

COMP9414: Artificial Intelligence

Informed Search

Wayne Wobcke

Room J17-433
wobcke@cse.unsw.edu.au
Based on slides by Maurice Pagnucco

Overview

- Heuristics
- Informed Search Methods
 - ▶ Best-First Search
 - ▶ Greedy Search
 - ▶ A* Search
 - ▶ Iterative Deepening A* Search
- Conclusion

Informed (Heuristic) Search

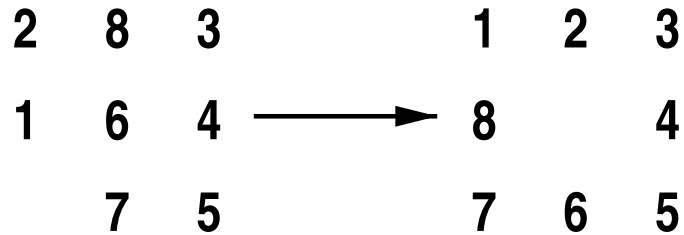
- We have seen that uninformed methods of search are capable of systematically exploring the state space in finding a goal state
- However, uninformed search methods are very inefficient in most cases
- With the aid of problem-specific knowledge, informed methods of search are more efficient
- References:
 - ▶ Ivan Bratko, [Prolog Programming for Artificial Intelligence](#), Fourth Edition, Pearson Education, 2012. (Chapter 12)
 - ▶ Stuart J. Russell and Peter Norvig, [Artificial Intelligence: A Modern Approach](#), Third Edition, Pearson Education, 2010. (Chapter 3)

Heuristics

- Heuristics are “rules of thumb”
- “Heuristics are criteria, methods or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal.”
Judea Pearl, [Heuristics](#)
- Can make use of heuristics in deciding which is the most “promising” path to take during search
- In search, heuristic must be an underestimate of actual cost to get from current node to goal — an [admissible heuristic](#)
- Denoted $h(n)$; $h(n) = 0$ whenever n is a goal node

Heuristics — Example

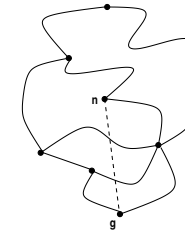
- 8-Puzzle — number of tiles out of place



- Therefore, $h(n) = 5$

Heuristics — Example

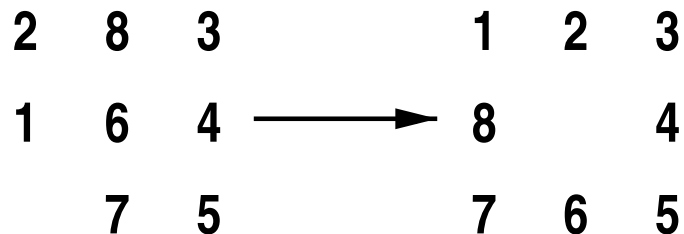
- Another common heuristic is the straight-line distance (“as the crow flies”) from node to goal



- Therefore, $h(n) = \text{distance from } n \text{ to } g$

Heuristics — Example

- 8-Puzzle — Manhattan distance (distance tile is out of place)



- Therefore, $h(n) = 1 + 1 + 0 + 0 + 0 + 1 + 1 + 2 = 6$

Best-First Search

function BestFirstSearch(problem, EvalFn) **returns** solution sequence

inputs: problem — a problem

EvalFn — evaluation function

QueueingFn ← function that orders nodes by EvalFn

return GeneralSearch(problem, QueueingFn)

Greedy Search

- **Idea:** expand node with the smallest estimated cost to reach the goal
- Use heuristic function $h(n)$ to select node from frontier for expansion (i.e. use $h(n)$ as **EvalFn** in Best-First Search)

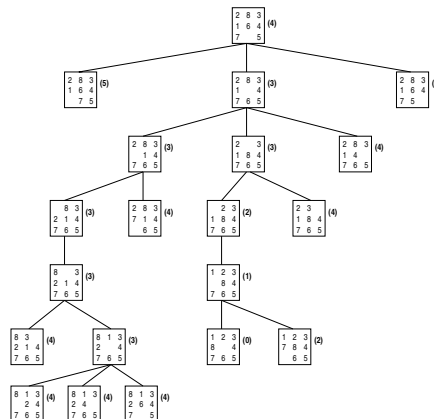
Greedy Search — Analysis

- Similar to depth-first search — tends to follow a single path to goal
- Not optimal, incomplete
- Time $O(b^m)$; Space $O(b^m)$
- However, good heuristic can reduce time and space complexity significantly

A* Search

- **Idea:** Use both cost of path generated thus far and estimate to goal to select node from frontier to expand
- $g(n)$ = cost of path from start node to n
- Accomplished by using an evaluation function $f(n) = g(n) + h(n)$
- $f(n)$ represents the estimated cost of the cheapest solution passing through n
- Expand node from frontier with smallest f -value
- Essentially combines uniform-cost search and greedy search

Greedy Search



A* Algorithm

OPEN — nodes on frontier; CLOSED – expanded nodes

OPEN = $\{\langle s, nil \rangle\}$

while OPEN is not empty

remove from OPEN the pair $\langle n, p \rangle$ with minimal $f(n)$

place $\langle n, p \rangle$ on CLOSED

if n is a goal node **return** success (path p)

for each edge e connecting n and n' with cost c

if $\langle n', p' \rangle$ is on CLOSED **then if** $p \oplus e$ is cheaper than p'

then remove n' from CLOSED and put $\langle n', p \oplus e \rangle$ on OPEN

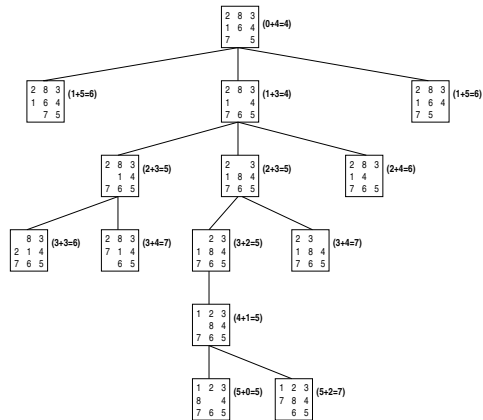
else if $\langle n', p' \rangle$ is on OPEN **then if** $p \oplus e$ is cheaper than p'

then replace p' with $p \oplus e$

else if n' is not on OPEN **then** put $\langle n', p \oplus e \rangle$ on OPEN

return failure

A* Search



```
% extend path by each node in Succs and insert into Paths in order of costs
extend_path([], _, _, Paths, Paths).           % no more paths

extend_path([S|Succs], Path, Cost, Paths, Paths2) :-
    Path = [LastNode_|_],
    s(LastNode, S, C),
    NewCost is Cost + C,                    % g(S) = NewCost
    h(S, H),                               % h(S) = estimate S->goal
    NewEst is NewCost + H,                 % f(S) = estimate of node's value
    insert([S|Path], NewCost, NewEst, Paths, Paths1),
    extend_path(Succs, Path, Cost, Paths1, Paths2).

% insert path in order in the list of paths
insert(Path, Cost, Estimate, [], [[Path, Cost]]).

insert(Path, Cost, Estimate, Paths, [[Path, Cost]|Paths]) :-
    Paths = [Path1|_],
    Path1 = [[Last1|_], Cost1],
    h(Last1, H),
    Estimate1 is Cost1 + H,
    Estimate1 > Estimate.                  % new path is better

insert(Path, Cost, Estimate, [Path1|Paths], [Path1|Paths1]) :-
    insert(Path, Cost, Estimate, Paths, Paths1).
```

A* Search

```
% A-Star Best-First Search. A simplified version of Bratko Fig. 12.3.
% Store paths and costs in a list of pairs [Path, Cost], ordered according to f(n)
% where f(n) = g(n) + h(n) with g(n) the path cost and h(n) the heuristic cost from n to a goal
% Nodes in a path are listed in reverse order, i.e. list [t,g,f,e,s] represents path s->e->f->g->t
% To search from the a node Start for a solution path SolPath, call ?- solve(Start, SolPath)

solve(Start, SolPath) :-
    astar([[[Start], 0]], SolPath).

astar([BestPath|Paths], Path) :-
    BestPath = [Path, Cost],
    Path = [LastNode|_],
    goal(LastNode).

astar([BestPath|Paths], SolPath) :-
    BestPath = [Path, Cost],
    Path = [LastNode|_],
    extend(Path, Cost, Paths, NewPaths),
    astar(NewPaths, SolPath).

% extend best path by successors of last node

extend(Path, Cost, Paths, NewPaths) :-
    Path = [LastNode_|_],
    findall(S, s(LastNode, S, _), Succs),
    not(Succs = []),
    extend_path(Succs, Path, Cost, Paths, NewPaths).
```

A* Search — Analysis

- Optimal (optimally efficient)
- Complete
- Number of nodes searched (and stored) still exponential in the worst case
- It has been shown that this will be the case unless the error in the heuristic grows no faster than the log of the actual path cost

$$|h(n) - h^*(n)| \leq O(\log h^*(n))$$
- For many heuristics, this error is at least proportional to the path cost!

Admissibility of the A* Algorithm

- We can impose conditions on graphs and the heuristic function $h(n)$ to guarantee that the A* algorithm, applied to these graphs, always finds an optimal solution
- **Admissibility** basically means an optimal solution is found (provided a solution exists) and it is the first solution found
- Conditions on state space graph
 - ▶ Each node has a finite number of successors
 - ▶ Every arc in the graph has a cost greater than some $\epsilon > 0$
- Condition on heuristic $h(n)$
 - ▶ For every node n , the heuristic never overestimates (i.e. $h(n) \leq h^*(n)$ where $h^*(n)$ is the actual cost from n to the goal — $h(n)$ is an **optimistic estimator**)

Proof of the Optimality of the A* Algorithm

Let G be an optimal goal state and let the optimal path cost be f^*

Let G_2 be a “worse” goal state (i.e. $g(G_2) > f^*$) but suppose A* selects G_2 from the frontier for expansion and returns the path to G_2

Then there must be a node n on the frontier that is on an optimal path to G such that A* chose G_2 rather than n : that is $f(G_2) \leq f(n)$

Since G_2 is a goal state, $f(G_2) = g(G_2)$, so this means $g(G_2) \leq f(n)$

Now $f(n) = g(n) + h(n)$ and since h is admissible, $f(n) \leq g(n) + h^*(n)$, where $h^*(n)$ is the true cost of reaching a goal state from n

But $g(n) + h^*(n)$ is f^* , the optimal path cost to G , so we have $g(G_2) \leq f^*$

This contradicts the hypothesis that G_2 is “worse” than G (i.e. $g(G_2) > f^*$)

Hence such a G_2 cannot exist and so A* returns an optimal path

Admissibility of the A* Algorithm

- A* is **optimally efficient** for a given heuristic: of the optimal search algorithms that expand search paths from the root node it can be shown that there is no other optimal algorithm that will expand fewer nodes in finding a solution
- **Monotonic** heuristic — along any path, the f -cost never decreases
- Common property of admissible heuristics
- If this property does not hold then we can use the following “trick” to modify the path cost for a node n' which is a child of node n (Pathmax Equation) $f(n') = \max(f(n), g(n') + h(n'))$

Completeness of the A* Algorithm

- Let G be an optimal goal state
- The only way that A* won't reach a goal state is if there are infinitely many nodes where $f(n) \leq f^*$
- This can only occur if there is a node with infinite branching factor or there is a path with a finite cost but with infinitely many nodes
- The former is taken care of by our first condition on the state graph
- The fact that arc costs are above some $\epsilon > 0$ means that $g(n) > f^*$ for some node n , taking care of the latter

Heuristics — Properties

- We say h_2 **dominates** h_1 iff $h_2(n) \geq h_1(n)$ for any node n
- A* will expand fewer nodes on average using h_2 than h_1
- We observe that every node for which $f(n) < f^*$ will be expanded
Which means n is expanded whenever $h(n) < f^* - g(n)$
But since $h_2(n) \geq h_1(n)$, any node expanded using h_2 will be expanded using h_1
- It is always better to use a heuristic with higher (underestimating) values
- Suppose you have identified a number of non-overestimating heuristics for a problem $h_1(n), h_2(n), \dots, h_k(n)$
Then $\max_{i \leq k} h_i(n)$ is a more powerful non-overestimating heuristic
- Therefore can design a whole range of heuristics to trap special cases

Generating Heuristics

- Admissible heuristics can be derived from the **exact** solution cost of a **relaxed** version of the problem
- If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**, then #tiles-out-of-place gives the shortest solution
- If the rules are relaxed so that a tile can move to **any adjacent square**, then Manhattan distance gives the shortest solution
- For TSP: let path be **any** structure that connects all cities
 \implies minimum spanning tree heuristic

Iterative Deepening A* Search

- IDA* performs repeated depth-bounded depth-first searches as in Iterated Deepening, however the bound is based on $f(n)$ instead of depth
- Start by using f -value of initial state
- If search ends without finding a solution, repeat with new bound of minimum f -value exceeding previous bound
- IDA* is optimal and complete with the same provisos as A*
- Due to depth-first search, space complexity = $O(\frac{bf^*}{\delta})$ (where δ = smallest operator cost and f^* = optimal solution cost) — often $O(bd)$ is a reasonable approximation
- Another variant — SMA* (Simplified Memory-Bounded A*) — makes full use of memory to avoid expanding previously expanded nodes

Conclusion

- Informed search makes use of problem-specific knowledge to guide progress of search
- This can lead to a significant improvement in the performance of search
- Much research has gone into admissible heuristics
- Even on the automatic generation of admissible heuristics