Pathfinding Search and Adversary Search

CS410/510CS Lecture 6

Bart Massey `<bart@cs.pdx.edu>`
PSU CS Department
February 15, 2001

Overview

- Pathfinding Search
- Adversary Search

Two Topics

Why these two topics together?

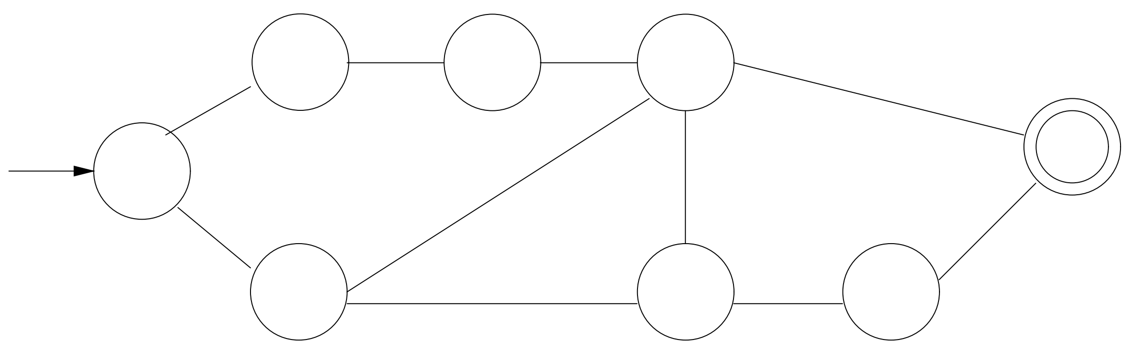
- Amount of material, obviously
- Common themes:
  - Optimization
  - Heuristics
  - BnB pruning
  - Search on graphs

  ...

Pathfinding Search

Concerned with finding *path* through graph

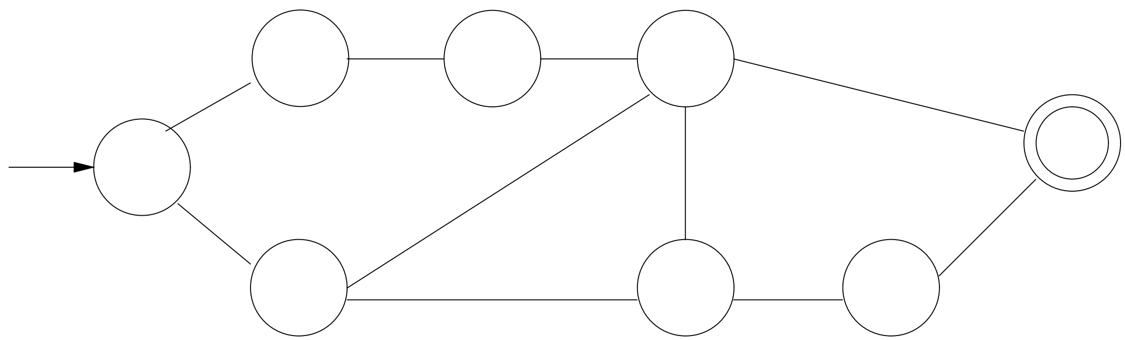
- Source: initial node
- Sink: goal node
- Graph: undirected?



Optimal Pathfinding Problem

Find some *shortest* path

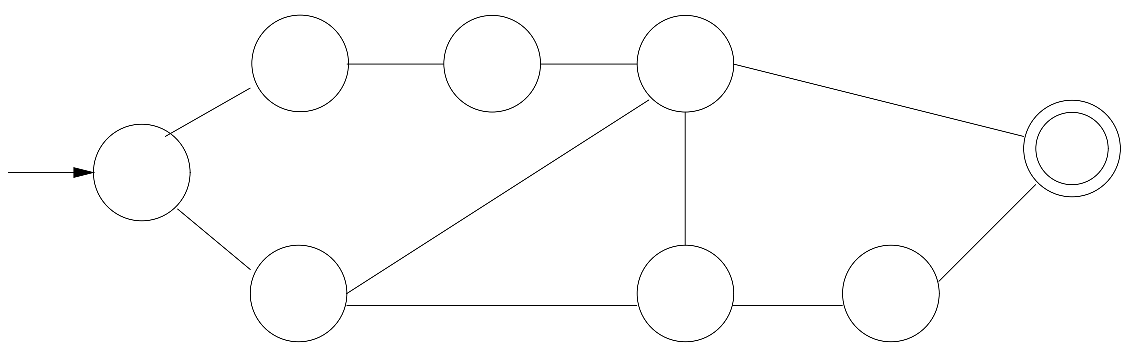
- May be many paths
- Combinatorial graph = *big*



DFS For Pathfinding

Find some path: via DFS

- Requires marking
- In NP: might "get lucky"
- Usual val (not var) order applies



DFS Marking: The Closed List

Two ways to *mark* (= keep track of visited nodes)

- Mark nodes themselves
- Keep set ("list") of *closed* (= marked) nodes

Advantages of closed list

- Can unmark trivially
- Can control space usage

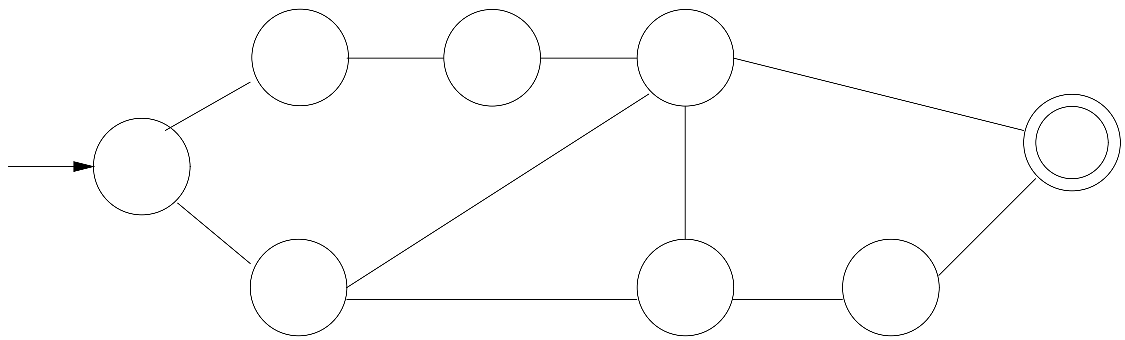
DFS Marking: The Open List

DFS uses (implicit or explicit) *stack*

- Contents is *open* (= to-do) nodes
- Can represent stack as *open list* (= priority-queue)
- Disadvantage: explicitly represents siblings

All implicit data in recursive DFS replaced with transparent data structures

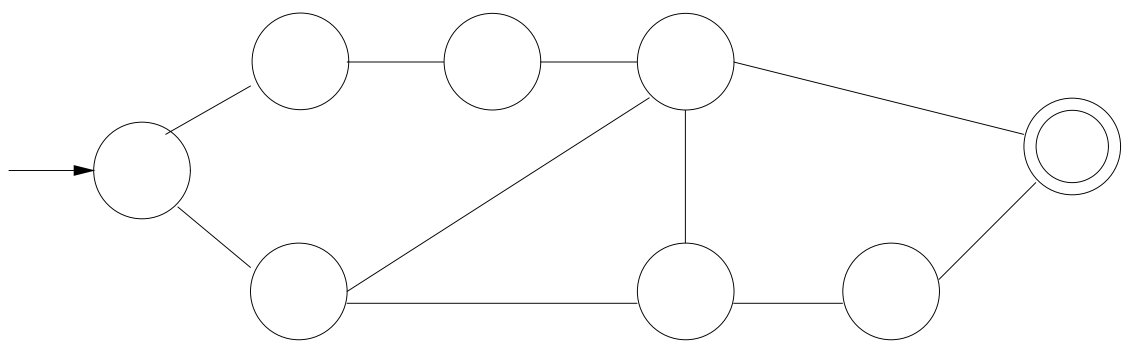
Explicit DFS: Example



BFS For Optimal Pathfinding

For optimal (shortest) pathfinding, use BFS

On undirected graph, must decide "parent" vs. "children"



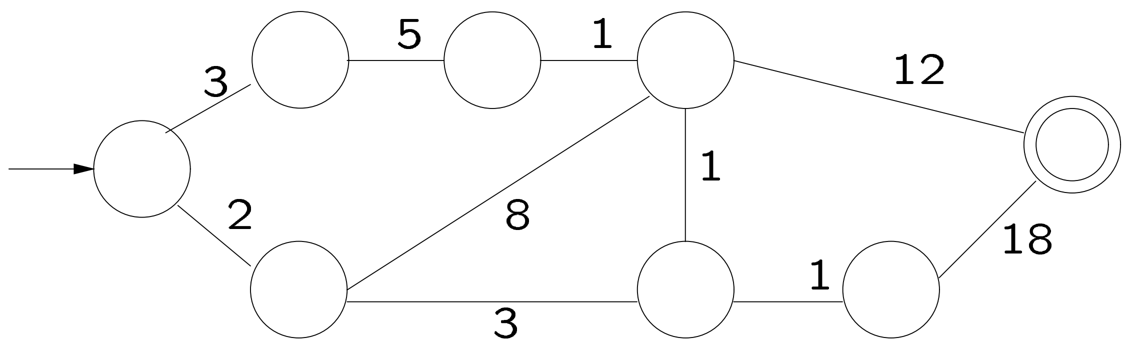
Open and Closed Lists In BFS

- Closed List: Same as DFS
- Open List: Still pqueue—least-recently, not most-recently visited
- BFS, so sizes problematic
- Can reduce open list through ID
- How to reduce closed?

Weighted Optimal Pathfinding

More generally

- Include edge weights
- Normally non-negative



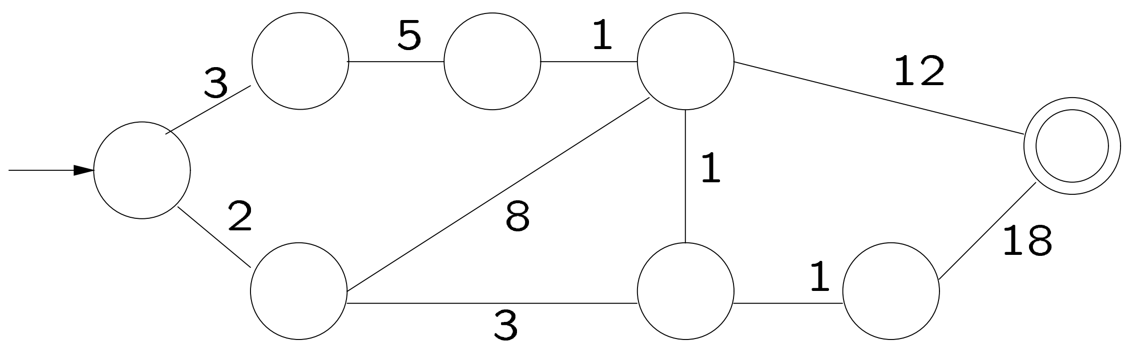
Dijkstra's Algorithm

Best-First Search: Expand open node on shortest path from source

- Open List: nodes pending expansion
- Closed List: expanded nodes
- Nodes can be reopened! Reopen sibling v of u iff path to v through u is shorter
- Must not stop until goal is *closed*

Dijkstra's Algorithm: Example

Keep frontier of nodes on shortest paths until goal closed



Proof: Dijkstra's Algorithm Works

Consider path from source s to node v

- Base case: v is neighbor of s; shortest path chosen
- Inductive case: last link of earliest shortest path is u

  By hypothesis, will find shortest path to u: then reopen neighbors of u. since earliest, will add $s, \ldots, u, v$

CS350: "Polytime" Pathfinding Algorithms

From CS 350: Dijkstra's Algorithm is "$O(|E|)$

Is pathfinding NP hard? Yes, for *combinatorial* problems:

- $|E|$ is $O(2^{\textbf{desc-size}})$
- Is pathfinding *in NP*? In general, no: bounds check might require tracing potentially exponential path!
- Many realistic pathfinding problems have $O(n^k)$ optimal path length bound

Admissible Distance Heuristics

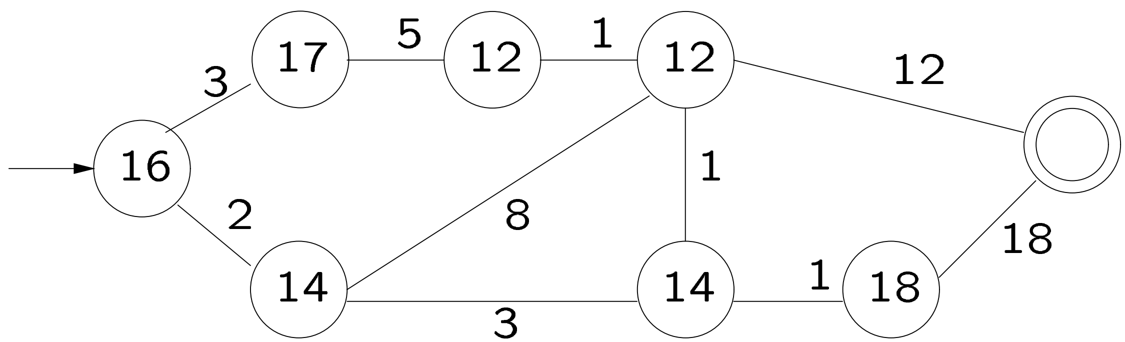
Apply admissible distance heuristic to BFS/Dijkstra? Sure!

Result is $A*$ (Nilsson, Pearl, Korf)

- Basically BnB on graph
- Because of graph/paths, prune may not be permanent
  - Score on open list is still $f(n) = g(n) + h(n)$
  - Node may be *re-opened* if shorter $g(n)$ found

A* Example: Dijkstra + BnB

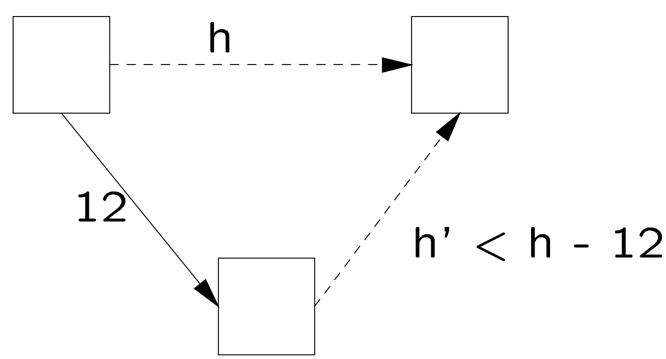
Consider typical lame heuristic $h(n)$



A* and The Triangle Inequality

If heuristic is *monotonic* (= *consistent*, obeys *triangle inequality*), need not reopen!

See handout for proofs



A*: Optimality

No need for fancy proof of correctness. Applying BnB to Dijkstra is safe

Provably time-optimal algorithm! But beware of assumptions

Time behavior in various cases

- DFS-like behavior with perfect heuristic
- BFS-like behavior with 0 heuristic

Iterative Deepening A*

Remove space problem of $A*$ via ID? Maybe

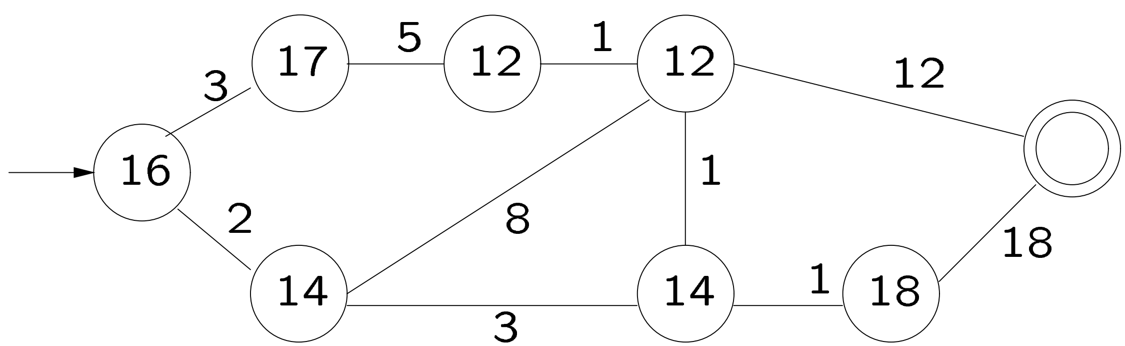
On trees, can do $IDA*$

- Performance tradeoff familiar from ID
- BnB still works

IDA* and Loops

Problem: ID only reduces open list size. Still must keep complete closed list for systematicity

$IDA*$ still *correct* without closed list. But DFS looping problem will dominate with bad heuristic. Tabu?



Tricks For A* and IDA*

- Enforcing monotonicity
- Scaling up heuristic
- Breaking ties: expand best $g(n) + h(n)$ in order of increasing $g(n)$

Pathfinding With Multiple Goals

Everything so far essentially OK with multiple goals. Common real-world case

In CS350, learned "all-pairs shortest-path". But requires too much memory here

Adversary Search

Search algorithms deal with imperfect information due to computational limits. AI question: "Is the world hostile"?

Consider world with hostile, capable adversary. Successful search there should be *robust*

Combinatorial Games

"Search in games" could mean many things

- Pathfinding search on game maps
- Realtime search and strategy
- Optimal move planning for e.g. chess or checkers

We take the last meaning. These games are *combinatorial*: a few simple rules, but many interesting game states

Specific Restrictions

We consider games which are

- Two-player
- "Zero-sum"
- Alternating
- Terminating
- Deterministic
- Perfect Knowledge

Still huge class of problems

The Value Of A Game

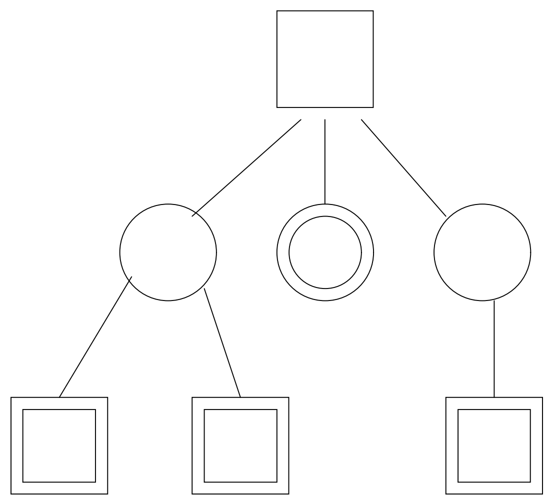
Value of game (state) G to player P is score P can expect for G given perfect play by P and opponent. Usually

- 1: P wins
- -1: P loses
- 0: a tie

If for all reachable states of some G, **value**(G) is tractable, G is "solved": can make perfect move by maximizing **value**(G)

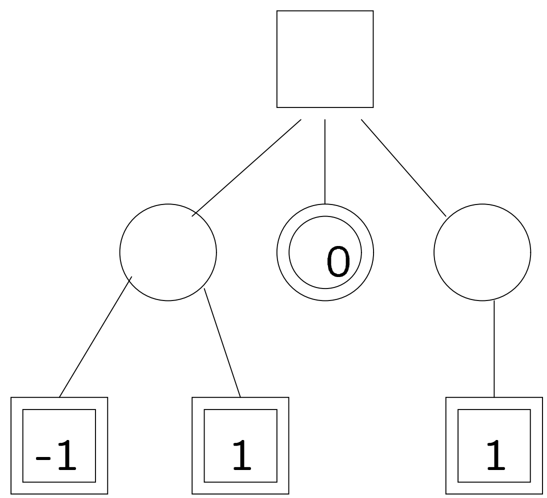
Game Trees

Value of a game state can be computed from value of all subsequent possible states: Game Tree



The Minimax Theorem (von Neumann)

In restricted Combinatorial Game, best move minimizes opponent's expected value



Game Tree Search

Minimax search is

- All solutions (DFS)
- Over combinatorial space
- Intractable

Approximate value? Yes: use heuristic. But hard to build good heuristics...

Soln: depth-limit search (like ID) and use heuristic at leaves!

Game Tree Search Converges

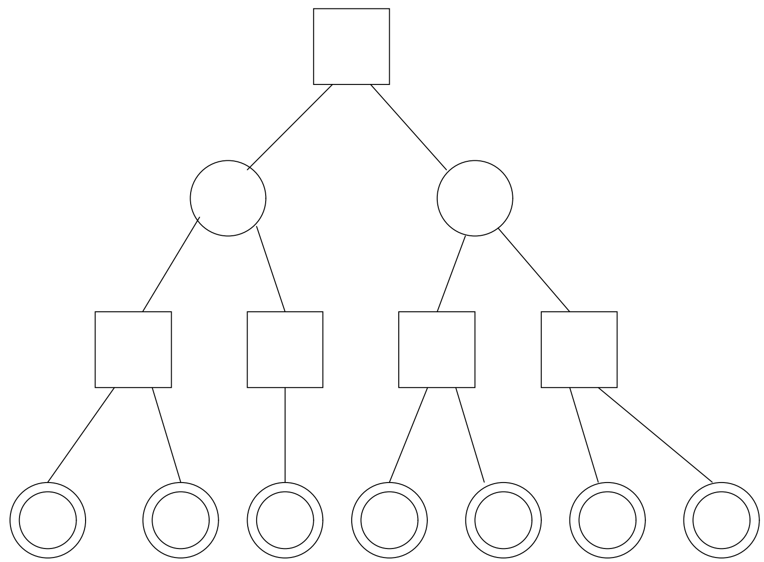
Is depth-limited search with heuristic at leaves better than heuristic at root?

- Intuitively, yes: early mistakes
- Theoretically, yes: proof by Korf
- In practice, yes: e.g. Chess

Searching a "ply" (tree level) deeper uniformly means (in practice) uniformly better idea of value of root

alpha-beta Pruning

Like BnB on depth-limited game tree. But adversary (= all-solutions) limits pruning



Additional Notes On alpha-beta

- Pruning depends on value ordering heuristic quality
  - Perfect pruning = double depth
  - Random pruning = 30
- Other search strategies are available
- Historically, enabled decent play

Game Tree Search and Closed Lists

Game states are graph, not tree! Build closed list, "transposition table"

- Stops looping
- Prevents re-search
- With random replacement, does OK for memory usage

Game Tree Search and Iterative Deepening

Why do ID? Planning to search anyhow...

- "Anytime" property: want to have backed up move when time expires
- Can use backed up values from transposition table for value ordering: dramatically improves $\alpha\beta$ performance

alpha-beta + ID + Transposition Tables

$\alpha\beta$ works very well with iterative deepening and transposition tables

- Transposition table remembers estimated values
- Get more accurate as search deepens
- Cause more $\alpha\beta$ pruning
- Increases search depth

More Advanced Methods, Tougher Problems

More Advanced Methods

- “Zero-window” methods
- MTD(f)

Tougher Problems

- Probability
- Hidden Information
- Nonconstant-Sum
- Multiplayer