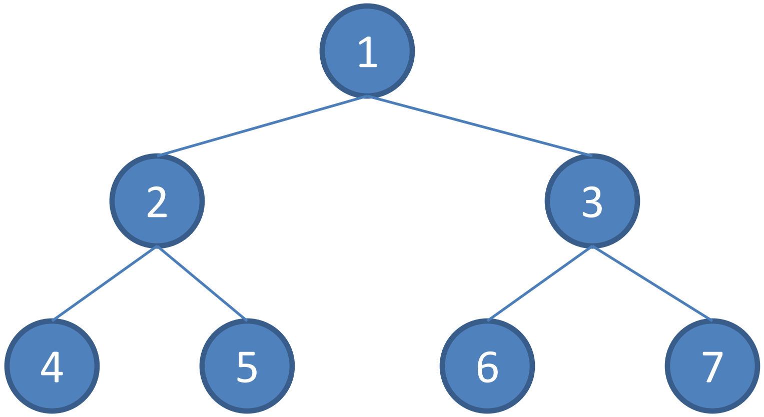
Breadth-first heuristic search

Paper by Rong Zhou, Eric A. Hansen

Presentation by Salomé Simon

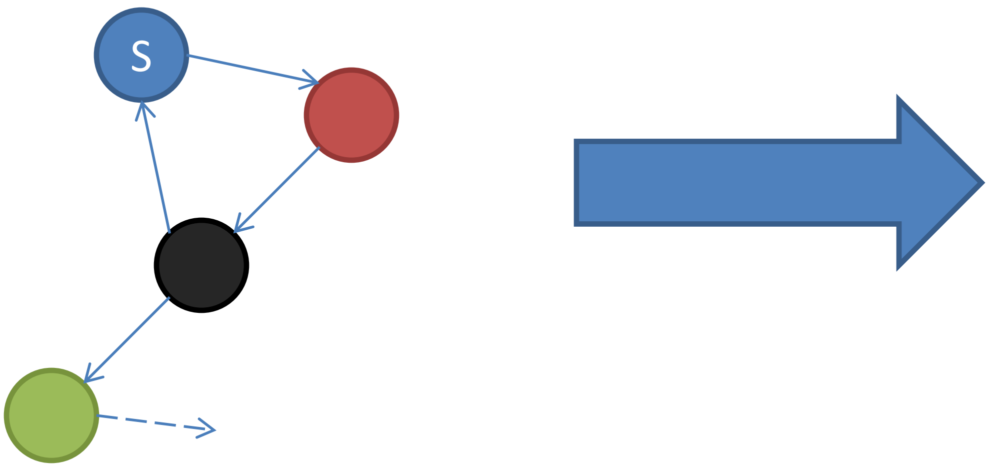
Breadth-first tree search

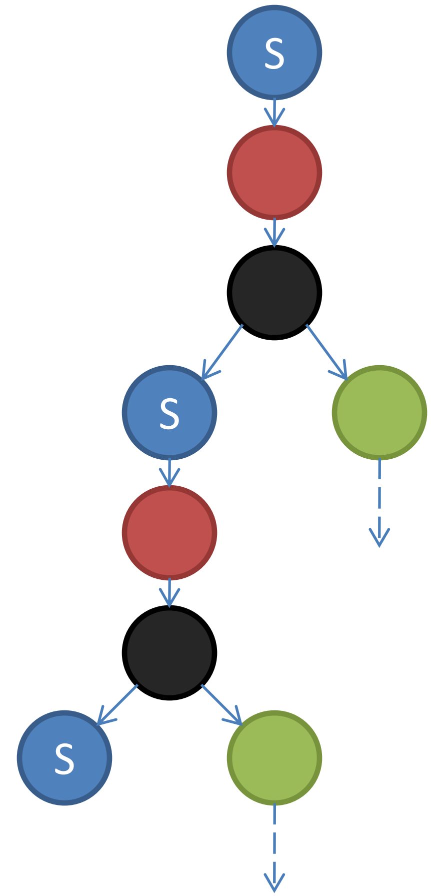


- **Used for search problems with uniform edge cost**
  - Prerequisite for presented techniques
- Drawback: all nodes need to be stored

Why store all nodes?

- Solution path reconstruction
- Duplicate detection

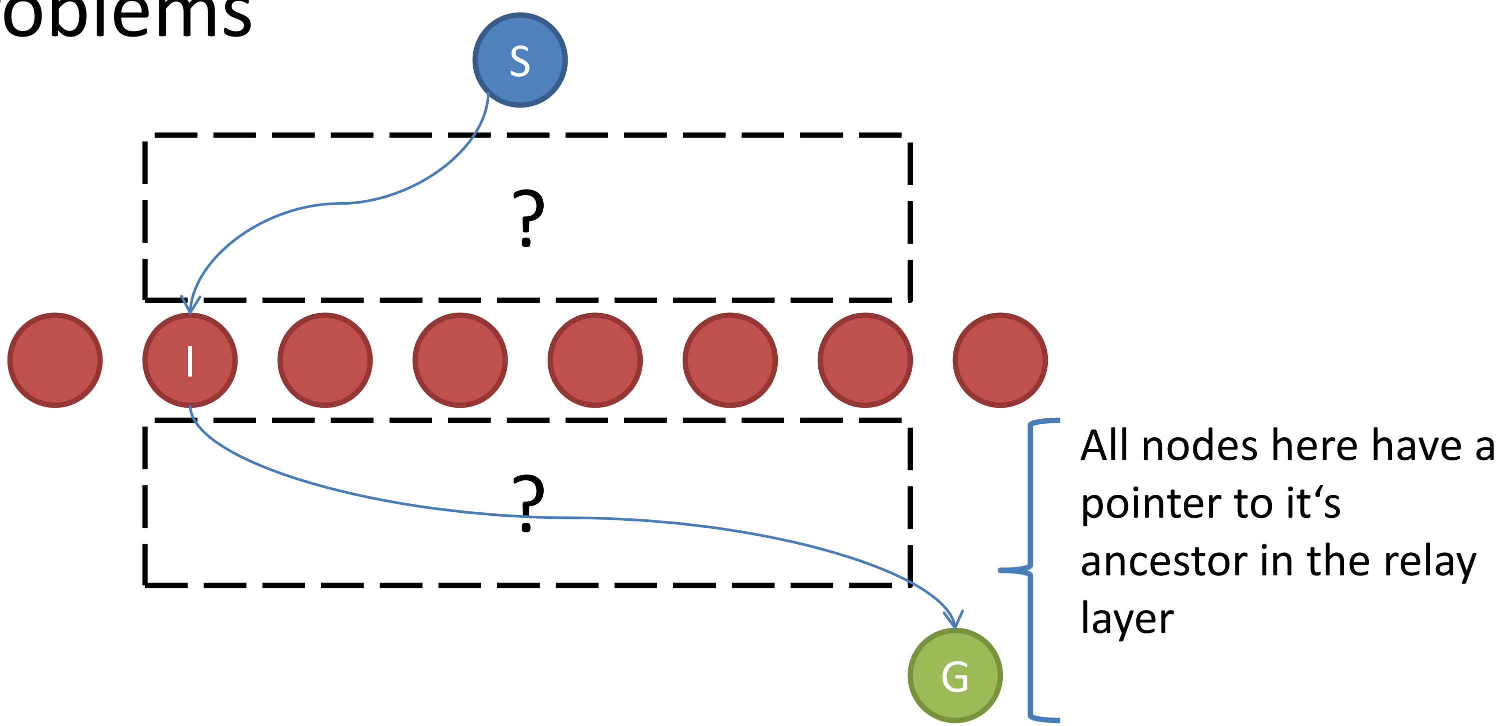




- Depth-first search can do this with a stack

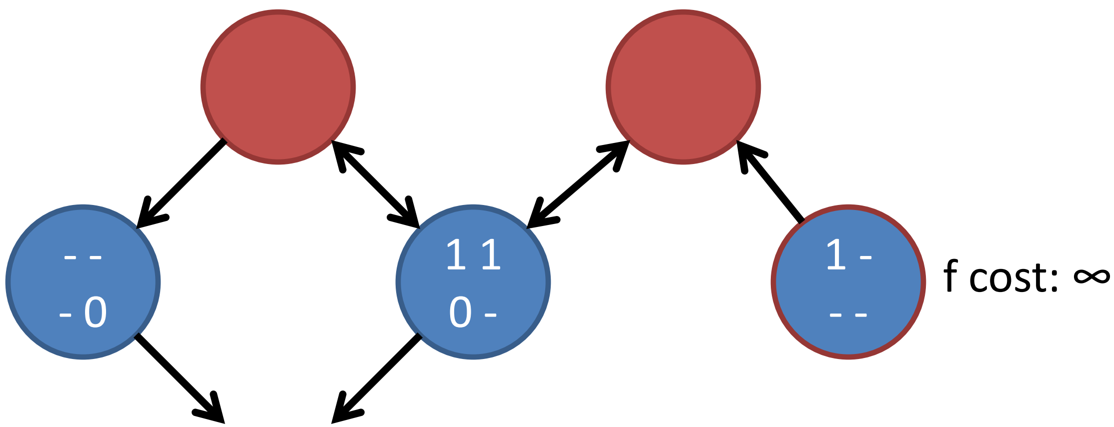
Memory efficient solution path recovery

- Divide and conquer principle
- Store one „relay layer“, recursively solve sub-problems



Memory efficient duplicate detection

- Frontier search: used operator bits
  - No closed list
  - Cannot use upper bound for pruning



- Sparse memory: counter for predecessors
  - Only nodes with counter ≠ 0 in closed list
  - Better for high branching factor

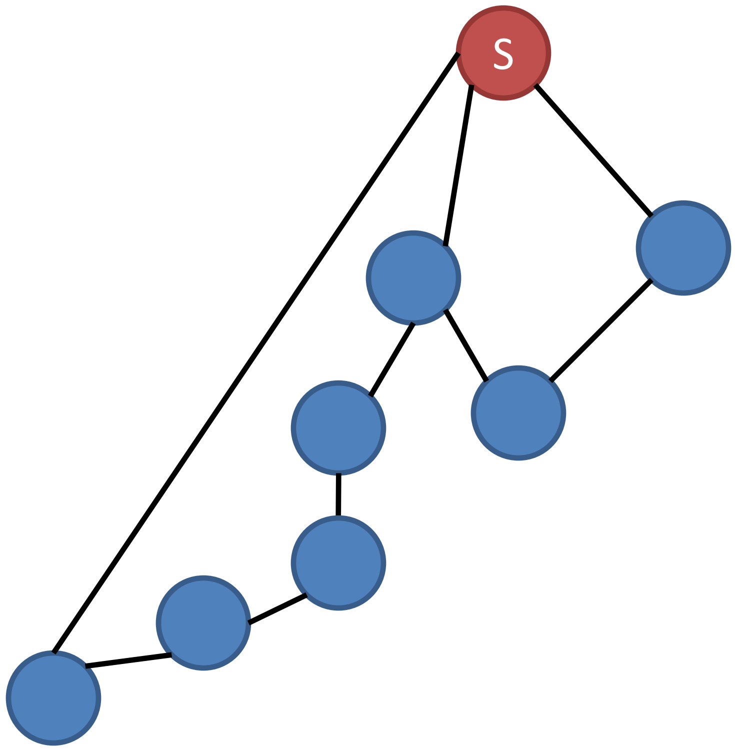
Memory efficient duplicate detection

- Drawbacks
  - Need to know predecessors
  - A* needs to have a consistent heurisitc

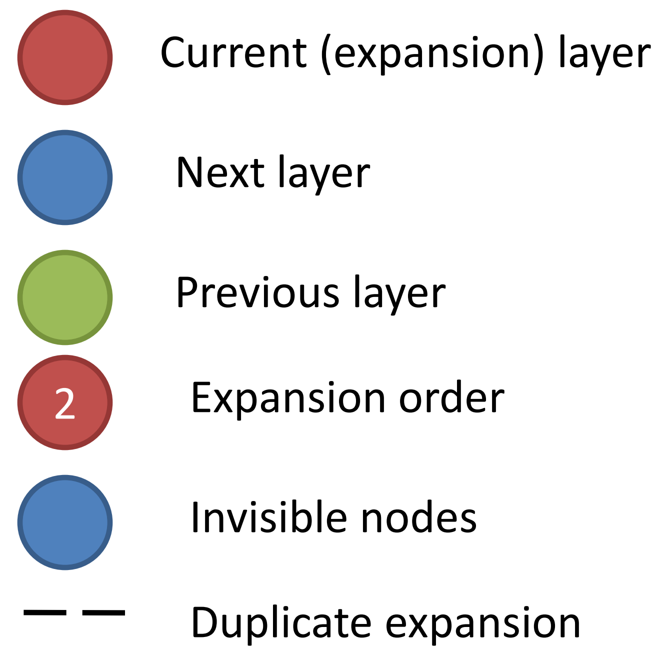
Layered duplicate detection

- Only usable for breadth-first search
- Open list: current and next layer
- Closed list: current and x previous layers
- For undirected graphs (& uniform edge cost): only one previous layer needs to be stored
- For directed graphs: Max g-cost difference between optimal g-cost of predecessor and sucessor (hard to determine)
  - But only linear regeneration when one layer is saved

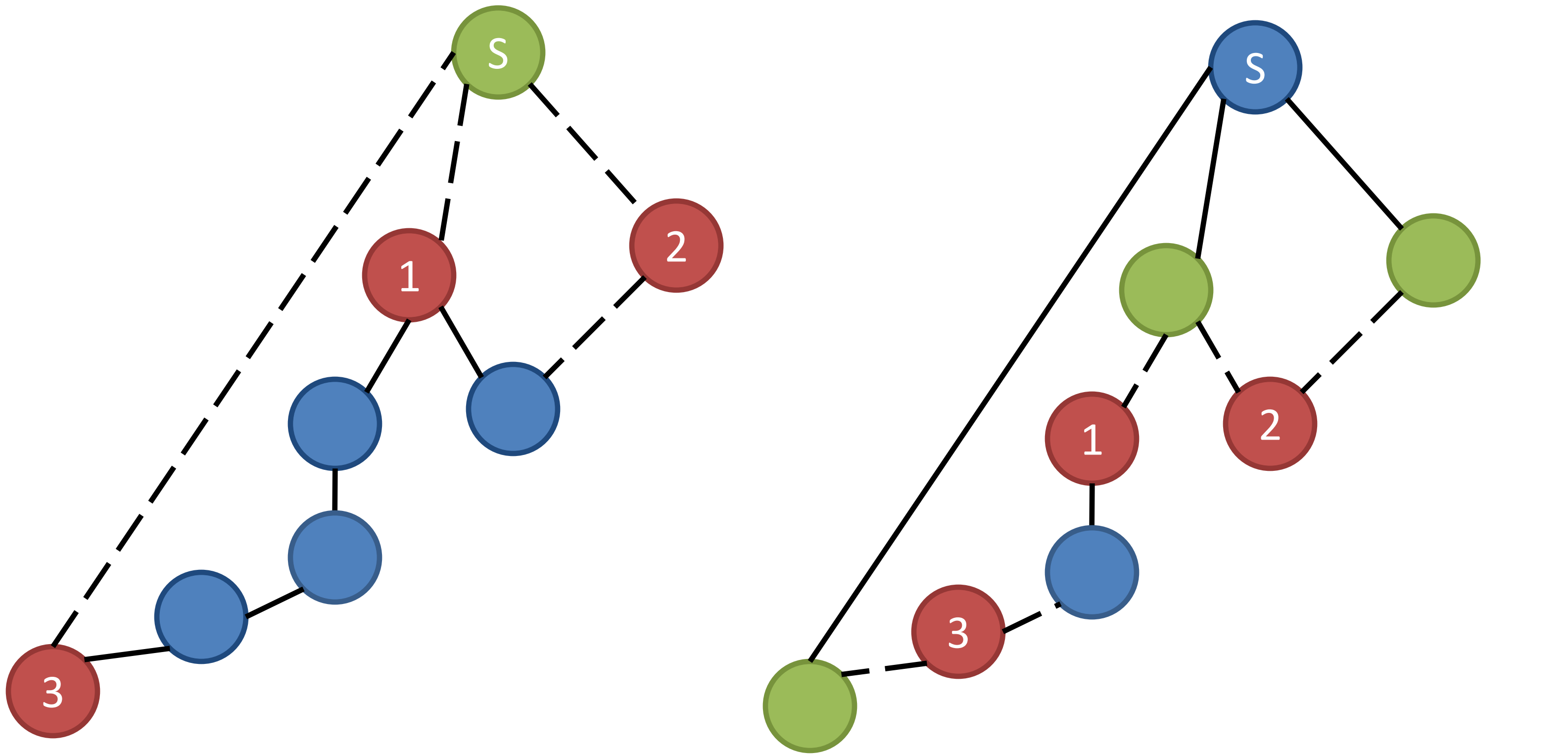
Layered duplicate detection: Example



Legend:



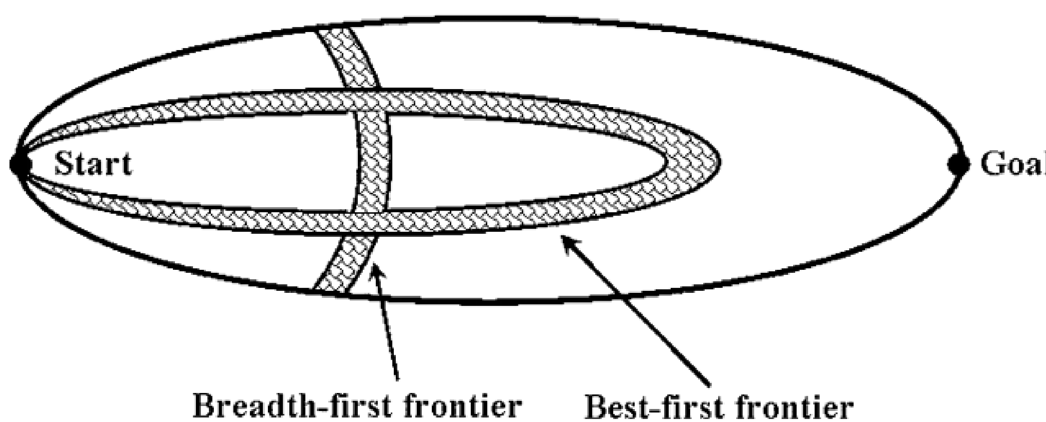
Layered duplicate detection: Example



Advantages of heuristic breadth-first search

- Frontier size is smaller (no proof):
  - Breadth-first #layers: f* + 1
  - Best first #layers: f* - h(start) + 1

  ≈ more layers means smaller layer size (if perfect upper bound)



Advantages of breadth-first search

- No sorting (FIFO)
- Frontier search
  - Can prune nodes above upper bound, since optimal g(n) is found once the node is expanded
    - → works also with admissible but not consistent heuristic
  - Easier memory allocation since no sorting needed
- Layered duplicate detection
  - Only for breadth-first search
  - Easiest to implement

Breadth-first branch-and-bound (BFBnB)

- Lower bound: $f(n) = g(n) + h(n)$
- Upper bound for pruning unpromising paths
  - With perfect upper bound BFBnB expands the same nodes as A* (disregarding ties), else more
- Solution path recovery: divide-and-conquer
  - Relay layer at 3/4 depth, since 1/2 are usually the biggest layers (more pruning later)
- Duplicate detection:
  - Frontier search
  - Sparse memory
  - Layered duplicate detection

Breadth-first iterative deepening A* (BFIDA*)

- BFBnB with iterative deepening
- Gives perfect upper bound
  - Useful when no good upper bound can be estimated
- Asymptotically optimal for node expansions
  - Even in directed graphs under certain conditions
- Does not use tie breaking (not beneficial)
  - Tie breaking only useful in last layer, but for breadth-first this layer is rather small

Results: Fifteen Puzzle

- Used algorithm: frontier-BFIDA*
  - Low branching factor
- 4 GB memory limitation
  - Frontier- / sparse-memory-A* only solved 96/100
  - Maximal memory usage by BFIDA*: 1.3 GB

	Frontier A*	Sparse memory A*	Frontier BFIDA*
#nodes stored	4.15x	2.74x	1x
Peak Memory	6.2x	4.1x	1x

Results: Fifteen Puzzle

- DFIDA* performs best
  - More nodes expanded (not all duplicates detected), but
  - Lower node-generation overhead

Results: 4-peg Towers of Hanoi

- Used algorithm: Frontier-BFBnB
  - A (probably) perfect upper bound can be found
- 2 GB memory limitation
  - Only BFBnB could solve 19-disk problem
- Inconsistent heuristic
  - Frontier-A* is not guaranteed to find optimal solution

Results: 4-peg Towers of Hanoi

Disks	Frontier A* Nodes Stored	Frontier A* Expanded	Frontier BFBnB Stored	Frontier BFBnB Expanded
17	2‘126‘885	10‘398‘240	390‘844	11‘628‘818
18	25‘987‘984	202‘577‘805	6‘987‘695	211‘993‘782
19	> 128‘000‘000	> 1‘193‘543‘025	55‘241‘327	1‘824‘553‘083

Results: Domain independent STRIPS planning

- Used algorithm: BFBnB with layered duplicate detection
  - easier to implement when not knowing the domain
  - even in domains with directed graphs one stored layer gives good results
- Compared to A* and DFIDA*
- BFBnB expands more nodes than A*, but uses significantly less memory
- DFIDA* performs poor due to excessive node regeneration

Results: General observations

- Compared to A*, Breadth-first has
  - **Significantly less memory usage**
  - More node expansions (especially if upper bound not perfect)
- Layered duplicate detection easy to implement and gives good results