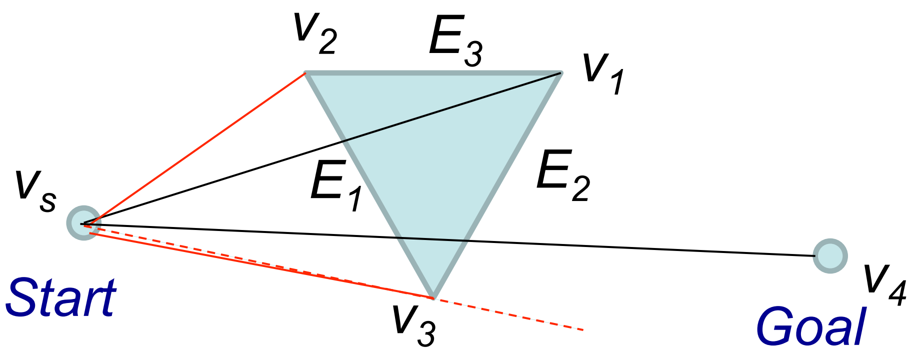
$\varepsilon=\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$

Iteration 2, stop at α_2 :
S={}

V_sV_2 is visible!

Top. Maps: Visibility Graph

Rotational plane sweep algorithm

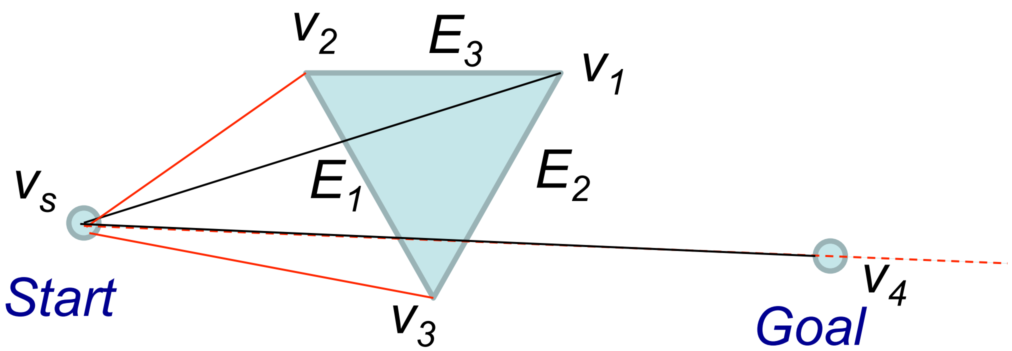


$\varepsilon=\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$

Iteration 3, stop at α_3 :
$S=\{E_1, E_2\}$

V_sV_3 does not intersect with E_1, it is visible!

Rotational plane sweep algorithm

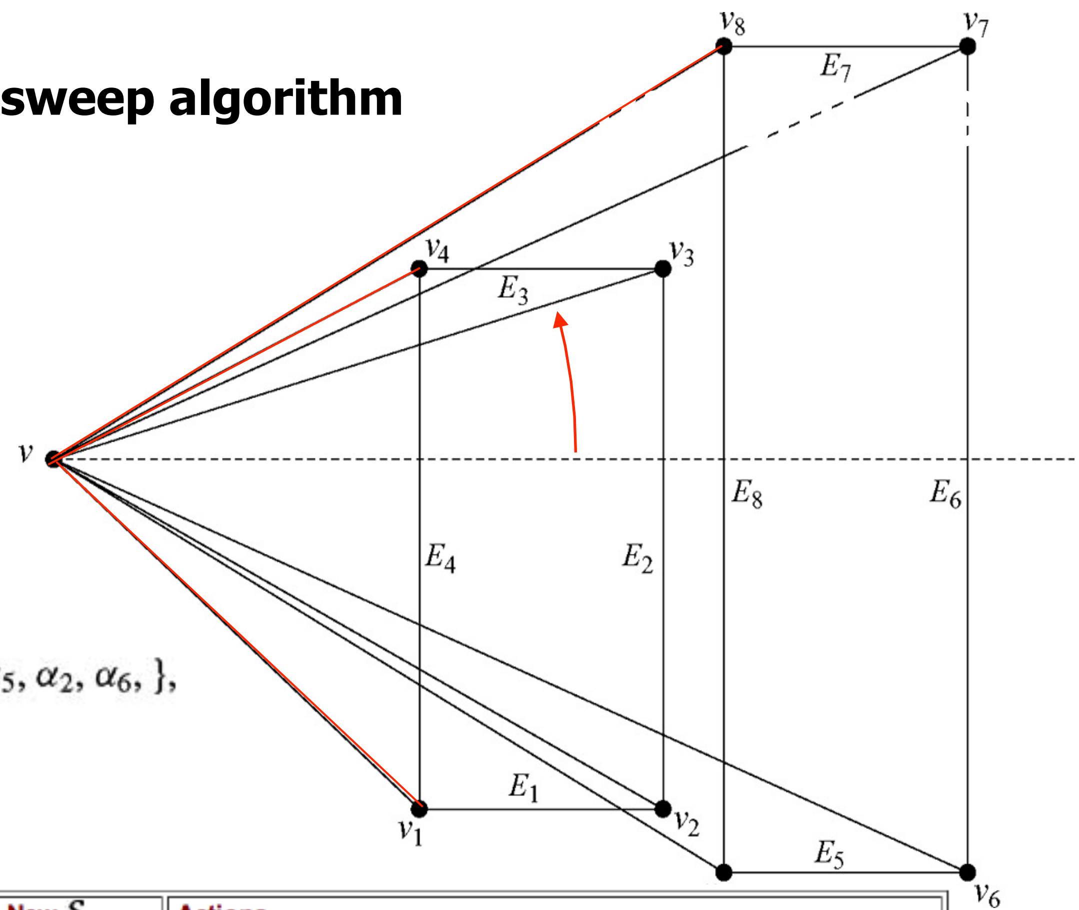


$\varepsilon=\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$

Iteration 4, stop at α_4 :
$S=\{E_1, E_2\}$

V_sV_4 intersects with E_1 and E_2!

Rotational plane sweep algorithm

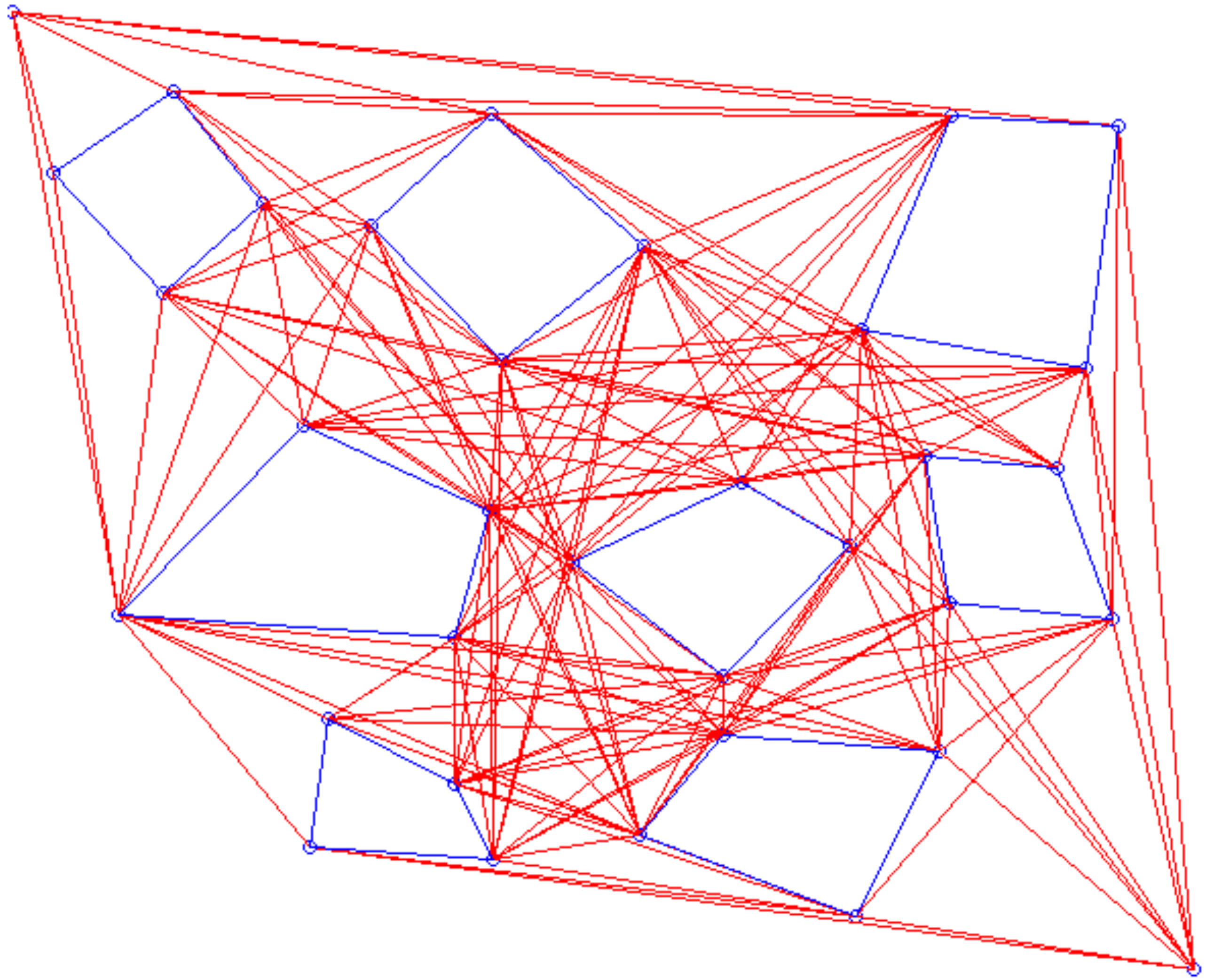


$$\mathcal{E} = \{\alpha_3, \alpha_7, \alpha_4, \alpha_8, \alpha_1, \alpha_5, \alpha_2, \alpha_6, \},$$

Vertex	New $\mathcal{S}$	Actions
Initialization	$\{E_4, E_2, E_8, E_6\}$	Sort edges intersecting horizontal half-line
α_3	$\{E_4, E_3, E_8, E_6\}$	Delete E_2 from $\mathcal{S}$. Add E_3 to $\mathcal{S}$.
α_7	$\{E_4, E_3, E_8, E_7\}$	Delete E_6 from $\mathcal{S}$. Add E_7 to $\mathcal{S}$.
α_4	$\{E_8, E_7\}$	Delete E_3 from $\mathcal{S}$. Delete E_4 from $\mathcal{S}$. ADD (v, v_4)to visibility graph
α_8	$\{\}$	Delete E_7 from $\mathcal{S}$. Delete E_8 from $\mathcal{S}$. ADD (v, v_8)to visibility graph
α_1	$\{E_1, E_4\}$	Add E_4 to $\mathcal{S}$. Add E_1 to $\mathcal{S}$. ADD (v, v_1)to visibility graph
α_5	$\{E_4, E_1, E_8, E_5\}$	Add E_8 to $\mathcal{S}$. Add E_5 to $\mathcal{S}$.
α_2	$\{E_4, E_2, E_8, E_5\}$	Delete E_1 from $\mathcal{S}$. Add E_2 to $\mathcal{S}$.
α_6	$\{E_4, E_2, E_8, E_6\}$	Delete E_5 from $\mathcal{S}$. Add E_6 to $\mathcal{S}$.
Termination		

Top. Maps: Visibility Graph

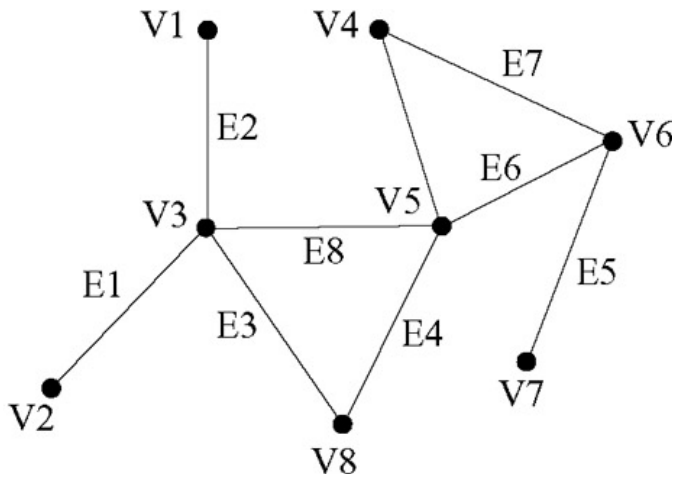
Rotational plane sweep algorithm



Graph search - A* algorithm

Graph:

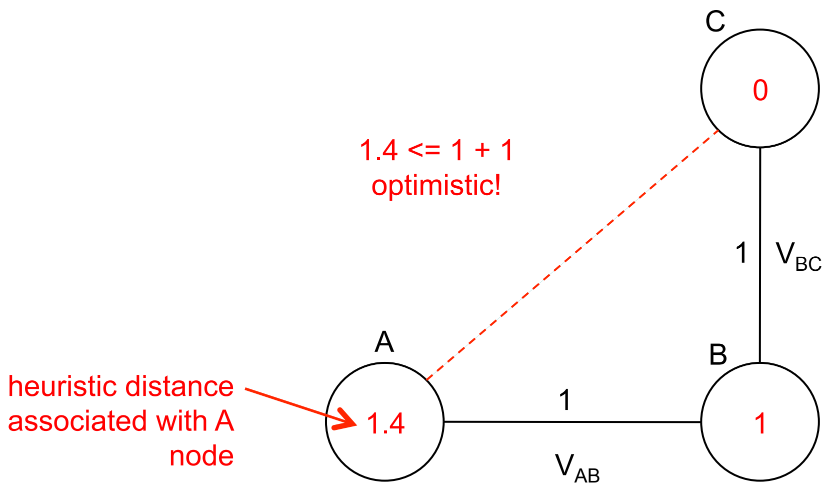
- Collection of nodes (Vi) and edges (Ei).
- In our application, nodes are interesting points generated by the Visibility Graph, Voronoi diagrams or free space positions in a grid map.
- The edges contain the euclidean distance between two nodes.
- Graph search consists in generating a sequence of connected nodes that has minimum length.
- Basic graph search algorithms can be very time-consuming if the number of nodes and edges is big.
- A* algorithm is a graph search algorithm that uses an heuristic to improve the search.



Graph search - A* algorithm

Heuristic Distance:

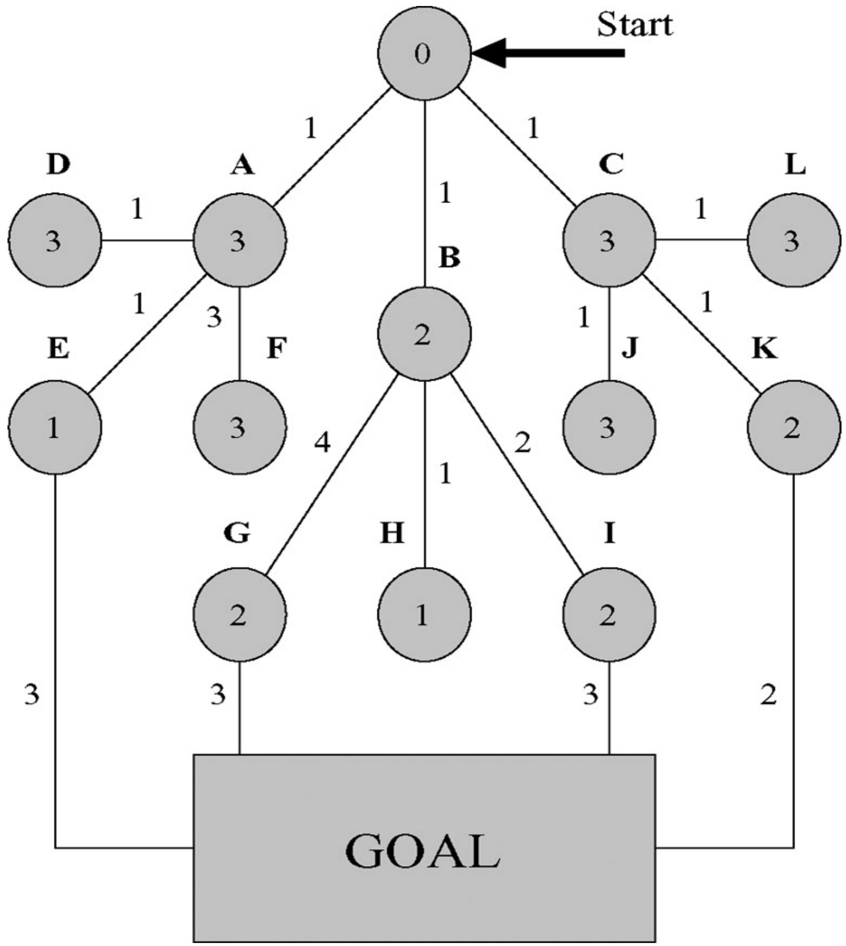
- The A* algorithm will search the graph efficiently with respect to a chosen heuristic.
- If the heuristic is good, then the search is efficient.
- If the heuristic is bad, the search will take more time although a path will be found.
- A* will produce an optimal path if its heuristic is optimistic:



Graph search - A* algorithm

Graph with heuristic distance on each node:

- The search starts in the top node
- The estimated cost of a node n is the sum of:
  - edge costs from n to start
  - heuristic distance from n to goal
- A* algorithm use two lists:
  - O list: set of nodes to explore
  - C list: set of explored nodes.



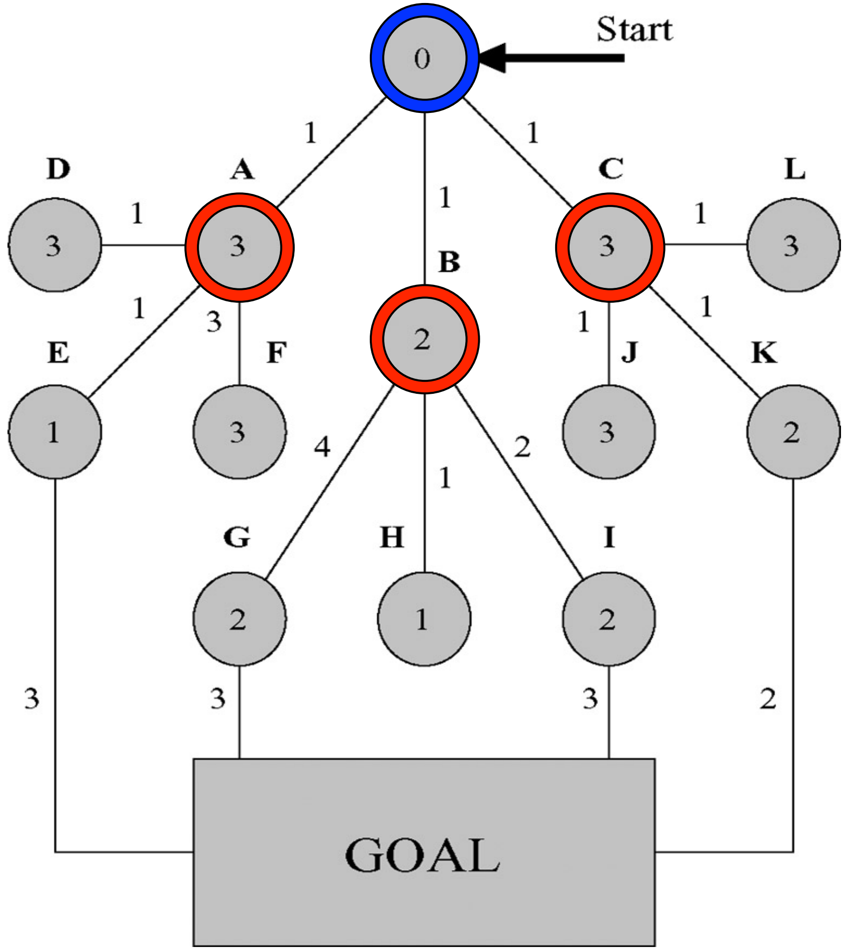
Graph search - A* algorithm

O list

Nodes	Cost
B	3
A	4
C	4

C list

Nodes	Backpointer
Start	-



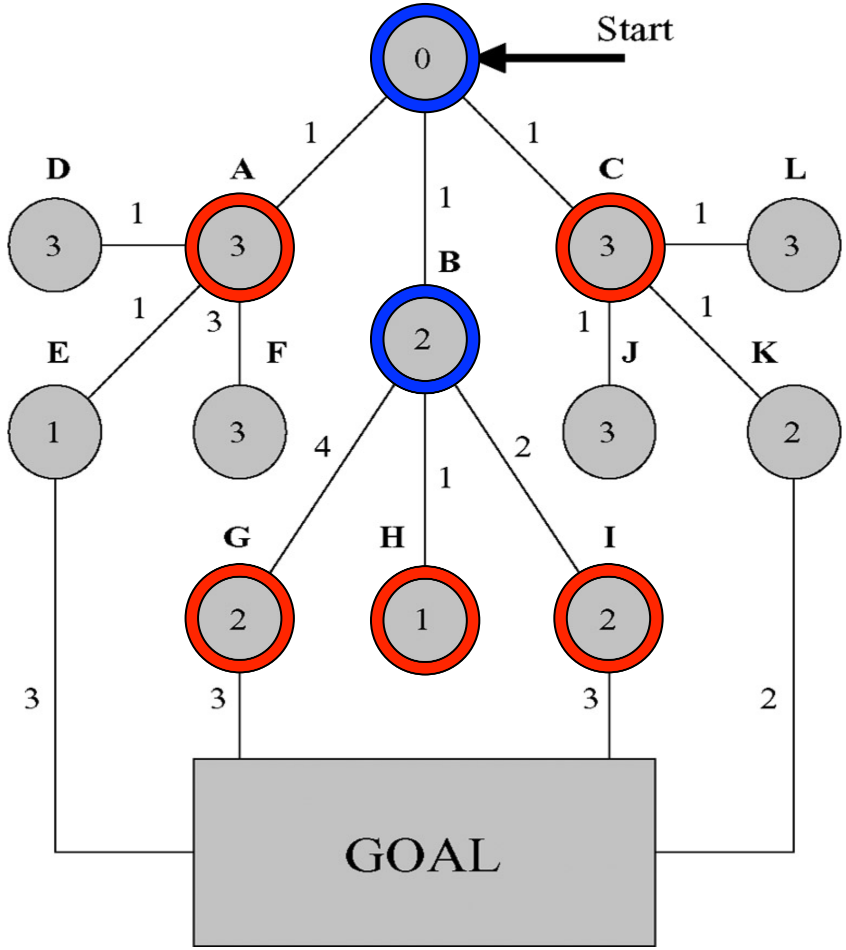
Graph search - A* algorithm

O list

Nodes	Cost
H	3
A	4
C	4
I	5
G	7

C list

Nodes	Backpointer
Start	-
B	Start



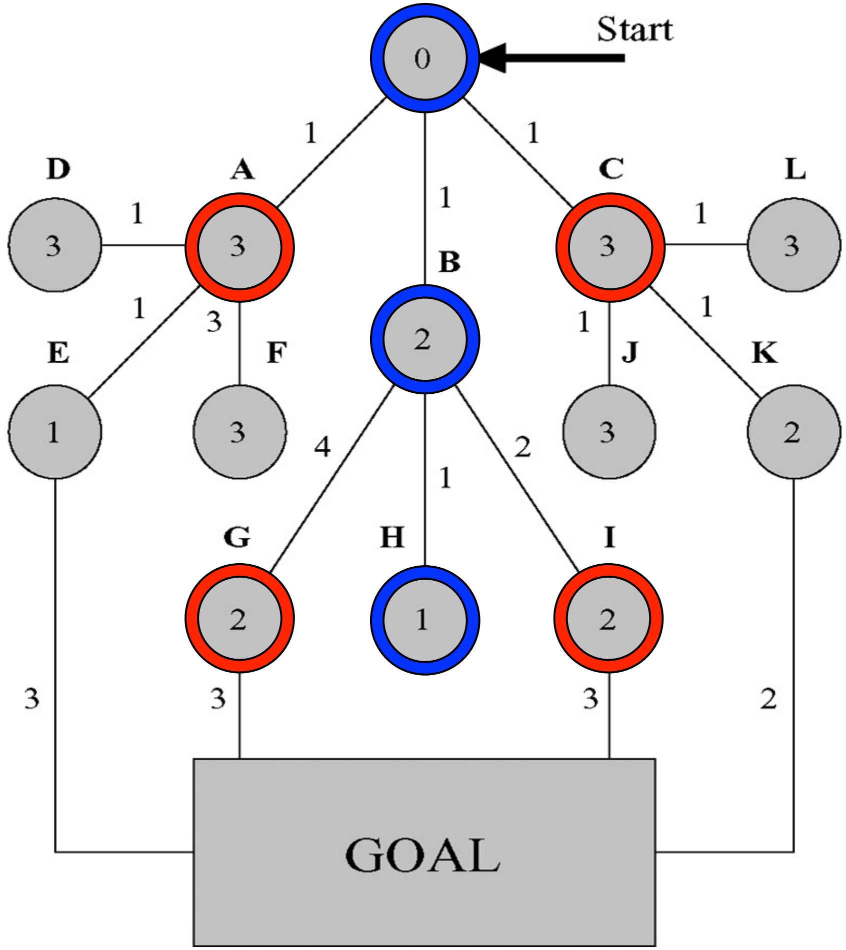
Graph search - A* algorithm

O list

Nodes	Cost
A	4
C	4
I	5
G	7

C list

Nodes	Backpointer
Start	-
B	Start
H	B



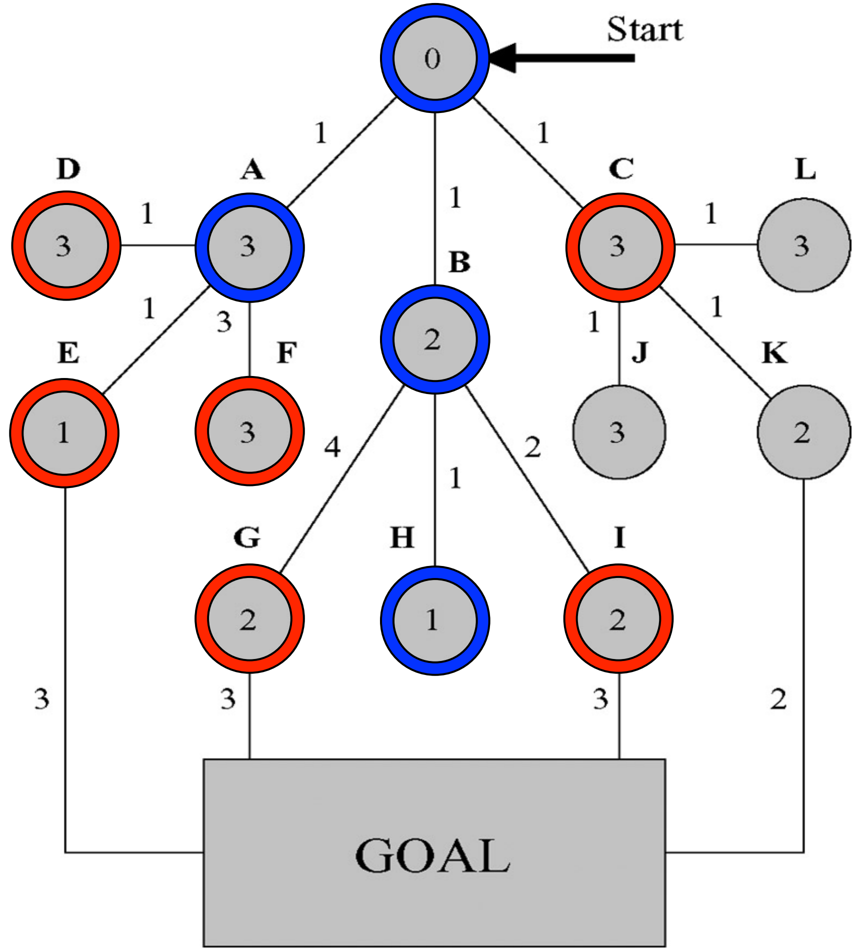
Graph search - A* algorithm

O list

Nodes	Cost
E	3
C	4
D	5
I	5
F	7
G	7

C list

Nodes	Backpointer
Start	-
B	Start
H	B
A	Start



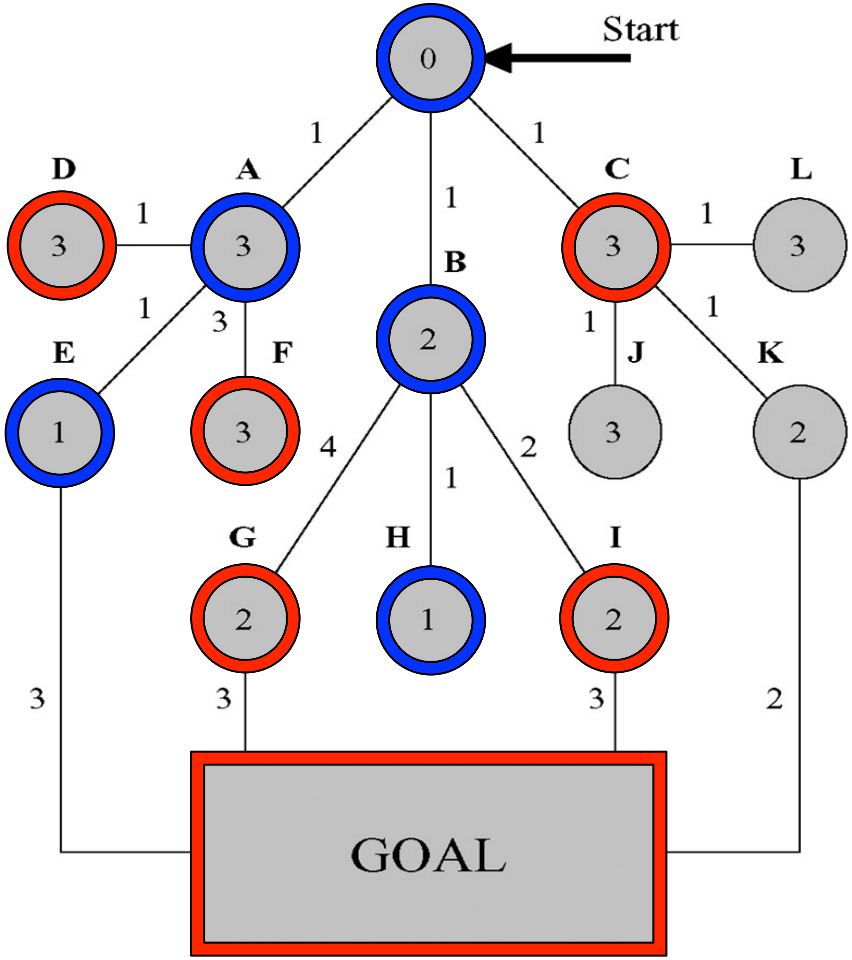
Graph search - A* algorithm

O list

Nodes	Cost
C	4
GOAL	5
D	5
I	5
F	7
G	7

C list

Nodes	Backpointer
Start	-
B	Start
H	B
A	Start
E	A



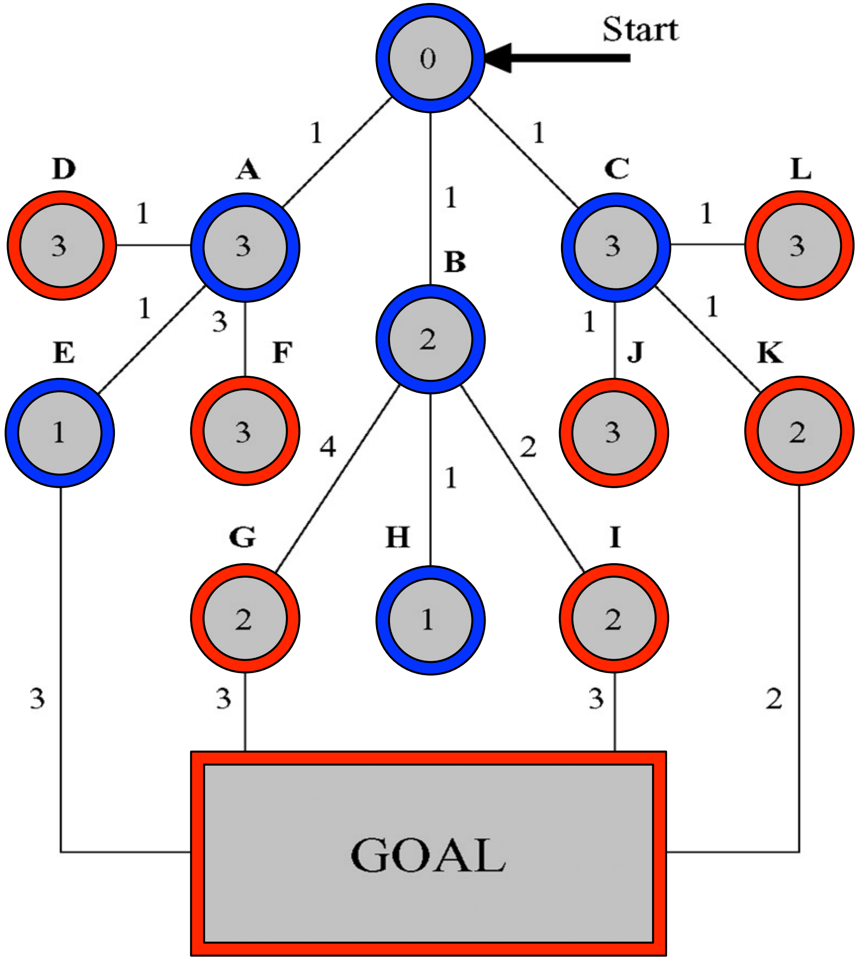
Graph search - A* algorithm

O list

Nodes	Cost
K	4
GOAL	5
L	5
J	5
D	5
I	5
F	7
G	7

C list

Nodes	Backpointer
Start	-
B	Start
H	B
A	Start
E	A
C	Start



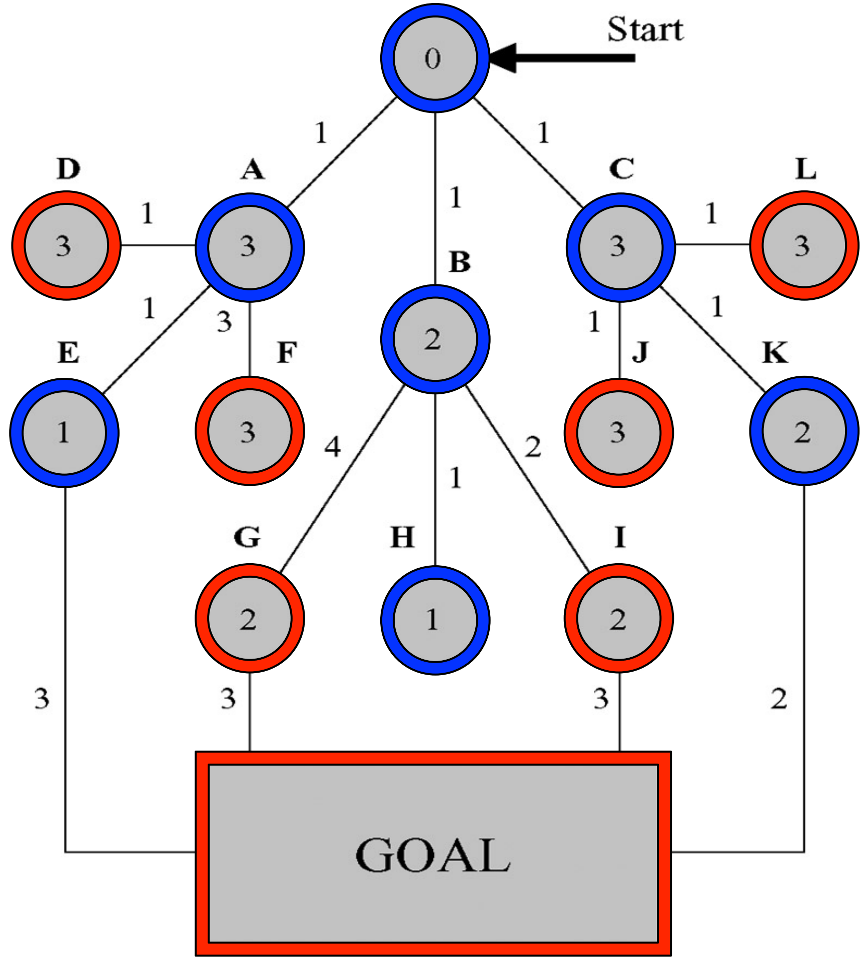
Graph search - A* algorithm

O list

Nodes	Cost
GOAL	4
L	5
J	5
D	5
I	5
F	7
G	7

C list

Nodes	Backpointer
Start	-
B	Start
H	B
A	Start
E	A
C	Start
K	C

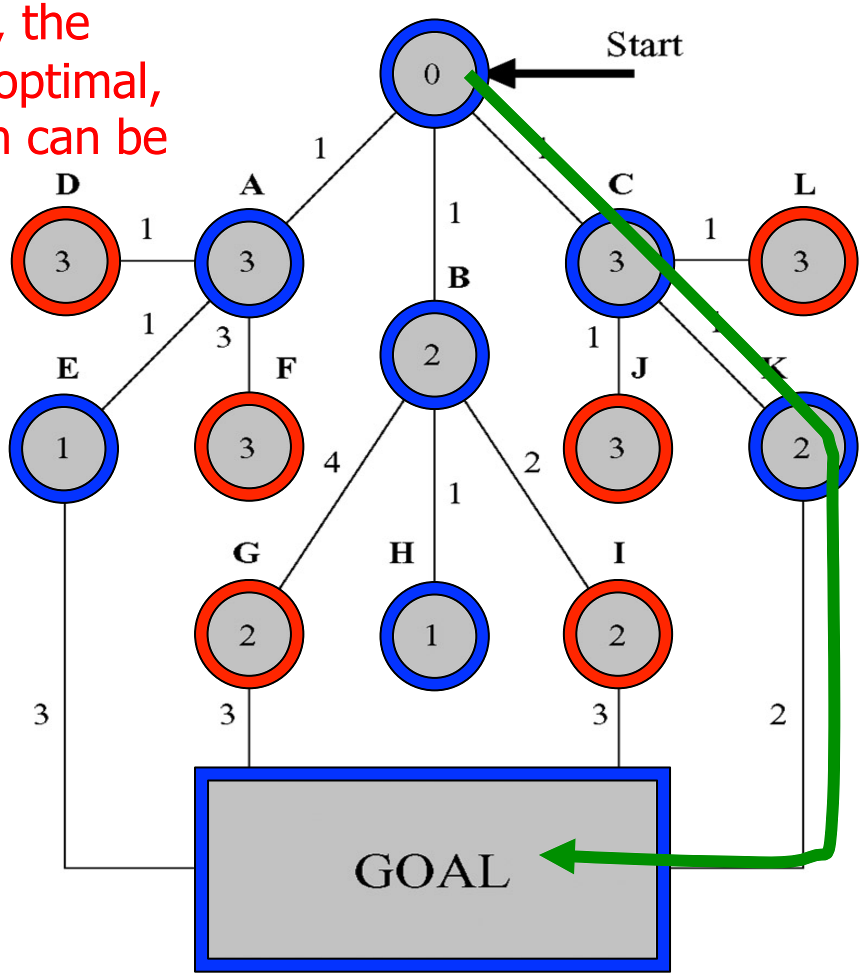


Graph search - A* algorithm

O list

Nodes	Cost
L	5
J	5
D	5
I	5
F	7
G	7

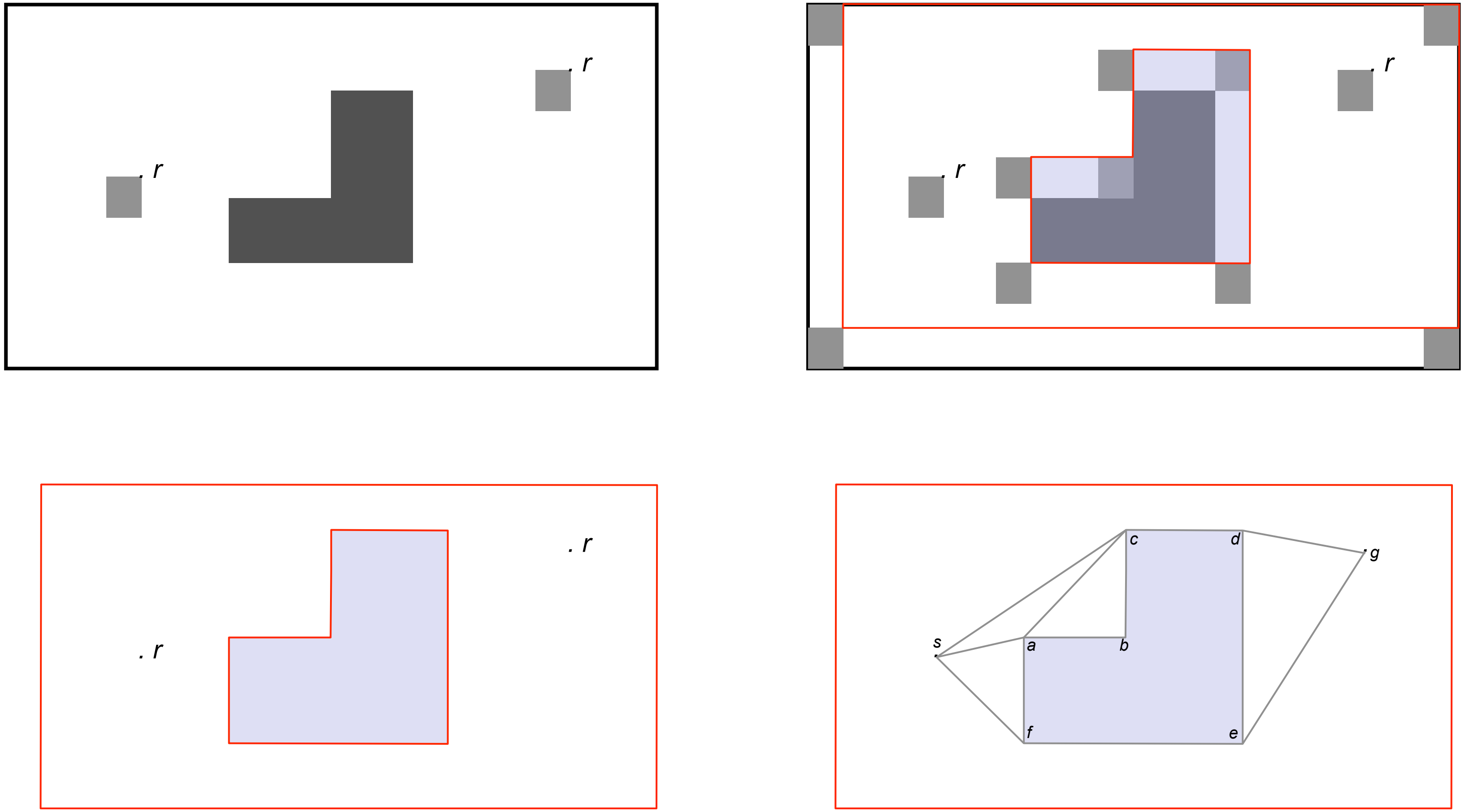
If the heuristic is optimistic, the results is optimal, the search can be stopped

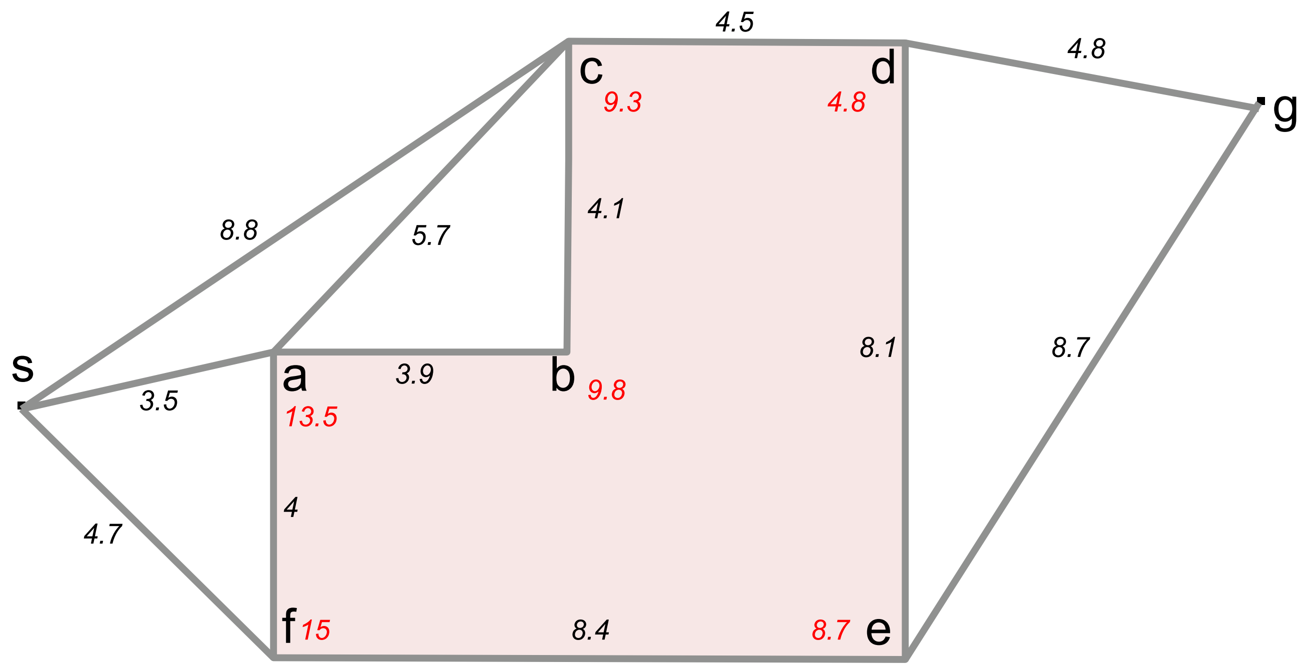


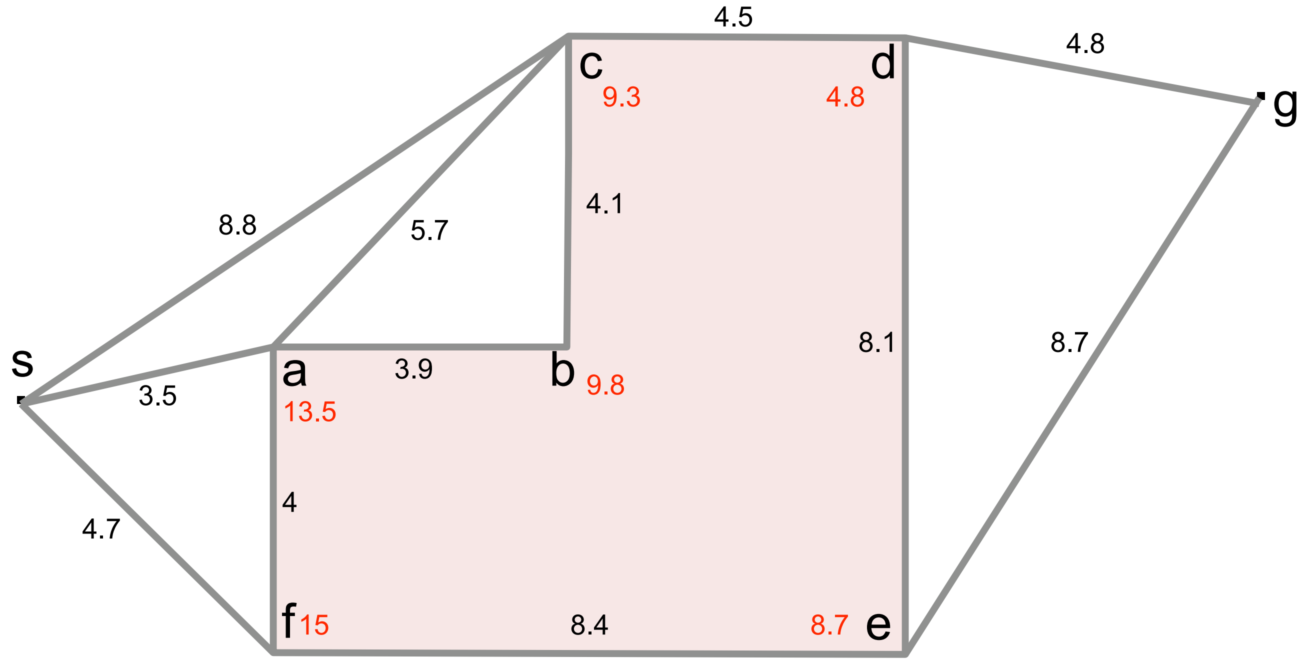
C list

Nodes	Backpointer
Start	-
B	Start
H	B
A	Start
E	A
C	Start
K	C
GOAL	K

Graph search - A* algorithm




4.5
4.8
c
d
9.3
4.8
g
8.8
5.7
4.1
8.1
8.7
s
a
3.9
b
9.8
3.5
13.5
4
4.7
f
15
8.4
8.7
e

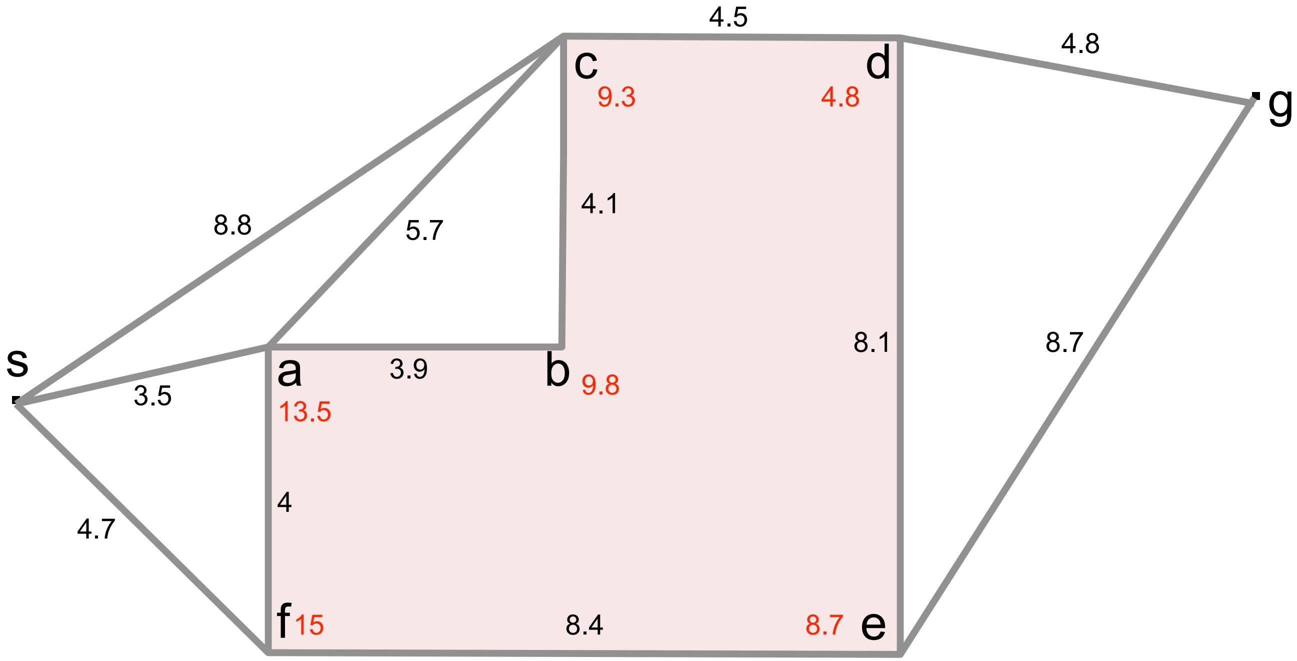


O list

Nodes	Cost
a	17
c	18.1
f	19.7

C list

Nodes	Backpointer
s	-

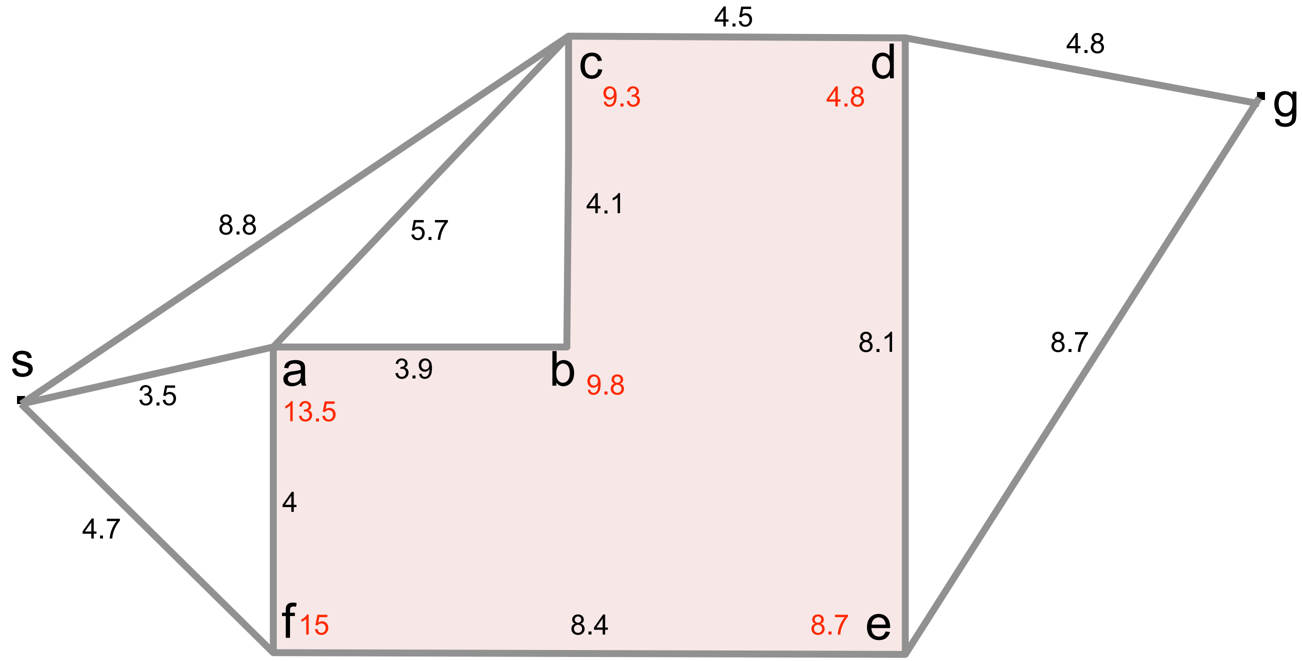


O list

Nodes	Cost
b	17.2
c	18.1
f	19.7

C list

Nodes	Backpointer
s	-
a	s

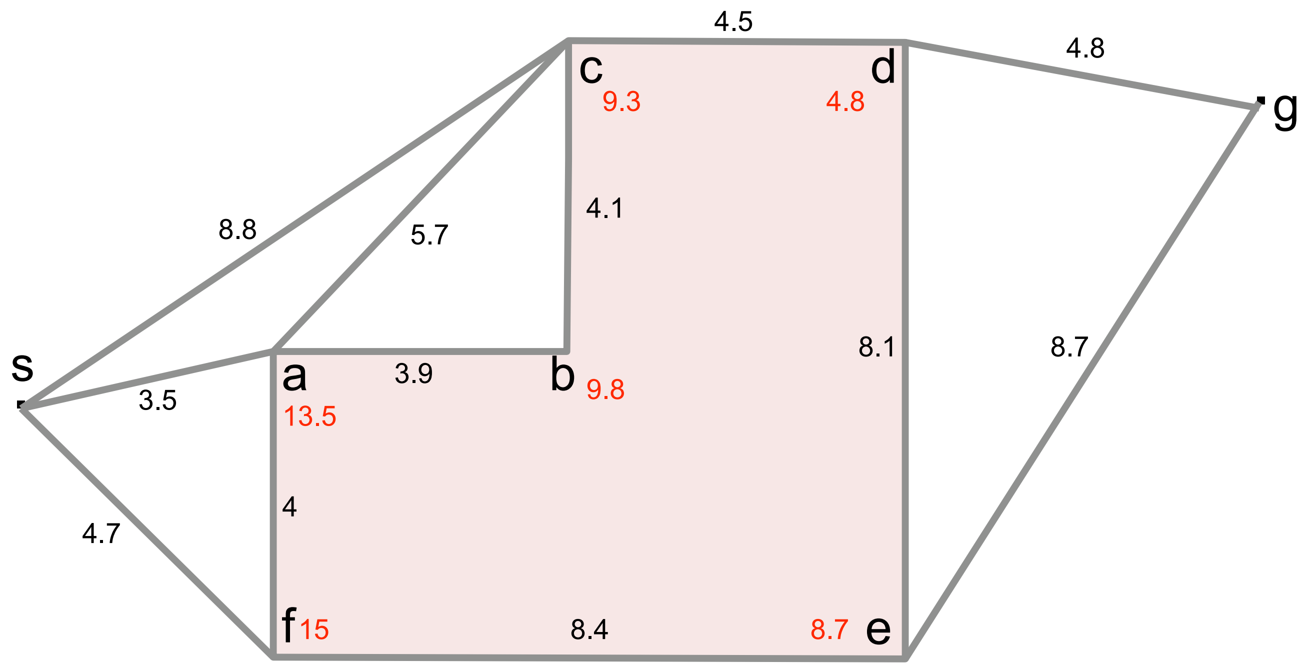


O list

Nodes	Cost
c	18.1
f	19.7

C list

Nodes	Backpointer
s	-
a	s
b	a

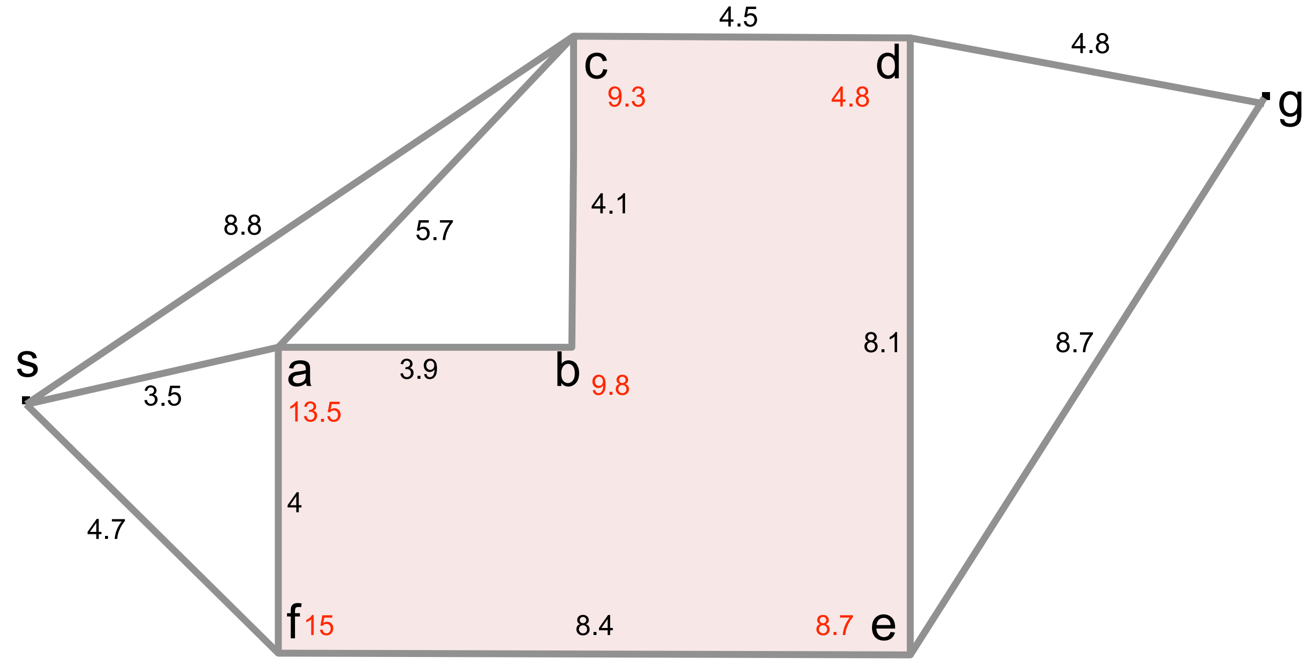


O list

Nodes	Cost
d	18.1
f	19.7

C list

Nodes	Backpointer
s	-
a	s
b	a
c	s

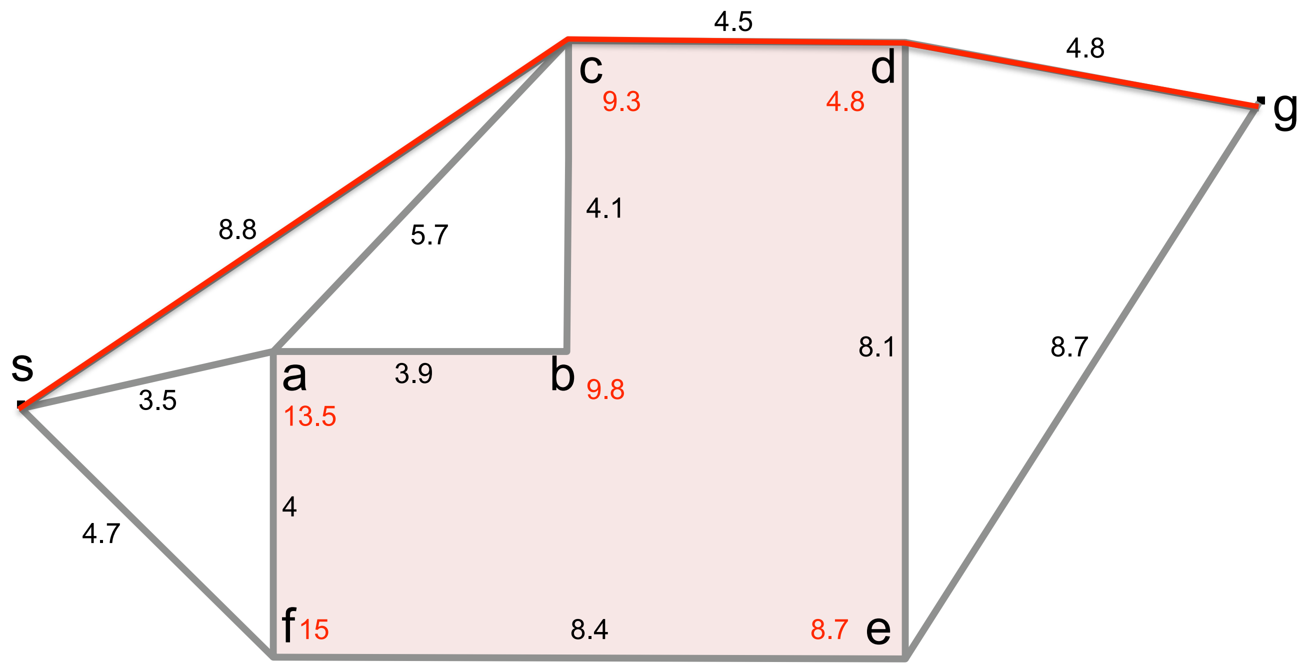


O list

Nodes	Cost
g	18.1
f	19.7
e	30.1

C list

Nodes	Backpointer
s	-
a	s
b	a
c	s
d	c



O list

Nodes	Cost
f	19.7
e	30.1

C list

Nodes	Backpointer
s	-
a	s
b	a
c	s
d	c
g	d