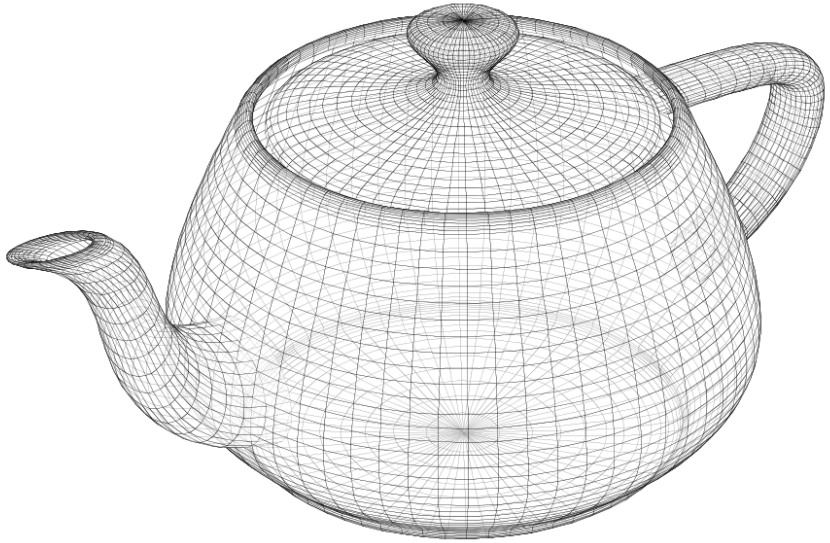


Game AI

Pathfinding: A*

Professor Eric Shaffer

Pathfinding in Games

Game characters typically need to move around a level

- Guards in a stealth game
- Units in a real-time strategy game

Scripted movement is easy for player to fool

Random wandering usually looks unrealistic…often unchallenging for player

ILLINOIS

A Brief History A*

The most famous search algorithm in AI

- Developed in 1968-ish
- It's a heuristic search…does not guarantee finding optimal solution
- Designed for point-to-point pathfinding
- Simple to implement and efficient to run
- Popularized in game industry in 1990s
  Brian Stout. *Smart moves: intelligent path-finding*. Game Developer Magazine, pages 28–35, October 1996.
- Virtually every pathfinding system in every game you play uses A*
- Can also be used as a decision-making tool to plan complex series of actions for characters.

ILLINOIS

The Problem

The problem is point-to-point pathfinding

Given a directed non-negative weighted graph and start and goal nodes

Generate a path with minimal total path cost

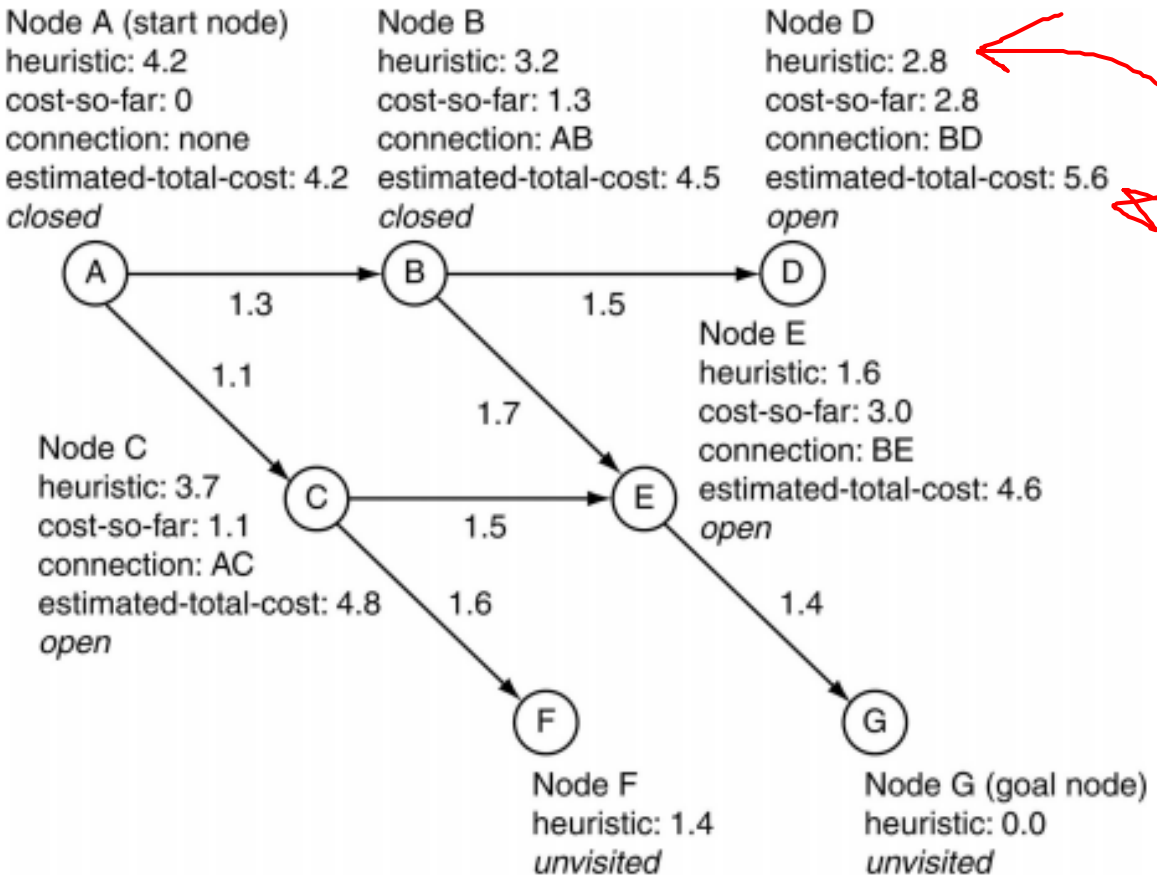
Any path with minimal cost will do

The Algorithm

Informally, the algorithm works in much the same way as Dijkstra does.

- But do NOT explore the open node with the lowest cost-so-far value
- Instead, choose the node that is most likely to lead to the shortest overall path.
- The notion of “most likely” is controlled by a heuristic.
- Heuristic guessed how far from goal the node is
- Each node then has an *estimated-total-cost*

Example


Node A (start node)
heuristic: 4.2
cost-so-far: 0
connection: none
estimated-total-cost: 4.2
closed
Node B
heuristic: 3.2
cost-so-far: 1.3
connection: AB
estimated-total-cost: 4.5
closed
Node D
heuristic: 2.8
cost-so-far: 2.8
connection: BD
estimated-total-cost: 5.6
open
A
B
D
1.3
1.5
1.1
1.7
Node E
heuristic: 1.6
cost-so-far: 3.0
connection: BE
estimated-total-cost: 4.6
open
Node C
heuristic: 3.7
cost-so-far: 1.1
connection: AC
estimated-total-cost: 4.8
open
C
E
1.5
1.6
1.4
F
G
Node F
heuristic: 1.4
unvisited
Node G (goal node)
heuristic: 0.0
unvisited

Differences from Djikstra

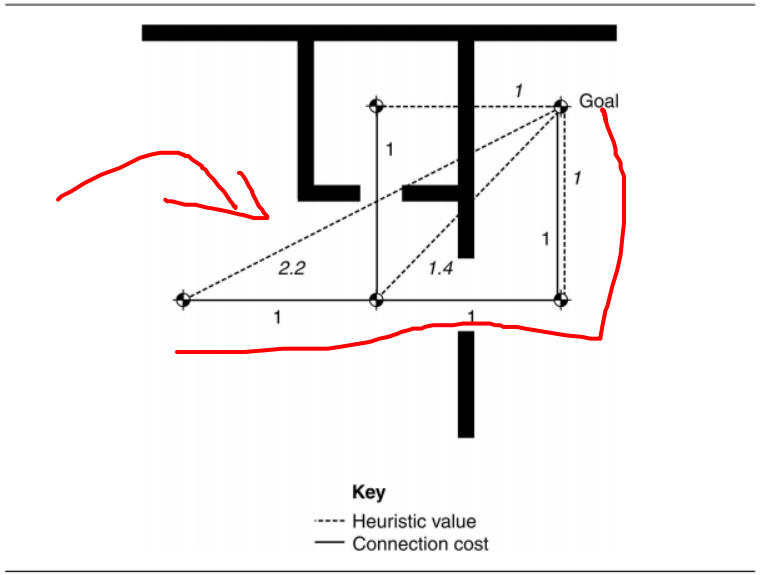
- Each iteration, explore node on Open with lowest estimated-total-cost
- A* can find better routes to nodes that are already on the Closed list.
- Such a node must be moved back to the Open list
- We do terminate when goal is on open and has lowest cost-so-far
  - BUT this may not generate the optimal solution
  - Price paid for faster execution

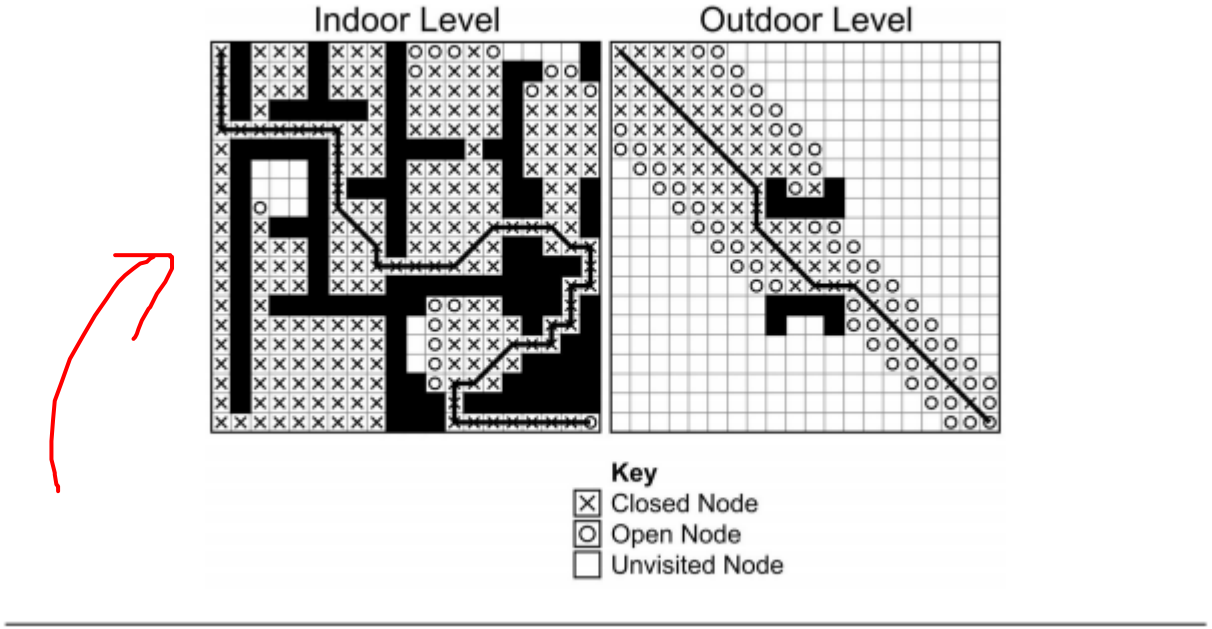
Heuristic Choices

- If heuristic always underestimates, you get best path
  - Possibly greater execution time than overestimating
  - Explores more around start node
- Overestimating can be faster…may not be optimal
  - Need to be sure you overestimate only a little or rarely
  - Error gets out of control, actually slow down search compared to underestimation

Euclidean Distance Heuristic

- Use direct “as-crow-flies” distance as an estimate
- This is an under-estimating heuristic
- Particularly good for outdoor levels

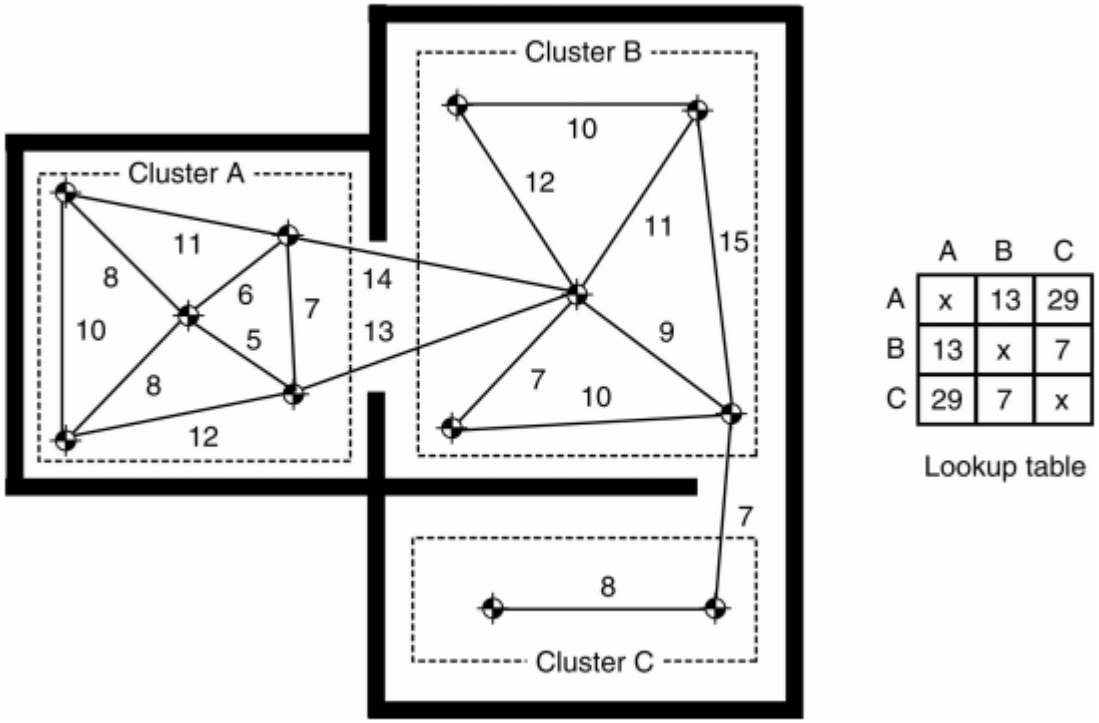




Cluster Heuristic

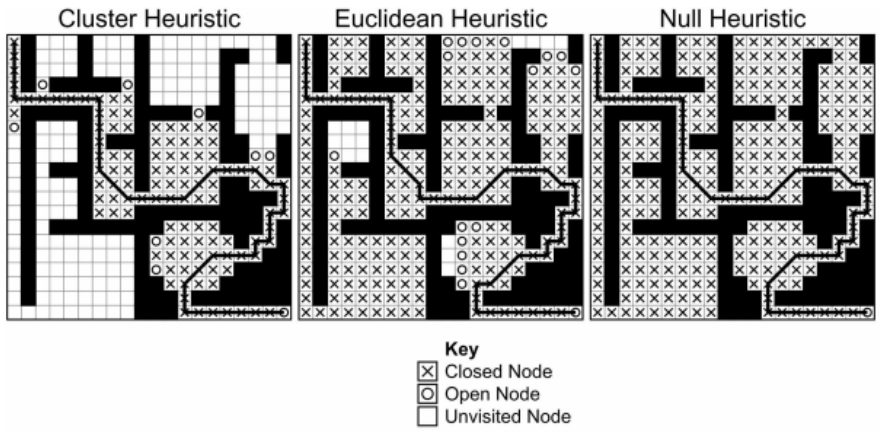
- Group nodes in clusters
  - Can be done automatically using graph clustering algorithm
  - Usually done manually for games
- Create lookup table of shortest path between clusters
  - Pre-process using A* or shortest path
  - Small enough set of clusters to be feasible…small table size
- During game, Euclidean used if start and goal are in same cluster
- Otherwise, use lookup table for estimated distance

Cluster Heuristic Example



	A	B	C
A	x	13	29
B	13	x	7
C	29	7	x

Lookup table



Indoor fill pattern

Probably best choice for indoor levels