CS 440: Introduction to Artificial Intelligence

Lecture 10

Matthew Stone

October 6, 2015

Recap— Search

- Initial state
- Possible actions in each state
- Transition model:
  Takes state and action and gives new state
- Goal test
  Describes whether state is what you want
- Path cost
  Says how easy or hard action sequence is

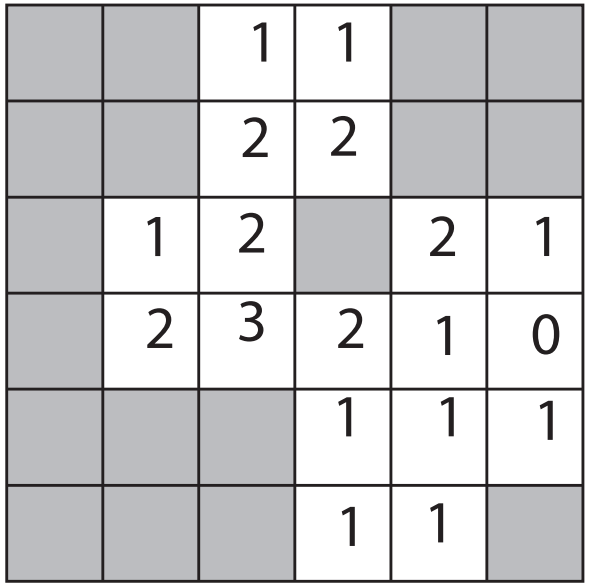
Recap— Search

Many ways to formalize any problem: Want

- few actions at each point
- short paths to solutions
- match with search algorithm

Example

Interpreting Minesweeper Boards



Example

Interpreting Minesweeper Boards

		1	1	X	X
	X	2	2		X
	1	2	X	2	1
	2	3	2	1	0
	X	X	1	1	1
			1	1	X

Formalizing Minesweeper

State:

- board with unexplained counts

Action:

- hypothesize a mine at (x, y)
- nuances to avoid duplication

Transition

- decrement counts adjacent to newly-placed mine

Goal

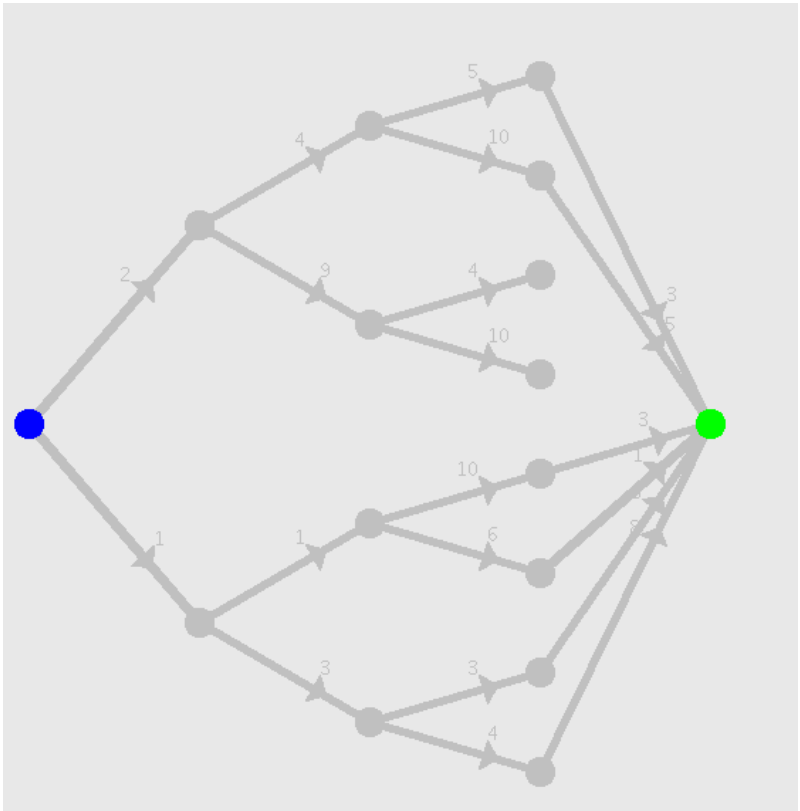
- unexplained counts are 0

Today's Focus: Path Cost

Generalize from steps of cost 1 to steps with positive cost

- Number of steps measures complexity of *constructing* solution
- Path cost measures complexity of *using* solution
- Key insight: These are not necessarily the same

Example



File is bfg.xml

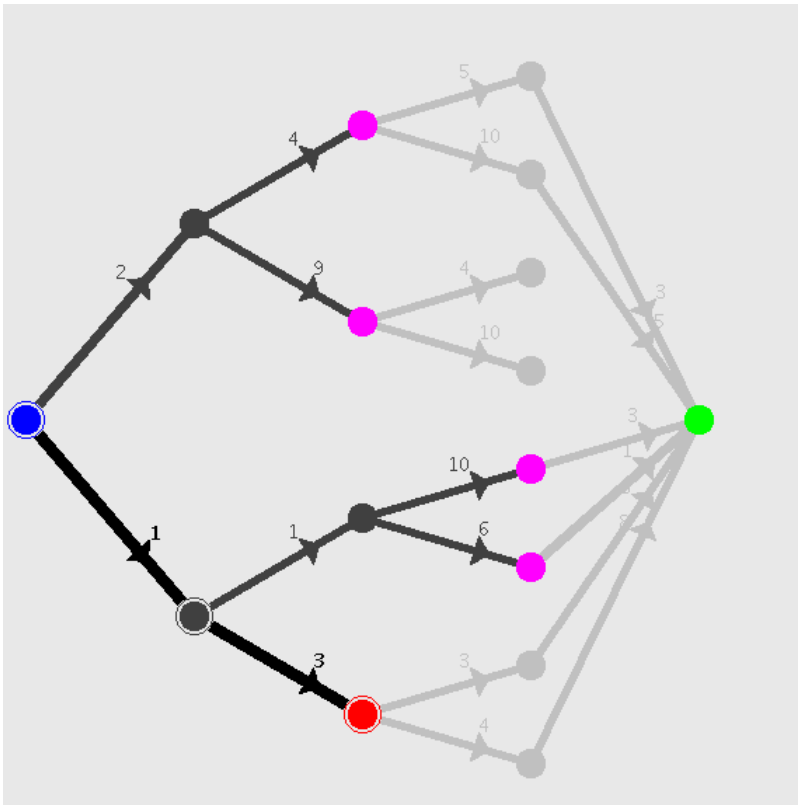
Costs more visible in upcoming slides

Natural first step

Best-first search

- Generalization of breadth-first search
- Sort frontier based on path cost rather than number of steps

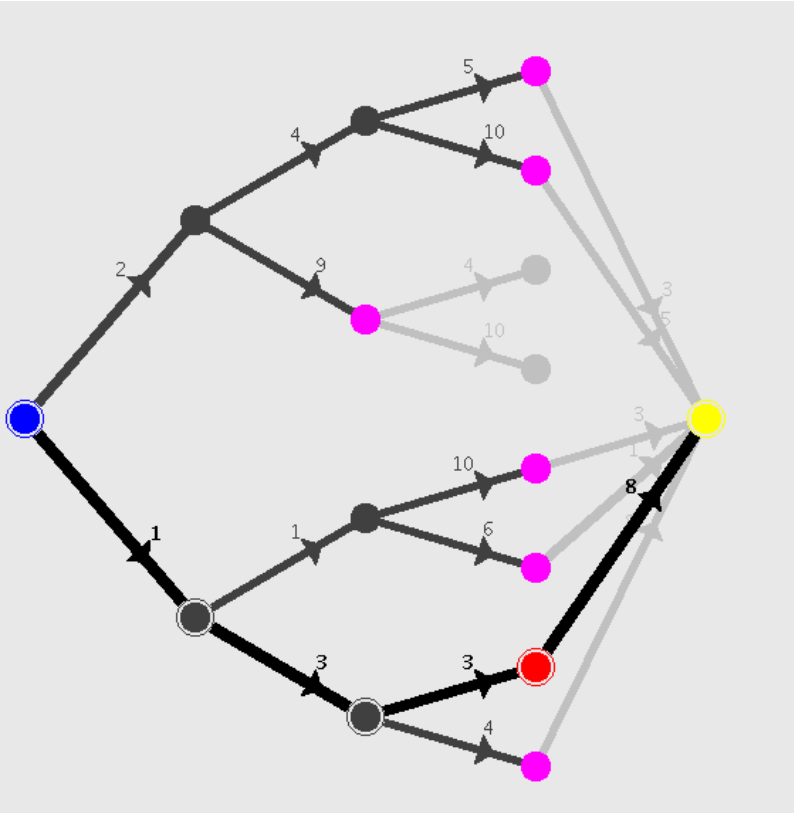
Snapshot



Here we've expanded all nodes of cost 2.
Move to next node, with cost 4.

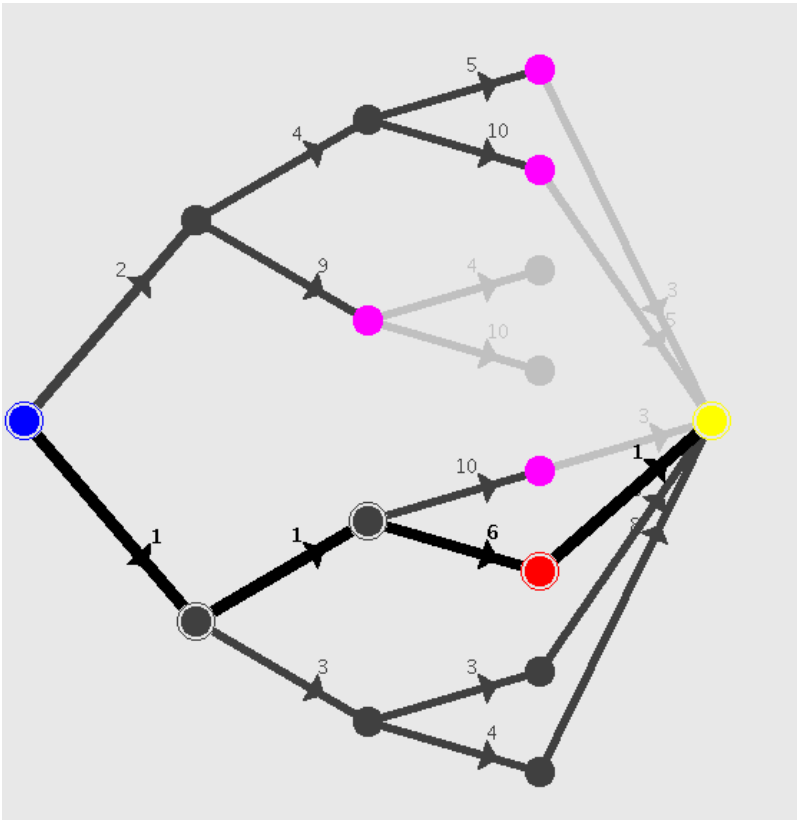
Complication

Solution as first added to frontier...



Complication

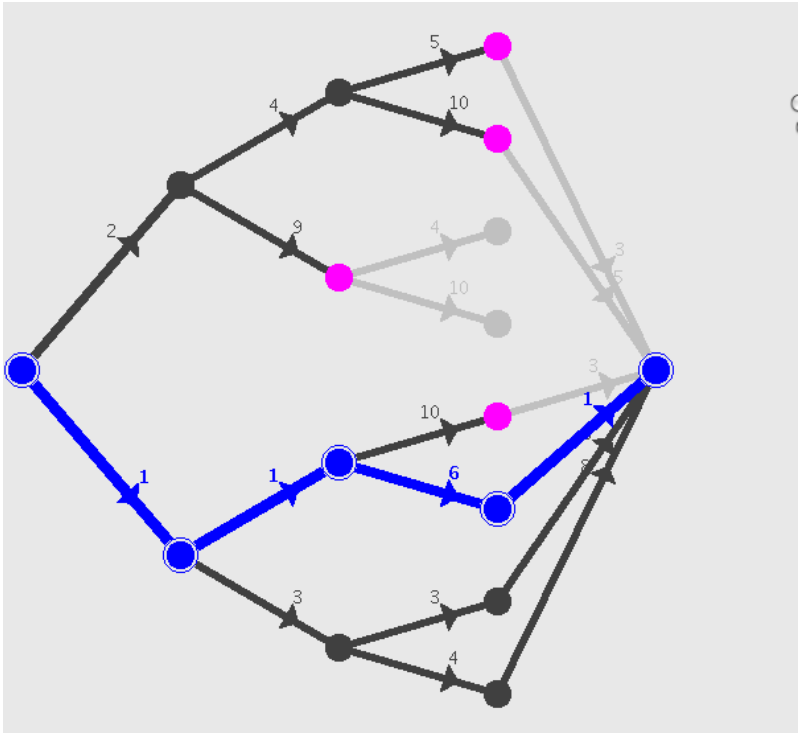
...may not have least-cost path.



Complication

Want to make sure

- search reports least-cost solution
- frontier associates states with least-cost path



Pseudocode - Variables

- *node*: current node being visited
- *frontier*: priority queue of nodes still to visit ordered by heuristic cost
- *explored*: set of nodes already visited

Pseudocode - Initialization

- *node* : initial state of problem, heuristic cost
- *frontier* : queue with *node* as only element
- *explored* : empty set

Pseudocode - Now repeat

- if frontier is empty return failure
- set *node* to result of popping best from *frontier*
- if *node* is a goal, return *node*
- add *node* to explored list
- process *node*'s children

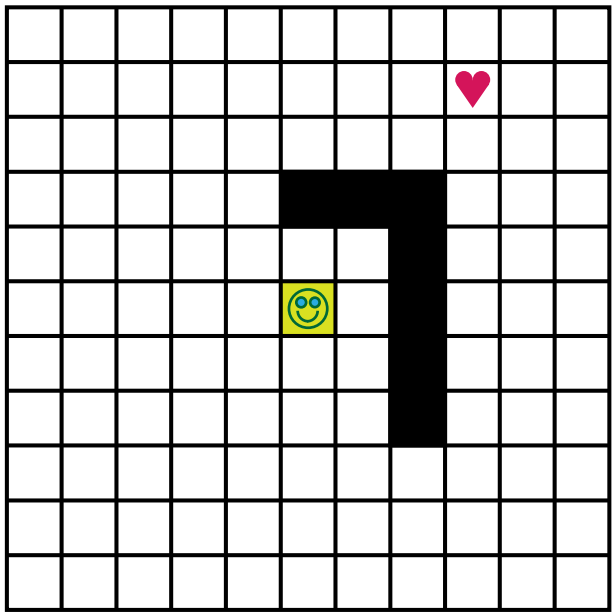
Process node's children

For each possible *action* from *node*

- construct *child* by applying *action* to *node*
- if *child* state is not in *explored* or *frontier* insert *child* on *frontier*
- if *child* state is on *frontier* with higher cost remove old node and add *child* to *frontier*

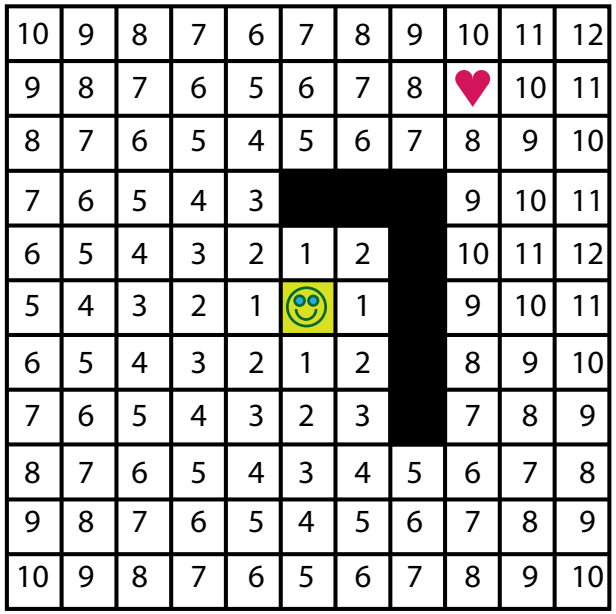
Path planning in games

- Position, destination, { n, s, e, w }



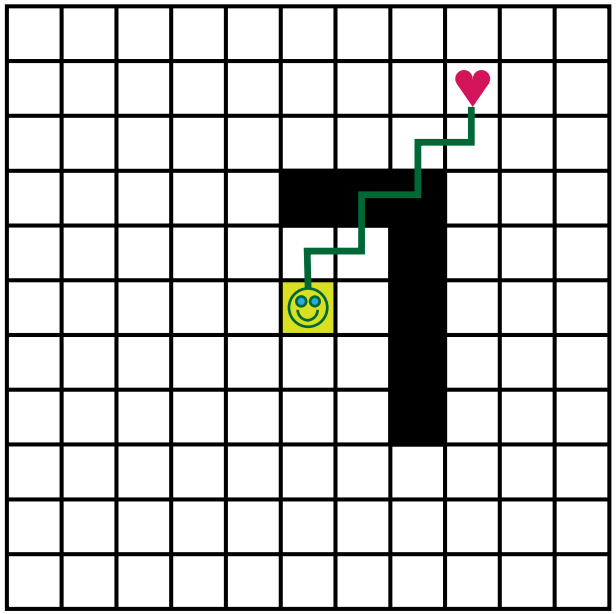
Breadth-first search

- Search by cost so far



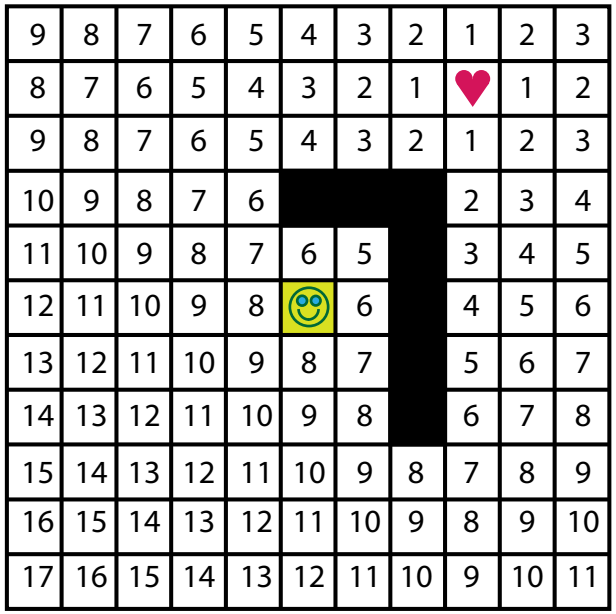
Heuristics and lookahead

- Can make a rough guess at work remaining



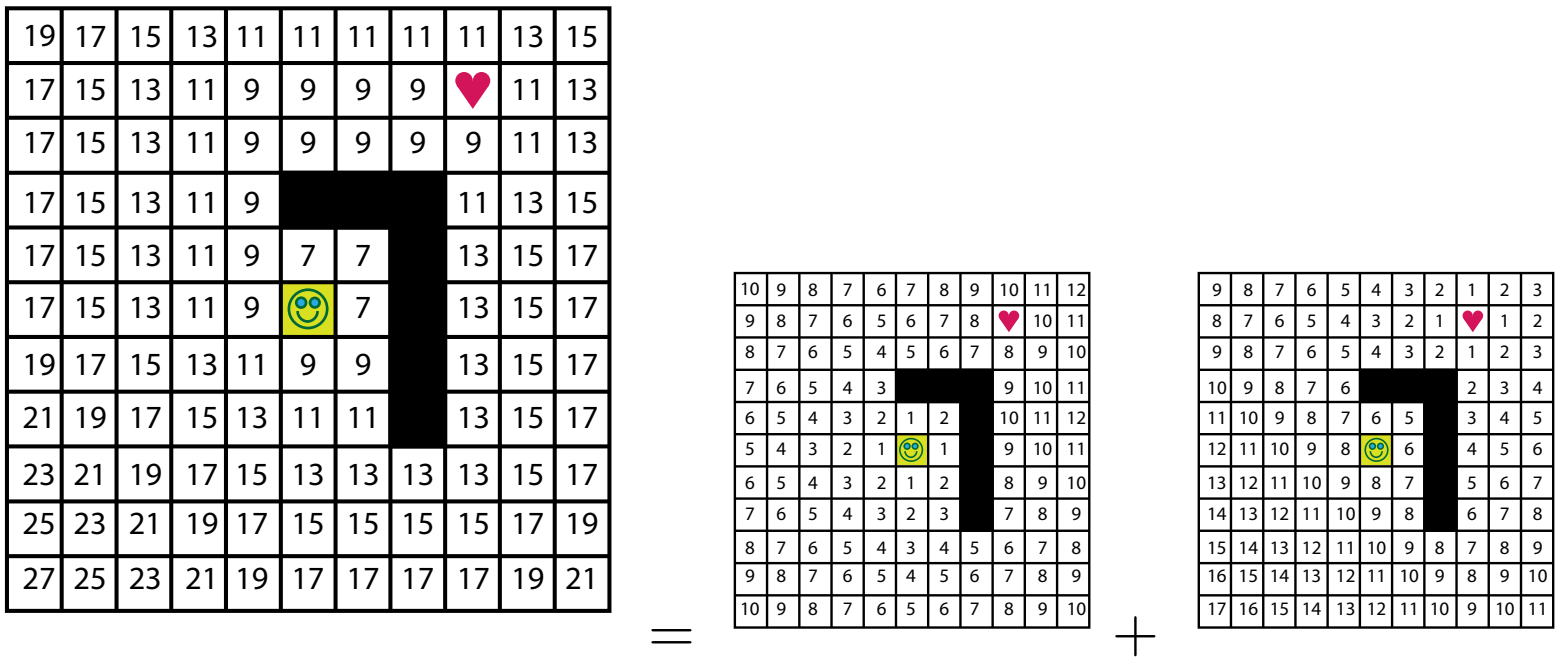
Shape of heuristic function

- Optimistic but useful



Combine cost and heuristic

- Measures



A* Search

Practical variant of heuristic search

- Go-to-method for informed search
- Assumes you have a good way to measure progress

Basic idea: Explore the search node that looks most promising

- Measure progress by actual cost already incurred
- *plus* estimate of cost remaining

Demonstration

http://www.vision.ee.ethz.ch/~cvcourse/astar/AStar.html

Observations about A* Search

- Finds optimal solutions
- Minimal search if you can find a solution without detours
  Just explore zone of optimal solutions
- Search can be exponential in size of smallest detour needed